Special issue on New Frontiers in Quantitative Methods in Informatics

Guest Editorial

Emiliano Casalicchio
Blekinge Institute of Technology
Karlskrona, Sweden
emiliano.casalicchio@bth.se

Salvatore Distefano
Università di Messina
Messina, Italy
Kazan Federal University
Kazan, Russia
sdistefano@{unime.it,kpfu.ru}

This special issue of *Performance Evaluation Review* collects eight papers selected from the proceedings of the workshop *InfQ2016 - New Frontiers in Quantitative Methods in Informatics* held in Taormina, Italy, on October 2016.

The InfQ workshop has been launched in 2010 by the Italian group on Quantitative Methods in Informatics as a forum for the presentation of all aspects of quantitative modeling and analysis of computer and communication system properties (e.g. performance and dependability). The past successful editions (Pisa '10, Lipari '11, Lucca '12, Sorrento '13 and Torino '14) were intended as national conferences. Starting from this year, the organizers have decided to also welcome contributions from researchers working in other countries and those whose interest in performance and dependability is derived from experience in other fields. From the proceedings of this year we selected eight full papers that have been extended with at least the 30% of new and unpublished contents. All the selected research works adopt quantitative methods and techniques to study complex systems and networks in challenging application fields, specifically: *Big Data architectures and applications* (Cardellini et al. and Gianniti et al.), *Cloud computing* (Canali et al., Longo et al. and Bianchi et al.), *Internet–of–Things (IoT)* (Donatiello et al. and Pinciroli et al.) and *Bioinformatics and systems biology* (Totis et al.).

In the area of Big data, *Cardellini et al.* study the problem of optimizing the performance of Data Stream Processing (DSP) applications that can be replicated and placed on multiple distributed computing nodes, to process the incoming data flow in parallel. The authors formulate the optimal DSP replication and placement as an integer linear programming problem. Predicting the performance of Big Data frameworks can be a costly task, hence the necessity to provide models that can be a valuable support for designers and developers. *Gianniti et al.* propose a novel modeling approach based on fluid Petri nets to predict MapReduce and Spark applications execution time.

Cloud computing is still a challenging research application area targeted in *Longo et al.* by proposing general steps and metrics to quantify the resiliency of large scale systems (e.g. Infrastructure-as-a-Service Cloud). The approach is formalized as a set of four algorithms that can be applied when large scale systems are modeled through stochastic state space-based models. In the same area, *Canali et al.* address the problem of network-aware virtual machine placement in software-defined Cloud data centers. Their work proposes a methodology to infer virtual machines (VM) communication patterns starting from the input/output network traffic time series of each VM and without relaying on a special purpose monitoring. Similarly, *Bianchi et al.* study how Infrastructure Providers can optimize the allocation of the virtual network requests into a substrate network while satisfying QoS requirements. The authors propose a two-stage virtual network embedding algorithm with delay and placement constraints.

Sensor networks are an enabling technology for IoT, investigated in *Donatiello et al.* to provide mechanisms to extend the lifetime of a barrier coverage sensor network deployed for target detection. The authors consider a scenario where sensors are randomly dropped on a bidimensional field to detect target traversals which occur in a stochastic way within a critical mission time. Another hot topic in the IoT domain is Mobile Crowdsensing (MCS), a contribution-based paradigm involving mobiles in pervasive application deployment and operation. *Pinciroli et al.* propose a new technique for the evaluation of relevant performance and energy figures of merit for MCS systems based on queuing networks including stations with variable number of servers to model the contribution dynamics.

A main aspect in computational modeling of biological systems is the identification of the model structure and parameters. *Totis et al.* propose an approach which overcomes model indetermination solving an optimization problem with an objective function that, similarly to Flux Balance Analysis, is derived from an empirical biological knowledge and does not require large amounts of data.

We would like to thank many people who contributed to the success of InfQ 2016 and of this special issues: the steering committee for their precious support and suggestions, the publicity chairs for their valuable contribution in advertising the workshop, the Technical Program Committee members for the time and effort that they have put into the paper selection process and the anonymous Reviewers which supported us in the special issue review process. These efforts make the InfQ2016 conference and this special issue possible.

Optimal Operator Replication and Placement for Distributed Stream Processing Systems

Valeria Cardellini
cardellini@ing.uniroma2.it
Vincenzo Grassi
vincenzo.grassi@uniroma2.it
Francesco Lo Presti
lopresti@info.uniroma2.it
Matteo Nardelli
nardelli@ing.uniroma2.it
Department of Civil Engineering and Computer Science Engineering
University of Rome Tor Vergata, Italy

ABSTRACT

Exploiting on-the-fly computation, Data Stream Processing (DSP) applications are widely used to process unbounded streams of data and extract valuable information in a near real-time fashion. As such, they enable the development of new intelligent and pervasive services that can improve our everyday life. To keep up with the high volume of daily produced data, the operators that compose a DSP application can be replicated and placed on multiple, possibly distributed, computing nodes, so to process the incoming data flow in parallel. Moreover, to better exploit the abundance of diffused computational resources (e.g., Fog computing), recent trends investigate the possibility of decentralizing the DSP application placement.

In this paper, we present and evaluate a general formulation of the optimal DSP replication and placement (ODRP) as an integer linear programming problem, which takes into account the heterogeneity of application requirements and infrastructural resources. We integrate ODRP as prototype scheduler in the Apache Storm DSP framework. By leveraging on the DEBS 2015 Grand Challenge as benchmark application, we show the benefits of a joint optimization of operator replication and placement and how ODRP can optimize different QoS metrics, namely response time, internode traffic, cost, availability, and a combination thereof.

1. INTRODUCTION

The ever increasing diffusion of cheap sensors and the presence of an almost ubiquitous Internet connectivity is producing an exponential growth in the amount of daily produced data, which carry valuable information about our surrounding environment. Indeed, by extracting valuable information from noisy Big Data, it is possible to develop new intelligent and pervasive services that can improve our everyday life in several domains, e.g., healthcare, energy management, logistic, and transportation. The volume and velocity of daily produced data have fostered the development of new processing approaches that rely on the usage of multiple computing machines. Nowadays, two opposite approaches are commonly adopted: batch processing and stream processing. The former stores all the data, usually on a distributed file system, and then operates on them on the basis of different programming models, among which the well-known MapReduce [8]. The latter processes all the data on-the-fly, i.e., without storing them, so it can produce results in a near real-time fashion. Therefore, Data Stream Processing (DSP) applications are widely used to process unbounded streams of data and timely extract valuable information. We focus on this kind of applications.

A DSP application is represented as a directed acyclic graph (DAG), with data sources, operators, and final consumers as vertices, and streams as edges. Each *operator* can be seen as a black-box processing element that continuously receives incoming streams, applies a transformation, and generates new outgoing streams. In the Big Data era, DSP applications should be capable to seamlessly process huge amount of data, which require to scale their execution on multiple computing nodes, because a single machine cannot provide enough processing power. To this end, these applications usually exploit *data parallelism*, which consists in increasing or decreasing the number of parallel instances for the operators, so that each instance can process a subset of the incoming data flow in parallel (e.g., [12, 16]). Moreover, since data sources can be geographically distributed, the execution of DSP applications can also take advantage of the ever increasing presence of distributed Cloud and Fog computing resources, which can improve the system scalability and reduce latency by moving the computation towards the network edge, closer to data sources. Nevertheless, the use of such distributed infrastructure poses new challenges, that include network and system heterogeneity, geographic distribution as well as non-negligible network latencies [23].

This paper focuses on the deployment of DSP applications over distributed computing nodes. Specifically, we investigate and evaluate Optimal DSP Replication and Placement (**ODRP**) [3], a unified general formulation of the operator replication and placement problem. Differently from most works in literature [9, 20, 21, 24], ODRP can jointly determine the application placement and the replication of its operators, while optimizing the Quality of Service (QoS) attributes of the application. With respect to our previous work [3], in this paper we describe the integration of ODRP as prototype scheduler in Apache Storm, an open-source and widely used DSP framework. Moreover, we extensively evaluate the prototyped solution in a real setting with the aim of highlighting its strengths and drawbacks. To this end, we

have designed and implemented a benchmark DSP application that solves the DEBS 2015 Grand Challenge [17].

This paper is organized as follows. We review related work in Section 2; in Section 3 we describe the system model and the problem under investigation, before presenting ODRP and its integration, as prototype scheduler, in Apache Storm in Sections 4 and 5, respectively. Then, in Section 6, relying on the Storm-based prototype and the benchmark application that addresses the DEBS 2015 Grand Challenge, we show the benefits of a joint optimization of replication and placement and how ODRP can optimize different QoS metrics. Finally, we conclude in Section 7.

2. RELATED WORK

To deploy a DSP application, a DSP system needs to determine the replication degree of the application operators and their placement on the computing infrastructure. Even though these problems have been widely investigated separately, only few works study their joint optimization.

Placement and Replication Problem. Most works in literature consider DSP operator placement and operator replication as independent and orthogonal decisions, where the operator placement is first carried out without determining the optimal number of replicas for each operator. Then, in response to some performance deterioration, the operators to be replicated and their new replication degree are identified. This two-stage approach requires to reschedule the DSP application in order to take the new application configuration into account and may incur in a significant overhead. In this paper, we propose a *single-stage* approach to determine both the placement and the parallelism degree of the operators in a DSP application. The DSP placement problem has been widely investigated in literature under different modeling assumptions and optimization goals, e.g., [4, 9, 24]. In [4], we proposed a general formulation of the optimal DSP placement which takes into account the heterogeneity of computing and networking resources and which encompasses the different solutions proposed in the literature. In [3] we extended that formulation so as to determine the optimal number of replicas for each operator contextually to their placement on the underlying infrastructure; in this paper, we integrate such formulation in a Storm-based prototype and evaluate it using a real application.

Since the placement problem is NP-hard, several heuristics have been proposed, e.g. [1, 22, 26]. They aim at minimizing a diversity of utility functions, such as the DSP application end-to-end latency, the inter-node traffic, and the network usage. Our problem formulation can be adjusted to take into account these different utility functions.

Many research efforts have focused on scaling the amount of operator replicas in response to changes observed in some monitored performance metric. Some works, e.g., [5, 15], exploited threshold-based policies based on the utilization of either the system nodes or the operator instances. Other works, e.g., [12, 20, 21], used more complex policies to determine the scaling decisions. Lohrmann et al. [20] proposed a strategy that enforces latency constraints by relying on a predictive latency model based on queueing theory. Mencagli [21] presented a game-theoretic approach where the control logic is distributed on each operator.

An important issue related to operator replication regards the management of stateful operators, since their state must be migrated in order to preserve the application integrity [12]. The approach we propose in this paper jointly places and replicates operators, thus saving the overhead and latency penalty incurred by stateful operator migrations in case of disjointed replication and placement decisions.

The works most closely related to ours have been presented by Heinze et al. [14, 16]. They proposed a model to estimate the latency spike created by a set of operator movements and used it to define an operator placement algorithm based on a bin packing heuristic that minimizes the latency violations and focuses only on the placement of the newly added operators. We present an optimal problem formulation that targets the initial placement decision and can be used to benchmark existing heuristics.

While most works, including ours, focus on replicating the operators at the level of the application logic, thus changing the graph topology, Fu et al. [11] proposed a queueing theory approach to determine the number of computing resources on which each operator is placed; however, their approach does not consider network delays.

DSP Frameworks. A great variety of DSP frameworks has been proposed so far; being interested in integrating our ODRP model, we focus mainly on the open-source ones. Apache Storm [25] is a DSP framework that provides an abstraction layer to execute event-based applications. It allows the user to merely focus on the application logic, while the effort of placing, distributing, and executing the application is handled by the framework. Several research efforts have used Storm to either evaluate new operator placement algorithms in a real environment or to propose some architectural improvements (e.g., [1, 13, 19, 25, 26]). Developed as the successor of Storm, Heron [18] preserves its abstraction layer while introducing some architectural improvements. Apache Spark [28] is a general-purpose framework for large-scale processing, which provides a batch and micro-batch processing approach. This latter alternative is throughput-oriented, whereas Storm, which is a pure DSP system, can further minimize the application latency and thus can be preferred in latency sensitive scenarios. Another emerging framework is Apache Flink[1], which provides a unified solution for batch and stream processing.

Aside the specific functionalities, these open-source frameworks use directed graphs to model DSP applications; therefore, our ODRP formulation well represents their placement problem and can be integrated into their scheduler. As regards the operator replication, so far these frameworks leave completely to the user the definition of the number of replicas that have to be instantiated. Nevertheless, since the user might over-/under-estimate the incoming load, this approach can lead to a sub-optimal utilization of the available resources (i.e., over-/under-provisioning). Since several placement policies [1, 2, 6, 10, 22, 26] and seminal replication policies [5] have been already integrated into Storm, we have also chosen it to implement our ODRP scheduler.

3. SYSTEM MODEL AND PROBLEM STATEMENT

In this section we present the resource and DSP application model and define the operator replication and place-

[1] `https://flink.apache.org/`

Table 1: Main notation adopted in the paper

Symbol	Description
G_{dsp}	Graph representing the DSP application
V_{dsp}	Set of vertices (operators) of G_{dsp}
E_{dsp}	Set of edges (streams) of G_{dsp}
C_i	Cost of deploying operator $i \in V_{dsp}$
R_i	Latency of $i \in V_{dsp}$ on a reference processor
Res_i	Resources required to execute $i \in V_{dsp}$
k_i	Maximum replication degree of $i \in V_{dsp}$
$\lambda_{(i,j)}$	Average tuple rate exchanged on $(i,j) \in E_{dsp}$
$b_{(i,j)}$	Avg. number of byte per tuple on $(i,j) \in E_{dsp}$
G_{res}	Graph representing computing and network resources
V_{res}	Set of vertices (computing nodes) of G_{res}
E_{res}	Set of edges (logical links) of G_{res}
A_u	Availability of node $u \in V_{res}$
Res_u	Amount of resources available at $u \in V_{res}$
S_u	Processing speed-up of $u \in V_{res}$
$A_{(u,v)}$	Availability of $(u,v) \in E_{res}$
$C_{(u,v)}$	Transmission cost per data on $(u,v) \in E_{res}$
$d_{(u,v)}$	Network delay on $(u,v) \in E_{res}$
$V^i_{res} \subseteq V_{res}$	Subset of nodes where $i \in V_{dsp}$ can be placed
$\mathcal{X} \sqsubset X$	Multiset of elements in X
$x_{i,\mathcal{U}}$	Placement of $i \in V_{dsp}$ on nodes in $\mathcal{U} \sqsubset V^i_{res}$
$y_{(i,j),(\mathcal{U},\mathcal{V})}$	Placement of $(i,j) \in E_{dsp}$ on the network paths from nodes in $\mathcal{U} \sqsubset V^i_{res}$ to nodes in $\mathcal{V} \sqsubset V^j_{res}$
z_u	Activation variable for $u \in V_{res}$
$z_{(u,v)}$	Activation variable for $(u,v) \in E_{res}$

ment problem. For the sake of clarity, in Table 1 we summarize the notation used throughout the paper.

3.1 Resource Model

Computing and network resources can be represented as a labeled fully connected directed graph $G_{res} = (V_{res}, E_{res})$, where the set of nodes V_{res} represents the distributed computing resources, and the set of links E_{res} represents the *logical connectivity* between nodes. Observe that, at this level, links represent the logical links across the networks corresponding to the network paths between nodes (as determined by the network operator routing strategies). Each node $u \in V_{res}$ is characterized by: Res_u, the amount of available resources; S_u, the processing speed-up on a reference processor; and A_u, its availability, i.e., the probability that u is up and running. Each link $(u,v) \in E_{res}$, with $u, v \in V_{res}$ is characterized by: $d_{(u,v)}$, the network delay between node u and v; $A_{(u,v)}$, the link availability, i.e., the probability that the link between u and v is active; and $C_{(u,v)}$, the cost per unit of data transmitted along the network path between u and v. This model considers also edges of type (u,u) (i.e., loops); they capture network connectivity between operators placed in the same node u, and are considered as perfect links, i.e., always active with no network delay. We assume that the considered QoS attributes can be obtained by means of either active/passive measurements or through some network support (e.g., SDN).

3.2 DSP Model

A DSP application can be represented at different levels of abstraction, and we can distinguish between an *abstract model*, which is defined by the user, and an *execution model*, which is used to run the application.

The DSP *abstract model* defines the streams and their characteristics, along with the type, role, and granularity of the stream processing elements. At this level, the DSP

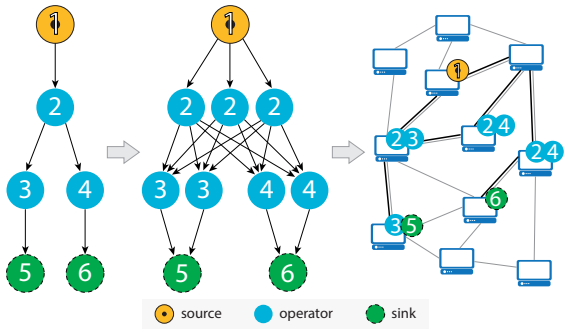


Figure 1: Replication of the application operators and their placement on the computing resources

application can be regarded as a network of operators connected by streams. An operator is a self-contained processing element that carries out a specific operation (e.g., filtering, aggregation, merging) or something more complex (e.g., POS-tagging), whereas a stream is an unbounded sequence of data (e.g., packet, tuple, file chunk). A DSP abstract model can be represented as a labeled directed acyclic graph (DAG) $G_{dsp} = (V_{dsp}, E_{dsp})$, where the nodes in V_{dsp} represent the application operators as well as the data stream sources (i.e., nodes without incoming links) and sinks (i.e., nodes without outgoing links), and the links in E_{dsp} represent the streams, i.e., data flows, between nodes. Due to the difficulties in formalizing the non-functional attributes of an abstract operator, we characterize it with the non-functional attributes of a reference implementation on a reference architecture: Res_i, the amount of resources required for running the operator; R_i, the operator latency (which accounts for the waiting time on the input queues as well as the execution time of a unit of data); C_i, the cost of deploying an instance of the operator. We characterize the stream exchanged from operator i to j, $(i,j) \in E_{dsp}$, with its average tuple rate $\lambda_{(i,j)}$ and average number of bytes per tuple $b_{(i,j)}$. To model load-dependent latency, we assume that the latency is function of λ_i, the operator input tuple rate, $R_i = R_i(\lambda_i)$, where $\lambda_i = \sum_{j \in V_{dsp}} \lambda_{(j,i)}$; without loss of generality, we also assume that R_i is an increasing function in λ_i. In this paper, we assume that Res_i is a scalar value, but our placement model can be easily extended to consider Res_i as a vector of required resources.

The DSP *execution model* is obtained from the abstract model by replacing each operator with its replicas. Indeed, in order to improve performance, multiple instances (or replicas) of the same DSP operator can be instantiated over different computing nodes. The premise is that by partitioning the streams over multiple processing elements, we reduce the load of each processing element which in turn yields lower operator (and overall application) latency. Differently from most of the existing solutions, ODRP computes the execution model by optimizing, in a single stage, the number of operator instances and their placement.

3.3 Operator Replication and Placement

The DSP replication and placement problem consists in determining, for each operator $i \in V_{dsp}$, the number of replicas and where to deploy them on the computing nodes in V_{res}. Figure 1 represents a simple instance of the problem.

Observe that a DSP operator cannot be usually placed on every node in V_{res}, because of physical (i.e., *pinned* operator) or other motivations (e.g., security, privacy). This observation allows us to consider for each operator $i \in V_{dsp}$ a subset of candidate resources $V_{res}^i \subseteq V_{res}$ where it can be deployed. For example, if sources and sinks ($I \subset V_{dsp}$) are external applications, their placement is fixed, that is $\forall i \in I, |V_{res}^i| = 1$. The operator placement can be represented by a function *map* which maps an operator $i \in V_{dsp}$ to a multiset of computing nodes in V_{res}^i. We recur to multisets because a deployment can place multiple replicas of the same operator on the same computing node. For instance, $map(i) = \{u, u, v\}$, $i \in V_{dsp}$, $u, v \in V_{res}^i$, indicates that operator i deployment consists of 3 replicas, two of which on node u and one on node v. A multiset $\mathcal{X}$ over a set X, which we denote as $\mathcal{X} \sqsubset X$, is defined as a mapping $\mathcal{X} : X \to \mathbb{N}$ where, for $x \in X$, $\mathcal{X}(x)$ denotes the multiplicity of x in $\mathcal{X}$. $x \in \mathcal{X}$ if and only if $\mathcal{X}(x) \geq 1$. The cardinality of a multiset $\mathcal{X}$, denoted $|\mathcal{X}|$, is defined by the number of elements in $\mathcal{X}$, that is $|\mathcal{X}| = \sum_{x \in \mathcal{X}} \mathcal{X}(x)$. Hereafter, without lack of generality, we will assume that in a deployment each operator $i \in V_{dsp}$ can be replicated at most k_i times. Therefore, we also find convenient to define the power multiset $\mathcal{P}(X)$ of a set X as the set of all multisets with elements taken from X and the subset $\mathcal{P}(X; k) \subset \mathcal{P}(X)$ of the multiset over X with cardinality no greater of k, that is $\mathcal{P}(X; k) = \{\mathcal{X} \in \mathcal{P}(X) | \sum_{x \in \mathcal{X}} \mathcal{X}(x) \leq k\}$.

4. OPTIMAL REPLICATION AND PLACEMENT MODEL

In this section we present our model for the ODRP problem. As the optimal solution depends on non-functional attributes, we first derive the expression for the different QoS metrics of interest and then present the ODRP formulation.

4.1 ODRP Variables

We model the ODRP problem with binary variables $x_{i,\mathcal{U}}$, $i \in V_{dsp}$ and $\mathcal{U} \sqsubset V_{res}^i$: $x_{i,\mathcal{U}} = 1$ if and only if the operator $map(i) = \mathcal{U}$, that is, i is replicated in $|\mathcal{U}|$ instances with exactly $\mathcal{U}(u)$ copies deployed in u, with $u \in \mathcal{U}$. We also find convenient to consider binary variables associated to links, namely $y_{(i,j),(\mathcal{U},\mathcal{V})}$, $(i,j) \in E_{dsp}$, $\mathcal{U} \sqsubset V_{res}^i$, $\mathcal{V} \sqsubset V_{res}^j$, which denotes whether the data stream flowing from operator i to operator j traverses the network paths from nodes in $\mathcal{U}$ to nodes in $\mathcal{V}$. By definition, we have $y_{(i,j),(\mathcal{U},\mathcal{V})} = x_{i,\mathcal{U}} \wedge x_{j,\mathcal{V}}$.

Finally, we also consider the variables z_u, $u \in V_{res}$ which denote whether at least one operator is deployed on node u and the variables $z_{(u,v)}$, $(u,v) \in E_{res}$, which denote whether a stream (or a portion of it) traverses the network path (u,v). By definition, we have $z_u = \vee_{i \in V_{dsp}, \mathcal{U} \in \mathcal{P}(V_{res}^i; k_i)} x_{i,\mathcal{U}}$ and $z_{(u,v)} = \vee_{(i,j) \in E_{dsp}, \mathcal{U} \in \mathcal{P}(V_{res}^i; k_i), \mathcal{V} \in \mathcal{P}(V_{res}^j; k_j)} y_{(i,j),(\mathcal{U},\mathcal{V})}$. For short, in the following we denote by $\boldsymbol{x}$ and $\boldsymbol{y}$ the placement vectors for nodes and edges, respectively, where $\boldsymbol{x} = \langle x_{i,\mathcal{U}} \rangle$, $\forall i \in V_{dsp}$, $\forall \mathcal{U} \in \mathcal{P}(V_{res}^i; k_i)$, and $\boldsymbol{y} = \langle y_{(i,j),(\mathcal{U},\mathcal{V})} \rangle$, $\forall x_{i,\mathcal{U}}, x_{j,\mathcal{V}} \in \boldsymbol{x}$. Similarly, we denote by $\boldsymbol{z_V}$ and $\boldsymbol{z_E}$ the vectors $\boldsymbol{z_V} = \langle z_u \rangle$, $\forall u \in V_{res}$, and $\boldsymbol{z_E} = \langle z_{(u,v)} \rangle$, $\forall (u,v) \in E_{res}$.

4.2 Qos Metrics

4.2.1 Operator QoS Metrics

Let us first consider an operator in isolation. For each $i \in V_{dsp}$, the QoS of the operator deployment depends on the deployment $\mathcal{U}$. Let $R_{i,\mathcal{U}}$, $C_{i,\mathcal{U}}$, and $A_{i,\mathcal{U}}$ denote the maximum latency, the cost, and the availability of the deployment $\mathcal{U}$, respectively. We readily have:

$$R_{i,\mathcal{U}} = \max_{u \in \mathcal{U}} \frac{R_i(\frac{\lambda_i}{|\mathcal{U}|})}{S_u} \quad (1)$$

$$C_{i,\mathcal{U}} = \sum_{u \in \mathcal{U}} \mathcal{U}(u) C_i Res_i \quad (2)$$

$$A_{i,\mathcal{U}} = \prod_{u \in \mathcal{U}} A_u \quad (3)$$

under the assumption that the traffic is equally split among the different operator replicas.

4.2.2 Stream QoS Attributes

We now turn our attention to the QoS attributes related to a stream. For a stream (i,j), the QoS depends on the upstream and downstream operators' deployments $\mathcal{U}$ and $\mathcal{V}$. Let $d_{(i,j),(\mathcal{U},\mathcal{V})}$, $C_{(i,j),(\mathcal{U},\mathcal{V})}$, and $A_{(i,j),(\mathcal{U},\mathcal{V})}$ denote the maximum latency, the cost, and the availability of the deployments $\mathcal{U}$ and $\mathcal{V}$, respectively. We readily have:

$$d_{(i,j),(\mathcal{U},\mathcal{V})} = d_{(\mathcal{U},\mathcal{V})} = \max_{u \in \mathcal{U}, v \in \mathcal{V}} d_{(u,v)} \quad (4)$$

$$C_{(i,j),(\mathcal{U},\mathcal{V})} = \sum_{u \in \mathcal{U}, v \in \mathcal{V}} \lambda_{(i,j),(\mathcal{U},\mathcal{V})} C_{u,v} \quad (5)$$

$$A_{(i,j),(\mathcal{U},\mathcal{V})} = \prod_{u \in \mathcal{U}, v \in \mathcal{V}} A_{(u,v)} \quad (6)$$

where

$$\lambda_{(i,j),(\mathcal{U},\mathcal{V})} = \frac{\lambda_{(i,j)}}{|\mathcal{U}||\mathcal{V}|} \quad u \in \mathcal{U}, v \in \mathcal{V} \quad (7)$$

is the amount of stream (i,j) traffic exchanged between two operator replicas under the deployments $\mathcal{U}$ and $\mathcal{V}$.

4.2.3 DSP Application QoS Metrics

We consider both user-oriented and system-oriented QoS metrics, such as application response time, cost, and availability for the former, and network related metrics for the latter.

Response Time: For a DSP application, with data flowing from several sources to several destinations, there is no unique definition of response time. In the following, we consider as response time R the worst end-to-end delay from a source to a sink. Given this definition, we have that:

$$R = \max_{\pi \in \Pi_{dsp}} R_\pi \quad (8)$$

being R_π the delay along path π and Π_{dsp} the set of all source-sink paths in G_{dsp}. Given a placement vector $\boldsymbol{x}$ (and resulting $\boldsymbol{y}$) and a path $\pi = (i_1, i_2, \ldots, i_{n_\pi})$, we have $R = R(\boldsymbol{x}, \boldsymbol{y}) = \max_{\pi \in \Pi_{dsp}} R_\pi(\boldsymbol{x}, \boldsymbol{y})$ with $R_\pi(\boldsymbol{x}, \boldsymbol{y})$ defined as:

$$R_\pi(\boldsymbol{x}, \boldsymbol{y}) = \sum_{p=1}^{n_\pi} R_{i_p}(\boldsymbol{x}) + \sum_{p=1}^{n_\pi - 1} D_{(i_p, i_{p+1})}(\boldsymbol{y}) \quad (9)$$

where

$$R_i(\boldsymbol{x}) = \sum_{\mathcal{U} \in \mathcal{P}(V_{res}^i; k_i)} R_{i,\mathcal{U}} x_{i,\mathcal{U}} \quad (10)$$

$$D_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{\mathcal{U} \in \mathcal{P}(V_{res}^i; k_i) \\ \mathcal{V} \in \mathcal{P}(V_{res}^j; k_j)}} d_{(\mathcal{U},\mathcal{V})} y_{(i,j),(\mathcal{U},\mathcal{V})} \quad (11)$$

denote respectively the execution time of operator i when deployed over the multiset $\mathcal{U}$ and the worst case network delay for transferring data from i to j when the two operators are mapped over $\mathcal{U}$ and $\mathcal{V}$, respectively.

Cost: We define the cost C of the DSP application as the monetary cost of all the computing resources and paths involved in the processing and transmission of the application data streams. We have:

$$C(\boldsymbol{x},\boldsymbol{y}) = \sum_{i\in V_{dsp}} C_i(\boldsymbol{x}) + \sum_{(i,j)\in E_{dsp}} C_{(i,j)}(\boldsymbol{y}) \tag{12}$$

where

$$C_i(\boldsymbol{x}) = \sum_{\mathcal{U}\in\mathcal{P}(V^i_{res};k_i)} C_{i,\mathcal{U}} x_{i,\mathcal{U}} \tag{13}$$

$$C_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{\mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} C_{(i,j),(\mathcal{U},\mathcal{V})} y_{(i,j),(\mathcal{U},\mathcal{V})} \tag{14}$$

Availability: We define the application availability A as the availability of all the nodes and paths involved in the processing and transmission of the application data streams. For the sake of simplicity, we assume the availability of the different components to be independent. However, we acknowledge that independence does not hold true in general and that a more detailed model is needed to capture the dependency relationship among logical components sharing physical nodes and networks links; we postpone it to future work. With the independence assumption, we readily have:

$$A(\boldsymbol{z_V},\boldsymbol{z_E}) = \prod_{u\in V_{res}:z_u=1} A_u z_u \cdot \prod_{(u,v)\in E_{res}:z_{(u,v)}=1} A_{(u,v)} z_{(u,v)} \tag{15}$$

To obtain a linear expression, we consider the logarithm of the availability, obtaining:

$$\log A(\boldsymbol{z_V},\boldsymbol{z_E}) = \sum_{u\in V_{res}} a_u z_u + \sum_{(u,v)\in E_{res}} a_{(u,v)} z_{(u,v)} \tag{16}$$

where $a_u = \log A_u$ and $a_{(u,v)} = \log A_{(u,v)}$. It is worth observing that in (16) we can take the summation over all $u \in V_{res}$ and $(u,v) \in E_{res}$ since the terms not appearing in (15) are those corresponding to $z_u = 0$ or $z_{(u,v)} = 0$, which do not affect the summation in (16).

Network Related QoS Metrics: In the DSP literature, several alternative network-aware metrics have been defined, including the inter-node traffic T [1], the network usage N [26], and an approximation of the elastic energy EE [22]. Let $Z(\boldsymbol{y})$, $Z = T|N|EE$, denote the QoS attribute of the DSP application under the placement policy $\boldsymbol{y}$, we have:

$$Z(\boldsymbol{y}) = \sum_{(i,j)\in E_{dsp}} Z_{(i,j)}(\boldsymbol{y}) \tag{17}$$

where $Z_{(i,j)}(\boldsymbol{y})$ is defined as follows.

The *inter-node traffic* T is the overall amount of data exchanged per time unit between operators placed on different nodes. Therefore, using the placement policy $\boldsymbol{y}$, the stream $(i,j) \in E_{dsp}$ generates an inter-node traffic equals to:

$$T_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{u\in\mathcal{U},v\in\mathcal{V},u\neq v\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} b_{(i,j)} \lambda_{(i,j),(\mathcal{U},\mathcal{V})} y_{(i,j),(\mathcal{U},\mathcal{V})} \tag{18}$$

The *network usage* N is the amount of data that traverses the network at a given time; therefore, the stream $(i,j) \in E_{dsp}$ imposes a load expressed by:

$$N_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{u\in\mathcal{U},v\in\mathcal{V},u\neq v\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} b_{(i,j)} \lambda_{(i,j),(\mathcal{U},\mathcal{V})} d_{(u,v)} y_{(i,j),(\mathcal{U},\mathcal{V})} \tag{19}$$

where $d_{(u,v)}$ is the network delay among nodes $u, v \in V_{res}$, with $u \neq v$.

In their paper [22], Pietzuch et al. indirectly minimize the network usage through the minimization of the elastic energy, which results from the equivalent system of springs that represents the application. Basically, their solution minimizes the amount of data that traverses each link weighted by the latency of the link itself. Hence, the stream $(i,j) \in E_{dsp}$ contributes to the elastic energy of the system with:

$$EE_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{u\in\mathcal{U},v\in\mathcal{V},u\neq v\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} b_{(i,j)} \lambda_{(i,j),(\mathcal{U},\mathcal{V})} d^2_{(u,v)} y_{(i,j),(\mathcal{U},\mathcal{V})} \tag{20}$$

Observe that, in all cases, $Z(\boldsymbol{y})$ is a linear function of $\boldsymbol{y}$.

4.3 ODRP Formulation

Depending on the usage scenario, a DSP replication and placement strategy could be aimed at optimizing different, possibly conflicting, QoS attributes. To this end, we use the Simple Additive Weighting (SAW) technique [27] to define the utility function $F(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z_V},\boldsymbol{z_E})$ as a weighted sum of the normalized QoS attributes of the application, as follows:

$$\begin{aligned} F(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z_V},\boldsymbol{z_E}) = {} & w_r \frac{R_{\max} - R(\boldsymbol{x},\boldsymbol{y})}{R_{\max} - R_{\min}} + w_a \frac{\log A(\boldsymbol{z_V},\boldsymbol{z_E}) - \log A_{\min}}{\log A_{\max} - \log A_{\min}} \\ & + w_c \frac{C_{\max} - C(\boldsymbol{x},\boldsymbol{y})}{C_{\max} - C_{\min}} + w_z \frac{Z_{\max} - Z(\boldsymbol{x},\boldsymbol{y})}{Z_{\max} - Z_{\min}} \end{aligned} \tag{21}$$

where $w_r,\ w_a,\ w_c,\ w_z \geq 0$, $w_r + w_a + w_c + w_z = 1$, are weights associated to the different QoS attributes. $R_{\max}$ ($R_{\min}$), $A_{\max}$ ($A_{\min}$), $C_{\max}$ ($C_{\min}$), and $Z_{\max}$ ($Z_{\min}$) denote, respectively, the maximum (minimum) value for the overall expected response time, availability, cost and network related metric. Observe that after normalization, each metric ranges in the interval $[0,1]$, where the value 1 corresponds to the best possible case and 0 to the worst case.

We formulate the ODRP problem as an Integer Linear Programming (ILP) model as follows:

$$\max_{\boldsymbol{x},\boldsymbol{y},r} F'(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z_V},\boldsymbol{z_E},r)$$

subject to:

$$\begin{aligned} r \geq {} & \sum_{\substack{p=1,\ldots,n_\pi\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)}} R_{i_p,\mathcal{U}} x_{i,\mathcal{U}} + \\ & \sum_{\substack{p=1,\ldots,n_\pi-1\\ q=p+1\\ \mathcal{U}\in\mathcal{P}(V^{i_p}_{res};k_{i_p})\\ \mathcal{V}\in\mathcal{P}(V^{i_q}_{res};k_{i_q})}} d_{(\mathcal{U},\mathcal{V})} y_{(i_p,i_q),(\mathcal{U},\mathcal{V})} \qquad \forall \pi \in \Pi_{dsp} \end{aligned} \tag{22}$$

$$Res_u \geq \sum_{\substack{i \in V_{dsp} \\ \mathcal{U} \in \mathcal{P}(V^i_{res})}} \mathcal{U}(u) Res_i x_{i,\mathcal{U}} \qquad \forall u \in V_{res} \tag{23}$$

$$z_u \geq \frac{\sum_{\substack{i \in V_{dsp} \\ \mathcal{U} \in \mathcal{P}(V^i_{res}; k_i)}} x_{i,\mathcal{U}}}{M} \qquad u \in V_{res} \tag{24}$$

$$z_{(u,v)} \geq \frac{\sum_{\substack{(i,j) \in E_{dsp} \\ \mathcal{U} \in \mathcal{P}(V^i_{res}; k_i) \\ \mathcal{V} \in \mathcal{P}(V^j_{res}; k_j)}} y_{(i,j),(\mathcal{U},\mathcal{V})}}{N} \qquad (u,v) \in E_{res} \tag{25}$$

$$1 = \sum_{\mathcal{U} \in \mathcal{P}(V^i_{res}; k_i)} x_{i,\mathcal{U}} \qquad \forall i \in V_{dsp} \tag{26}$$

$$x_{i,\mathcal{U}} = \sum_{\mathcal{V} \in \mathcal{P}(V^j_{res}; k_j)} y_{(i,j),(\mathcal{U},\mathcal{V})} \qquad \begin{matrix} \forall (i,j) \in E_{dsp}, \\ \mathcal{U} \in \mathcal{P}(V^i_{res}; k_i) \end{matrix} \tag{27}$$

$$x_{j,\mathcal{V}} = \sum_{\mathcal{U} \in \mathcal{P}(V^i_{res}; k_u)} y_{(i,j),(\mathcal{U},\mathcal{V})} \qquad \begin{matrix} \forall (i,j) \in E_{dsp}, \\ \mathcal{V} \in \mathcal{P}(V^j_{res}; k_j) \end{matrix} \tag{28}$$

$$x_{i,\mathcal{U}} \in \{0,1\} \qquad \begin{matrix} \forall i \in V_{dsp}, \\ \mathcal{U} \in \mathcal{P}(V^i_{res}; k_i) \end{matrix} \tag{29}$$

$$y_{(i,j),(\mathcal{U},\mathcal{V})} \in \{0,1\} \qquad \begin{matrix} \forall (i,j) \in E_{dsp}, \\ \mathcal{U} \in \mathcal{P}(V^i_{res}; k_i) \\ \mathcal{V} \in \mathcal{P}(V^j_{res}; k_j) \end{matrix} \tag{30}$$

$$z_u \in \{0,1\} \qquad \forall u \in V_{res} \tag{31}$$

$$z_{(u,v)} \in \{0,1\} \qquad \forall (u,v) \in E_{res} \tag{32}$$

In the problem formulation we use the objective function $F'(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z_V}, \boldsymbol{z_E}, r)$ that is obtained from $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z_V}, \boldsymbol{z_E})$ by replacing $R(\boldsymbol{x}, \boldsymbol{y})$ with the auxiliary variable r, which represents the application response time in the optimization problem, in order to obtain a linear objective function. Observe that, indeed, while F is nonlinear in $\boldsymbol{x}$, $\boldsymbol{y}$ since $R(\boldsymbol{x}, \boldsymbol{y}) = \max_{\pi \in \Pi_{dsp}} R_\pi(\boldsymbol{x}, \boldsymbol{y})$ is a nonlinear term, F' is linear in r as well as in $\boldsymbol{x}$ and $\boldsymbol{y}$. Equation (22) follows from (8)–(11). Since r must be larger or equal than the response time of any path and, at the optimum, r is minimized, $r = \max_{\pi \in \Pi_{dsp}} R_\pi(\boldsymbol{x}, \boldsymbol{y}) = R(\boldsymbol{x}, \boldsymbol{y})$. The constraint (23) limits the placement of operators on a node $u \in V_{res}$ according to its available resources. Constraints (24) and (25) are the activation constraints for the variable z_u and $z_{(u,v)}$, respectively, with M and N large constants. Equation (26) guarantees that each operator $i \in V_{dsp}$ is placed on one and only one node $u \in V^i_{res}$. Finally, constraints (27)–(28) model the logical `AND` between the placement variables, that is, $y_{(i,j),(u,v)} = x_{i,u} \wedge x_{j,v}$.

THEOREM 1. *The ODRP problem is an NP-hard problem.*

PROOF. It is sufficient to observe that the Optimal DSP Replication and Placement problem is a generalization of the Optimal DSP Placement problem we presented in [4], which has been shown to be NP-hard. □

5. STORM INTEGRATION

To enable the usage of ODRP in a real DSP framework, we have developed a prototype scheduler for Apache Storm, named S-ODRP. We first briefly describe the main features of Storm and how it represents and executes DSP applications. Then, we present the prototype design in details.

5.1 Apache Storm

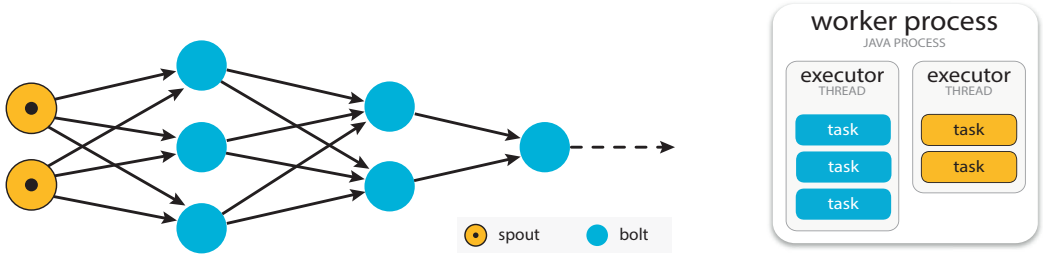
Storm is an open source, real-time, and scalable DSP system maintained by the Apache Software Foundation. It provides an abstraction layer where event-based applications can be executed over a set of worker nodes interconnected by an overlay network. A *worker node* is a generic computing resource (e.g., a physical host, a mobile device, a virtual machine, a Docker container), whereas the overlay network comprises the logical links among these nodes.

In Storm, an application is represented by its *topology*, which is a DAG with spouts and bolts as vertices and streams as edges. A *spout* is a data source that feeds the data into the system through one or more streams. A *bolt* is either a processing element, which extracts valuable information from incoming tuples, or a final information consumer; a bolt can also generate new outgoing streams, like spouts do. A *stream* is an unbounded sequence of *tuples*, which are key-value pairs. We refer to spouts and bolts as operators. Figure 2a shows an example of a DSP application.

Storm uses three types of entities with different grain to execute a topology: tasks, executors, and worker processes. A *task* is an instance of an application operator (i.e., spout or bolt), and it is in charge of a share of the incoming operator stream. An *executor*, which is the smallest schedulable unit, can execute one or more tasks related to the same operator; in other words, t_i, the number of tasks of an operator, is always greater or equal than e_i, the number of executors of the same operator, that is, $t_i \geq e_i$. Since the operator executors run concurrently, they can increase the operator throughput when subject to heavy incoming load. A *worker process* is a Java process that runs a subset of the executors of the *same* topology, i.e., a topology can be distributed across different worker processes. As represented in Figure 2b, there is an evident hierarchy among the Storm entities: a group of tasks runs sequentially in the executor, which is a thread within the worker process, that in its turn serves as container on the worker node. To date, Storm leaves completely to the user the definition of the number of worker processes, executors, and tasks for the DSP application. Moreover, the framework enables the user to change at runtime the parallelism degree of an application through the `rebalance` API; however, its implementation is not efficient, because it restarts the whole application with new executors, leading to possible data loss.

Besides the computing resources, i.e., the worker nodes, the Storm architecture includes two additional components: Nimbus and ZooKeeper. *Nimbus* is a centralized component in charge of coordinating the topology execution; it uses its *scheduler* to define the placement of the application operators on the pool of available worker nodes. The assignment plan determined by the scheduler is communicated to all the worker nodes through *ZooKeeper*[2], which is a shared in-memory service for managing configuration information and enabling distributed coordination. Since each worker

[2] `http://zookeeper.apache.org/`



(a) Topology of a DSP application (b) Execution model

Figure 2: Storm abstractions

node can execute one or more worker processes, a *Supervisor* component on each worker node starts or terminates worker processes on the basis of the Nimbus assignments. Each worker node can concurrently run a limited number of worker processes, based on the number of available *worker slots*.

5.2 S-ODRP: ODRP in Storm

We develop a new scheduler for Storm, named **S-ODRP**, whose core is the model presented in Section 4. In order to design S-ODRP, we have to address two issues: (1) to adapt the DSP and resource model to consider the specific execution entities of Storm, and (2) to instantiate the ODRP model with the proper QoS information about computing and networking resources.

As regards the first issue, we have to model the fact that Storm runs multiple executors to replicate an operator, and that a Storm scheduler deploys these executors on the available worker slots, considering that at most $EPS_{\max}$ executors can be co-located on the same slot. Hence, S-ODRP defines $G_{dsp} = (V_{dsp}, E_{dsp})$, with V_{dsp} as the set of operators and E_{dsp} as the set of streams exchanged between them. Since in Storm an operator is considered as a black box element, we conveniently assume that its attributes are unitary, i.e., $C_i = 1$ and $Res_i = 1$, $\forall i \in V_{dsp}$. By solving the replication and placement model, S-ODRP determines the number of executors for each operator $i \in V_{dsp}$, leveraging on the cardinality of $\mathcal{U}$ when $x_{i,\mathcal{U}} = 1$, with $\mathcal{U} \sqsubset V^i_{res}$. The resource model $G_{res} = (V_{res}, E_{res})$ must take into account that a worker node $u \in V_{res}$ offers some worker slots $WS(u)$, and each worker slot can host at most $EPS_{\max}$ executors. For simplicity, S-ODRP considers the amount of available resources C_u on a worker node $u \in V_{res}$ to be equal to the maximum number of executors it can host, i.e., $C_u = WS(u) \times EPS_{\max}$. To enable the parallel execution of executors, C_u should be equal (or proportional) to the number of CPU cores available on u.

As regards the second issue, Storm allows us to easily develop new centralized schedulers with the pluggable scheduler APIs. However, Storm is unaware of the QoS attributes of its networking and computing resources, except for the number of available worker slots. Since we need to know these QoS attributes in order to apply the ODRP model, we rely on Distributed Storm, a Storm extension[3] we presented in [2], that enables the QoS awareness of the scheduling system by providing intra-node (i.e., availability) and inter-node (i.e., network delay and exchanged data rate) information. This extension estimates network latencies using a network coordinate system, which is built through the Vivaldi algorithm [7], a decentralized algorithm having linear complexity with respect to the number of network locations. S-ODRP retrieves, from the monitoring components of the extended Storm, the information needed to parametrize the nodes and edges in G_{dsp} and G_{res}. Specifically, it considers: the average data rate exchanged between communicating executors (i.e., $\lambda_{(i,j)}, \forall (i,j) \in E_{dsp}$), the node availability ($A_u, \forall u \in V_{res}$), and the network latencies ($d_{(u,v)}, \forall u, v \in V_{res}$). Once built the ODRP model, S-ODRP relies on CPLEX©[4], the state-of-the-art solver for ILP problems, for its resolution.

From an operational prospective, Nimbus uses S-ODRP to compute the optimal operator replication and placement when a new application is submitted to Storm and when a failure of the worker process compromises the application execution. In the latter case, S-ODRP invalidates the existing assignment and computes the new optimal placement. Algorithm 1 summarizes the runtime execution of S-ODRP, which has to face two main issues: to collect the exchanged data rate between the operators, and to replicate the operators as needed. When information on the exchanged data rate is unknown (line 3), e.g., the first time the application is scheduled, S-ODRP defines an early assignment and monitors the application execution to harvest the needed information (lines 4–6). As soon as this information is available, S-ODRP reassigns the application by solving the updated ODRP model with the network-related QoS attributes (line 8). To enact the replication decision computed by ODRP (lines 5 and 9), S-ODRP leverages on the Storm API `rebalance`, which restarts the application with the correct number of executors, before assigning them to the worker nodes as specified by the computed placement solution.

Algorithm 1 Application placement with S-ODRP

```
1: function SCHEDULE(G_dsp, G_res)
2:     RP = [ ]                          ▷ replication and placement
3:     if not streamsDataRateAvailable(G_dsp) then
4:         RP ← ODRP(G_dsp, G_res)
5:         enact(RP)
6:         G_dsp ← collectStreamsDatarate(G_dsp)
7:     end if
8:     RP ← ODRP(G_dsp, G_res)
9:     enact(RP)
10: end function
```

[3]Source code available at `http://bit.ly/extstorm`

[4]`http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

6. EXPERIMENTAL RESULTS

Our experiments revolve around S-ODRP, the prototype scheduler for Storm whose core is ODRP, and they aim to show the generality and flexibility of the proposed formulation as well as its impact in terms of achievable application performance. After introducing the experimental setup and the reference application (Section 6.1), we provide a general overview on the runtime execution of S-ODRP (Section 6.2). Then, in Section 6.3, we show the benefits of the joint optimization of placement and replication when the DSP application is subject to an increasing load. Finally, in Section 6.4, we evaluate how S-ODRP can optimize several QoS metrics, such as response time, cost, availability, and inter-node traffic.

6.1 Experimental Setup

We perform the experiments using Apache Storm 0.9.3 on a cluster of 6 worker nodes, each with 2 worker slots, and a further node to host Nimbus and ZooKeeper. Each node is a machine with a dual CPU Intel Xeon E5504 (8 cores at 2 GHz) and 16 GB of RAM. To better exploit the presence of independent CPU cores, we define that a worker slot can host at most 4 executors, i.e., $EPS_{\max} = 4$; therefore, a worker node can host at most 8 operator replicas, one for each available CPU core. We emulate wide-area network latencies among the Storm nodes using `netem`, which applies to outgoing packets a Gaussian delay with mean and standard deviation in the ranges $[12, 32]$ ms and $[1, 3]$ ms, respectively. As regards the pricing policy, we charge only the usage of computing resources, i.e., we set a unitary cost for each operator replica. We solve the ILP problem using CPLEX© (version 12.6.3) on the node hosting Nimbus.

As test-case application we developed a benchmarking application that solves the first query of the DEBS 2015 Grand Challenge [17]: by processing data streams originated from the New York City taxis, the goal of the query is to find the top-10 most frequent routes during the last 30 minutes. Its topology is represented in Figure 3. The *data source* reads the dataset from Redis, an in-memory data store, and pushes data towards a *parser* operator, which parses them and filters out irrelevant and invalid data. Afterwards, *filterByCoordinates* forwards only the events related to a specific observation area, whose extension is about 22 500 Km^2. The operator *computeRouteID* is in charge of identifying the route covered by taxis, and *countByWindow* counts the route frequency in the last 30 minutes; the notion of time is managed by a coordinator component, called *metronome*, which pulses when the time related to the dataset events advances. The following operators, *partialRank* and *globalRank*, compute the top-10 most frequent routes by leveraging on a two-step approach that enables to compute the ranking in a distributed and parallel manner. Finally, *globalRank* publishes the top-10 updates on a message queue, implemented with RabbitMQ. We assume that *data source* and *globalRank* are pinned operators. Moreover, since we investigate the initial application placement, we have set the data source so to feed the topology with a constant data rate, defined a-priori.

In the experiments, we define that each operator can be replicated at most three times (i.e., $k_i = 3, \forall i$), except for the pinned ones (i.e., *data source* and *globalRank*) and the *metronome*, which cannot be easily parallelized. Without loss of generality, in the ODRP model we estimate the response time R_i of operator i subject to the incoming load $\lambda_i/|\mathcal{U}|$ by modeling the underlying computing node as an M/M/1 queue, i.e., $R_i(\lambda_i/|\mathcal{U}|) = (\mu_i - \lambda_i/|\mathcal{U}|)^{-1}$, where μ_i is the service rate of i measured on a reference processor. The operators service rate and the other configuration parameters have been obtained through preliminary experiments and are shown in Table 2.

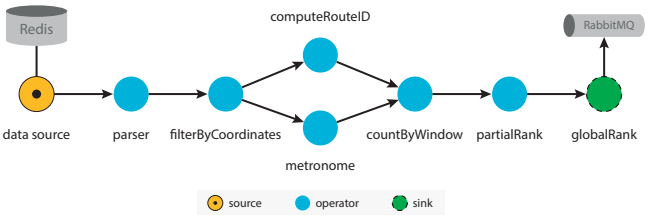


Figure 3: Reference DSP application

Table 2: Parameters of the experimental setup

Application: service rate per operator, expressed in tuples/s (tps)

Operator	μ_i	Operator	μ_i
data source	284 tps	metronome	190 tps
parser	233 tps	countByWindow	335 tps
filterByCoordinates	253 tps	partialRank	2371 tps
computeRouteID	253 tps	globalRank	185 tps

Normalization factors for the ODRP utility function

Parameter	Value	Parameter	Value
$R_{\min}$	5 ms	$R_{\max}$	450 ms
$A_{\min}$	95%	$A_{\max}$	100%
$C_{\min}$	8	$C_{\max}$	18
$Z_{\min}$	0 tps	$Z_{\max}$	3400 tps

6.2 Evaluation of S-ODRP

This first experiment aims at showing the runtime execution of the S-ODRP scheduler, described by Algorithm 1, and its impacts on the application performance. As baseline we use S-ODP, a prototype scheduler for Storm whose core is the ODP model that optimizes only the operator placement [4]. Note that ODP is a special case of ODRP where $k_i = 1, \forall i \in V_{dsp}$. To simplify the presentation, we consider only the response time R and the monetary cost C as QoS metrics; all worker nodes have an availability of 100%. Both the optimization models focus on the minimization of the application response time R (i.e., $w_r = 1$, $w_c = 0$), so we name them as **S-ODRP_R** and **S-ODP_R**, respectively. Differently from S-ODP_R, S-ODRP_R computes R relying on the exchanged data-rate between the operators; as presented in Section 5, it deploys the application with a preliminary placement so to harvest the relevant data; this preliminary placement is computed by minimizing the deployment cost (i.e., $w_c = 1$, $w_r = 0$). The rescheduling event takes place after 100 s of execution and is represented in Figure 4 with a vertical line.

We set the data rate of the source operator to 80 tuples/s and launch the application. Figure 4 reports the resulting application response time. We observe that as soon as the placement is defined, i.e., after 0 s and after 100 s (the latter for S-ODRP_R only), a transient period is experimented where R is quite high and exceeds 1 s. This behavior depends on a well-known issue of the Storm framework [26], which starts the operators as soon as they are ready without coordination at level of the whole application; as a consequence, data emitted by an operator wait in inter-operator

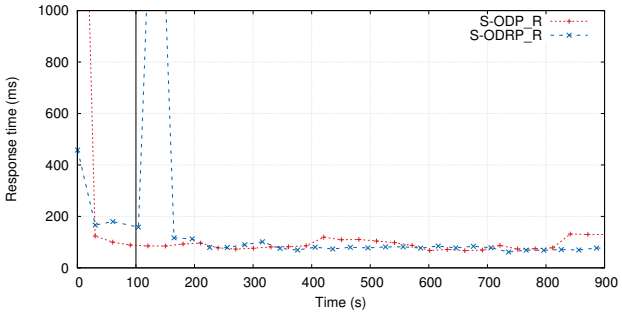


Figure 4: Runtime execution of S-ODRP and S-ODP

Table 3: Operator replication

Operator	S-ODP	S-ODRP		
		20, 40, 60 tuples/s	80, 100 tuples/s	120 tuples/s
data source	1	1	1	1
parser	1	1	1	3
filterByCoordinates	1	1	1	2
computeRouteID	1	1	2	2
metronome	1	1	1	1
countByWindow	1	1	1	3
partialRank	1	1	2	3
globalRank	1	1	1	1

buffers until the following operator is up and running for processing. When the transient period ends, after 200 s, the applications deployed with the two schedulers experience quite similar performance in terms of response time.

Table 3 reports the total number of operator replicas deployed by the two schedulers. S-ODP_R cannot replicate the operators, therefore it instantiates a replica for each of them. With a source data rate of 80 tuples/s, S-ODRP_R replicates twice two operators, namely *computeRouteID* and *partialRank*, and runs the application with a total of 10 executors. S-ODRP_R replicates the operators as much as possible while considering that, when a new replica has to be located on a new worker node, the latter introduces network latencies that can overcome the benefits of replication in reducing the operator execution time. In this experiment, it is worth to observe that, although the scheduler is forced to use a second worker node, it places the replicas so to minimize the response time; in particular, only the replicas of *computeRouteID*, which have the lowest data rate exchanged with the other operators, are located on a separate node.

6.3 Impact of Replication

In the second set of experiments, we want to investigate the replication benefits when the application is subject to different incoming loads. We use the same settings of the previous experiment except for the source data rate and we compare how S-ODRP_R and S-ODP_R determine the placement of the reference application. In each single experiment, which lasts 900 s, the source data rate is constant and is set in the range $[20, 120]$ tuples/s with step 20. We collect the resulting QoS metrics as soon as the transient period ends (i.e., after 200 s) and we summarize the results in Figure 5 leveraging on a boxplot, which represents the QoS metric distribution through the minimum value, the 5th percentile, 50th percentile, 95th percentile, and the maximum value; the average value is also represented using a full dot. We define as *active node utilization*, the average utilization of all the worker nodes involved in the application execution; each node contributes with its average utilization calculated on a time window of 60 s.

Figure 5a reports the application response time and Table 3 the number of replicas per operator. Although S-ODP_R cannot replicate the operators, it finds the optimal placement that minimizes R, as S-ODRP_R does. Indeed, when the replication is not needed, i.e., up to 60 tuples/s, the two schedulers achieve the same application performance. Note that, also in this case, network delays prevent S-ODRP_R from further replicating the operators: due to the absence of inter-node traffic (see Figure 5c), we can easily detect that all the 8 replicas run on the same node.

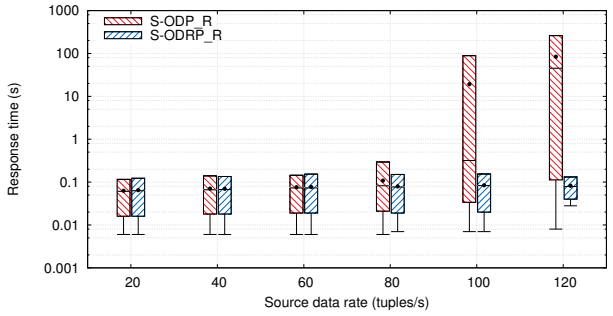
When the data source emits 80 tuples/s, the replication is needed: the *partialRank* operator represents a bottleneck, because it receives on average 2500 tuples/s, i.e., 5% more than its service rate. The utilization of the worker node that hosts the whole application for S-ODP_R, reported in Figure 5b, is around 20%, therefore the overload situation cannot be easily detected if not relying on fine-grain monitoring tools, which work at the level of single operator or CPU core. Conversely, S-ODRP_R detects the bottleneck and replicates the operator, which is then executed on a second worker node (see Table 3 and Figure 5c).

A similar behavior can be observed when the data source emits 100 tuples/s. This time the bottleneck operator, *partialRank*, receives on average 30% more tuples than a single replica can process per unit of time. With S-ODP_R, the application response time is unstable and continuously grows during the experiment, up to 106 s per single tuple. Conversely, thanks to replication, with S-ODRP_R the application maintains the same response time of the configuration with 80 tuples/s.

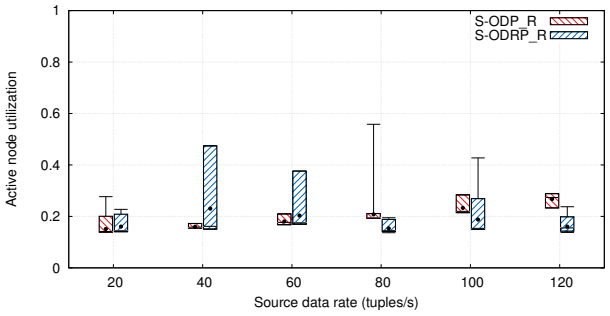
The need of replication is further exacerbated when the data source emits 120 tuples/s. With S-ODP_R, the application response time explodes up to $\sim$ 300 s per tuple. On the contrary, S-ODRP_R obtains an application response time that is not influenced by the increased load. To keep up with the incoming data rate, S-ODRP_R needs to replicate every operator. It instantiates 2 replicas for *filterByCoordinates* and *computeRouteID*, and 3 replicas for *parser*, *countByWindow*, and *partialRank*; comprising also the other operators, the application runs with a total of 16 executors (see Table 3) on 2 worker nodes. In spite of the increased incoming data rate, Figure 5c shows that the application deployment produces a fairly limited inter-node traffic. Finally, we observe from Figure 5b that, although the need of replication, the average value of the active node utilization is quite low. This behavior highlights that, for this specific use case, a static mapping between replicas and CPU cores does not lead to an efficient usage of resources; a possible solution to better exploit the available resources might be based on a dynamic multiplexing of replicas on the same CPU core. Since this optimization calls for the run-time adaptation of the application deployment, it falls out of the scope of this paper, but we plan to investigate more on it as future work.

6.4 Optimal Replication and Placement

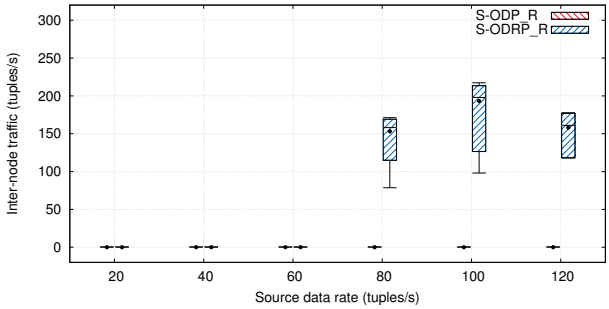
This experiment evaluates the effect of different optimization objectives on QoS metrics, i.e., availability A, cost C, response time R, and inter-node traffic T. To this end,



(a) Response time



(b) Active node utilization



(c) Inter-node traffic

Figure 5: Impact of replication on the application performance

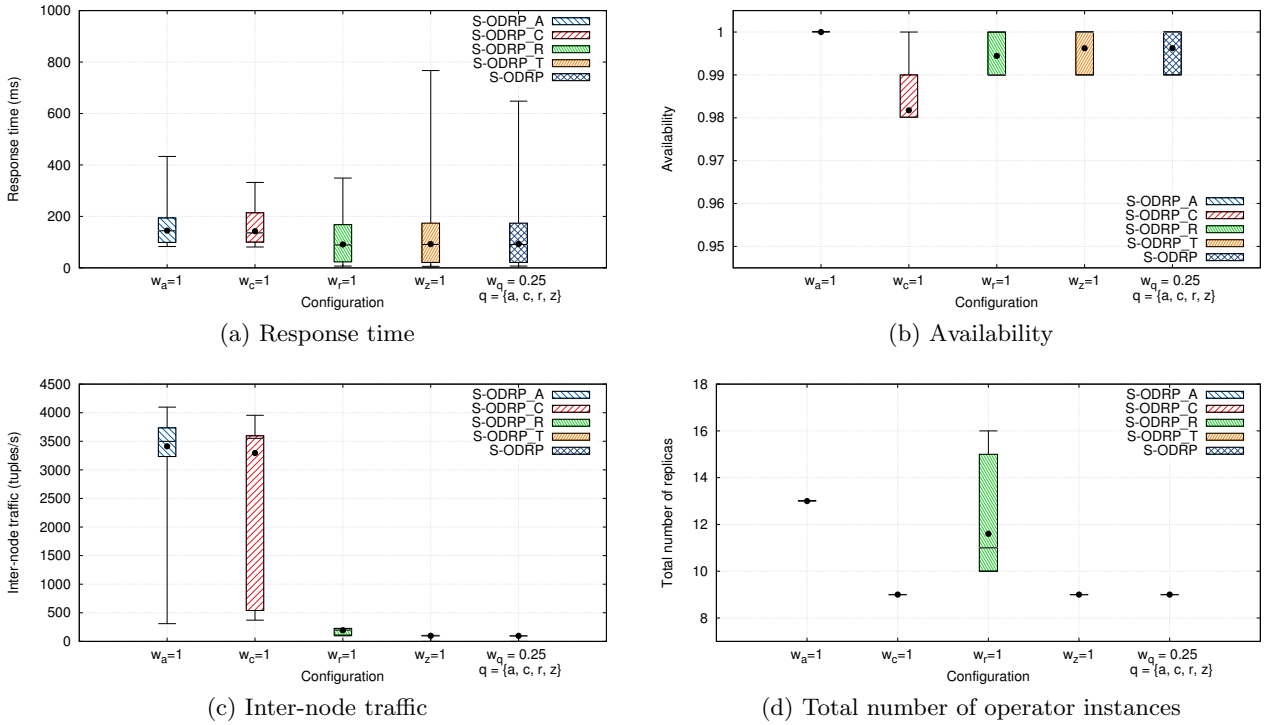
we set our experimental environment so that half of the worker nodes has an availability of 99% and the other of 100%, whereas the links are always available. We place the pinned operators (i.e., *data source* and *globalRank*) on a single worker node chosen randomly among those 100% available. S-ODRP determines the placement of the reference application when the data source emits 100 tuples/s. Figure 6 summarizes 25 runs, each of 900 s, where we collect QoS metrics after the initial transient period of 200 s. We first compute the replication and placement solution by optimizing a single QoS metric. For example, to optimize the response time we set the weights as $w_r = 1$, $w_a = w_c = w_z = 0$. Then, we optimize the multi-objective function by uniformly weighting each metrics contribution (i.e., $w_a = w_c = w_r = w_z = 0.25$); we report in Table 2 the normalization factors used in Equation (21). Figure 6 presents the results in term of the different QoS metrics.

When S-ODRP optimizes the application availability A (for short, **S-ODRP_A**), the solution finds the configuration where all the replicas are on the most available worker nodes. Observe that the placement of pinned operators, which are not relocated by S-ODRP, impacts on the overall application availability. In our experiments we placed these operators on nodes with 100% of availability. From Figure 6d and Equation (15), we observe that, although this configuration of S-ODRP does not optimize the number of operator instances, multiple replicas can be executed until a new worker node with availability lower than 100% has to be activated. This setting of weights does optimize neither the response time nor the inter-node traffic (see Figure 6a and 6c): on average, the response time is almost 1.6 times higher than the minimum achievable, whereas the inter-node traffic is almost 35 times higher than the optimal one.

When S-ODRP optimizes the cost C (**S-ODRP_C**), the placement solution tries to use fewer replicas as possible, as shown in Figure 6d. However, the explicit modeling of the operator service rate enables to instantiate a configuration that can properly handle the incoming traffic: S-ODRP instantiates 9 replicas, i.e., replicates twice the bottleneck operator (*partialRank*). Nothing can be concluded about the other QoS metrics, i.e., response time, application availability, and inter-node traffic (see Figures 6a, 6b, and 6c): since these metrics are not optimized, there is a set of equally optimal solutions that differ each other only for the operator placement on the same set of computing resources.

When S-ODRP optimizes response time (**S-ODRP_R**), the placement solution experiences the minimum achievable value for this metric, as shown in Figure 6a, but it uses up to 16 replicas (Figure 6d). Observe that 16 replicas completely occupy two worker nodes, and the presence of network delays prevents the scheduler from instantiating other replicas. Since the cost of running a configuration is directly proportional to the total number of operator instances, this is also the most expensive solution. From Figure 6c, we can also observe that the inter-node traffic is quite low (almost double with respect to the optimal value), because transmitting data over the network rather than locally introduces network delays that penalize the response time.

When S-ODRP optimizes the network-related QoS metric, that is the inter-node traffic T (**S-ODRP_T**), the solution tries to co-locate the operator instances on the least number of nodes, so to reduce the amount of data transmitted over the network (see Figure 6c). In this configuration the number of replicas is neither minimized nor maximized; however, if the load is equally split among replicas, S-ODRP might take advantage of replication in order to reduce the amount



(a) Response time

(b) Availability

(c) Inter-node traffic

(d) Total number of operator instances

Figure 6: Impact of different optimization objectives on the QoS metrics

of data transmitted on the network. As side effect of the co-location, the application response time is, on average, very close to the optimal one. Nothing can be concluded about the application availability (Figure 6b), which is not considered as an optimization objective of S-ODRP_T.

When S-ODRP optimizes all the considered QoS metrics, the resulting placement and replication solution has response time and inter-node traffic which are very close to the values achievable when optimizing a single-objective function. The total number of operator instances is equal to 9, as shown in Figure 6d, therefore only the bottleneck operator is replicated. The application availability ranges from 100% to 99%, and assumes 99.6% as average value. Although it might appear counter-intuitive, this result follows from the trade-off between the minimization of response and the maximization of the availability pursued in the utility function F. In particular, when the application availability is 99%, on the basis of the pinned operators placement, whose location is randomly defined a-priori, choosing a worker node that minimizes the response time, rather than one that maximizes the availability, provides a bigger contribution to the optimization of the utility function F.

On ODRP Resolution Time. We now discuss about the *resolution time* of ODRP and its relationship with the optimization goals. We consider as resolution time the time needed to compute the exact solution of the ILP problem. Although the investigated placement problem is fairly limited in size ($|V_{res}| = 6$, $|V_{dsp}| = 8$), the ODRP model includes about 55.8 K variables and, considering all the 25 runs of the last experiment, its average resolution time is 10.84 s. Figure 7 provides more details on the average resolution time of ODRP with respect to the different weights (w_a, w_c, w_r, and w_z) used for the utility function F. Determining a placement that minimizes the application response

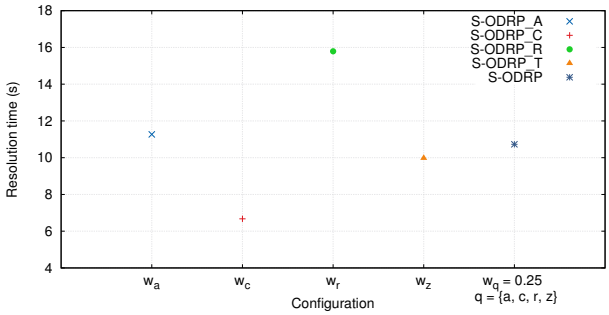


Figure 7: Average resolution time of ODRP with respect to the optimization objective

time is the most computationally demanding configuration; conversely, the minimization of the application deployment costs registers the fastest resolution time. Note that the former is twice slower than the latter. Being ODRP an NP-hard problem, as demonstrated in Theorem 1, it does not scale well as the problem instance increases in size. Nevertheless, by determining the optimal replication and placement of DSP operators, ODRP provides a benchmark for evaluating heuristics, for developing new ones, and for identifying the most suitable ones with respect to the specific optimization objectives.

7. CONCLUSIONS

In this paper, we have presented and evaluated ODRP, an ILP formulation that jointly optimizes the replication and placement of DSP applications. ODRP is a general and flexible model that can take into account the heterogeneity

of computing and networking resources and can be conveniently configured to optimize different QoS metrics, whose importance depends on the application scenario. With S-ODRP, we have developed an ODRP-based prototype scheduler for Apache Storm, one of the widely used open-source DSP frameworks. Then, relying on an application that processes real time data generated by taxis moving in a urban environment (DEBS 2015 Grand Challenge), we have conducted a thorough experimental evaluation. The latter has shown the benefits of a joint optimization of operators replication and placement on the application performance and how ODRP can contextually optimize several QoS metrics.

As future work, we plan to develop efficient heuristics to deal with large problem instances for the initial application replication and placement. Moreover, we plan to extend ODRP in order to support run-time adaptations of the operator deployment, so that a DSP application can efficiently handle input streams or execution environments with continuously changing characteristics. With the aim of optimizing reconfigurations, we will model their impact in terms of application performance degradation (e.g., temporary increment of response time due to operator state migrations).

8. ACKNOWLEDGMENTS

This publication is supported by COST Action ACROSS (IC1304).

9. REFERENCES

[1] L. Aniello, R. Baldoni, and L. Querzoni. Adaptive online scheduling in Storm. In *Proc. of ACM DEBS '13*, pages 207–218, 2013.

[2] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. Distributed QoS-aware scheduling in Storm. In *Proc. of ACM DEBS '15*, pages 344–347, 2015.

[3] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. Joint operator replication and placement optimization for distributed streaming applications. In *Proc. of InfQ '16 (in conjunction with VALUETOOLS '16)*, 2016.

[4] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. Optimal operator placement for distributed stream processing applications. In *Proc. of ACM DEBS '16*, pages 69–80, 2016.

[5] V. Cardellini, M. Nardelli, and D. Luzi. Elastic stateful stream processing in Storm. In *Proc. of HPCS '16*, pages 583–590. IEEE, 2016.

[6] A. Chatzistergiou and S. D. Viglas. Fast heuristics for near-optimal task allocation in data stream processing over clusters. In *Proc. of ACM CIKM '14*, 2014.

[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. *SIGCOMM Comput. Commun. Rev.*, 34(4), 2004.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. of OSDI '04*, pages 137–150. USENIX Association, 2004.

[9] R. Eidenbenz and T. Locher. Task allocation for distributed stream processing. In *Proc. of IEEE INFOCOM '16*, 2016.

[10] L. Fischer, T. Scharrenbach, and A. Bernstein. Scalable linked data stream processing via network-aware workload scheduling. In *Proc. of SSWS '13*, pages 81–96, 2013.

[11] T. Z. J. Fu, J. Ding, R. T. B. Ma, M. Winslett, et al. DRS: Dynamic resource scheduling for real-time analytics over fast streams. In *Proc. of IEEE ICDCS 2015*, pages 411–420, 2015.

[12] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu. Elastic scaling for data stream processing. *IEEE Trans. Parallel Distrib. Syst.*, 25(6):1447–1463, 2014.

[13] T. Heinze, L. Aniello, L. Querzoni, and Z. Jerzak. Cloud-based data stream processing. In *Proc. of ACM DEBS '14*, pages 238–245, 2014.

[14] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer. Latency-aware elastic scaling for distributed data stream processing systems. In *Proc. of ACM DEBS '14*, pages 13–22, 2014.

[15] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. Auto-scaling techniques for elastic data stream processing. In *Proc. of IEEE ICDEW '14*, pages 296–302, 2014.

[16] T. Heinze, L. Roediger, A. Meister, Y. Ji, et al. Online parameter optimization for elastic data stream processing. In *Proc. of ACM SoCC '15*, pages 276–287, 2015.

[17] Z. Jerzak and H. Ziekow. The DEBS 2015 grand challenge. In *Proc. of ACM DEBS '15*, pages 266–268. ACM, 2015.

[18] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, et al. Twitter Heron: Stream processing at scale. In *Proc. of ACM SIGMOD '15*, pages 239–250, 2015.

[19] T. Li, J. Tang, and J. Xu. A predictive scheduling framework for fast and distributed stream data processing. In *Proc. of IEEE Big Data '15*, 2015.

[20] B. Lohrmann, P. Janacik, and O. Kao. Elastic stream processing with latency guarantees. In *Proc. of IEEE ICDCS '15*, pages 399–410, 2015.

[21] G. Mencagli. A game-theoretic approach for elastic distributed data stream processing. *ACM Trans. Auton. Adapt. Syst.*, 11(2):13:1–13:34, 2016.

[22] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, et al. Network-aware operator placement for stream-processing systems. In *Proc. of IEEE ICDE '06*, 2006.

[23] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things J.*, 3(5):637–646, Oct. 2016.

[24] C. Thoma, A. Labrinidis, and A. Lee. Automated operator placement in distributed data stream management systems subject to user constraints. In *Proc. of IEEE ICDEW '14*, pages 310–316, 2014.

[25] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, et al. Storm@Twitter. In *Proc. of ACM SIGMOD '14*, pages 147–156, 2014.

[26] J. Xu, Z. Chen, J. Tang, and S. Su. T-Storm: traffic-aware online scheduling in Storm. In *Proc. of IEEE ICDCS '14*, pages 535–544, 2014.

[27] K. P. Yoon and C.-L. Hwang. *Multiple Attribute Decision Making: an Introduction*. Sage Pubs, 1995.

[28] M. Zaharia, M. Chowdhury, T. Das, A. Dave, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. of NSDI '12*. USENIX Association, 2012.

Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications

Eugenio Gianniti
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Via Golgi 42 20133 Milano, Italy
eugenio.gianniti@polimi.it

Alessandro Maria Rizzi
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Via Golgi 42 20133 Milano, Italy
alessandromaria.rizzi@polimi.it

Enrico Barbierato
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
via Ponzio 34/5 20133 Milano, Italy
enrico.barbierato@polimi.it

Marco Gribaudo
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
via Ponzio 34/5 20133 Milano, Italy
marco.gribaudo@polimi.it

Danilo Ardagna
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Via Golgi 42 20133 Milano, Italy
danilo.ardagna@polimi.it

ABSTRACT

Big Data applications allow to successfully analyze large amounts of data not necessarily structured, though at the same time they present new challenges. For example, predicting the performance of frameworks such as Hadoop and Spark can be a costly task, hence the necessity to provide models that can be a valuable support for designers and developers. Big Data systems are becoming a central force in society and the use of models can also enable the development of intelligent systems providing Quality of Service (QoS) guarantees to their users through runtime system reconfiguration.

This paper provides a new contribution in studying a novel modeling approach based on fluid Petri nets to predict MapReduce and Spark applications execution time which is suitable for runtime performance prediction.

Models have been validated by an extensive experimental campaign performed at CINECA, the Italian supercomputing center, and on the Microsoft Azure HDInsight data platform. Results have shown that the achieved accuracy is around 9.5% for Map Reduce and about 10% for Spark of the actual measurements on average.

Keywords

Spark, MapReduce, Hadoop, fluid Petri nets

1. INTRODUCTION

The implementation of Big Data applications is steadily growing today [17]. According to a recent analysis, the Big Data market reached $16.9 billion in 2015 with a compound annual growth rate (CAGR) of 39.4%, about seven times the one of the overall ICT market [12].

From the technological perspective, MapReduce is capable of analyzing very efficiently large amounts of unstructured data, i.e., it is a viable solution to support both the variety and volume requirements of Big Data analyses [19]. MapReduce has been adopted in multiple application domains, e.g., machine learning, graph processing, and data mining [32]. Its open source implementation, Hadoop 2.x, recently introduced a wide set of performance enhancements (e.g., SSD support, caching, and I/O barriers mitigation). IDC estimates that Hadoop touched half of the world data last year [18], supporting both traditional batch and interactive data analysis applications [30] and at a CAGR of 58%.[1].

Nevertheless the rigid division between Map and Reduce requires to subdivide a complex application into a directed acyclic graph (DAG) of MapReduce jobs, comprising tasks that perform a specific computation on partitions/splits of the input data. In this case, the MapReduce paradigm forces the storage of each intermediate phase results on disk, thus being unsuitable for applications requiring a low latency between different phases, along with general application Quality of Service (QoS) guarantees. Other frameworks, such as Tez and Spark, have been introduced [29, 35] to address this problem. Although Tez can handle general DAGs of MapReduce phases, it still requires to write each stage result on disk. On the other hand, Spark can exploit a set of primitives to request the cashing of partial results in memory, thus allowing lower latency and better performance.

Spark has been developed on the resilient distributed dataset (RDD) concept [34], a novel distributed memory abstraction providing a restricted form of memory sharing. In practice, Spark can easily obtain a 10x speedup over Hadoop on specific scenarios [36].

In this context, one of the main challenges [22, 31] is that the execution time of MapReduce and Spark applications is generally unknown in advance. Because of this, predicting the execution time of Hadoop/Spark jobs is usu-

[1]http://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#5138509c6c1d

ally done empirically through experimentation, requiring a costly setup [15].

In addition Big Data systems are becoming a central force in society; thus requiring the development of intelligent systems providing QoS guarantees to their users.

Performance prediction models are extremely useful to aid development and deployment of Big Data applications; either for design time decisions or run time system reconfiguration. Design time models can help, e.g., to determine the appropriate size of a cluster or to predict the budget required to run Hadoop/Spark in public Clouds: a trending scenario, since by 2020 nearly 40% of Big Data analyses will be supported by public Clouds [12]. Such models can be used also at run time, allowing a dynamic adjustment of the system configuration [3, 28], e.g., to cope with workload fluctuations or to reduce energy costs.

While in early Hadoop versions CPU *slots* and other resources were separated between mappers and reducers using a static approach, in Hadoop 2.x *containers* (both for MapReduce and Spark) are distributed among ready tasks in a dynamic fashion. On the one hand, this allows a better cluster utilization, on the other hand performance modeling became much more difficult. In Spark, cluster resources are scheduled to process part of the operations on RDDs: to obtain an RDD, Spark first builds a DAG with its dependencies, then processes each stage providing a certain amount of resources, based on data locality.

The originality of this paper consists in a new modeling technique capable of catching the system behavior under the dynamic assignment of the available cluster resources. We assume that the cluster is governed by the Capacity Scheduler, which partitions the available resources among multiple customers through queues, each queue being regulated by a FIFO policy.

This work proposes a fluid Petri net (FPN) model to estimate at runtime MapReduce and Spark jobs execution time, combining real experimentation and model-based evaluation, while also exploring different properties of the considered Big Data frameworks to unveil the characteristics of the YARN Capacity Scheduler that have the highest influence in its performance and therefore should be represented in the models, so as to obtain a reliable evaluation.

The accuracy of the model is evaluated on real systems by performing experiments based on the TPC-DS industry benchmark for business intelligence data warehouse applications. The Italian supercomputing center, CINECA, and the Microsoft Azure HDInsight [2] data platform have been considered as target deployment.

Results and experiments performed on real systems have shown that the achieved accuracy is within around 9.5% for Map Reduce and about 10% for Spark of the actual measurements on average. With respect to previous literature, to the best of our knowledge this article is one of the first contributions able to study the performance of Hadoop-2.x MapReduce and Spark-based clusters, with dynamic allocation of resources.

This work extends the previous paper presented in [13] by adding the linear blending of the deterministic and exponential approximation, and validates the Spark models by taking into account generic instead of two-phases Map and Reduce DAGs. It is organized as follows: Section 2 compares this work with other proposals available in the literature; Section 3 presents our novel proposals for MapReduce and Spark modeling, via FPNs. Next, Section 4 reports some experimental results to validate and study the properties of our models. Finally, Section 5 shows how to deploy models supporting capacity planning decisions and Section 6 draws the conclusions, providing the lines for future work.

[2] https://azure.microsoft.com/en-us/services/hdinsight/

2. RELATED WORK

The literature provides a large number of performance studies for Hadoop 1.x, since the framework has been widely adopted in the ICT industry, often supporting core business activities. Two main approaches have been explored: i) simulation-based models implement the single constituents of Hadoop and of the job, replaying in a simulated environment the steps and delays of the real system; ii) analytical models, on the other hand, define a mathematical representation of those constituents, avoiding the costs of running multiple simulations. Both approaches make use of information such as input dataset size, cluster resources, and Hadoop specific parameters. The computational effort and the time spent in running simulation-based models make them hardly fit for the purposes of runtime cluster management: thus, we hereby consider only analytical models, according to the focus of our paper.

The authors of [31] propose the ARIA framework for estimating the makespan of jobs in MapReduce clusters. This approach relies on information extracted from the logs of previous executions of similar jobs. Adopting scheduling techniques, the authors prove lower and upper bounds on makespans. From these results, they derive formulae for performance prediction. They obtain both a conservative estimate, suitable for hard deadlines, and an alternative that does not offer guarantees of meeting deadlines, but boasts a relative error below 10% in the validation against measured timings.

Another performance model estimating the execution time by considering the single costs of the various phases of a MapReduce job is described in [23]. In this work, the authors go down to the very low level elements that determine the cost of single job phases, writing a 37-parameter model that provides execution times within 10% of those measured in a real cluster. Even with such an accurate model, the validation considers just single job executions.

In queueing network (QN) literature, the fork/join paradigm (see [26] and [5]) is used to denote the modeling of the concurrent execution of many tasks within higher level jobs. Specifically, this approach operates through two steps: i) jobs are spawned at a fork node in multiple tasks, then ii) they are submitted to queueing stations that, in turn, model the available servers.

Once all the tasks have been served, they can synchronize at a join node. It has to be noted that when a fork/join network has more than two queues, a closed form solution is not possible (see [24]). That said, it is possible to mitigate the issue by using a special kind of structure, as shown in [21], which considers the Markov chain (MC) underlying the QN representing the possible states of the system. Unfortunately, as noted in [10, 24], the state space grows exponentially when the tasks number corresponds to realistic MapReduce jobs—in the order of thousands—thus making the above approaches unsuitable.

An interesting alternative in the shape of approximation

methods is introduced in [25], which proposes a method based on exponentially distributed service times, though it is not applicable to Hadoop deployments. Indeed, as per preliminary experiments conducted in this work, it is safe to assume that phase type (or in some cases Erlang, see also [2]) can approximate the general distributions of mapper and reducer times, while assuming an exponential distribution led to around 50–60% [11] relative error on the prediction of the overall job execution time. To this concern, see also [21], where the authors propose an approximate mean value analysis technique using an iterative hierarchical approach.

Other approaches prefer adopting a MapReduce modeling based on Petri nets (PNs). For example, in [9] the authors use a stochastic PN, applying mean field analysis to calculate average metrics and estimate the performance of a Big Data architecture based on Hadoop.

In order to capture the performance of Hive queries, generalized stochastic Petri nets together with other formalisms (i.e., process algebras or MCs) are used to create multi-formalism models in [6]. Specifically, the authors show how performance can depend on some configuration parameters.

In [1], the authors propose colored PNs to determine the feasibility of a distributed file system project. After designing a deployment of HDFS, which uses a set of spare resources in a cluster of workstations to provide a sufficiently available distributed file system, the system availability is assessed by exploiting PNs.

Though earlier methods exploited static resource allocation (at different levels of detail) to model Hadoop 1.0 clusters, the FPN models used here can capture the dynamic assignment of YARN resource containers (for example, by using the model presented in [8]) and are capable of estimating performance of Hadoop 2.x jobs very efficiently.

Spark is a powerful framework oriented to big data processing. It allows users to quickly build applications, combining SQL and Data Analytics. Spark performance prediction can be a challenging task due to different factors: in [33], the authors proposed a simulation driven prediction model that, taking in account only part of the traces log, predict the performance of jobs execution. Recently, given the framework complexity, black box approaches based on machine learning have been proposed to predict Spark applications performance as a function of data sets size, executors memory and number of cores [14]. However, machine learning models usually require a long training phase and are usually not suitable to generalize their prediction to different configuration settings [4].

3. FLUID MODELS

Very often modelers are bound to face several problems when trying to provide a description of a physical system by using a formalism. Typically, these problems are more evident in techniques using discrete states, since the analysis produces a state space explosion, with an exponential growth in the number of states following the complexity of the model. Different alternatives to mitigate this problem have been proposed. Specifically, *hybrid* techniques involving a continuous and discrete part have proved to reduce significantly the severity of the issue. Continuous variables can be exploited in different scenarios: for example, they can be used to represent the rising temperature in a closed room or the water leaking out of a full bucket.

In literature, fluid models have been presented in different ways for what concerns the realization of the continuous aspect. Among the different flavors, in [16], the authors refer to i) fluid stochastic Petri nets (FSPNs), ii) reward and iii) fluid models. FSPNs add to stochastic Petri nets introducing the ability to handle continuous parts. In reward models, the idea consists in using a MC and associating a reward rate, a positive weight whose value depends on the time spent in a particular state.

Fluid models can be used to successfully study the MapReduce framework and even more sophisticate ones, such as Spark [35], including paradigms like concurrent programming.

In the following, Section 3.1 describes the FSPN used, while Section 3.2 describes how to generate an approximation of the average execution time of jobs in the deterministic limiting case, while Section 3.3 considers the exponential limiting case. Section 3.4 considers a convex combination of the previously presented limiting cases; finally, Section 3.5 adds Spark generalization.

3.1 FSPN Model for a MapReduce Job

In order to formalize the fluid model proposed in this paper, let us focus on the MapReduce paradigm, and describe it using the FSPN shown in Figure 1. In Section 3.5 the model will be generalized to Spark jobs. Note that the purpose of using a FSPN model is just to specify the underlying fluid model without enumerating its states: for this reason we will just give a brief description of the conventions adopted in the formalism used.

In this formalization, single circles represent *discrete places*, which can hold an integer number of tokens; single boxes represent *timed transitions* that can fire after an exponential amount of time; bars represent *immediate transitions*, which fire as soon as they are enabled; thin lines define *standard arcs* that move tokens among places and enable the connected transitions; and thin lines with circular ends define *inhibitor arcs*, which prevent the corresponding transitions from firing whenever the input place is marked. Double circles represent *fluid places*, which can hold a continuous amount of fluid; double boxes identify *fluid transitions*, which continuously pump fluid in and out of continuous places; double arrows represents *fluid arcs* that can remove fluid from their input places; and thick arrows represents *set arcs*, that immediately insert a given amount of fluid in their destination place when their input transition fires.

The discrete amount of tokens is moved across the discrete arcs as usual. With regard to the fluid places, they include a level denoted by a continuous variable, which flows according to an instantaneous rate. The discrete FSPN component regulates the fluid flow through the continuous part, while the conditions enabling a transition depend on the discrete component only.

The model of the MapReduce paradigm considers N users (corresponding to the marking of place `Users`) that can submit jobs to the system after a think time Z. The submission of jobs from a user is modeled by the firing of transition `Think` characterized by the infinite server semantic. According to the YARN Capacity Scheduler, we consider that only one job at a time can be executed by the system: this is obtained via place `Available`, which holds a token whenever the infrastructure is ready to run a new job, thus enabling the corresponding `Start` transition. Both the Map and Reduce

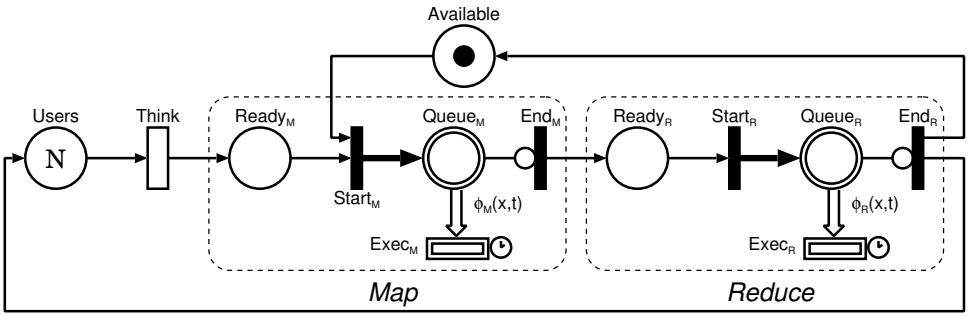


Figure 1: FSPN representation of the fluid model underlying a MapReduce job.

phases (as a generic Spark stage) are modeled by a similar sub-network. Place `Ready` holds a token whenever the phase is ready to start, and its beginning is modeled by the firing of the immediate transition `Start`. This transition inserts, via its output set arc, the number of tasks (either map or reduce) that need to be executed in the corresponding phase in fluid place `Queue`. The execution of the jobs is modeled by the time-dependent fluid transition `Exec`, which is connected to place `Queue` with an output arc. Transitions `Exec` have a special time-dependent semantic that will be described in the following paragraphs: for this reason it is represented with a small clock drawn at its side. Whenever the queue is empty, the next phase can start or the job can end thanks to the firing of transition `End`.

Following [7], the fluid evolution is defined as a function $\phi : \mathbb{R}^2 \to \mathbb{R}$, which defines how the fluid level of the corresponding place changes with time. In particular, $\phi(x,t)$ represents the fluid level reached by a place at time t, given that it starts with level x at time $t=0$. This semantic is represented graphically by drawing a fluid arc that connects a fluid place (`Queue` in Figure 1) to a time-dependent fluid transition (`Exec` in Figure 1). In the model, two functions $\phi^{\mathrm{M}}(x,t)$ and $\phi^{\mathrm{R}}(x,t)$ are associated respectively to the map and reduce phases. These functions regulate the evolution of the fluid in the corresponding places, so that the fluid level x represents the average remaining number of tasks that still need to be executed in a phase.

Task duration distributions can be generally regarded as phase type or Erlang [2], parameterized according to the observed coefficient of variation (CV). The CV ranges between 0 and 1, whence the two limiting cases of deterministic and exponential distribution, respectively. In this paper we approximate general distributions with proper convex combinations of the limiting cases that have either deterministic or exponential task execution times.

3.2 Deterministic Task Execution Time

Let us suppose that our MapReduce jobs are executed by C containers. Let us also assume that a phase is composed of N tasks, and that each task requires a deterministic time T to be executed. Figure 2 represents the evolution of the number of remaining tasks as function of time, which in our fluid model corresponds to the fluid evolution function $\phi(N,t)$. At time $t=0$, C out of N tasks are immediately assigned to all the containers. Since their duration is deterministic, all the C tasks will end at the same time T, leaving just $N-C$ tasks left to be executed in the system. Then the next batch of C tasks will start, and it will end at $2T$, leaving in

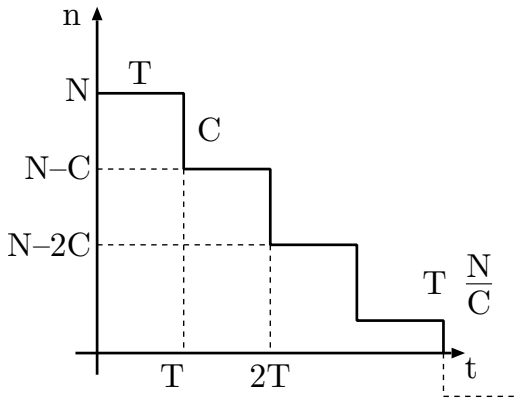


Figure 2: Fluid evolution: Deterministic task execution time

the system $N-2C$ jobs. The function $\phi_{\mathrm{d}}(x,t)$ can then be defined as follows:

$$\phi_{\mathrm{d}}(x,t) = \max\left(0, x - \left\lfloor \frac{t}{T} \right\rfloor C\right) \tag{1}$$

In this scenario, the time $\tau_{\mathrm{d}}(N,C,T)$ required to run N tasks with deterministic running time T on C containers can be computed as:

$$\tau_{\mathrm{d}}(N,C,T) = T\left\lceil \frac{N}{C} \right\rceil \tag{2}$$

3.3 Exponential Task Execution Time

Let us now suppose that the task execution time is exponentially distributed, with average execution time T. Since the minimum of n exponential distributions of rate μ is exponentially distributed with rate $n\mu$, we can describe the evolution of the number of remaining jobs with the death-only birth-death process represented in Figure 4, where $\mu = \frac{1}{T}$. In particular, the average sojourn time in states N to C is $\frac{T}{C}$, while for the states $c \in \{C-1,\ldots,1\}$ is $\frac{T}{c}$. This can lead us to the definition of $\phi_{\mathrm{e}}(x,t)$, as shown in Figure 3. In particular, when $x > C$, the number of jobs decreases at rate $\frac{C}{T}$. Then, the decrease rate reduces at $\frac{\lceil x \rceil}{T}$ when $x < C$. In particular we have:

$$\phi_{\mathrm{e}}(x,t) = \begin{cases} x - t\frac{C}{T}, & t < \frac{N-C}{C}T \\ c - (t - t_c)\frac{c}{T}, & t_c \le t < t_{c-1} \end{cases} \tag{3}$$

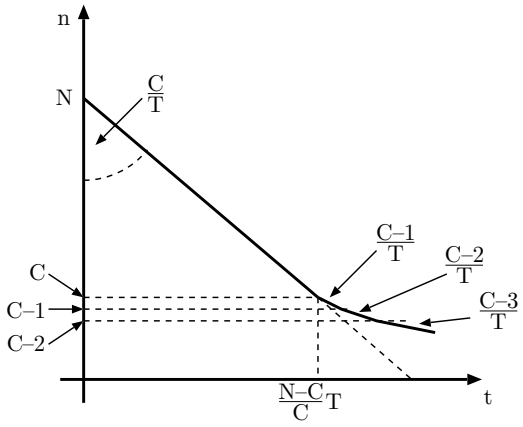


Figure 3: Fluid evolution: Exponential task execution time

where t_c represents the time at which only c of the nodes are still in use, and it can be computed in a recursive way starting from $c = C$ down to $c = 1$ in the following way:

$$t_c = \begin{cases} \frac{N-C}{C}T, & c = C \\ t_{c+1} + \frac{T}{c}, & 1 \leq c < C \end{cases} \quad (4)$$

In this scenario, the time $\tau_{\mathrm{e}}(N,C,T)$ required to run N tasks with exponentially distributed running time with average T on C nodes can be computed as:

$$\tau_{\mathrm{e}}(N,C,T) = T\left(\frac{N-C}{C} + \sum_{c=1}^{C}\frac{1}{c}\right) \quad (5)$$

3.4 General Task Execution Time

Typically, real data does not follow exactly neither deterministic nor exponential service time distributions, instead we can observe samples with diverse CVs. In order to take into account this variability and apply the proposed technique in the general case, our approach considers a convex combination of the previously presented limiting cases. The fluid evolution function becomes:

$$\phi(x,t) = \alpha\phi_{\mathrm{d}}(x,t) + (1-\alpha)\,\phi_{\mathrm{e}}(x,t) \quad (6)$$

which, in turn, yields:

$$\tau(N,C,T,\alpha) = \alpha\tau_{\mathrm{d}}(N,C,T) + (1-\alpha)\,\tau_{\mathrm{e}}(N,C,T) \quad (7)$$

Given the experimental data, we can profile each phase to extract the number of involved tasks and the average service time. Moreover, the α parameter can be obtained minimizing the absolute error between the measured phase duration and the predicted time τ. For instance, a MapReduce fluid job profile consists of the following parameters: N^{M}, T^{M}, α^{M}, N^{R}, T^{R}, and α^{R}.

3.5 Spark DAG stages generalization

Spark applications and jobs are characterized by generic DAGs, as shown in Figure 5. A generalized model to support Spark jobs should then use more fluid places to represent the various stages of the DAGs, and exploit the discrete part of the model to properly share the available resources among the tasks that are ready to start. This in-depth analysis is however future work and it will not be carried out in this paper. However, in many cases a DAG can be simplified as a sequence of stages. This result has been proven in scheduling theory for real systems in which the number of tasks N is greater than the available resources C (see [27]) stating that a DAG execution can be approximated with a stage sequence respecting precedence constraints. In this paper we will take a critical path approach where we consider only the set of stages which determines the job execution time: in particular, we will start from the execution traces of the considered application, and remove the stages that are not relevant to determine their duration. Figure 5 shows a Spark query (analyzed in the experimental section) consisting of different jobs of which only 4 are taken in account during the execution, as the parallel stages not belonging to the critical path are discarded (see Section 4.3 for additional details).

4. EXPERIMENTS

In this section, we describe the results of the experiments we conducted to validate our approach. Experiments have been performed at CINECA, the Italian supercomputing center and on HDInsight, the Microsoft Azure cloud solution.

The experiments consist of a set of Hive queries taken from the TPC-DS benchmark suite and run on a dedicated Hadoop MapReduce and Spark cluster.

The rest of the section is organized as follows: Section 4.1 thoroughly describes how the experiments have been executed; sections 4.2 and 4.3 present the obtained results by applying the fluid models for Map Reduce and Spark, respectively.

4.1 Experimental Settings

The models presented in the previous sections have been validated with an experimental campaign on CINECA's (the Italian supercomputing center) PICO cluster.

PICO[3], the Big Data cluster available at CINECA, is composed of 74 nodes, each of them boasting two Intel Xeon 10-core 2670 v2@2.5GHz, with 128 GB RAM per node. Out of this 74 nodes, up to 66 are available for computation. In our experiments on PICO, we used several configurations ranging from 60 to 120 cores and set up the scheduler to provide one container per core.

The cluster is shared among different users, hence resources are managed by the Portable Batch System (PBS). PBS allows for submitting jobs and checking their progress, configuring at a fine-grained level the computational requirements: for all submissions it is possible to request a number of nodes and to define how many CPUss and what amount of memory are needed on each of them. Since the cluster is shared among different users, the performance of single jobs depends on the overall system load, even though PBS tries to split the resources. Due to this, it is possible to have large variations in performance according to the total usage of the cluster. In particular, storage is not handled directly by PBS, thus leading to an even greater impact on performance.

We tried to mitigate this variability first of all by requesting entire nodes of the cluster for the execution of our experiments. In such a way, we could be sure that nobody else could run other jobs on the same nodes, thus interfering with

[3] `http://www.hpc.cineca.it/hardware/pico`

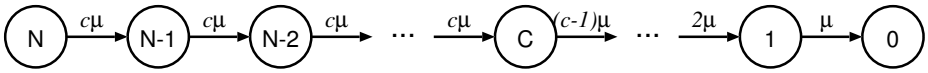


Figure 4: Markov model for MapReduce or Spark phases with exponential service times.

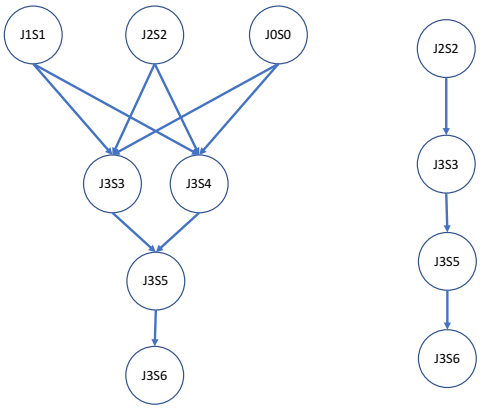


Figure 5: TPC-DS Q26 DAG and critical path

the performance measurement.

An ephemeral Hadoop cluster has been created at the beginning of each experiment on the allocated nodes. The PICO cluster provides the myHadoop tool for setting up a Hadoop 2.5.1 cluster, upon which we used Hive 1.2.1. HDFS is kept locally on the selected nodes, in order to experience lower variability than the one observed using the centralized storage.

In spite of these settings, still the experiments showed high variability, in particular with a few runs characterized by extremely high execution time. In our analysis we discarded runs with an anomalous execution time, taking out all the experiments that lie more than three standard deviations away from the average computed for the same configuration.

The Azure HDInsight cluster [4] used during our experiments is composed of two *D3* virtual machines (VMs) for resource manager and a varying number of *A3* VMs for worker nodes, containing the Spark executors. Each *D3* and *A3* machine provides four CPU cores and 7 GB of memory. We used Spark version 1.6.2, configured with a driver using 2 GB of memory and each executor using two cores and 2 GB of memory. The total number of available physical cores varied between 6 and 24 with step 2 (the number of physical cores to containers mapping is 1:1 also for this second scenario).

The dataset used for testing has been generated using the TPC-DS benchmark[5] data generator, creating at a scale factor ranging from 250 GB to 1 TB several files directly used as external tables by Hive. We chose the TPC-DS benchmark as it is the industry standard for benchmarking data warehouses.

As MapReduce benchmark, we introduced five *ad hoc*

```
select avg(ws_quantity),
       avg(ws_ext_sales_price),
       avg(ws_ext_wholesale_cost),
       sum(ws_ext_wholesale_cost)
from web_sales
where (web_sales.ws_sales_price
       between 100.00 and 150.00)
  or (web_sales.ws_net_profit
      between 100 and 200)
group by ws_web_page_sk
limit 100;
```

(a) R1

```
select inv_item_sk, inv_warehouse_sk
from inventory
where inv_quantity_on_hand > 10
group by inv_item_sk, inv_warehouse_sk
having sum(inv_quantity_on_hand) > 20
limit 100;
```

(b) R2

```
select avg(ss_quantity),
       avg(ss_net_profit)
from store_sales
where ss_quantity > 10
  and ss_net_profit > 0
group by ss_store_sk
having avg(ss_quantity) > 20
limit 100;
```

(c) R3

```
select cs_item_sk, avg(cs_quantity) as aq
from catalog_sales
where cs_quantity > 2
group by cs_item_sk;
```

(d) R4

```
select inv_warehouse_sk,
       sum(inv_quantity_on_hand)
from inventory
group by inv_warehouse_sk
having sum(inv_quantity_on_hand) > 5
limit 100;
```

(e) R5

Figure 6: Interactive queries

[4]https://azure.microsoft.com/en-us/pricing/details/virtual-machines/

[5]http://www.tpc.org/tpcds/

Table 1: Fitted parameters

Query	d [GB]	n^{M}	$\bar{t}^{\mathrm{M}}$ [ms]	n^{R}	$\bar{t}^{\mathrm{R}}$ [ms]
R1	250	144	25970	151	2346
R1	500	287	32159	300	1958
R1	750	434	34244	455	1996
R1	1000	591	40534	619	3063
R2	250	4	57326	4	8785
R2	500	2	42202	2	6582
R2	750	3	48869	3	7500
R2	1000	65	1082274	68	13086
R3	250	381	28659	400	2369
R3	500	757	37018	793	2570
R3	750	1148	42348	1009	2785
R3	1000	1560	41961	1009	3048
R4	250	288	25087	302	2958
R4	500	573	41007	601	2961
R4	750	868	43902	910	3259
R4	1000	1183	42615	1009	8667
R5	250	4	13456	4	1424
R5	500	2	11774	2	1499
R5	750	3	12682	3	1462
R5	1000	64	19557	68	1610

queries[6], which are mapped as a single MapReduce job in Hive. This queries are dubbed R1–5 and are shown in Figure 6. The profiling phase has been conducted by extracting average task durations from at least twenty runs of each query. The numbers of map and reduce tasks varied, respectively, in the ranges $[2, 1560]$ and $[2, 1009]$. The discussed parameters are shown in Table 1. In the table we report the scale factor d, the number of tasks of each phase, respectively n^{M} and n^{R} for mappers and reducers, and the average task durations, respectively $\bar{t}^{\mathrm{M}}$ and $\bar{t}^{\mathrm{R}}$.

As Spark benchmark, we used the official TPC-DS queries, addressed respectively as queries Q26 and Q52. The DAG of query Q26 is shown in Figure 5, while Figure 17 reports Q52. Results where computed using a number of containers that varies from 6 to 24. These parameters are shown in Table 4. In the table we report the scale factor d, the number of tasks of each stage n, and the average task duration $\bar{t}$ measured for the cases with the minimum and maximum number of containers. The DAG execution takes into account the critical path only. In case of jobs that can be run in parallel (such as J1S1, J2S2 and J0S0 and J3S3, J3S4), the path will choose the slowest job being the one who has the most significant weight. In a way, the process simplifies the given DAG.

Models accuracy will be evaluated as the relative error (ε) between the average jobs execution time measured in the system (t) and execution time predicted by the model (τ) :

$$\varepsilon = \frac{\tau - t}{t} \tag{8}$$

Models evaluation took always less than one second on an Intel i7@3.1 GHz laptop, demonstrating their suitability for runtime performance prediction.

4.2 Analysis of MapReduce Jobs

[6] https://github.com/deib-polimi/Hive-Experiment-Runner

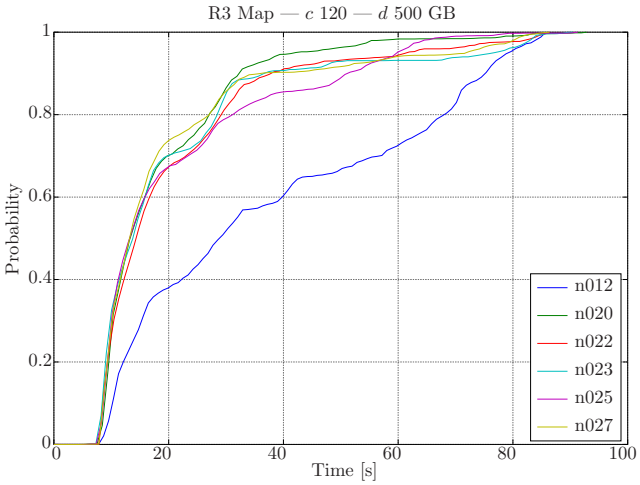


Figure 7: R3 map, 120 containers, 500 GB dataset

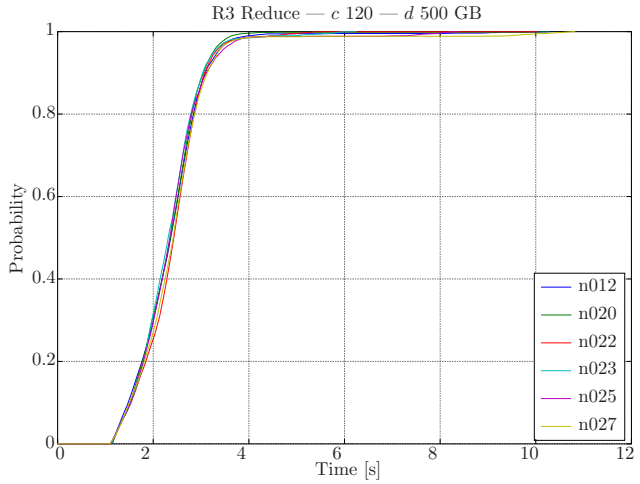


Figure 8: R3 reduce, 120 containers, 500 GB dataset

In this section we present and discuss the results obtained with the previously described fluid techniques. First of all, we studied the empirical cumulative distribution functions (CDFs) of task durations on different nodes, in order to identify the cases in which performance was strongly affected by exogenous interference. We then considered the accuracy that can be reached adopting the approach based on a convex combination of the deterministic and exponential limiting cases.

Figures 7, 8, 9 and 10 show the empirical CDFs derived from the measurements. Figures 7 and 8 refer to subsequent phases of the same experimental run, namely, query R3 running on the six-node, 120-container cluster deployment over the 500 GB dataset. Accordingly, Figures 9 and 10 show query R5 on the three-node, 60-container cluster over the 750 GB dataset.

As a general trend well represented in Figures 8 and 10, reducers tend to behave quite regularly. Most likely, possible variabilities spread across the shuffle stage that overlaps with the map phase, hence end up being hardly noticeable in each reducer task duration.

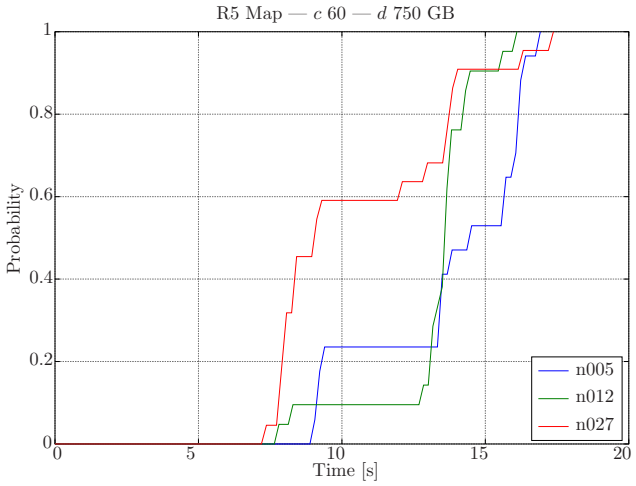


Figure 9: R5 map, 60 containers, 750 GB dataset

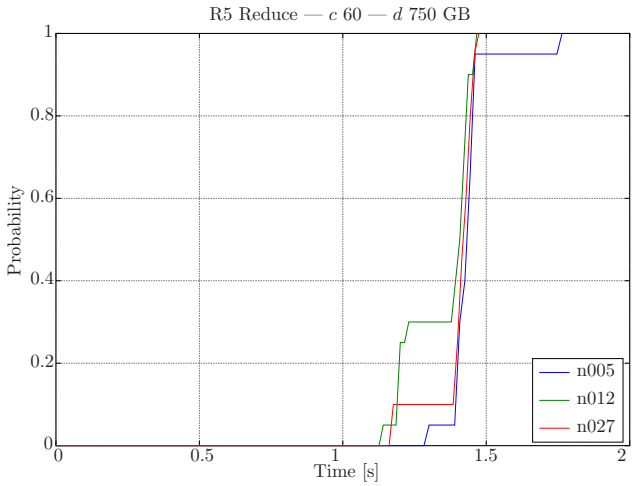


Figure 10: R5 reduce, 60 containers, 750 GB dataset

On the other hand, mappers suffer a stronger impact from external interference. In the reported graphics we have examples of both good and problematic performance. Figure 7 shows how all but one node have a very similar behavior. Further, the only different node is not significantly distant from the others, hence probably the low variability is imputable to the physiological effects of data locality. Instead, Figure 9 reports that the three involved nodes are quite variable in performance. In this case, since we are dealing with a small interactive query, any exogenous interference may cause a strong impact on the overall response time, thus making prediction harder.

Table 2 reports the results of the accuracy assessment for the proposed method, considering MapReduce jobs only. In particular, we consider the fluid evolution function (7), discussed in Section 3.4. For several experiments we report the involved query, the total number of containers available in the cluster (c), the scale factor for the dataset generator (d), the average measured (t) and predicted (τ) execution time, and the relative error (ε).

The average accuracy achieved with the approximate formula is 9.23%, perfectly in line with the expectations in the performance prediction field [20], where a 30% accuracy on response times is acceptable. Only a handful of experiments show a relatively high error, peaking at 28.91% in absolute value. Nonetheless, the standard error of the mean is 1.33%. The largest relative errors tend to appear in the experiments involving R5, a small query both in terms of number of tasks and of overall duration. This behavior suggests that the relative abundance of resources might amplify the effects of variability, making prediction a harder task to accomplish.

4.3 Analysis of Spark Jobs

We have fit the model presented in Section 3.4 against the measures, for each stage and number of containers for both Q26 and Q52. In particular, we have minimized the sum of the squared difference of the completion times of each task measured in the trace and predicted by the model, varying parameters T and α. The minimization has been performed in Apache Octave using the SQP procedure of the `optim` package. As representative cases, Figure 11 compares the measured average completion time and the fluid model approximation for $C = 12$ and $C = 24$ containers for stage J3S3 of query Q26; Figure 12 does the same for stage J2S2 of query Q52. In all cases the completion time shows a ladder-shaped behavior for the first tasks, which gradually reduces as the tasks are executed: this feature is captured by the deterministic component of the fluid model. Table 3 shows the minimum, average and maximum relative errors of the fitted model for what concerns the completion time of the job, among the different configurations in terms of containers. Note that for jobs with just two tasks the model error is always zero, since with two parameters (T and α defined as per section Section 3.4) it is always able to perfectly capture the evolution of the average completion time of the tasks.

We have then characterized the traces to provide a compact fluid model that completely describes the queries. In particular, we have drawn a GANTT diagram to study how tasks are run in parallel on the available cores. Figure 13 shows how tasks are executed at the beginning of Q26. The three initial stages, J0S0, J1S1 and J2S2, are independent and are all composed by two tasks. Since all configurations have more than three containers, the three stages are run in

Table 2: Accuracy with the approximate formula

Query	c	d [GB]	t [ms]	τ [ms]	ε [%]
R1	60	250	80316	82314	2.49
R2	60	250	84551	78100.96	−7.63
R3	60	250	275684	282737.30	2.56
R4	60	250	219243	221205.13	0.89
R5	60	250	25924	19134.99	−26.19
R1	60	750	389562	387158.42	−0.62
R2	60	750	80090	74975.87	−6.39
R3	60	750	1027329	1052517.40	2.45
R4	60	750	808490	834140	3.17
R5	60	750	24392	17339.16	−28.91
R1	60	1000	556680	564379.06	1.38
R2	60	1000	2009929	2546188	26.68
R3	60	1000	1374024	1419558.01	3.31
R4	60	1000	1374244	1562961.30	13.73
R5	60	1000	48839	60180	23.22
R1	80	500	143139	138049.55	−3.56
R2	80	500	73243	69092.11	−5.67
R3	80	500	526760	533398.48	1.26
R4	80	500	410376	423152	3.11
R5	80	500	23558	17090.94	−27.45
R1	80	750	268821	266099.64	−1.01
R2	80	750	78080	73276.07	−6.15
R3	80	750	791314	807333.48	2.02
R4	80	750	618045	634756.77	2.70
R5	80	750	23894	17100.00	−28.43
R1	80	1000	439052	442487.10	0.78
R2	80	1000	1110685	1129076.31	1.66
R3	80	1000	1019973	1043506.48	2.31
R4	80	1000	960985	1061851.90	10.50
R5	80	1000	39206	28972.61	−26.10
R1	100	250	49726	59324	19.30
R2	100	250	82861	77078.03	−6.98
R3	100	250	168209	172945.69	2.82
R4	100	250	138238	141341.11	2.24
R5	100	250	25316	18281.41	−27.79
R1	100	500	132383	127974.74	−3.33
R2	100	500	73870	69572.08	−5.82
R3	100	500	401827	416948.55	3.76
R5	100	500	24619	18226.33	−25.97
R1	100	750	203531	203550.04	0.01
R2	100	750	78291	73068.51	−6.67
R3	100	750	635991	651107.92	2.38
R4	100	750	514310	526617.13	2.39
R5	100	750	24887	17988.72	−27.72
R1	120	250	46215	50790	9.90
R2	120	250	83136	76901.37	−7.50
R3	120	250	143650	144140.92	0.34
R4	120	250	97829	90498.44	−7.49
R5	120	250	26072	18720.46	−28.20
R1	120	500	91809	83580.16	−8.96
R2	120	500	72543	68232.80	−5.94
R3	120	500	303843	298283.35	−1.83
R4	120	500	275407	279794.24	1.59
R5	120	500	25265	18030.72	−28.63
R1	120	750	199234	200260.15	0.52
R2	120	750	79042	74269.17	−6.04
R3	120	750	661214	660269.38	−0.14
R4	120	750	507861	513599.16	1.13
R5	120	750	24882	18143.36	−27.08

Table 3: Fitting errors for the stages of the two considered queries Q26 and Q52

Stage	min ε [%]	$\bar{\varepsilon}$ [%]	max ε [%]
Q26 - J3S3	1.01	1.79	3.47
Q26 - J3S4	11.57	15.45	19.23
Q26 - J3S5	1.48	8.28	17.27
Q26 - J3S6	2.19	6.89	12.39
Q52 - J2S2	0.15	1.04	2.53
Q52 - J2S4	9.71	15.32	19.36

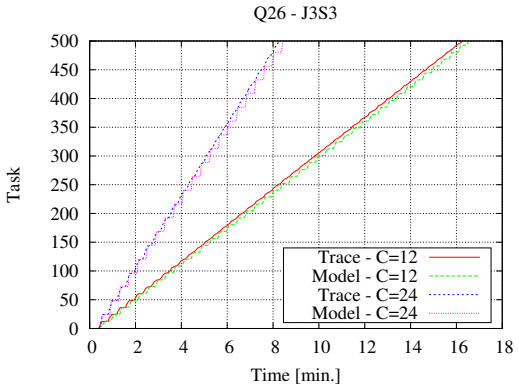


Figure 11: Q26 - J3S3 - Comparison of fitting and measured data

parallel at the beginning of the query. The next stage, J3S3 starts when the three preceding one are completed. J0S0, J1S1 and J2S2 are all of comparable duration, but from the measurements, stage J2S2 always ends a few ms later than the other. Hence, we can include stages J0S2 and J1S1 into J2S2, and put only the latter in the model of the query. The same feature is also observable in Figure 14 for what concerns tasks J1S0 and J0S1 of query Q52: following the same reasoning, we include only stage J0S1 in the model of Q52. For what concerns Q26, as shown from the DAG depicted in Figure 5, jobs J3S3 and J3S4 are can be run in parallel. Figure 15 is a GANTT of Q26 focused on the time in which these two jobs are executed: in particular, it shows that Spark starts executing the tasks belonging to J3S3, and when they are completed, the systems starts allocating container to J3S4. However, the latter tasks are much shorter than the former, and they can be completed before the end of J3S3 with the containers that have been released. For this reason the execution of job J3S4 is masked by the running of J3S3, allowing us to removing it from the model. To summarize, query Q26 can be modeled by the sequence of jobs J2S2, J3S3, J3S5 and J3S6; query Q52 by the sequence of jobs J0S1, J2S2 and J3S3.

From the GANTT analysis of the completion times of the tasks, it is also visible that there is a non-negligible delay between the completion of one job and the beginning of following one: see for example Figure 13 for the delay between the end J2S2 and J3S3, and Figure 14 for the one between J0S1 and J2S2. Moreover it can also be seen that this delay in some cases depends on the number of containers used: this should be a consequence of the communication overhead required to perform synchronization among the

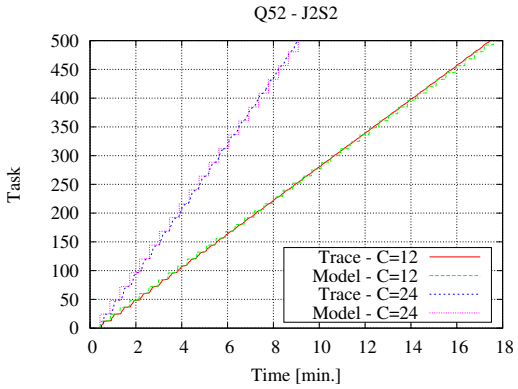


Figure 12: Q52 - J2S2 - Comparison of fitting and measured data

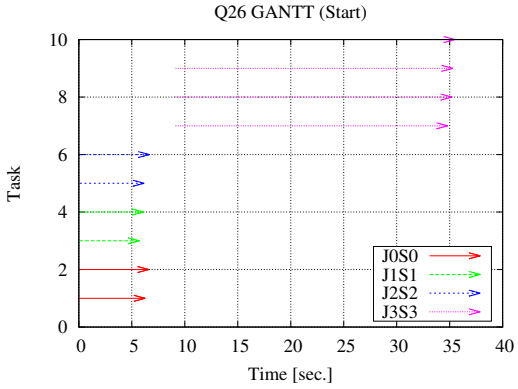


Figure 13: Q26 - GANTT (Start)

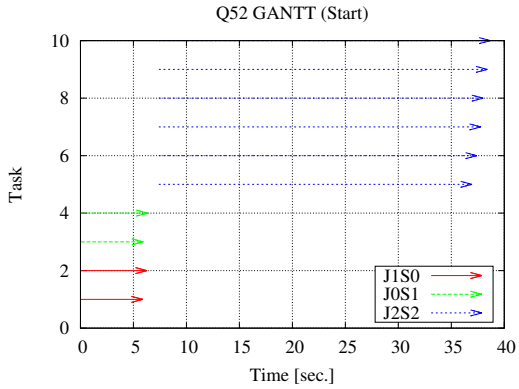


Figure 14: Q52 - GANTT (Start)

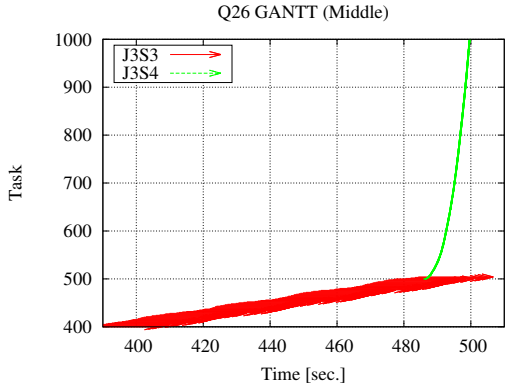


Figure 15: Q26 - GANTT (Middle)

nodes.

For example Figure 16 shows the delay between stages J2S2 and J3S3 of Q26, and between stages J0S1 and J2S2 of Q52. While for query Q52 the delay is basically constant, there is a clear linear increase in length as function of the number of containers for Q26. To obtain more faithful representation of the query, these delays must be included in the corresponding models as they can lead to a performance degradation as the number of container grows as in other parallel frameworks (see [**?**] and [**?**]).

Extending the techniques given in Section 3.1, we have modeled query Q26 with the FSPN shown in Figure 18, and query Q52 with FSPN given in Figure 19. Only the stages whose execution was not masked have been modeled. Delays between the stages have been introduced by replacing the immediate transitions that determines the ends of the stages ($\texttt{End}_{J2S2}$,$\texttt{End}_{J3S3}$,$\texttt{End}_{J3S5}$ for Q26, and $\texttt{End}_{J0S1}$ and $\texttt{End}_{J2S2}$ for Q52) with timed transitions. The delays have been assumed to be of exponential duration, with the average equal to the one measured from the traces.

In order to validate our model, we have initially put $N = 1$ (only one user in the system) and the firing time of transition `Think` to zero. We have analyzed the FSPN with a custom built trace generator written in Lua[7]. We have then compared the completion time distribution obtained from the model with the empirical distribution measured from the traces. Figure 20 and Figure 21 shows respectively the results for queries Q26 and Q52 run on a system with $C = 24$ containers. As it can be seen, results are quite satisfactory for Q52, while the model overestimates the completion time for Q26: this can be partially due to the increased complexity of the considered trace, and to the availability of a more reduced set of measures to train the fluid model.

An important parameter of the system is the number of containers C onto which the queries are executed. This parameter however is not explicitly represented in the FSPN, but it is used in the fluid flow functions that characterize the fluid transitions. We have included it in the model by performing regressions (either linear, exponential or constant) of the parameters of the fluid flow functions, and used it to compute parameters T and α of the fluid models, and the average firing time of transitions that represent the delay between the stage, as reported in Table 5.

To validate the approach, we have performed the regression using only measures until $C = 20$ containers, extrapolated the values for $C = 24$, and solved the model with the computed parameters. Results have been compared with both the data obtained from the real trace, and the model obtained using the parameters fit directly from the corresponding measures. Figure 22 shows the extrapolation process for the parameters of job J3S3 of query Q26, while Figure 23 does it for job J2S2 of Q52. The extrapolation of the delay duration is instead

[7] `http://www.lua.org`

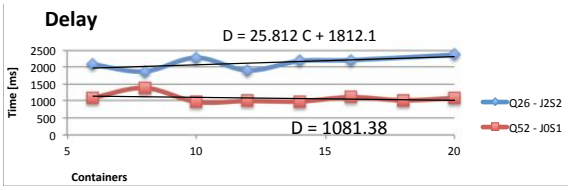


Figure 16: Delay extrapolation

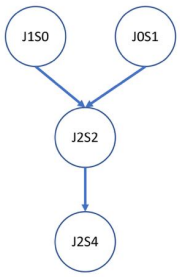


Figure 17: Q52 DAG

shown in Figure 16. Results of the extrapolated models are instead shown in Figure 20 and Figure 21. For query Q52, the extrapolation introduces an additional error, while for Q26 it provides results that are even more accurate than the one obtained with the best fit procedure. This can be explained by the fact that fitting aims at modeling the entire evolution of the tasks, while the computation of the delay uses only the completion time of the job (which is determined only by the last task). For Q26, even if the extrapolated values provided a poorer description for the completion time of the intermediate tasks, they provide a better fit for the ending time of the job. In both cases the error is around 10%, that could be considered acceptable for many practical applications.

5. USE OF MODELS TO SUPPORT CAPACITY PLANNING DECISIONS

We now exploit the model of the Q26 query to analyze the considered system. In particular, we first study the completion time of the query increasing the number of containers from $C = 25$ to $C = 500$. In the case with largest amount of resources, all the tasks of the longest phase can be executed in parallel in a single run. From the plot it appears clear that there is a clear advantage in increasing the number of containers up to $C = 100$. After that point, the advantages of increasing the number of containers become less obvious, and a larger value of C might result in a degradation of the performance. This can be due to several issues: first, as the number of container increases, the communication overhead becomes more significative, compromising the advantage that a larger number of the resources can give. Moreover, since some of the jobs are divided into $n = 200$ tasks, from $C = 100$ on, some of the nodes will be used for just one task, making them idle for most the time, and making their use less-effective in participating at the completion of the job. This effect is even more evident for $C = 250$, since the maximum number of tasks of the longest stage of the system is $n = 500$.

Table 4: Spark parameters

Query	d [GB]	s	n	$\bar{t}$ [ms]
Q26	500	J0S0	2	[6456, 7512]
Q26	500	J1S1	2	[6023, 6929]
Q26	500	J2S2	2	[5863, 7174]
Q26	500	J3S3	500	[497604, 1932021]
Q26	500	J3S4	500	[14202, 27974]
Q26	500	J3S5	200	[339369, 643977]
Q26	500	J3S6	200	[2752, 4179]
Q52	500	J1S0	2	[6135, 6982]
Q52	500	J0S1	2	[6055, 7070]
Q52	500	J2S2	500	[544216, 2077535]
Q52	500	J2S4	200	[9346, 15205]

We then use the model to answer the following question: considering a given number of users N in the range $[5, 25]$, which could be optimal number of containers to give the system an average response time of less than 10 minutes? We compute this result by performing a parameter space exploration of the model, by varying both the number of containers and the number of users. In particular, observing that only on query at a time can be executed, we first compute the completion time distribution as done in Section 4.3, for different values of C. Then, we reduce the system to a simple QN model composed by a delay station to represent the think time of the users, and a FCFS queue to model the execution of the query as shown in Figure 25. We have analyzed the model using the JMT - Java Modeling Tools[8] package by performing a what-if analysis over the number of users N, and by assigning a *replay* distribution to the queue representing the query, connected to the results pre-computed with the fluid model.

Results of the analysis are shown in Figure 26. For a number of containers $C < 10$, the target deadline cannot be achieved for any number of users. The same is also valid if the number of users $N > 20$: in these cases none of the possible solutions would be able to solve the problem and provide a response in less than 10 minutes on average. Should this be the case, the solution must be sought by either increasing the parallelization in a larger number of tasks (if possible), changing the algorithm, or look for faster computing nodes. For all the other case, the optimum number of containers to achieve the performance goal is in the range $[50, 250]$, noting that increasing the parallelization over $C = 250$ might even reduce the performance and violate the considered constraints.

6. CONCLUSIONS

In this paper we proposed FPNs able to model the execution of MapReduce jobs running in Hadoop 2.x clusters governed by the Capacity Scheduler, and to model Apache Spark jobs. The experimental results for MapReduce have shown that the average percentage error that can be achieved is around around 9.5% for Map Reduce of the actual measurements on average, making the approach suitable for design time capacity planning or runtime cluster management. When applied to study two queries performed using Apache Spark, we have obtained similar results (about 10%

[8] http://jmt.sourceforge.net

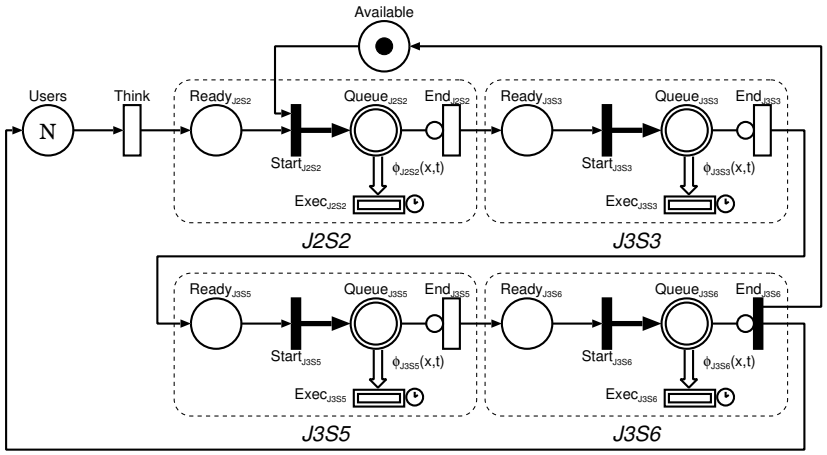


Figure 18: FSPN model for Q26

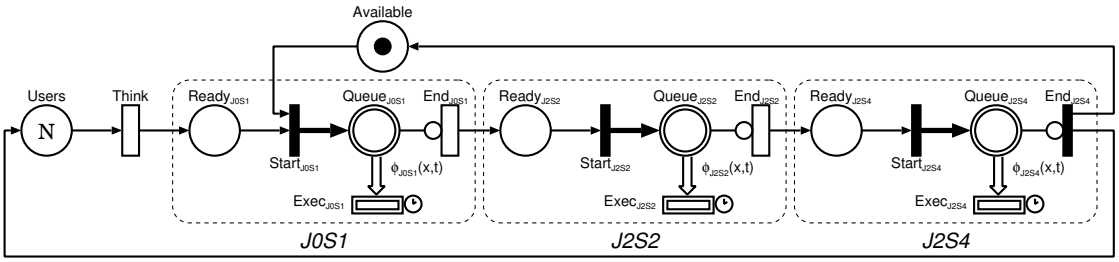


Figure 19: FSPN model for Q52

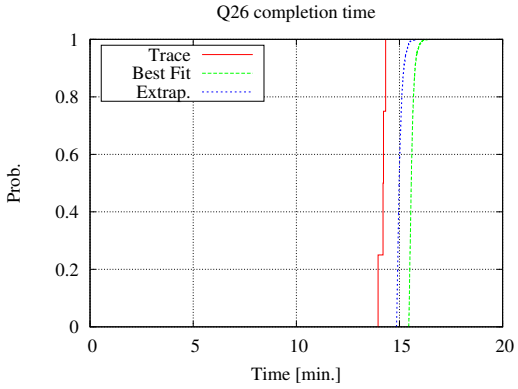


Figure 20: Q26 - model validation of fitting and extrapolation with 24 containers

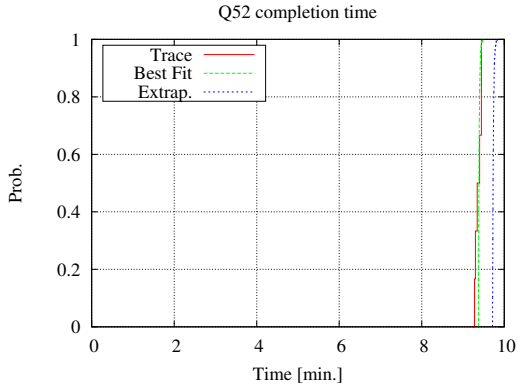


Figure 21: Q52 - model validation of fitting and extrapolation with 24 containers

average percentage error): this has shown that the proposed approach can be generalized to several advanced settings.

On-going work is considering the analysis of Spark jobs characterized by general DAGs and considering machine learning applications workloads. Moreover, the framework will be integrated within an optimization tool for runtime cluster management.

Acknowledgments

The results of this paper have been partially funded by EUBra-BIGSEA (grant agreement no. 690116), a Research and Innovation Action funded by the European Commission under the Cooperation Programme, Horizon 2020 and the Ministério de Ciência, Tecnologia e Inovação, RNP/Brazil (grant GA0000000650/04).

Eugenio Gianniti is also partially supported by the DICE H2020 research project (grant agreement no. 644869).

Spark experiments have been supported by Microsoft under the *Top Compsci University Azure Adoption* program.

Table 5: Parameter extrapolation

Stage	T	α	Delay [ms]
Q26 - J2S2	$6863.16 \cdot e^{-0.008775C}$	$0.0931 \cdot e^{-0.0224029C}$	$25.812C + 1812.11$
Q26 - J3S3	$52.99C + 22725.7$	$0.6312 \cdot e^{-0.06005C}$	$528.02C - 5009.21$
Q26 - J3S5	$39208,46$	0.1839	65.76
Q26 - J3S6	$17.4226C$	0	
Q52 - J0S1	$6749.1154 \cdot e^{-0.006279C}$	$0.1065 \cdot e^{-0,0432C}$	1081.38
Q52 - J2S2	$91.63C + 24437.97$	$0.1448 \cdot e^{-0,02399C}$	108.46
Q52 - J2S4	$40.234C + 359.59$	0	

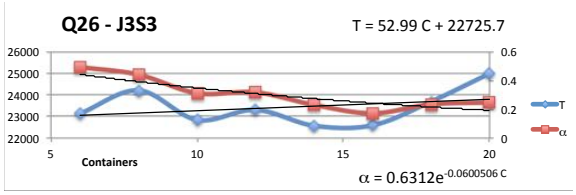


Figure 22: Q26 - J3S3 - Model parameters extrapolation

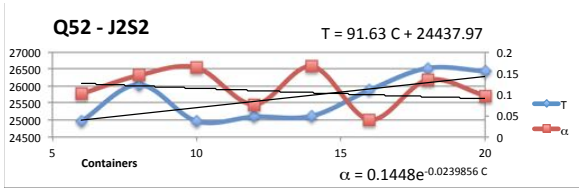


Figure 23: Q52 - J2S2 - Model parameters extrapolation

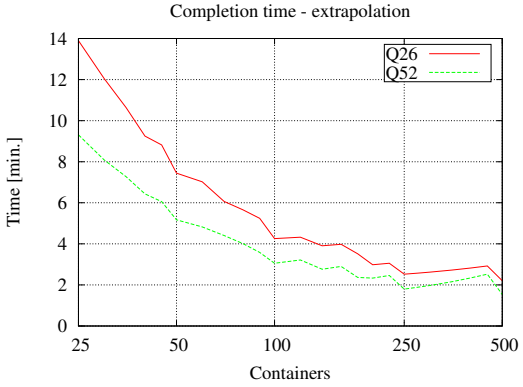


Figure 24: Estimated response time of Q26 and Q52

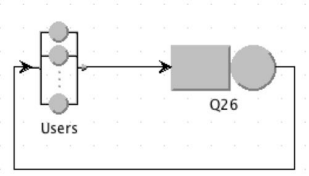


Figure 25: The model of Q26 in JMT

REFERENCES

[1] L. Aguilera-Mendoza and M. T. Llorente-Quesada. Modeling and simulation of Hadoop Distributed File System in a cluster of workstations. In A. Cuzzocrea and S. Maabout, editors, *Model and Data Engineering*, volume 8216 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2013.

[2] D. Ardagna, S. Bernardi, E. Gianniti, S. K. Aliabadi, D. Perez-Palacin, and J. I. Requeno. Modeling performance of hadoop applications: A journey from queueing networks to stochastic well formed nets. In *Algorithms and Architectures for Parallel Processing - 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings*, pages 599–613, 2016.

[3] D. Ardagna, C. Ghezzi, and R. Mirandola. Rethinking the use of models in software architecture. In *QoSA 2008*, pages 1–27. Springer Berlin Heidelberg, Oct. 2008.

[4] E. Ataie, E. Gianniti, D. Ardagna, and A. Movaghar. A combined analytical modeling machine learning approach for performance prediction of mapreduce jobs in hadoop clusters. In *MICAS-SYNASC 2016 Workshops Proceedings. To appear.*

[5] S. Balsamo, P. G. Harrison, and A. Marin. Methodological construction of product-form stochastic petri nets for performance evaluation. *Journal of Systems and Software*, 85(7):1520–1539, 2012.

[6] E. Barbierato, M. Gribaudo, and M. Iacono. Modeling Apache Hive based applications in Big Data architectures. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '13, pages 30–38. ICST, 2013.

[7] E. Barbierato, M. Gribaudo, and M. Iacono. Modeling hybrid systems in SIMTHESys. In *Proceedings of the 8th International Workshop on Practical Applications of Stochastic Modelling*. to appear, 2016.

[8] E. Barbierato, M. Gribaudo, and D. Manini. *Fluid Approximation of Pool Depletion Systems*, pages 60–75. Springer International Publishing, Cham, 2016.

[9] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri. Exploiting mean field analysis to model performances of Big Data architectures. *Future Generation Computer Systems*, 37:203–211, 2014.

[10] W. W. Chu, C.-M. Sit, and K. K. Leung. Task response time for real-time distributed systems with resource contentions. *IEEE Trans. Softw. Eng.*, 17(10):1076–1092, Oct. 1991.

[11] M. Ciavotta and D. Ardagna.

[12] J. Gantz. The digital universe in 2020.

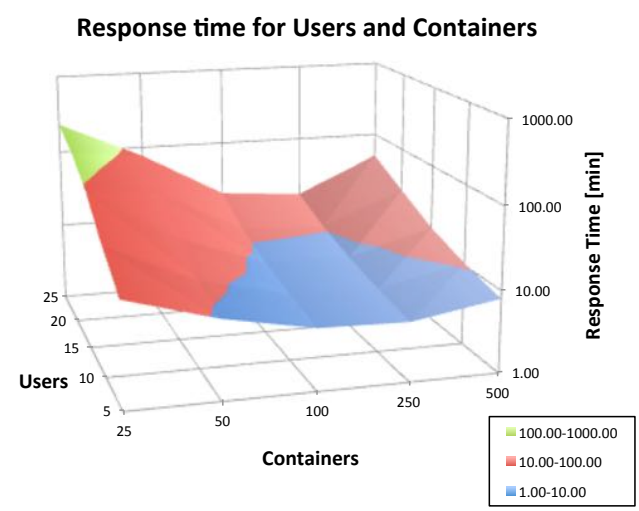


Figure 26: Determining the best number of containers for a given number of users to obtain a target average response time for Q26

http://idcdocserv.com/1414.

[13] E. Gianniti, A. M. Rizzi, E. Barbierato, M. Gribaudo, and D. Ardagna. Fluid petri nets for the performance evaluation of mapreduce applications. In *INFQ 2016 Workshop Proceedings*.

[14] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna. Stage Aware Performance Modeling of DAG Based in Memory Analytic Platforms. In *In IEEE Cloud 2016 Proceedings*.

[15] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna. Stage aware performance modeling of DAG based in memory analytic platforms. In *Cloud*, 2016.

[16] M. Gribaudo and M. Telek. *Fluid Models in Performance Analysis*, pages 271–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[17] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big Data and its technical challenges. *Commun. ACM*, 57(7):86–94, July 2014.

[18] K. Kambatla, G. Kollias, V. Kumar, and A. Grama. Trends in Big Data analytics. *Journal of Parallel and Distributed Computing*, 74(7):2561–2573, 2014.

[19] D. Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, 2012.

[20] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[21] D.-R. Liang and S. K. Tripathi. On performance prediction of parallel computations with precedent constraints. *IEEE Trans. Parallel Distrib. Syst.*, 11(5):491–508, May 2000.

[22] M. Lin, L. Zhang, A. Wierman, and J. Tan. Joint optimization of overlapping phases in MapReduce. *SIGMETRICS Performance Evaluation Review*, 41(3):16–18, 2013.

[23] X. Lin, Z. Meng, C. Xu, and M. Wang. A practical performance model for Hadoop MapReduce. In *International Conference on Cluster Computing Workshops*. IEEE, 2012.

[24] V. W. Mak and S. F. Lundstrom. Predicting performance of parallel computations. *IEEE Trans. Parallel Distrib. Syst.*, 1(3):257–270, July 1990.

[25] R. D. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Trans. Computers*, 37(6):739–743, 1988.

[26] R. Osman and P. G. Harrison. Approximating closed fork-join queueing networks using product-form stochastic petri-nets. *J. Syst. Softw.*, 110(C):264–278, Dec. 2015.

[27] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[28] J. Polo, Y. Becerra, D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé. Deadline-based MapReduce workload management. *IEEE Trans. Network and Service Management*, 10(2):231–244, 2013.

[29] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino. Apache tez: A unifying framework for modeling and building data processing applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1357–1369, New York, NY, USA, 2015. ACM.

[30] C. Shanklin. Benchmarking Apache Hive 13 for enterprise Hadoop. https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.

[31] A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic resource inference and allocation for MapReduce environments. In *ICAC '11*, June 2011.

[32] E. Vianna, G. Comarela, T. Pontes, J. M. Almeida, V. A. F. Almeida, K. Wilkinson, H. A. Kuno, and U. Dayal. Analytical performance models for MapReduce workloads. *International Journal of Parallel Programming*, 41(4):495–525, 2013.

[33] K. Wang and M. Khan. Performance prediction for apache spark platform. pages 166–173, 2015. cited By 2.

[34] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, 2012. USENIX.

[35] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

An Approach for Resiliency Quantification of Large Scale Systems

Francesco Longo
Università degli Studi di Messina, Italy
flongo@unime.it

Rahul Ghosh
Xerox Research Center, India
rahul.ghosh@xerox.com

Vijay K. Naik
IBM T. J. Watson Research Center, USA
vkn@us.ibm.com

Andrew J. Rindos
IBM, USA
rindos@us.ibm.com

Kishor S. Trivedi
Duke University, USA
ktrivedi@duke.edu

ABSTRACT

We quantify the resiliency of large scale systems upon changes encountered beyond the normal system behavior. Formal definitions for resiliency and change are provided together with general steps for resiliency quantification and a set of resiliency metrics that can be used to quantify the effects of changes. A formalization of the approach is also shown in the form of a set of four algorithms that can be applied when large scale systems are modeled through stochastic analytic state space models (monolithic models or interacting sub-models). In particular, in the case of interacting sub-models, since resiliency quantification involves understanding the transient behavior of the system, fixed-point variables evolve with time leading to non-homogenous Markov chains. At the best of our knowledge, this is the first paper facing this problem in a general way. The proposed approach is applied to an Infrastructure-as-a-Service (IaaS) Cloud use case. Specifically, we assess the impact of changes in demand and available capacity on the Cloud resiliency and we show that the approach proposed in this paper can scale for a real sized Cloud without significantly compromising the accuracy.

Keywords

Infrastructure-as-a-Service Cloud, interacting sub-models, large scale systems, non-homogeneous Markov chains, resiliency quantification

1. INTRODUCTION

Large enterprise systems are difficult to manage due to their scale and complexity. Although such systems are carefully designed, they may still fail to deliver expected services due to failures or other unexpected variations in the working conditions. For business critical systems, such deviations from normal behavior may cause violations of service level agreements (SLAs). Hence, during the design stage of such systems, a notion of built-in *resiliency* is necessary to encounter the uncertainties that can occur beyond the normal behavior.

However, quantifying the system resiliency is difficult because of two key reasons: (1) lack of consensus on the definition of resiliency and (2) lack of systematic approaches for quantifying resiliency. The term resiliency is used in different fields and several definitions are present. Many researchers interpret resiliency as a synonym for fault-tolerance. However, the effect of fault tolerance can be captured by traditional dependability measures, such as reliability, availability, maintainability, and safety [7, 2]. DeBardeleben *et al.* [6] suggested that the goal of high end computing is to enable effective and resource-efficient use of computing systems at extreme scale in the presence of system degradation and failures. They defined resiliency of a high end computing system as the ability to keep applications running and maintain an acceptable level of service when facing transient, intermittent, and permanent faults. However, the quantification of the effects of failures/repairs on a system is covered under the umbrella of dependability, joint analysis of contention/queuing and failures/repairs is known as performability [14], and the effect of failures/repairs on individual tasks is covered under the umbrella of task completion time analysis [5, 18]. Impact of (large scale) failures (e.g., natural disasters), intrusions, or attacks on systems/services and subsequent transient behavior is quantified by survivability analysis [15, 21]. Sterbenz *et al.* [26] defined resiliency as the combination of trustworthiness (dependability, security, and performability) and tolerance (survivability, disruption tolerance, and traffic tolerance).

Our work about resiliency starts from the definition of Sterbenz *et al.* and is inspired from Laprie's [19] and Simoncini's [25] work, in which resiliency is defined as the persistence of service delivery that can justifiably be trusted when facing *changes* - usually not encountered under normal operating conditions. We use this notion of change in our resiliency quantification approach. Note that dynamic redundancy features in traditional fault tolerance with detection followed by reconfiguration are already included in classic dependability measures. The changes we are referring to here are beyond the envelope of system configurations already considered during the design stage and thus beyond fault tolerance. We also emphasize that survivability quantification only deals with changes due to (large scale) failures, intrusions, or attacks. The notion of resiliency is broader than survivability as the changes encountered by the system can be any unexpected variations in the working conditions, e.g., workload, faultload, or system capacity.

This paper presents a general method for resiliency quantification of large scale systems through analytic models. A formal definition of resiliency is provided together with a definition of change. Moreover, a set of metrics is defined to quantify the resiliency of a system with respect to a variety of measures of interest. In fact, contrary to the traditional view of synonymizing resiliency with dependability, we believe that resiliency can be quantified even with respect to performance measures.

The approach is formally presented through a set of algorithms. Such algorithms represent an extension of the one presented in a previous work of ours [10]. Here, first of all, a general algorithm is presented illustrating the approach in the case in which the large scale system is modeled through a monolithic state-space model. However, such a kind of models are usually affected by the state space explosion problem limiting the scale of the system that is actually possible to deal with. As a solution to this problem, the use of a set of interacting sub-models to manage the complexity is getting more attention. For this reason, we provide a second algorithm showing how the approach is applicable also in the case of interacting sub-models.

In the interacting sub-models approach, interdependencies are resolved by fixed-point iterations. Resiliency quantification involves understanding the transient behavior of the system. Transient analysis of interacting sub-models coupled with fixed-point iterations becomes non-trivial as the values of fixed-point parameters change with time – leading to non-homogeneous sub-models. Thus, we also present a third algorithm for resiliency quantification when dealing with such non-homogeneous, interacting sub-models, based on a piece-wise constant approximation. To the best of our knowledge, ours is the first attempt to perform transient analysis of state-space based fixed-point iterative sub-models. The approach is illustrated by an Infrastructure-as-a-Service (IaaS) Cloud use case for which we provide several numerical results.

The paper is organized as follows. Section 2 presents an overview of the state of the art about resiliency quantification and discusses the main differences with respect to our approach while Section 3 provides an overview of the proposed resiliency quantification approach also presenting a set of resiliency metrics. Section 4 introduces our IaaS Cloud use case and shows how to model it through a stochastic reward net monolithic model and a set of continuous time Markov chain interacting sub-models. In Section 5, a set of algorithms is presented formalizing our approach and showing how the non-homogeneity issue in the case of interacting sub-models can be solved through a piece-wise constant approximation. Finally, Section 7 concludes the paper and points out some future research directions.

2. RELATED WORK

We observe that resiliency is mostly interpreted as one of the dependability and security attributes (e.g., reliability, availability). Table 1 shows a classification of the related research based on the authors' interpretation of resiliency.

In [24], Puggelli *et al.* defined resiliency of a sensor network by taking into account node failures and battery lifetime. The authors tried to maximize the resiliency via redundancy. Najjar *et al.* [22] defined network resiliency as the probability of no disconnection in a family of regular graph network topologies. In the context of high performance computing (HPC), Ostrouchov *et al.* [23] interpreted resiliency as fault tolerance, i.e., the identification of failures and diagnosis of their root causes. Clearly, in [24, 22, 23], notion of resiliency is synonymous to reliability of the system.

Table 1: Classification of related research based on the authors' interpretation of resiliency.

Resiliency as	**References**
Reliability	[24, 22, 23]
Availability	[3, 8, 20, 17]
Survivability	[1, 4]
Security	[27, 9, 30]

Similar to [24], in [3], Cappello *et al.* analyzed resiliency of HPC systems from a fault tolerance perspective. The authors observed that automatic or application level checkpoint-restart does not work in such a context as the time for checkpointing and restarting usually exceed the mean time to failure of the system as a whole. They proposed different recovery mechanisms to address this problem. In a similar work, Engelmann *et al.* [8] introduced the notion of resiliency supporting models (e.g., failure prediction, checkpointing, rejuvenation scheduling) to analyze and improve the resiliency of HPC systems. In [20], Liang *et al.* quantified the resiliency of Platform-as-a-Service (PaaS) Clouds. The authors defined resiliency of PaaS applications as the ability to maintain a low failure rate by dynamically changing the affected components. In [17], Jhawar *et al.* survey approach for the characterization of recurrent failures in a typical Cloud computing environment, analyzing the effects of failures and evaluating fault tolerance solutions. Observe that the approach for resiliency quantification described in [3, 8, 20] falls under traditional availability analysis.

In [1], Agarwal *et al.* evaluated resiliency of telecommunication networks upon probabilistic large-scale failures such as physical attacks or natural disasters over a geographical area. Such failures could affect both the physical and the logical layer of the networks. The authors used computational geometric techniques in order to identify the most vulnerable locations in the network and computed the expected number of failed components or the expected total data loss. In [4], Cappello *et al.* survey recent work on resiliency in the HPC community. They mostly focus on errors in computation due to unstable systems. Such errors usually propagate and generate various kinds of malfunctions (e.g., process crashes, result corruptions). Analysis of (large scale) failure or other undesired events, e.g., intrusion or disaster and their impact on the system is covered by survivability analysis [15].

In [27], Bu *et al.* defined resiliency of attack-resistant networks as the ability to provide alternative communication paths in the case of failures or attacks on existing paths. Using approximation algorithms, the authors computed the probability that a path is blocked and used this metric to evaluate different network designs. In [9], Erdene-Ochir *et al.* studied the resiliency of a set of routing protocols for wireless sensor networks. They defined the resiliency as the ability of a network to continue to operate in presence of a certain number of malicious compromised nodes that could disrupt the routing functionality. Using simulation, the authors evaluated the resiliency of four routing protocols in

the presence of compromised nodes. In the context of peer to peer (P2P) networks, Xuan *et al.* [30] defined resiliency as the ability of a system to defend against threats due to node churns and malicious nodes. Using Markov chains, they proposed an analytic approach to quantify resiliency of structured P2P systems which have relatively stable size and uniformly distributed nodes. Thus, the authors in [27, 9, 30], mainly quantify the resiliency of a system from security perspectives.

In contrast to all the past efforts, in this paper, we take a broader approach for system resiliency quantification. Our proposed approach can be applied to any performance, dependability, or security measure. In the following sections, we describe the approach and show its application for the resiliency quantification of IaaS Cloud performance taken as a use case.

3. RESILIENCY QUANTIFICATION

In this section, we provide our definition of resiliency and change. Starting from such definitions, we show a high level overview of the proposed approach. Finally, we introduce a set of metrics that can be used to quantify the resiliency of a system with respect to a set of measures of interest.

3.1 Definitions of Resiliency and Change

As discussed in Section 2, contrary to the several definitions of resiliency that are found in the literature, we believe that the resiliency of a system can be quantified with respect to several measures of interest. For such a reason, our definition of resiliency is the following.

DEFINITION 3.1 (RESILIENCY). *The resiliency of a system Σ with respect to a specific measure of interest M is the capacity of the system to maintain the considered metric within reasonable intervals when facing one or more changes.*

The notion of change is thus important to our definition of resiliency.

DEFINITION 3.2 (CHANGE). *A change is any variation beyond the normal operating conditions of system Σ caused by any internal or external factor.*

Within our definition of change we thus take into consideration both internal causes of change (strictly related to the way the system works) and external causes of change (more related to how the system environment affects its behavior).

3.2 Overview of our approach

The approach proposed in this paper allows the quantification of the resiliency of large scale systems via the use of analytic models. In the following, we outline the general steps of the approach.

(i) Given a system Σ, construct a stochastic analytic state-space model Ω of Σ to find the measure of interest M under normal behavior. With respect to such measure, Ω can be a performance, availability, or a performability model of Σ.

(ii) Evaluate the system measure of interest M under normal conditions using the model developed in step (i). This step consists in computing the steady state value $M_{ss}^{(bc)}$ of measure of interest M without any application of changes.

(iii) Apply changes to the system. Changes can be classified into two broad categories: (a) non-structural changes and (b) structural changes. Non-structural changes do not affect the state space of the underlying model. Examples of such changes are variation of arrival rates or hardware/software failure rates. Structural changes cause the number of states in the state space of the underlying model to vary and/or affect its reachability graph by adding/removing transitions between states. Examples of structural changes can be addition or removal of system components.

(iv) Analyze the transient behavior of the system model Ω to compute the progress of measure of interest M after applying the changes. Initial probabilities for this transient analysis are obtained from the steady state probabilities as computed from the normal behavior of the system model Ω in step (ii). The transient analysis can be stopped as soon as the measure of interest M reaches a new steady state value $M_{ss}^{(ac)}$.

(v) After the changes are applied and the transient analysis of step (iv) has been performed, the transient response of measure of interest M can be compared with the value of the same measure as computed in step (ii) (M_{ss}^{bc}) and the resiliency of system Σ can be quantified by computing a set of resiliency metrics as described in the following.

3.3 Resiliency metrics

We define resiliency metrics to quantify the resiliency of a system when changes are enforced. Let $M_{ss}^{(bc)}$ and $M_{ss}^{(ac)}$

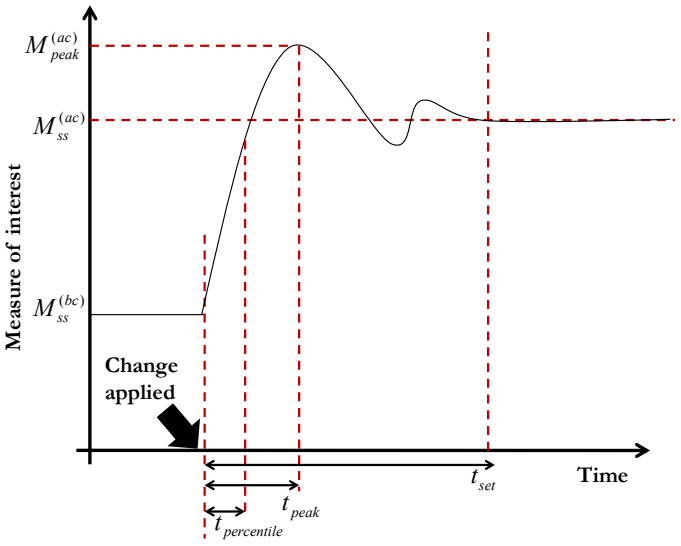


Figure 1: Conceptual view of resiliency metrics.

denote the steady state values of measure of interest M (for which resiliency is computed) before and after the application of change, respectively. The following four metrics can be defined.

(1) Settling Time (t_{set})**.** This is defined as the elapsed duration from the time when the change is applied to the system, until the measure of interest reaches and stays within $M_{ss}^{(ac)} \pm \delta\%$ of $|M_{ss}^{(bc)} - M_{ss}^{(ac)}|$.

(2) Peak Overshoot (Undershoot) (PO) Let $M_{peak}^{(ac)}$ be the maximum deviation of the measure from its steady state value after change is applied. Peak overshoot (undershoot)

in percentage is defined as:

$$PO(\%) = \frac{|M_{peak}^{(ac)} - M_{ss}^{(ac)}|}{M_{ss}^{(ac)}} \times 100 \quad (1)$$

(3) Peak Time (t_{peak})**.** It is defined as the elapsed duration from the time when the change is applied to the system until the measure of interest reaches the maximum deviation $M_{peak}^{(ac)}$.

(4) γ-Percentile Time ($t_{percentile}$)**.** It is defined for two cases: (1) when $M_{ss}^{(bc)} \geq M_{ss}^{(ac)}$, it is the elapsed duration from the time when the change is applied to the system until the output measure reaches $M_{ss}^{(ac)} + (100-\gamma)\%$ of $|M_{ss}^{(bc)} - M_{ss}^{(ac)}|$ for the first time, and (2) when $M_{ss}^{(bc)} \leq M_{ss}^{(ac)}$, it is the elapsed duration from the time when the change is applied to the system until the output measure reaches $M_{ss}^{(ac)} - (100-\gamma)\%$ of $|M_{ss}^{(bc)} - M_{ss}^{(ac)}|$ for the first time. $(100-\gamma)$ is always greater than δ as used in settling time definition.

Fig. 1 shows the conceptual view of resiliency metrics.

4. CASE STUDY FOR IAAS CLOUD

In this section, we introduce a case study for Infrastructure-as-a-Service Cloud. We provide both a monolithic model and a set of interacting sub-models in order to show how our technique can be applied in both cases and how the use of interacting sub-models provide better scalability still maintaining a high level of accuracy.

4.1 System Description

In an IaaS Cloud, when a request is processed, a pre-built image is used to deploy one or more VM instances or a pre-deployed VM is customized and made available to the requester. VMs are deployed on physical machines (PMs) each of which may be shared by multiple VMs. The deployed VMs are provisioned with request specific CPU, RAM, and disk capacity. In this paper, we show the analysis for an IaaS Cloud where the PMs are grouped into three pools: hot (running), warm (turned on, but not ready) and cold (turned off). A pre-instantiated VM can be readily provisioned and brought to ready state on a running PM (hot PM) with minimum provisioning delay. Instantiating a VM from an image and deploying it on a warm PM needs additional provisioning time. PMs in the cold pool are turned-off when not in use and deploying a VM on such a PM requires additional startup delays. Thus, organization of PMs in multiple pools helps to minimize power and cooling costs without incurring into high startup delays for all VMs. We assume that all PMs in a pool are identical and all requests are homogeneous, where each request is for one VM.

Submitted user requests enter a first-come, first-served (FCFS) queue. The request at the head of the queue is processed by a Resource Provisioning Decision Engine (RPDE) as follows. The request is provisioned on a hot PM if a pre-instantiated but unassigned VM exists. If no hot PM is available, a PM from the warm pool is used for provisioning the requested VM. If all warm PMs are busy, a PM from the cold pool is used. If no PM is available, the request is rejected (service unavailable). When a running job exits, the capacity used by that VM is released and becomes available for provisioning the next job.

Problem Statement. Performance of the above described IaaS Cloud can be quantified using state-space analytic models. In particular, we will show both monolithic and interacting sub-models. If all the inter-event times are assumed to be exponentially distributed, such models are homogeneous continuous time Markov chains (CTMC). However, during transient analysis, if interacting sub-models are used, their generator matrices evolve with time thus leading to non-homogeneous CTMCs. This paper describes a set of algorithms for resiliency quantification when dealing both with monolithic model and with non-homogeneous, interacting sub-models. We quantify the resiliency of IaaS Cloud for two performance measures - (i) job rejection rate (ρ_{reject}) and (ii) mean number of jobs in RPDE ($E[N_{RPDE}]$). Resiliency metrics are computed to quantify the effects on the system due to changes in job arrival rate and system capacity. Finally, the scalability and accuracy of the resiliency quantification performed with the interacting sub-models are compared with the resiliency quantification approach for a monolithic model.

4.2 Monolithic model

An SRN monolithic model for request provisioning in an IaaS Cloud is shown in Fig. 2. A detailed description of the model can be found in [11]. Transition T_{arr} with rate λ models the job arrivals. The conflict among transitions $T_d^{(h_i)}$ model the searching delay to find a hot PM for resource provisioning. VM provisioning delay on a hot PM is modeled by transitions $T_{inst}^{(h_i)}$ (with rate β_h). Transitions $T_{serv}^{(h_i)}$ represent servicing of jobs. If no token is present in place $P_{buf}^{(h_i)}$ (buffer of i^{th} PM in the hot pool is full) or if one token is present in place $P_{buf}^{(h_i)}$ but no token is present in place $P_{vm}^{(h_i)}$ (the PM is full), then the PM is not able to accept any further request. If this happens for all the PMs in the hot pool, transition $T_n^{(h)}$ is enabled moving a token from place $P_d^{(h)}$ to place $P_d^{(w)}$.

We assume that the RPDE can process only one request at a time and this is represented by the inhibitor arc from place $P_d^{(w)}$ to transitions $T_d^{(h_i)}$ and $T_n^{(h)}$. The modeling of the warm pool is similar to that of the hot pool. However, when there is no job being executed or provisioned on a warm PM, the first request to the warm PM requires an additional startup delay. For this reason, two additional immediate transitions are present for each warm PM, $t_y^{'(w_j)}$ and $t_y^{''(w_j)}$ (with $1 \leq j \leq n_w$), with proper guards. We model the additional delay by means of places $P_{stup}^{(w_j)}$ and transitions $T_{stup}^{(w_j)}$ (with rate γ_w).

When PMs in the warm pool are all busy, transition $T_n^{(w)}$ is enabled and the RPDE tries to provision the request in the cold pool. The modeling of the cold pool is equivalent to that of the warm pool. Main difference with warm pool is that when all PMs are busy in the cold pool, transition T_{drop} will fire modeling the rejection of a request due to insufficient PM capacity.

Outputs of the model are obtained by defining a function that assigns an appropriate reward rate to each marking of the SRN and then computing the expected reward rate in either transient or steady state. From monolithic model, we can compute: job rejection rate due to buffer full (ρ_{block}), job rejection rate due to insufficient capacity (ρ_{drop}), their sum (ρ_{reject}), and the mean number of jobs in RPDE ($E[N_{RPDE}]$).

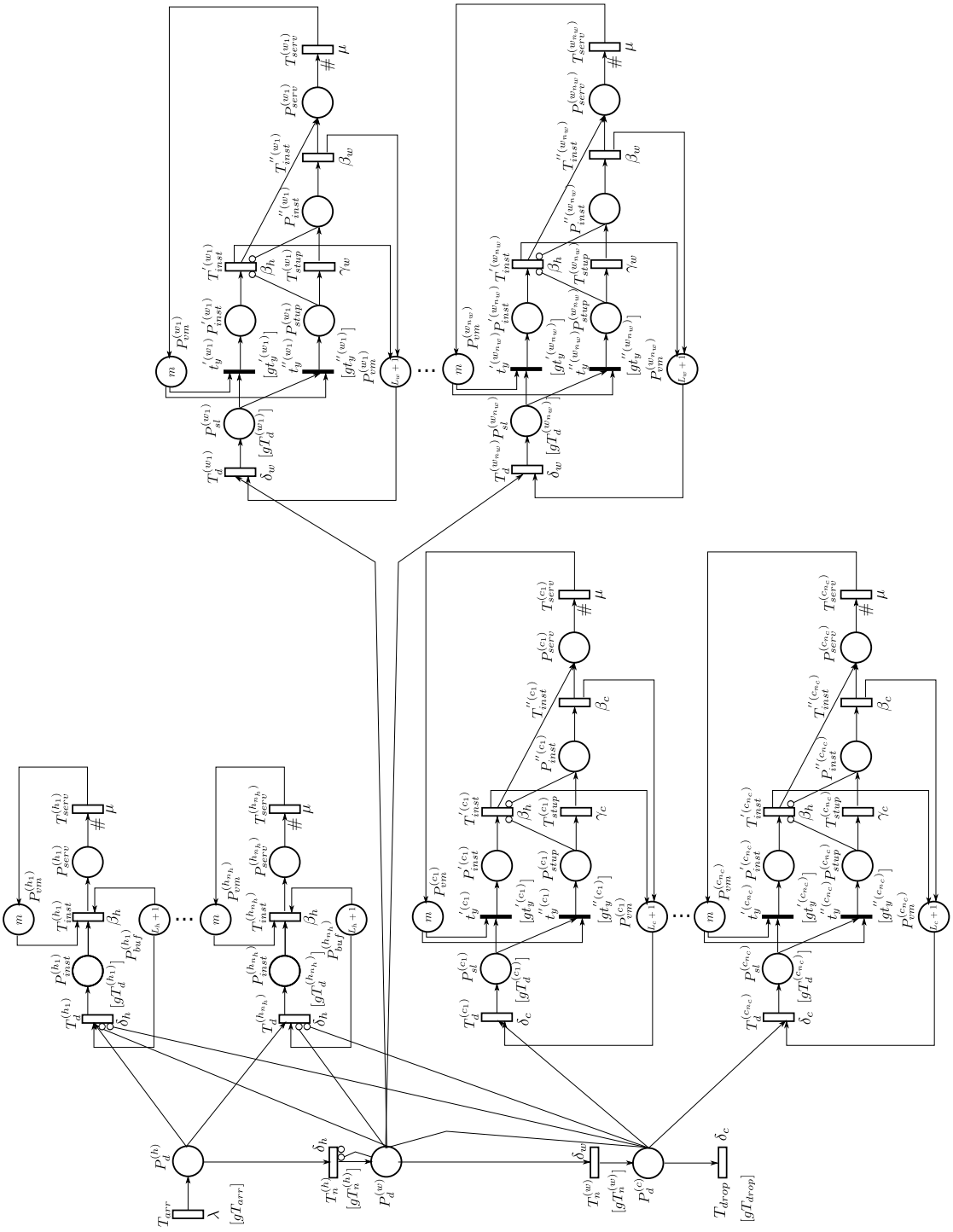


Figure 2: Stochastic reward net model for IaaS Cloud.

4.3 Interacting sub-models

We use plain CTMCs as interacting sub-models. The detailed performance model can be found in [13]. Final solution of the overall model is obtained by fixed-point iteration over individual sub-models. For each sub-model we highlight the fixed-point parameters that are exchange with other sub-models.

Resource provisioning decision engine (RPDE) sub-model. Fig. 3 shows RPDE sub-model. States of the sub-model in Fig. 3 are indexed by (q, x), where q denotes the number of jobs in the RPDE queue and $x \in \{h, w, c\}$ denotes the pool in which the RPDE is searching for available PMs. Input parameters for this sub-model are: (i) job arrival rate (λ), (ii) mean search delays to find a PM from hot/warm/cold pool that can be used for resource provisioning ($1/\delta_h$, $1/\delta_w$ and $1/\delta_c$, respectively), (iii) probabilities that a hot/warm/cold PM can accept a job for resource provisioning (P_h, P_w and P_c, respectively) and (iv) maxi-

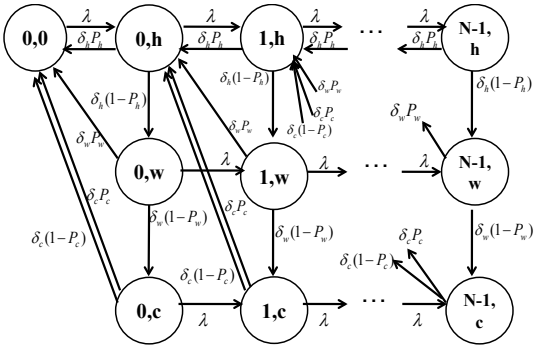


Figure 3: CTMC sub-model for RPDE.

mum number of jobs in RPDE (N). Among all the input parameters, P_h, P_w, and P_c are computed as outputs from the VM provisioning sub-models described later. From the RPDE sub-model, using Markov reward approach, we compute job rejection probability due to buffer full (P_{block}), rejection probability due to insufficient capacity (P_{drop}), their sum: $P_{reject} = P_{block} + P_{drop}$, and mean number of jobs in RPDE ($E[N_{RPDE}]$). Then, multiplying by arrival rate, we can compute job rejection rate due to buffer full (ρ_{block}), rejection rate due to insufficient capacity ρ_{drop}, and their sum: $\rho_{reject} = \rho_{block} + \rho_{drop}$.

VM provisioning sub-models. VM provisioning sub-models capture the provisioning and deployment of requested VMs to accepted jobs. For each hot, warm, and cold PM, we have one CTMC which keeps track of the number of running VMs. VM provisioning sub-model of a pool is the union of individual provisioning sub-models of each PM in that pool.

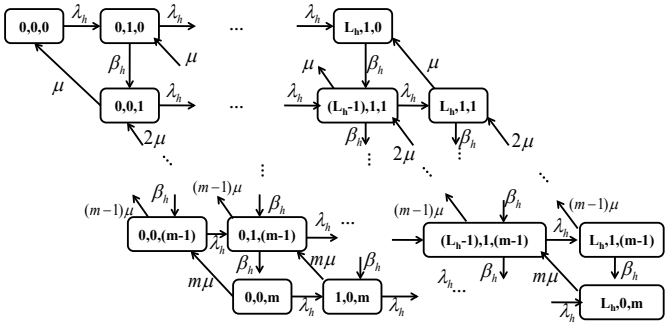


Figure 4: VM provisioning sub-model for a hot PM.

Fig. 4 shows VM provisioning sub-model for a hot PM. Conceptually, the overall hot pool is modeled by a set of independent hot PM sub-models. Note that only one PM sub-model needs to be solved. States of the sub-model in Fig. 4 are indexed by (i, j, k), where i denotes the number of jobs in the PM buffer, j denotes the number of VMs currently being provisioned, k denotes the number of running VMs. Input parameters for a hot PM CTMC are: (i) effective job arrival rate to each hot PM (λ_h), (ii) rate at which VM instantiation, provisioning, and configuration occurs (β_h), (iii) job service rate (μ), (iv) buffer size of hot PM (L_h), and (v) maximum number of VMs that can be deployed on a PM (m). Assuming a total of n_h PMs in the hot pool, λ_h is given by $\lambda(1 - P_{block})/n_h$. Observe that P_{block} is computed from RPDE sub-model. Output of the hot pool sub-model is the steady state probability P_h that at least one PM in the hot pool can accept a job for provisioning.

The CTMCs for a warm and a cold PM [13] are similar to the hot PM sub-model, with few differences, mainly related to the effective arrival rate to each PM and to startup delays. Output of the warm (cold) pool sub-model is the steady state probability P_w (P_c) that at least one PM in the warm (cold) pool can accept a job for provisioning. Probabilities P_h, P_w, and P_c as obtained from hot, warm, and cold pool sub-models, respectively, are used in RPDE sub-model as input parameters.

In Fig. 5, using import graphs, input-output dependencies among VM provisioning sub-models and RPDE sub-model are shown. Such dependencies are resolved via fixed-point iteration.

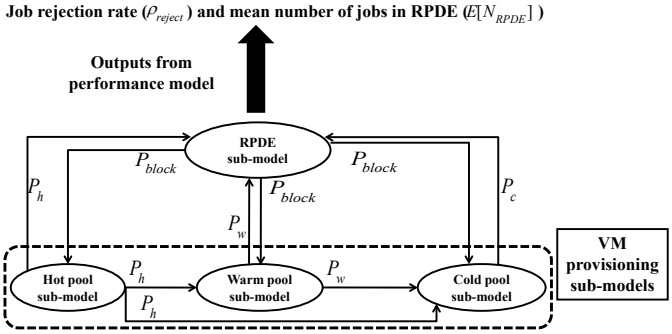


Figure 5: Interactions among the sub-models.

5. FORMALIZATION OF THE APPROACH

In this section, we provide a set of algorithms for our resiliency quantification approach as described in Section 3.2. First, an algorithm is presented in the case in which a single monolithic model is used for the considered system. Then, we show how such an approach can be applied in the case in which an interacting sub-models approach is used, by providing two additional algorithms.

5.1 Resiliency quantification with a monolithic model

Algorithm 1 presents the pseudocode for resiliency quantification using a monolithic model. Given a monolithic model Ω for the system under consideration Σ, we wish to analyze the progress of a measure of interest M before and after a change and to compute the resiliency metrics for it. Both non-structural and structural changes can be modeled by altering the value of a model parameter ω. Let ω_1 and ω_2 be the values of such a parameter before and after the change. Moreover, let T be the length of the time interval for transient analysis and Δ the length of each sub-interval.

To understand the system behavior before the change is applied, we solve the monolithic model for its steady state behavior, using function *steady(.)* (see line 5). From such a steady state analysis, we obtain: (i) steady state probability vector of the monolithic model $\mathbf{S}'$, (ii) steady state value $M_{ss}^{(bc)}$ of performance measure M. Before the transient analysis can be performed, it is necessary to map the state space of the model before the change into the state

Algorithm 1 Resiliency quantification for monolithic model

Input: (i) Ω: monolithic model, (ii) ω_1 and ω_2: values of the input parameter ω before and after the change respectively, (iii) M: measure of interest, (iv) T: length of the time interval for transient analysis, (v) Δ: length of the sub-intervals for transient analysis.

Output: Progress of measure of interest M during the whole analysis and resiliency metrics.

1: declare $\mathbf{S}'$: vector of *probability*;
2: declare $\mathbf{S}''$: vector of *probability*;
3: declare $M_{ss}^{(bc)}, M_{ss}^{(ac)}$: *real number*;
4: declare $M(t)$: vector of *real number*;
5: $\{\mathbf{S}', M_{ss}^{(bc)}\} \leftarrow steady(\Omega, \omega_1, M)$;
6: $\mathbf{S}'' \leftarrow map(\mathbf{S}', \Omega, \omega_1, \omega_2)$;
7: $\{M(t), M_{ss}^{(ac)}\} \leftarrow transient(\Omega, \omega_2, M, \mathbf{S}'', T, \Delta)$;
8: output $M_{ss}^{(bc)}$, $M_{ss}^{(ac)}$ and $M(t)$;
9: $\{t_{set}, PO, t_{peak}, t_{percentile}\} \leftarrow res(M_{ss}^{(bc)}, M_{ss}^{(ac)}, M(t))$;
10: output t_{set}, PO, t_{peak}, and $t_{percentile}$;

Algorithm 2 Transient analysis

Input: (i) Ω: model, (ii) ω_2: value of the input parameter ω, (iii) M: measure of interest, (iv) $\mathbf{S}$: initial probability vector, (v) T: length of the time interval, (vi) Δ: length of the sub-intervals.

Output: Progress of measure of interest M during the transient analysis and corresponding steady state value.

1: declare t: *time instant*;
2: declare *stop*: *boolean*;
3: declare $\mathbf{S}'$: vector of *probability*;
4: declare $M(t)$: vector of *real number*;
5: declare M', M_{ss}: *real number*;
6: $t \leftarrow 0$;
7: *stop* $\leftarrow$ false;
8: do:
9: $\quad t \leftarrow t + \Delta$;
10: $\quad \{\mathbf{S}', M'\} \leftarrow new\_point(\Omega, \omega_2, M, \mathbf{S}, \Delta)$;
11: $\quad \mathbf{S} \leftarrow \mathbf{S}'$;
12: $\quad M(t) \leftarrow add\_point(M(t), M')$;
13: $\quad$ if $t \geq T$:
14: $\quad\quad$ *stop* $\leftarrow$ true;
15: while *stop* is false;
16: $M_{ss} = M'$;
17: output $M(t)$ and M_{ss};

space of the model after the change and to compute the initial probability vector to be used for transient analysis $\mathbf{S}''$. This is performed by function *map(.)* (see line 6). The exact behavior of such a function is out of the scope of this paper. However, it is important to understand that one-to-one, one-to-many, many-to-one mappings are possible. A mapping is one-to-one if a non-structural change is applied to the model (i.e., if the state space of the model before and after the change is exactly the same). Otherwise, one-to-many or many-to-one mapping is performed (i.e., the state space of the model is modified after the change).

Next, the model is solved for its transient behavior through function *transient(.)* (see line 7). This is performed considering the value of the model parameter ω after the change (ω_2) and setting the initial probability vector to be $\mathbf{S}''$. The progress $M(t)$ of measure of interest M is computed till a new steady state value $M_{ss}^{(ac)}$ is reached.

Finally, function *res(.)* is used to compute resiliency metrics as described in Section 3.3 (see line 9).

To clarify further, we also report the pseudocode for function *transient(.)* in Algorithm 2. The function performs a simply cycle in which, at each iteration, a new point of the measure of interest M is computed (see line 10). The loop runs until the transient analysis is completed for all sub-intervals (see line 13-14). For transient analysis in the first sub-interval, the initial probability vector $\mathbf{S}$ is used. For subsequent sub-intervals, values of $\mathbf{S}$ are obtained from the transient analysis in previous sub-intervals (see line 11).

5.2 Resiliency quantification with interacting sub-models

Algorithm 3 Resiliency quantification for interacting sub-models

Input: (i) n: number of interacting sub-models, (ii) $\mathbf{\Omega}$: vector of sub-models, (iii) ω_1 and ω_2: values of the model parameter ω before and after the change respectively, (iv) M: measure of interest, (v) T: length of the time interval for transient analysis, (vi) Δ: length of the sub-intervals for transient analysis, (vii) ϵ: upper bound on error in fixed-point iterations.

Output: Progress of measure of interest M during the whole analysis and resiliency metrics.

1: declare $\mathbf{S_s}', \mathbf{S_s}''$: vector of vector of *probability*;
2: declare $\mathbf{S_t}(t)$: vector of vector of vector of *probability*;
3: declare $\mathbf{f_s}', \mathbf{f_s}''$: vector of *fixed-point parameter*;
4: declare $\mathbf{f_t}'(t), \mathbf{f_t}''(t)$: vector of vector of *fixed-point parameter*;
5: declare $M_{ss}^{(bc)}, M_{ss}^{(ac)}$: *real number*;
6: declare $M(t)$: vector of *real number*;
7: $\mathbf{f_s}' \leftarrow initial\_parameters\_steady(\mathbf{\Omega})$;
8: do:
9: $\quad \mathbf{f_s}'' \leftarrow \mathbf{f_s}'$;
10: $\quad$ for i in $0...n$:
11: $\quad\quad \{\mathbf{S_s}'[i], \mathbf{f_s}'\} \leftarrow steady(\mathbf{\Omega}[i], \omega_1, \mathbf{f_s}')$;
12: while $continue\_iterations\_steady(\mathbf{f_s}', \mathbf{f_s}'', \epsilon)$ is true
13: $M_{ss}^{(bc)} \leftarrow compute\_measure(\mathbf{\Omega}, \mathbf{S_s}', M)$;
14: $\mathbf{S_s}'' \leftarrow map(\mathbf{S_s}', \mathbf{\Omega}, \omega_1, \omega_2)$;
15: $\mathbf{f_t}'(t) \leftarrow initial\_parameters\_transient(\mathbf{\Omega}, \mathbf{S_s}'')$;
16: do:
17: $\quad \mathbf{f_t}''(t) \leftarrow \mathbf{f_t}'(t)$;
18: $\quad$ for i in $0...n$:
19: $\quad\quad \{\mathbf{S_t}(t)[i], \mathbf{f_t}'(t)\} \leftarrow transient(\mathbf{\Omega}[i], \omega_2, \mathbf{f_t}'(t), \mathbf{S_s}''[i], T, \Delta)$;
20: while $continue\_iterations\_transient(\mathbf{f_t}'(t), \mathbf{f_t}''(t), \epsilon)$ is true
21: $\{M(t), M_{ss}^{(ac)}\} \leftarrow compute\_measure(\mathbf{\Omega}, \mathbf{S_t}(t), M)$;
22: output $M_{ss}^{(bc)}$, $M_{ss}^{(ac)}$ and $M(t)$;
23: $\{t_{set}, PO, t_{peak}, t_{percentile}\} \leftarrow res(M_{ss}^{(bc)}, M_{ss}^{(ac)}, M(t))$;
24: output t_{set}, PO, t_{peak}, and $t_{percentile}$;

The main motivation behind using an interacting sub-models approach is the following. A global monolithic model to capture all the details of a large scale system tends to be complex. Even by using methods for the automated generation of models such as stochastic Petri nets, such models become intractable and may not scale to large sized systems. Hence, interacting sub-models approach is frequently used. It can be demonstrated that such an approach reduces the complexity of analysis without significantly affecting accuracy.

Final solution of the overall model is usually performed at steady state and obtained by fixed-point iterations over individual sub-models steady state solutions. Sub-models exchange parameters among each other at each fixed-point

iteration and the iterations are concluded as soon as the values of the exchanged parameters remain constant or within a certain interval (upper bound on error in fixed-point iterations). However, our resiliency quantification approach implies a transient solution of the considered model. In the case of interacting sub-models, this causes exchanged parameters to become functions of time rather than single values and thus gives rise to non-homogeneous, interacting sub-models. We explain this phenomenon with an example. During transient analysis of interacting sub-models for our IaaS Cloud use case, P_{block}, obtained as an output measure from RPDE sub-model, is used in VM provisioning sub-models as an input parameter to compute P_h, P_w, and P_c. Since the transient value of P_{block} changes with time, values of job-arrival rates in hot, warm, and cold sub-models, as derived from P_{block}, also change with time. As a result, the generator matrices of hot, warm, and cold sub-models become time-dependent. The transient values of P_h, P_w, and P_c are used to compute transition rates in the RPDE sub-model. This leads to a time-dependent generator matrix of RPDE sub-model.

First of all, we introduce an algorithm for resiliency quantification using interacting sub-models approach without considering this issue. Later, we modify the algorithm to avoid non-homogeneity still maintaining accuracy. Algorithm 3 presents the pseudocode for resiliency quantification using interacting sub-models. The system under consideration Σ is modeled through a vector $\mathbf{\Omega}$ of n interacting sub-models and also in this case we want to analyze the progress of a measure of interest M before and after a change and to compute the resiliency metrics for it. Let ω_1 and ω_2 be the values of a model parameter ω before and after the change and let T and Δ be the length of the time interval for transient analysis and the length of each sub-interval, respectively. Moreover, let ϵ be the upper bound on the error due to the fixed-point iterations. This parameter is used to check if fixed-point iterations can stop and it is computed at each iteration as a function of the fixed-point parameters exchanged among the sub-models (see lines 15 and 23).

The structure of Algorithm 3 is similar to the one of Algorithm 1 with the difference that steady state and transient analysis (performed at lines 5 and 7 of Algorithm 1) are replaced with a set of fixed-point iterations. First of all, we consider steady state analysis. At the beginning of such an analysis, initial values for the fixed-point parameters are guessed (see line 10). Then, a loop starts (see lines 11-15) and at each iteration all the sub-models are solved at steady state and fixed-point parameters are exchanged among sub-models (see line 14). As soon as the upper bound on error is reached, iterations stop and the value of the measure of interest at steady state before the change $M_{ss}^{(bc)}$ is computed (see line 16). At line 17, a mapping is performed to compute the initial probability vector for each sub-model based on the corresponding steady state probability vector before the change, in order to enforce the change. Finally, another round of fixed-point iterations is performed (see lines 19-23) for transient analysis.

The main difference between the fixed-point iterations performed at lines 11-15 and the ones performed at lines 19-23 is that in the latter case, fixed-point parameters exchanged among sub-models become functions of time. This causes two main problems: (i) the upper bound error on fixed-point iterations needs to be computed on functions instead of single values to understand if fixed-point iterations should stop; (ii) each sub-model receives from other sub-models fixed-point parameters that are functions of time, thus becoming a non-homogeneous CTMC.

Solution of non-homogeneous models could be problematic especially in the presence of multiple model parameters varying with time. For this reason, in this paper we introduce a third algorithm that allows us to apply our resiliency quantification approach to the case of interacting sub-models without incurring the non-homogeneity issue but still maintaining an acceptable level of accuracy. Algorithm 4 is a variant of Algorithm 3 in which the transient analysis is not conducted through a single set of fixed-point iterations and by exchanging fixed-point parameters in the form of functions of time. We use an approach inspired

Algorithm 4 Resiliency quantification for interacting sub-models with piece-wise approximation.

Input: (i) n: number of interacting sub-models, (ii) $\mathbf{\Omega}$: vector of sub-models, (iii) ω_1 and ω_2: values of the model parameter ω before and after the change respectively, (iv) M: measure of interest, (v) T: length of the time interval for transient analysis, (vi) Δ: length of the sub-intervals for transient analysis, (vii) ϵ: upper bound on error in fixed-point iterations.
Output: Progress of measure of interest M during the whole analysis and resiliency metrics.

1: declare t: *time instant*;
2: declare *stop*: *boolean*;
3: declare $\mathbf{S_s}$: vector of vector of *probability*;
4: declare $\mathbf{S_t'}, \mathbf{S_t''}$: vector of vector of *probability*;
5: declare $\mathbf{f_s'}, \mathbf{f_s''}$: vector of *fixed-point parameter*;
6: declare $\mathbf{f_t'}, \mathbf{f_t''}, \mathbf{f_t'''}$: vector of *fixed-point parameter*;
7: declare $M_{ss}^{(bc)}, M_{ss}^{(ac)}, M'$: *real number*;
8: declare $M(t)$: vector of *real number*;
9: $\mathbf{f_s}' \leftarrow initial\_parameters\_steady(\mathbf{\Omega})$;
10: do:
11: $\quad \mathbf{f_s}'' \leftarrow \mathbf{f_s'}$;
12: $\quad$ for i in $0...n$:
13: $\quad\quad \{\mathbf{S_s}[i], \mathbf{f_s}'\} \leftarrow steady(\mathbf{\Omega}[i], \omega_1, \mathbf{f_s}')$;
14: while $continue\_iterations\_steady(\mathbf{f_s}', \mathbf{f_s}'', \epsilon)$ is true
15: $M_{ss}^{(bc)} \leftarrow compute\_measure(\mathbf{\Omega}, \mathbf{S_s}, M)$;
16: $\mathbf{S_t}'' \leftarrow map(\mathbf{S_s}, \mathbf{\Omega}, \omega_1, \omega_2)$;
17: $\mathbf{f_t}' \leftarrow initial\_parameters\_transient(\mathbf{\Omega}, \mathbf{S_t}'')$;
18: do:
19: $\quad t \leftarrow t + \Delta$;
20: $\quad \mathbf{f_t}''' \leftarrow \mathbf{f_t'}$;
21: $\quad$ do:
22: $\quad\quad \mathbf{f_t}'' \leftarrow \mathbf{f_t'}$;
23: $\quad\quad$ for i in $0...n$:
24: $\quad\quad\quad \{\mathbf{S_t}'[i], \mathbf{f_t}'\} \leftarrow new\_point(\mathbf{\Omega}[i], \omega_2, \mathbf{f_t}', \mathbf{S_t}''[i], \Delta)$;
25: $\quad\quad\quad \mathbf{f_t}' \leftarrow approximate(\mathbf{f_t}', \mathbf{f_t}''')$;
26: $\quad$ while $continue\_iterations\_transient(\mathbf{f_t}', \mathbf{f_t}'', \epsilon)$ is true
27: $\quad M' \leftarrow compute\_measure\_point(\mathbf{\Omega}, \mathbf{S_t}', M)$;
28: $\quad M(t) \leftarrow add\_point(M(t), M')$;
29: $\quad \mathbf{S_t}'' \leftarrow \mathbf{S_t}'$;
30: $\quad$ if $t \geq T$:
31: $\quad\quad stop \leftarrow$ true;
32: while *stop* is false;
33: $M_{ss}^{(ac)} \leftarrow M'$;
34: output $M_{ss}^{(bc)}$, $M_{ss}^{(ac)}$ and $M(t)$;
35: $\{t_{set}, PO, t_{peak}, t_{percentile}\} \leftarrow res(M_{ss}^{(bc)}, M_{ss}^{(ac)}, M(t))$;
36: output t_{set}, PO, t_{peak}, and $t_{percentile}$;

from piece-wise constant approximation method [28]. We divide the time interval for transient analysis into small sub-intervals and we assume that the values of model transition rates are constant within these sub-intervals. Fixed-point iteration among the sub-models is carried out for each sub-interval and outputs from one sub-interval are transferred as inputs to the next sub-interval.

In Algorithm 4, outer loop (lines 18-32) runs until we finish the transient analysis for all sub-intervals. The inner loop (lines 21-26) represents the fixed-point iteration within each sub-interval. For transient analysis in the first sub-interval, initial state probability vectors ($\mathbf{S_t}''$) and fixed point variables ($\mathbf{f_t}'$) are obtained from the steady state analysis. For subsequent sub-intervals, values of $\mathbf{S_t}''$ and $\mathbf{f_t}'$ are obtained from transient analysis in previous sub-intervals. However, an average is computed between the current value of each fixed-point parameter and the value obtained from previous sub-interval. Such averaging process reduces the errors introduced due to non-homogeneous behavior of the system and it is performed by function *approximate*(.) at line 25. Before starting the fixed-point iteration within a sub-interval, we store the values of fixed point variables obtained from previous sub-intervals in $\mathbf{f_t}'''$. Then, we check the values of $\mathbf{f_t}'$ and $\mathbf{f_t}''$ in successive iterations to determine the condition for convergence in fixed-point iteration (see line 26). After the fixed-point iteration in a sub-interval finishes, we update the values of initial state probability vectors ($\mathbf{S_t}''$) at line 29. We use these updated values in the next sub-interval. Observe that within a sub-interval, fixed-point variables (used as input parameters to the sub-models) are updated in each iteration while the initial state probability vectors used to obtain the transient solutions of the sub-models remain unchanged.

6. NUMERICAL RESULTS

Case	Description
case-I	Increase in λ, 100 PMs in each pool
case-II	Increase in λ, 1000 PMs in each pool
case-III	Decrease in λ, 100 PMs in each pool
case-IV	Decrease in λ, 1000 PMs in each pool
case-V	Removal of 900 PMs from hot, 750 PMs from warm and cold pool each
case-VI	Removal of 850 PMs from hot and warm each, 700 PMs from cold pool
case-VII	Removal of 800 PMs from each pool
case-VIII	Addition of 150 PMs to hot pool
case-IX	Addition of 75 PMs to hot and warm pool each
case-X	Addition of 50 PMs to each pool

Table 2: Description of the considered cases.

In this section, we discuss key results for large scale IaaS Cloud resiliency quantification using scalable interacting sub-models. Subsequently, model solution time and accuracy of our proposed resiliency algorithm for interacting sub-models are compared with the monolithic model.

6.1 Resiliency quantification for large scale IaaS Cloud

We use SHARPE [29] software package to solve interacting sub-models and quantify the IaaS Cloud resiliency of the performance measures. Using Python scripts, we automate the generation of SHARPE input files and implement the following tasks: (i) mapping the state space of the sub-models before the change into the state space of the sub-models after the change, (ii) transferring the initial probability vectors and fixed-point variables from current sub-interval to the next sub-interval. We quantified large scale IaaS Cloud resiliency upon two types of changes: change in job-arrival rate and change in the number of PMs. In Table 2, we define ten cases that we have analyzed and we numerically compare the corresponding resiliency metrics (with $\gamma = 90$ and $\delta = 0.1$) in Table 3.

Effect of changing job-arrival rate (λ)**.** Fig. 6(a) shows the transient effects of changing the job arrival rate (λ) on job rejection rate. We analyze the resiliency of two Cloud configurations with 100 PMs and 1000 PMs in each pool. For both configurations, maximum 2 VMs can run simultaneously on a PM. At time instance $t = 0$, arrival rate is increased from 1000 to 1300 jobs/hr. In case-I and case-II, settling times are $t_{set}^{(I)}$ and $t_{set}^{(II)}$ respectively. At $t = 1.5$ hr, job arrival rate is reduced to initial value, i.e., 1000 jobs/hr. When the change is removed, settling times are $t_{set}^{(III)}$ and $t_{set}^{(IV)}$ respectively.

In Fig. 6(b), similar transient effects are shown for the mean number of jobs in RPDE (a measure of congestion in the Cloud).

Effect of changing system capacity. Next, we quantify the resiliency of IaaS Cloud due to addition or removal of PMs. Fig. 7(a) shows the transient effects of removing PMs on the job rejection rate. We start with a scenario where 1000 PMs are equally distributed in each pool (as denoted by $(1000, 1000, 1000)$). At $t = 0$, we remove total 2400 PMs in three different ways: (i) 900 PMs are removed from hot pool and 750 PMs are removed from warm and cold pool each (denoted by $(100, 250, 250)$), (ii) 850 PMs are removed from hot and warm pool each and 700 PMs are removed from cold pool (denoted by $(150, 150, 300)$), and (iii) 800 PMs are removed from each pool (denoted by $(200, 200, 200)$). After the removal of the PMs, rejection rate increases and finally reaches a steady state. Settling time is maximum when 800 PMs are removed from each pool, i.e., the Cloud configuration is changed from $(1000, 1000, 1000)$ to $(200, 200, 200)$. Although, observe that the job rejection rate is maximum when the Cloud configuration is changed from $(1000, 1000, 1000)$ to $(100, 250, 250)$. Similar effects on the mean number of jobs in RPDE are shown in Fig. 7(b).

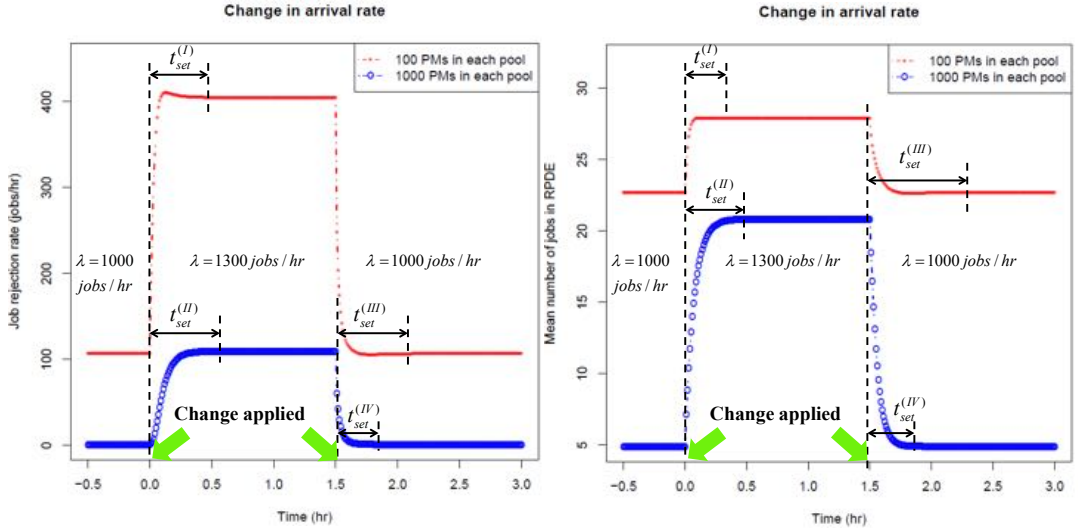
Fig. 8(a) shows the transient effects of adding new PMs on job rejection rate. We start with a scenario where 150 PMs are equally distributed in each pool (as denoted by $(150, 150, 150)$). At $t = 0$, we add total 150 PMs in three different ways: (i) 150 PMs are added to hot pool (denoted by $(300, 150, 150)$), (ii) 75 PMs are added to hot and warm pool each (denoted by $(225, 225, 150)$), and (iii) 50 PMs are added to each pool (denoted by $(200, 200, 200)$). After the addition of the PMs, rejection rate decreases and finally reaches a steady state. Among the three changes, settling time is minimum when PMs are added only to the hot pool. Similar effects on the mean number of jobs in RPDE are shown in Fig. 8(b).

6.2 Comparison of scalability with monolithic model

We use Stochastic Petri Net Package (SPNP) [16] for resiliency quantification using the monolithic model. SPNP allows to easily map the state space of the model before the change into the state space of the model after the change and to compute the initial probability vectors in a straightforward manner. Both the interacting sub-models and the

Cases	Resiliency metrics for job rejection rate				Resiliency metrics for mean number of jobs in RPDE			
	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$
case-I	0.47	1.45	0.13	0.06	0.35	0.16	0.13	0.05
case-II	0.52	-	-	0.21	0.48	-	-	0.16
case-III	0.55	1.1	0.29	0.06	0.78	0.27	0.32	0.13
case-IV	0.28	-	-	0.07	0.36	-	-	0.13
case-V	0.87	-	-	0.37	0.75	-	-	0.29
case-VI	1.22	-	-	0.61	1.17	-	-	0.56
case-VII	1.73	-	-	0.90	1.75	-	-	0.88
case-VIII	0.78	-	-	0.15	0.89	-	-	0.20
case-IX	0.98	-	-	0.21	1.05	-	-	0.26
case-X	0.91	-	-	0.22	0.97	-	-	0.27

Table 3: Comparison of resiliency metrics for the different cases described in Table 2.



(a) (b)

Figure 6: Effect of changing job arrival rate (a) job rejection rate and (b) mean number of jobs in the RPDE.

(#PMs in each pool, maximum # VMs per PM)	Monolithic model	Interacting sub-models
(1, 1)	912	27
(1, 2)	4, 655	35
(1, 4)	20, 691	47
(1, 8)	116, 603	71
(1, 16)	768, 075	119
(1, 32)	5, 543, 915	215
(1, 38)	9, 848, 061	251
(1, 39)	Memory overflow	257
(2, 1)	43, 776	27
(3, 1)	2, 101, 248	27
(4, 1)	Memory overflow	27
(500, 64)	–	407

Table 4: Comparison of model states.

(#PMs in each pool, maximum # VMs per PM)	Monolithic model	Interacting sub-models
(1, 1)	0.124	0.066
(1, 2)	0.196	0.074
(1, 4)	0.556	0.088
(1, 8)	2.998	0.141
(1, 16)	79.563	0.208
(1, 32)	188.174	0.346
(1, 38)	293.672	0.399
(1, 39)	Memory overflow	0.406
(2, 1)	2.024	0.063
(3, 1)	166.704	0.062
(4, 1)	Memory overflow	0.060
(500, 64)	–	0.055

Table 5: Comparison of model solution times in seconds.

monolithic model were solved using a desktop PC with Intel Core 2 Duo processor (E8400, 3.0 GHz) and 4 GB memory. In Table 4, we report the state space for both the monolithic model and interacting sub-models. Monolithic model runs into a memory overflow problem when the number of PMs in each pool increases beyond 3 and the number of VMs per PM increases beyond 38. We observe that the state space size of the monolithic model increases quickly and becomes too large to construct the reachability graph even for small number of PMs. However, with interacting sub-models approach, the state space increases at a slower rate as the maximum number of VMs per PM is increased and remains constant with increasing number PMs. A comparison of solution times is shown in Table 5. Solution time for monolithic model increases almost exponentially with the increase in model size. Solution time for interacting sub-models remains almost constant with the increase in model size. Clearly, interacting sub-models approach facilitates fast quantification of large scale IaaS Cloud resiliency.

6.3 Comparison of accuracy

Using 2 PMs per pool and 1 VM per PM, we compare the accuracy of resiliency quantification using monolithic model and interacting sub-models approach. We quantify the resiliency of two performance measures: job rejection rate and mean number of jobs in RPDE respectively, when subjected to a change in arrival rate (λ). At $t = 0$, we change the job arrival rate from 350 jobs/hr to 400 jobs/hr. In Table 6, we

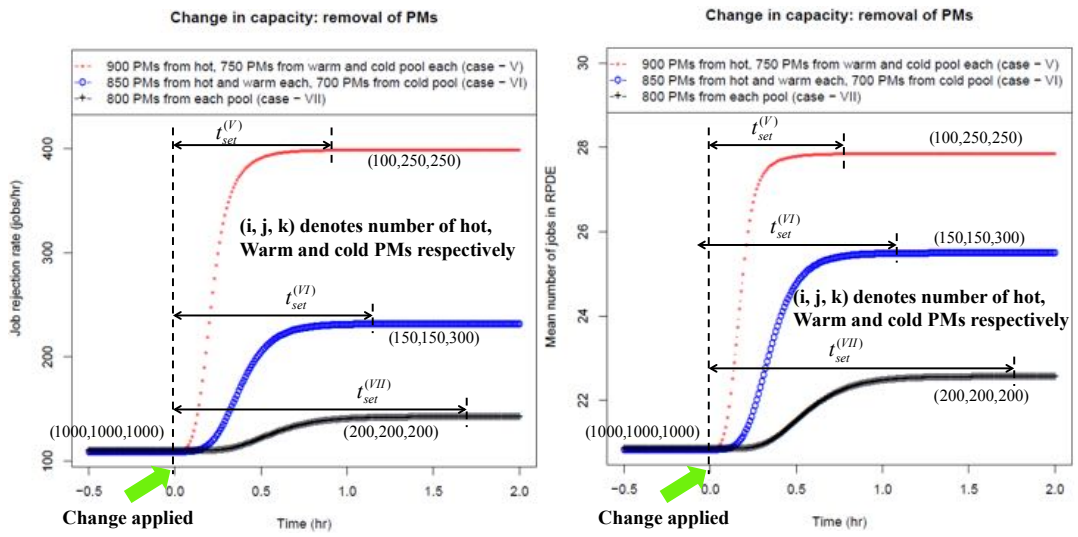


Figure 7: Effect of removing PMs on (a) job rejection rate and (b) mean number of jobs in the RPDE.

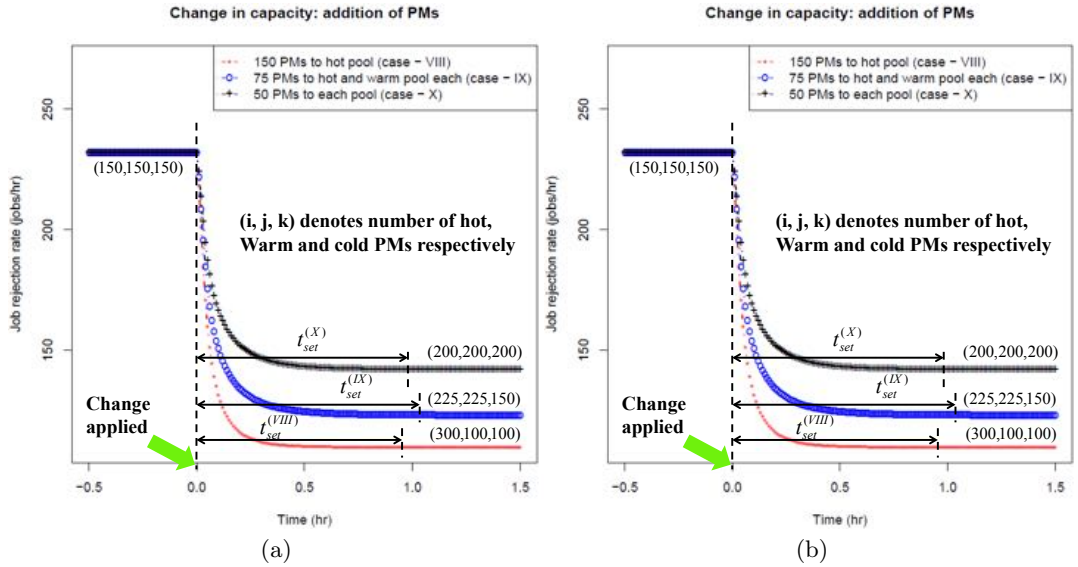


Figure 8: Effect of adding PMs on (a) job rejection rate and (b) mean number of jobs in the RPDE.

observe that the resiliency metrics are comparable in both the approaches. There are two sources of errors in interacting sub-models based resiliency quantification approach. They are: (i) errors due to decomposition of the monolithic model and (ii) errors due to non-homogeneous behavior of the system during transient phase. In this paper, we only focus on resiliency metrics that capture the errors in the transient analysis. Analysis of errors due to model decomposition, is shown in [12].

7. CONCLUSIONS AND FUTURE WORK

In this paper, a general approach for resiliency quantification of large scale systems using stochastic analytic state space models is presented. Monolithic models or interacting sub-models can be used. In the case of interacting sub-models non-homogeneity in time arises due to the combination of transient analysis and fixed-point iterations. At the best of our knowledge, this is the first paper facing this problem in a general way. We apply our technique to an IaaS Cloud use case. Resiliency metrics described in this paper can be used to quantify the transient behavior of a IaaS Cloud under sudden changes and can help during design stage of the IaaS Cloud. In our future work, we plan on investigating the effects of different types of failures (such as failures of PMs and VMs) and computing the resiliency of reliability, availability and performability metrics of IaaS Cloud. Moreover, we plan to further extend our analytic models in order to take into account a larger number of pools, different provisioning strategies, and heterogeneity in requests and PMs. Also different types of job arrivals and

Algorithms	Resiliency metrics for job rejection rate				Resiliency metrics for mean number of jobs in RPDE			
	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$	t_{set}	$PO(\%)$	t_{peak}	$t_{percentile}$
Interacting sub-models	0.12	$1.52e-04$	0.20	0.04	0.10	$0.21e-04$	0.18	0.04
Monolithic model	0.10	$0.85e-04$	0.21	0.04	0.11	$0.24e-04$	0.23	0.04

Table 6: Comparison of resiliency metrics between the two algorithms when arrival rate is increased.

service time distributions can be easily taken into consideration so that interesting evaluation about the impact of such assumptions on the resiliency of the system can be performed.

8. REFERENCES

[1] P. K. Agarwal, A. Efrat, S. Ganjugunte, D. Hay, S. Sankararaman, and G. Zussman. The resilience of WDM networks to probabilistic geographical failures. In *INFOCOM*, 2011.

[2] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa. Reliability assessment of wireless sensor nodes with non-linear battery discharge. 2010.

[3] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir. Toward exascale resilience. *International Journal of High Performance Computing Applications*, 23(4):374–388, 2009.

[4] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomputing Frontiers and Innovations*, 1, 2014.

[5] P. F. Chimento and K. S. Trivedi. The completion time of programs on processors subject to failure and repair. *IEEE Trans. on Computers*, 42(10):1184–1194, 1993.

[6] N. DeBardeleben et al. *High-End Computing Resilience: Analysis of Issues Facing the HEC Community and Path-Forward for Research and Development*. 2010.

[7] S. Distefano, F. Longo, and M. Scarpa. Symbolic representation techniques in dynamic reliability evaluation. pages 45–53, 2010.

[8] C. Engelmann and C. Leangsuksun. Modeling techniques towards resilience. In *National HPC Workshop on Resilience*, 2009.

[9] O. Erdene-Ochir, M. Minier, F. Valois, and A. Kountouris. Resiliency of wireless sensor networks: Definitions and analyses. In *ICT*, 2010.

[10] R. Ghosh, F. Longo, V. Naik, R. A.J., and K. Trivedi. Resiliency quantification for large scale systems: An iaas cloud use case. In *InfQ 2016 workshop*, 2016.

[11] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi. Modeling and performance analysis of large scale iaas clouds. *Future Generation Computer Systems*, 29(5):1216 – 1234, 2013. Special section: Hybrid Cloud Computing.

[12] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi. Modeling and performance analysis of large scale IaaS clouds. *Elsevier Future Generation Computer Systems*, 29(5):1216–1234, 2013.

[13] R. Ghosh, K. S. Trivedi, V. K. Naik, and D. S. Kim. Performability analysis for Infrastructure-as-a-Service cloud: An interacting stochastic models approach. In *PRDC*, 2010.

[14] B. Haverkort, R. Marie, G. Rubino, and K. S. Trivedi (eds.). *Performability Modeling Tools and Techniques*. Wiley, 2001.

[15] P. Heegard and K. S. Trivedi. Network survivability modeling. *Computer Networks*, 53(8):1215–1234, 2009.

[16] C. Hirel, B. Tuffin, and K. S. Trivedi. SPNP: stochastic Petri nets. version 6. In *Lecture Notes in Computer Science*, 2000.

[17] R. Jhawar and V. Piuri. Fault tolerance and resilience in cloud computing environments. In J. R. Vacca, editor, *Computer and Information Security Handbook, 2nd Edition*. Morgan Kaufmann, MA, USA, 2013.

[18] V. G. Kulkarni et al. The completion time of a job on multimode systems. *Adv. Appl. Prob.*, 19(4):932–954, 1987.

[19] J. C. Laprie. From dependability to resilience. In *DSN*, 2008.

[20] Q. Liang and B.-S. Lee. Delivering high resilience in designing Platform-as-a-Service clouds. In *IEEE CLOUD*, 2011.

[21] Y. Liu and K. S. Trivedi. Survivability quantification: The analytical modeling approach. *Intl. Journal of Performability Eng.*, 2(1):29–44, 2006.

[22] W. A. Najjar and J.-L. Gaudiot. Network resilience: A measure of network fault tolerance. *IEEE Trans. Computers*, 39(2):174–181, 1990.

[23] G. Ostrouchov, T. Naughton, C. Engelmann, G. Vallee, and S. Scott. Nonparametric multivariate anomaly analysis in support of HPC resilience. In *ESCIW*, 2009.

[24] A. Puggelli, M. Mozumdar, A. Sangiovanni-Vincentelli, and L. Lavagno. A routing-algorithm-aware design tool for indoor wireless sensor networks. In *ICNC*, 2012.

[25] L. Simoncini. Resilient computing: An engineering discipline. In *IPDPS*, 2009.

[26] J. P. Sterbenz et al. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Elsevier Computer Networks*, 54(8):1245 – 1265, June 2010.

[27] B. Tian, S. Norden, and T. Woo. Trading resiliency for security: model and algorithms. In *ICNP*, 2004.

[28] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Wiley, 2001.

[29] K. S. Trivedi and R. Sahner. Sharpe at the age of twenty two. *SIGMETRICS Perform. Eval. Rev.*, 36(4):52–57, Mar. 2009.

[30] D. Xuan, S. Chellappan, and X. Wang. Resilience of structured peer to peer systems: Analysis and enhancement. In *Jie Wu, editor, Handbook On Theoretical And Algorithmic Aspects Of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*. Auerbach Publications, 2005.

Identifying Communication Patterns between Virtual Machines in Software-Defined Data Centers

Claudia Canali
Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia
Via P. Vivarelli 10
41125, Modena, Italy
claudia.canali@unimore.it

Riccardo Lancellotti
Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia
Via P. Vivarelli 10
41125, Modena, Italy
riccardo.lancellotti@unimore.it

ABSTRACT

Modern cloud data centers typically exploit management strategies to reduce the overall energy consumption. While most of the solutions focus on the energy consumption due to computational elements, the advent of the Software-Defined Network paradigm opens the possibility for more complex strategies taking into account the network traffic exchange within the data center. However, a network-aware Virtual Machine (VM) allocation requires the knowledge of data communication patterns, so that VMs exchanging significant amount of data can be placed on the same physical host or on low cost communication paths. In Infrastructure as a Service data centers, the information about VMs traffic exchange is not easily available unless a specialized monitoring function is deployed over the data center infrastructure. The main contribution of this paper is a methodology to infer VMs communication patterns starting from input/output network traffic time series of each VM and without relaying on a special purpose monitoring. Our reference scenario is a software-defined data center hosting a multi-tier application deployed using horizontal replication. The proposed methodology has two main goals to support a network-aware VMs allocation: first, to identify couples of intensively communicating VMs through correlation-based analysis of the time series; second, to identify VMs belonging to the same vertical stack of a multi-tier application. We evaluate the methodology by comparing different correlation indexes, clustering algorithms and time granularities to monitor the network traffic. The experimental results demonstrate the capability of the proposed approach to identify interacting VMs, even in a challenging scenario where the traffic patterns are similar in every VM belonging to the same application tier.

1. INTRODUCTION

The need to reduce the energy consumption of large scale cloud data centers is attracting a lot of attention towards energy-efficient strategies for resource management and VMs allocation. Most research studies merely consider VMs allocation as a way to reduce the number of active servers [5, 8, 7, 15], while more recent proposals also take into account the energy consumption related to the data transfers among VMs, for example with the aim to co-locate on the same server VMs that intensively communicate among themselves [14, 18].

The recent advent of Software-Defined Data Centers (SDDCs), where all elements of the infrastructure – including networking – are virtualized as software components and the separation of the network control plane from the data plane allows flexible network management and reconfiguration, gave a further impulse to energy-efficient strategies taking into account data traffic exchanges within the data center. However, the adoption of a network-aware VMs allocation strategy requires the knowledge about the communication patterns between VMs. In this paper, we consider the point of view of the provider of an Infrastructure as a Service (IaaS) data center, where each VM is seen as a black box without any further knowledge of the software running on it. In this scenario, obtaining a data transfer matrix between the VMs is not possible unless a specialized monitoring infrastructure is developed and deployed over the data center, as in [16, 19]. The most popular monitoring services in industry[1] and literature [3], indeed, just provide the input/output traffic rate of each VM, without a per-source/per-destination breakdown. Hence, specific strategies are required to infer the VMs communication patterns starting from these low level information.

In order to address this issue, we propose a novel approach explicitly aiming to identify interacting VMs in the case multi-tier applications are horizontally replicated over a SDDC, as shown in Fig. 1. The tiers of each application are organized in vertical stacks that are replicated over the data center; each tier of an application is hosted on a separate VM, and we assume that each VM communicates only with other VMs within the same vertical stack, while no communication between VMs belonging to two different vertical stacks occurs. In this scenario, a request dispatcher typically distributes the incoming workload among the vertical stacks of the application. It is important to note that the presence of the load-balancing request dispatcher leads to very similar utilization and communication patterns of VMs belonging to the same tier but to different vertical stacks of the same application (e.g., VMs VM1 and VM2 in the figure). This similarity represents one of the most challenging aspects for the problem of identifying communicating VMs, because input/output traffic patterns may be correlated even if no communication occurs between them.

The main contribution of this paper is the proposal of a methodology that, starting from the time series of the aggregated network traffic of each VM, is able to identify which VMs intensively communicate each other and which ones belong to the same vertical stack of the applications deployed over a SDDC. Both these pieces of information are extremely valuable as input to VMs allocation strategies to reduce the energy consumption due to the network traffic exchange. Ideally, indeed, VMs exchanging large amount of data and/or belonging to the same vertical stack of an application should be placed on the same physical host or on hosts connected by low cost links to reduce energy consumption. The pro-

[1] https://aws.amazon.com/cloudwatch/

posed methodology exploits correlation techniques to identify co-occurring similarities in the communication patterns of different VMs, and clustering algorithms to identify VMs belonging to the same vertical stacks of an application. In this paper, we compare the use of different correlation indexes and clustering algorithms applied to VMs network utilization time series collected with varying time granularities in order to evaluate the capability of the proposed methodology of identifying the interacting VMs. Our experiments, based on the deployment of a benchmark over a cloud infrastructure, show that the analysis of the correlation among VMs input/output time series and the application of clustering algorithms represent a promising way to identify VMs that intensively communicate within a SDDC.

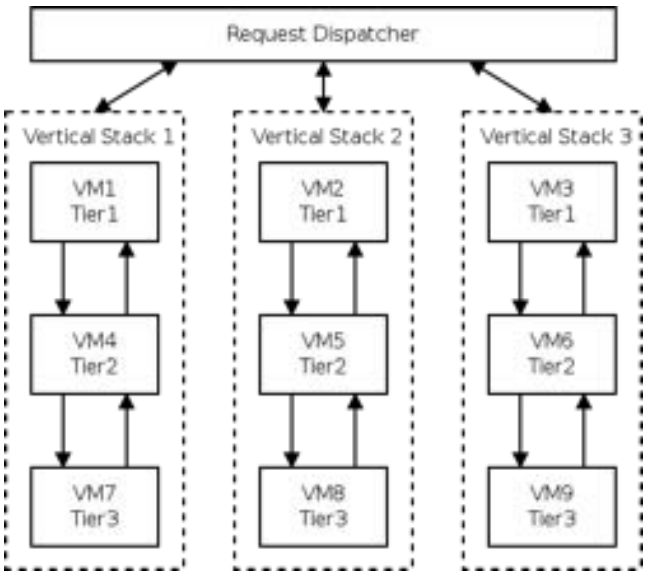


Figure 1: Multi-tier application deployed over a cloud data center

The remainder of this paper is organized as follows. Section 2 describes the architecture of a Software-Defined Data Center. Section 3 models the problem and outlines the proposed methodology. Section 4 describes the experimental results used to validate our proposal. Finally, Section 5 concludes the paper with some final remarks.

2. SOFTWARE-DEFINED DATA CENTER

Fig. 2 shows the structure of a Software-Defined Data Center (SDDC) where virtualized physical hosts run multiple VMs (that are logically organized into applications, tiers and vertical stacks as in Figure 1); the figure represents the physical infrastructure of the data center and the logical organization of the management processes.

The physical infrastructure is divided into manageable subareas, called PODs [4], representing building blocks including physical hosts connected to first level network switches; these switches are in turn connected to a Data Center Network Core, that may be contain more levels of switches. This network structure represents a typical fat-tree topology, which is one of the most popular used in both traditional and Software-Defined data centers because it can provide fault tolerance and scalability thanks to the ability to exploit multiple paths between end-nodes [1], as shown in Fig. 2.

Within each physical host, multiple VMs are running different tiers of the software applications. The VMs are connected by a virtual network to the physical infrastructure. On each host we have a monitoring subsystem, typically operating at the level of the virtualized hypervisor, that periodically collects resources usage time series for each VM (e.g., CPU, memory, input/output). The collected information usually includes also aggregated data about the input and output network traffic of each VM. In a IaaS data center, this is the only information available about the traffic exchange of a VM.

The dashed box on the left side of the figure describes the Data Center Management and the related components. This block receives two types of input. First, the VMs Monitor process receives the data collected by the hypervisors located on each physical host: these data include the usage time series for different resources (CPU, memory, input/output and network traffic). Second, the Network Monitor receives the information coming from the data plane of the data center network infrastructure, that is the set of network switches. All these collected data are sent to the core components of the Data Center Management of a SDDC, which are the VMs Manager and the Network Manager. The former is responsible for running the strategies for VMs allocation, that are communicated to the hypervisors local to each host; it is worth to note that, differently from VMs allocation based only on computational resources, network-aware allocation strategies require the VMs Manager to take into account also information about the network traffic exchange between VMs. The Network Manager includes the control plane, which is responsible for implementing strategies of network management and reconfiguration, that are communicated to the data plane; this separation between control and data plane introduces the benefits of a centralized approach to network configuration, that is programmable at the software level rather than manually reconfigurable through distributed low-level interfaces [10].

From an energy point of view, SDDCs realize a more seamless integration of the network within the data center IT processes, opening up to the possibility of novel energy-efficient resource strategies for the cloud infrastructures integrating complex and adaptive network management. Network-aware VMs allocation strategies to reduce energy consumption in this kind of data centers not only aim to minimize the number of physical host turned on, but also to place strongly interacting VMs on low cost links. The preferable solution from an energy point of view for couples of VMs characterized by an intense traffic exchange is to be placed on the same physical host, and secondarily on low cost links (e.g., within the same POD). Other strategies to save energy due to network traffic exchange are based on the possibility to take advantage of underused links to perform traffic consolidation in few active communication links and network devices [12, 20]. To implement these strategies, the SDDC management needs not only information about the input/output traffic of each VM but also to know which couples of VMs are intensively communicating with each other and/or which VMs are running the different tiers of the same vertical stack of an application. The proposed methodology to infer these information from the aggregated data on the input and output network traffic of each VM is presented in the next section.

3. METHODOLOGY DESCRIPTION

In our reference scenario, the SDDC hosts multi-tier applications. The tiers of each application are organized in vertical stacks that are horizontally replicated and each tier of an application is hosted on a separate VM, as shown in Figure 1. We recall that each VM communicates only with other VMs within the same vertical stack. However, the presence of the request dispatcher leads to similar communication patterns for VMs belonging to the same tier but to different vertical stacks of the same application, thus complicating the identification of communicating VMs based only on the

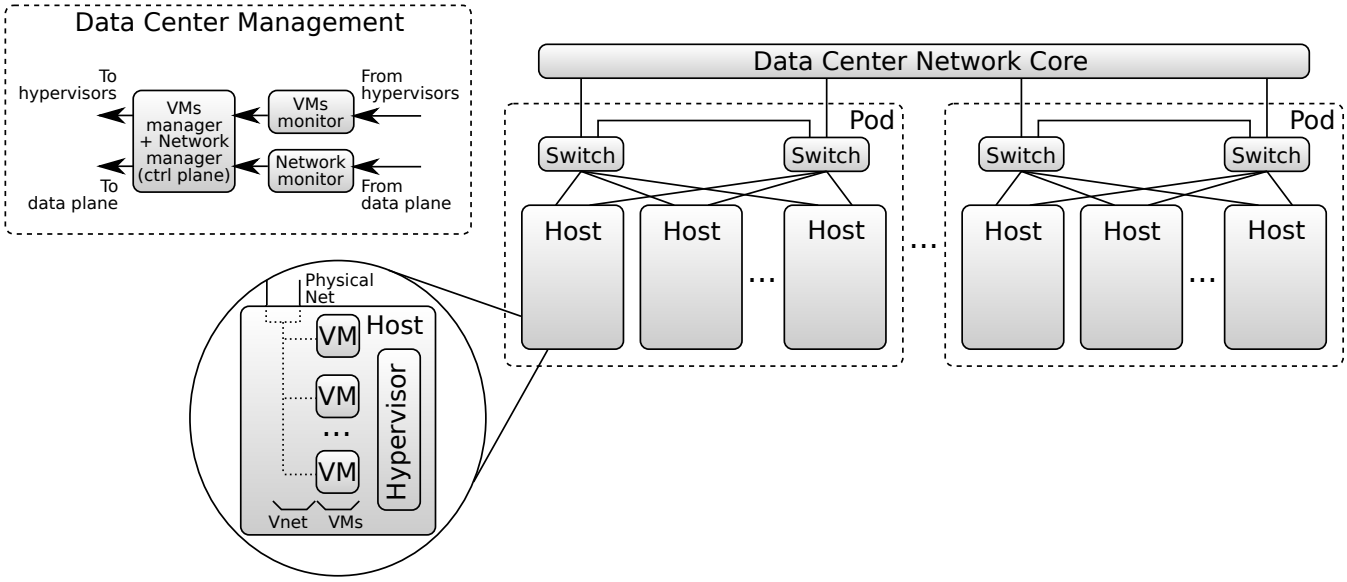


Figure 2: Software-Defined Data Center

input/output packet rate information of each VM.

To support a network-aware VMs allocation in this scenario, the proposed methodology has two main goals. A first goal is to identify couples of VMs exchanging data; this information may be directly used in the VMs allocation problem, for example by adding in the objective function of the underlying optimization problem a term that takes into account the cost of data exchange [6]. A second goal is to achieve a more detailed knowledge about how the deployment of the applications over the data center in order to provide useful input information to higher level VMs allocation approaches; for example, higher availability and survivability of the application can be achieved simply by locating non-interacting vertical stacks of the same application on different pods of the data center.

We now introduce a more formal definition of the basic principles previously sketched. Let us consider a set $\mathcal{N}$ of VMs. We define $P_{j_1}^{out}$ as the time series describing the output packet rate of a generic VM $j_1 \in \mathcal{N}$, and $P_{j_2}^{in}$ as the time series of input packet rate for a VM j_2. We denote as τ the time interval between the samples of the time series. Starting from these time series describing the network activity of the VMs in the data center, we aim to infer which VMs communicate among themselves by considering the correlation in their input and output traffic. We assume that the vertical stacks remain the same (that is, they are not re-organized) during the time series data collection. The proposed methodology to identify communicating VMs is based on the following steps:

1. Interpolation and synchronization of the time series;
2. Computation of a correlation matrix between input and output traffic of couples of VMs;
3. Identification of interacting VMs through correlation-based analysis and clustering.

3.1 Traces interpolation and synchronization

The cloud monitor that collects information about the VMs network utilization is based on the prototype described in [3]. The data collector produces a sequence of tuples in the form `<timestamp, pkt_in, pkt_out>`, where the last two values contain the number of packets received and transmitted since the last data sampling. However, the agents monitoring each VM are not explicitly synchronized. As a result, traces collected from different VMs may be not synchronized.

Preliminary tests with artificially generated traces demonstrate the detrimental effect of not synchronizing traces before applying the correlation analysis of network traffic patterns belonging to different VMs. For this reason, we introduce in our system a trace synchronization step: to achieve this result we rely on data interpolation. Our approach can be summarized as follows:

- we remove samples at the beginning from each time series such that each time series starts in the time interval $[t_0, t_0+\tau]$ and we make sure that each time series contains T samples;
- we compute a synchronization time t_0^* that is the average of the starting times of each trace; from this time we have the interpolated time series timestamps that we define as $t_0^*, t_0^* + \tau, \ldots, t_0^* + i\tau, \ldots, t_0^* + T\tau$;
- for each time series $P_{j_1}^{out}$, we define a synchronized time series $P_{j_1}^{*out}$ using cubic interpolation. In our prototype the implementation of the cubic interpolation is provided by the python *Pandas*[2] framework. A similar procedure is carried out also for the time series of inbound packets $P_{j_2}^{in}$.

Figure 3 provides an example of the above described process. We start with two time series, represented with squares and circles in the upper part of the figure. For both time series we have an interval of τ between two consecutive samples. In the middle part of the figure we show a spline interpolation of the samples that creates a continuous space of values for each possible sampling instant. Finally, the bottom part of the figure shows the re-sampling of the time series. The output is new couple of time series (again represented with squares and circles) where the sampling instants are synchronized.

As a simplified notation, in the following of the paper we will use $P_{j_1}^{*out}(i)$ and $P_{j_2}^{*in}(i)$ for the i-th sample of a synchronized time series.

[2]http://pandas.pydata.org/

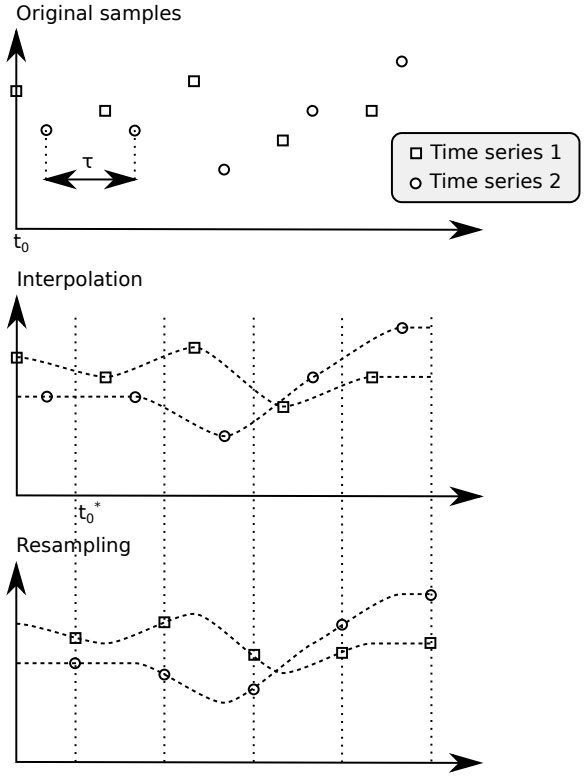


Figure 3: Interpolation and synchronization

3.2 Correlation matrix

Starting from the synchronized time series we can compute the correlation matrix $\mathbf{C}$ between input and output packet rates of each pair of VMs. Our choice is to focus on the correlation between output packet of a VM and input packet of another, to identify data exchange flows from one VM to another. This approach is consistent with other approaches that take into account data exchange to optimize VM communication such as [19, 20] where not just the amount of data exchanged but also the direction of this data exchange is taken into account. Specifically, we define the generic element c_{j_i,j_2} of the matrix as:

$$c_{j_i,j_2} = \mathrm{Cor}(P_{j_1}^{*out}, P_{j_2}^{*in}) \tag{1}$$

where j_1 and j_2 are two generic VMs and $\mathrm{Cor}(\cdot)$ is a function that computes the correlation among two different time series. In literature multiple functions have been proposed to measure the correlation between data series. It is worth to note that this approach leads to the creation of a correlation matrix that may not be symmetric because we may have a strong correlation of data exchange from VM i to VM j (e.g., if VM i sends large amount of data to VM j) but low correlation in the opposite direction (in the previous example the main data flow from VM j to VM i could consist of just ACK messages). In our analysis we consider two different alternatives, that are the *Pearson* and the *Spearman* correlation functions [17]. The first is the most widely used solution to compare time series, while the second emerged in preliminary results as one of the most interesting alternatives in terms of ability to discriminate groups of time series characterized by similar long-term trends, as is the case of our experiments.

We recall that the Pearson correlation index ρ is defined as:

$$\rho(P_{j_1}^{*out}, P_{j_2}^{*in}) = \frac{E[(P_{j_1}^{*out} - \mu(P_{j_1}^{*out}))(P_{j_2}^{in} - \mu(P_{j_2}^{*in}))]}{\sigma(P_{j_1}^{*out})\sigma(P_{j_2}^{*in})} \tag{2}$$

where $\mu(\cdot)$ is the mean value of a time series, $\sigma(\cdot)$ is the standard deviation, and $E[\cdot]$ is a shorthand for the average function (in equation 2 the average is used to compute the covariance of the two time series).

The Spearman correlation index ρ_s corresponds to the correlation between two time series where the original values have been substituted by their ranks in the time series:

$$\rho_s(P_{j_1}^{*out}, P_{j_2}^{*in}) = 1 - \frac{6\sum_{i=0}^{T} r(P_{j_1}^{*out}(i)) - r(P_{j_2}^{*in}(i))}{T(T^2-1)} \tag{3}$$

where T is the number of samples in the time series and $r(\cdot)$ is the rank of a sample among the other samples of the same time series. Switching from the values to the ranks in a time series increases significantly the capacity of this function to discriminate the small fluctuations in the packet rate that may place apart two time series with the same long-term behavior. This is a promising solution to cope with the characteristics of our scenario, where we assume to have a similar workload on every element of the replicated vertical stacks of the considered application.

3.3 Identification of interacting VMs

The final step of our proposal is the definition of couples/groups of interacting VMs based on the correlation matrix obtained in the previous step.

This step can either focus on identifying single couples of VMs that intensively exchange network traffic or focus on providing a broader view of the system grouping together VMs that belong to the same vertical stack of an application (as in Figure 1).

If we consider the first task, that is identifying couples of interacting VMs, the most straightforward approach is to apply a threshold on the correlation matrix to distinguish between *correlated* and *uncorrelated* time series of network resource utilization. Basically, we consider as correlated every couple of time series with a correlation value higher or equal than a defined threshold, while if the correlation value is lower than the threshold, we assume the corresponding time series to be uncorrelated.

On the other hand, if our goal is to identify the vertical clusters of VMs running tiers of the same application, we need a more complex approach aiming to cluster VMs that show a similar behavior in terms of network traffic patterns. The correlation matrix is considered as an *affinity matrix* between VMs and a clustering algorithm is applied to group together the VMs. Supporting an input in the form of an affinity/distance matrix (instead of a feature vector) is a critical point in the choice of the clustering algorithm. Specifically, we take into account three possible solutions: *spectral clustering* [13], *affinity propagation* [11], and *agglomerative clustering* [9].

The spectral clustering algorithm computes the Laplacian operator from the input similarity matrix. The eigenvalues and eigenvectors of the Laplacian are then used to extract a new coordinate system that is fed into a k-means clustering phase [13]. Affinity propagation is a fast clustering approach that aims to identify cluster representatives by simulating the exchange of messages among data points. Once the cluster representatives are found, clustering is carried out by assigning to each representative the most affine other elements. Finally, the agglomerative clustering is a hierarchical clustering algorithm that starts with each VM in its own cluster

and, then, at each iteration merges the two most affine clusters (we use the average distance between elements of the two clusters as a representation of the cluster affinity, but preliminary experiments show that other metrics such as the minimum of the maximum distance provides similar results). The resulting *dendogram* is then cut depending on the number of clusters we want to obtain.

It is worth to note that all the considered algorithms can automatically determine the number of clusters to use, either directly (as for the affinity propagation clustering that includes the estimation of the number of clusters in the algorithm) or indirectly (for example, by performing a spectral gap analysis for the spectral clustering algorithm by adding constraints on the cluster size to determine when the agglomerative clustering stops). For all the three algorithms, we rely on their implementation in the *SciKit-Learn* library[3].

4. EXPERIMENTAL RESULTS

4.1 Experimental setup

We test our infrastructure using an experimental setup where multiple copies of the TPC-W benchmark[4] are executed in parallel over a cloud data center hosted by the Amazon EC2 infrastructure[5]. The monitoring is based on the architecture described in [3]. Our traces refer to a TPC-W run with 12 VMs divided into 4 vertical stacks for a period of 12 hours, with samples collected by default every 30 seconds (but we present also experiments with traces where data collection has a granularity of 1 and 2 minutes).

As metrics for the comparison of the different correlation functions, we distinguish between the two goals of identifying the interacting couples of VMs and clustering together VMs to identify the vertical stacks.

For the first goal we consider the classic measures of classification problems, that are *precision*, *recall*, and *F-measure*. *Precision* is defined as the fraction of identified couples of VMs that are actually communicating, while *recall* is the fraction of communicating couples of VMs that are correctly identified. In terms of true/false positives/negatives, we can define the precision $P = \frac{TP}{TP+FP}$, and the recall $R = \frac{TP}{TP+FN}$. F-measure is the harmonic mean of precision and recall, that is: $F = 2\frac{P \cdot R}{P+R}$.

For the clustering, we consider the clustering *purity* [2] as the main metric to compare different solutions. Purity, that is one of the most popular metrics for cluster evaluation, considers the fraction of correctly identified VMs as the measure of the clustering performance. Specifically, purity is defined by comparing the generic final solution $\mathbf{S}$ of the VM clustering with the ground truth vector $\mathbf{S}^*$, which represents the correct classification of the VMs into the vertical stacks. Purity is thus defined as:

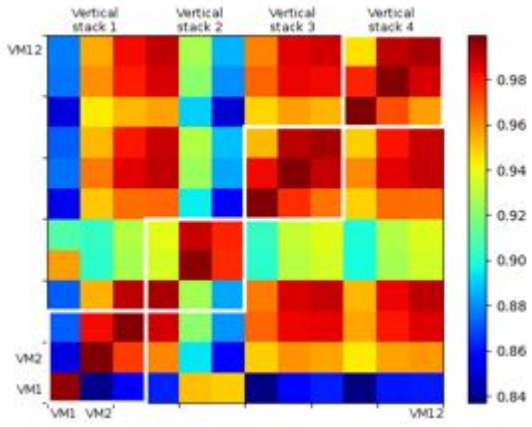
$$purity = \frac{|\{s^j : s^j = s^{j*}, \forall j \in \mathcal{N}\}|}{|\mathcal{N}|}$$

where s^j is the cluster to which VM j is assigned in the considered solution, s^{j*} is the *correct* classification of VM j, and $\mathcal{N}$ is the set of clustered VMs.

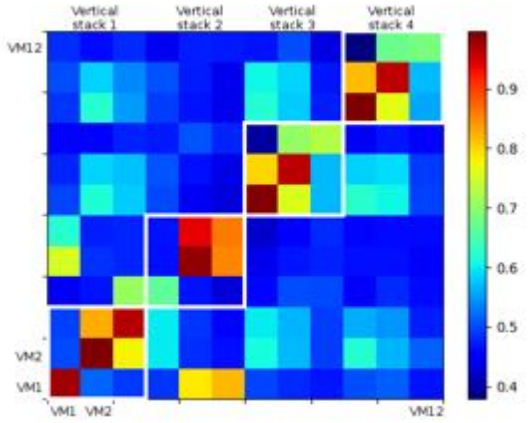
4.2 Comparison of correlation indexes

We first present the correlation matrices computed over every couple of time series using both Pearson and Spearman correlation indexes.

[3]http://scikit-learn.org/
[4]www.tpc.org/tpcw/
[5]https://aws.amazon.com/ec2/



(a) Pearson correlation index



(b) Spearman correlation index

Figure 4: Heatmap of the correlation matrices

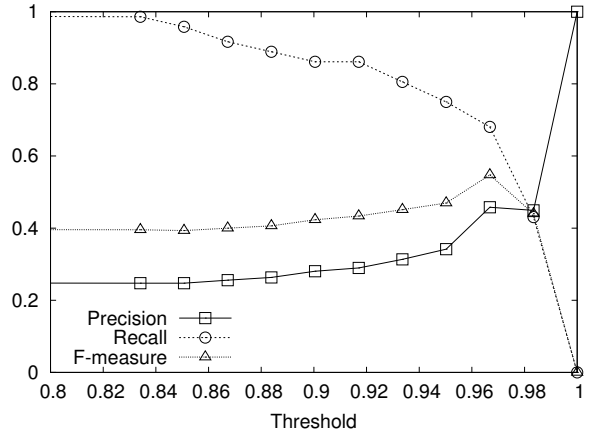
Figure 4 represents the two correlation matrices represented as heat maps. For each VM (numbered 1 to 12 in the figure) we consider the two time series of input and output packets. Each picture presents also an outline of the vertical stacks (shown as four white square frames on the main diagonal of the matrix). Ideally the correlation should present high values within the four squares delimiting each vertical stack and low values outside.

A qualitative analysis of Figures 4a and 4b provides some interesting insights: we observe clearly that the Pearson correlation index (Figure 4a) tends to return higher values (as testified by the general predominance of reddish colors). Particularly critical is the presence of red-orange shades far from the main diagonal (and outside from the vertical stack frames), that suggests high correlation also between VMs belonging to different vertical stacks. On the other hand, the Spearman correlation index (Figure 4b) provides a clearer distinction between groups of VMs in the same vertical stack (the red areas close to the diagonal) and VMs belonging to different stacks, suggesting a better capacity to distinguish communicating from not interacting VMs.

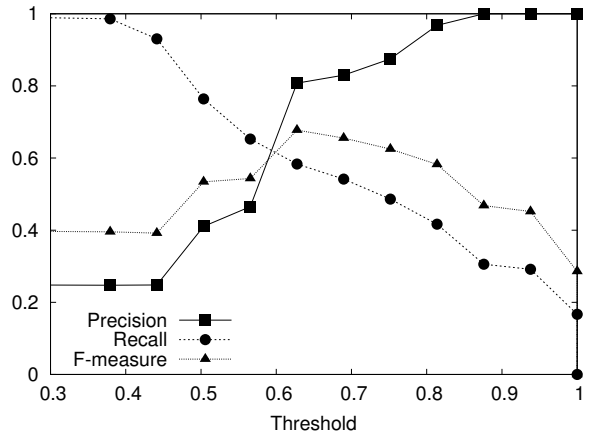
4.3 Identification of interacting VMs

We now consider how the two considered correlation indexes

can be used in a threshold-based identification of the interacting VMs couples. In particular, we evaluate sensitivity of the VMs classification with respect to the threshold value.



(a) Pearson correlation index



(b) Spearman correlation index

Figure 5: Precision, Recall, and F-measure

Figure 6 summarizes this comparison showing the accuracy for both correlation indexes as a function of the threshold value. It is clear that the Spearman correlation index provides a twofold advantage. First, it achieves better performance than the Pearson index, with significantly higher maximum F-measure values. Second, it provides more stable performance with a range of threshold values returning an accuracy higher than 0.5 that spans from 0.5 to 0.8, while the best values for the Pearson function are limited in a single peak around 0.96.

4.4 Clustering of VMs

A further analysis concerns the case where the correlation indexes can be used to create the input affinity matrix for the different matrices

Table 1: Clustering purity

Clustering algorithm	Spearman	Pearson
Spectral clustering	0.75	0.66
Affinity propagation	0.50	0.42
Agglomerative clustering	0.38	0.38

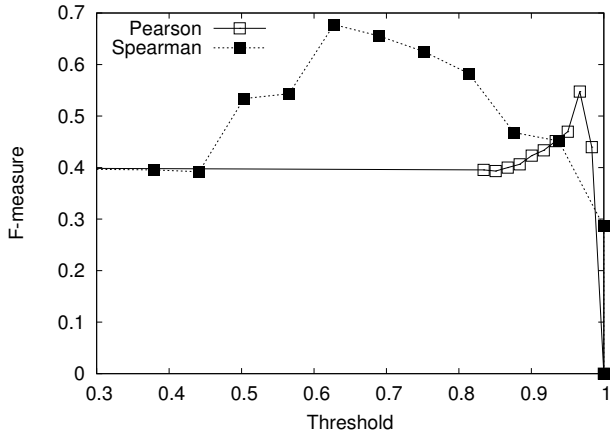


Figure 6: F-measure comparison

Table 1 provides an insight on the clustering purity of the three considered clustering algorithms for each of the two correlation indexes taken into account. The message from the table is rather straightforward, concerning both the clustering algorithms and the correlation indexes. If we observe the different algorithms with affinity input matrices based on the two correlation indexes, it is evident that the Spearman index leads to significantly higher clustering purity with respect to the alternative. Furthermore, if we compare the three considered clustering algorithms, we observe that the spectral clustering algorithm clearly outperforms the other two alternatives.

If we consider the reasons for the performance of the different clustering algorithms, we may refer to the example in Figure 7. It shows an example of clustering output (on the top – the example refers to the spectral clustering algorithm) against the correct clustering (bottom). We observe two types of errors: VMs 1 and 4 are swapped (each is assigned to the cluster where the other belongs), while VM 12 is simply misplaced. For the affinity propagation algorithm the poor clustering performance are reduced by the presence of an additional problem, that is a wrong estimation in the number of clusters. Finally, for the agglomerative clustering, we observe a large number of cluster swaps between VMs (as in the case of VMs 1 and 4 in the figure).

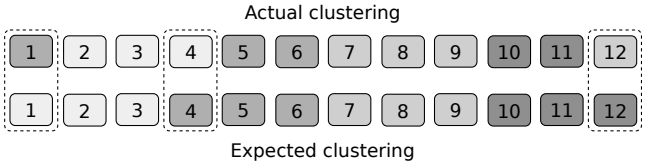


Figure 7: Clustering example

4.5 Comparison of time sampling intervals

As a final analysis, we evaluate how the data sampling granularity affects the quality of the classification of correlated and non-correlated time series and of the VMs clustering. We start our analysis focusing on the classification problem. To this aim, Figure 8 compares the F-measure achieved by the classification based on the Spearman correlation index as the sampling period τ of the network utilization ranges from 30 seconds to 2 minutes. We observe that, even with this small change in the sampling granularity, the impact on the classification F-measure is very significant. Specifically, we

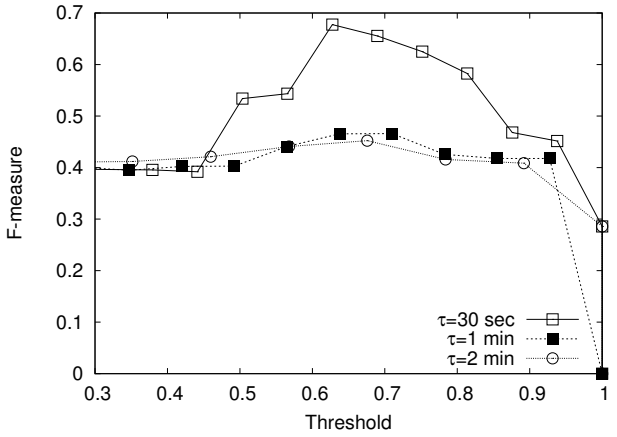


Figure 8: Sensitivity to sampling period τ

observe two main effects of a coarse-grained data collection. First, the sensitivity of the Spearman correlation to the threshold value is further reduced if we consider a large sampling period. This higher robustness is an effect of the smoothed time series: basically, we work with a time series where only the most evident fluctuations are taken into account. This means that the low-frequency and highly evident fluctuations, that are the only way to distinguish two vertical stacks, are easier to identify even over a wider range of threshold values, thus explaining the observed stability with respect to different threshold values. Second, if we consider the threshold range related to the higher F-measure ([0.6-0.8]), we observe that the accuracy tends to decrease dramatically as the sampling interval increases. This effect is rather intuitive, because by increasing the sampling period, we obtain a smoother time series of network resource utilization. This effect still captures the main effects of correlation, but the high frequency fluctuations that allow us to distinguish two vertical stacks are lost, with a consequent reduction of the classification performance.

Table 2: Clustering purity

τ	**Spectral clustering**	**Affinity propagation**	**Agglomerative clustering**
$\tau = 30$ sec	0.75	0.50	0.38
$\tau = 1$ min	0.50	0.27	0.38
$\tau = 2$ min	0.50	0.27	0.33

The second analysis concerns the impact of sampling frequency on the clustering purity. To this aim, Table 2 compares the clustering purity for the three considered clustering algorithms when the Spearman correlation index is used. The table provides two clear confirmations. On one hand, it confirms the better performance of the spectral clustering over the other algorithms, with a purity that is 97% to 31% higher compared with the alternatives. On the other hand, we observe that, even for the spectral clustering algorithm, the increase of the sampling period τ has a major detrimental effect, with the purity reduced by 33% as τ grows from 30 seconds to 1 minute. This purity reduction is caused by the same reasons that determine a reduction of the F-measure for the identification of interacting VMs, that is the smoother time series makes more difficult to distinguish between different vertical stacks.

5. CONCLUSIONS

In this paper we analyzed the problem of identifying groups of VMs in a cloud infrastructure that exchange information without having access to a detailed model of the data transfer among them. In particular, we focus on the case of horizontally replicated vertical stacks, where each VM in a vertical stack corresponds to a tier of an application. We pointed out that this case is very common and challenging in a cloud scenario where scalability is achieved though replication.

We described a methodology to discover communicating VMs that is based on the analysis of correlation between the time series of network traffic of each VMs and on clustering algorithms for identifying the vertical stacks of VMs. We compare different correlation indexes to identify the most suitable alternative: our experiments show that the use of ranking-based techniques (as in the Spearman correlation index) clearly outperforms traditional correlation metrics, such as the Pearson index. Furthermore, we compare three clustering algorithms to identify the most suitable alternative for detecting the vertical stack: our experiments demonstrate that spectral clustering far outperforms the alternatives. Finally, we perform a sensitivity analysis demonstrating that fine-grained network data collection is essential to achieve high accuracy in the identification of communicating VMs.

Acknowledgement

The authors acknowledge the support of the University of Modena and Reggio Emilia through the project *S^{2}C: Secure Software-defined Cloud*. The authors also thank Davide Ferrari for his help in conducting the experiments.

6. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[2] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. A Comparison of Extrinsic Clustering Evaluation Metrics Based on Formal Constraints. *Journal of Information Retrieval*, 12(4):461–486, Aug. 2009.

[3] M. Andreolini, M. Colajanni, and M. Pietri. A scalable architecture for real-time monitoring of large information systems. In *Proc. IEEE Symposium on Network Cloud Computing and Applications*, London, UK, Dec. 2012.

[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. *ACM SIGCOMM Computer Communication Review*, 41(4):242–253, 2011.

[5] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[6] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya. Energy-efficient data replication in cloud computing datacenters. *Cluster Computing*, 18(1):385–402, 2015.

[7] C. Canali and R. Lancellotti. Exploiting ensemble techniques for automatic virtual machine clustering in cloud systems. *Automated Software Engineering*, 21(3):319–344, Sept 2014.

[8] C. Canali and R. Lancellotti. Exploiting Classes of Virtual Machines for Scalable IaaS Cloud Management. In *Proc. of the 4th Symposium on Network Cloud Computing and Applications (NCCA)*, Jun. 2015.

[9] W. H. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of classification*, 1(1):7–24, 1984.

[10] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, 17(2):20–27, 2013.

[11] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[12] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proc. of 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, San Jose, California, 2010.

[13] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec. 2007.

[14] A. Marotta and S. Avallone. A Simulated Annealing Based Approach for Power Efficient Virtual Machines Consolidation. In *Proc. of 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015.

[15] C. Mastroianni, M. Meo, and G. Papuzzo. Probabilistic Consolidation of Virtual Machines in Self-Organizing Cloud Data Centers. *IEEE Transactions on Cloud Computing*, 1(2):215–228, 2013.

[16] X. Meng, V. Pappas, and L. Zhang. Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In *Proc. of the 29th Conference on Information Communications (INFOCOM)*, San Diego, California, USA, Mar. 2010.

[17] L. Myers and M. J. Sirois. *Spearman Correlation Coefficients, Differences between*. John Wiley & Sons, Ltd, 2014.

[18] M. Shojafar, C. Canali, R. Lancellotti, and E. Baccarelli. Minimizing computing-plus-communication energy consumptions in virtualized networked data centers. In *Proc. of 21st IEEE Symposium on Computers and Communications (ISCC)*, Messina, Italy, Jun. 2016.

[19] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra. Starling: Minimizing Communication Overhead in Virtualized Computing Platforms Using Decentralized Affinity-Aware Migration. In *Proc. of 39th International Conference on Parallel Processing (ICPP)*, San Diego, CA, Sept 2010.

[20] Y. Zhang and N. Ansari. Hero: Hierarchical energy optimization for data center networks. *IEEE Systems Journal*, 9(2):406–415, 2013.

A Markov Reward based Resource-Latency Aware Heuristic for the Virtual Network Embedding Problem

Francesco Bianchi
Department of Civil Engineering and Computer Science Engineering
University of Rome "Tor Vergata", Italy
f.bianchi@ing.uniroma2.it

Francesco Lo Presti
Department of Civil Engineering and Computer Science Engineering
University of Rome "Tor Vergata", Italy
lopresti@info.uniroma2.it

ABSTRACT

An ever increasing use of virtualization in various emerging scenarios, e.g.: Cloud Computing, Software Defined Networks, Data Streaming Processing, asks Infrastructure Providers (InPs) to optimize the allocation of the virtual network requests (VNRs) into a substrate network while satisfying QoS requirements. In this work, we propose MCRM, a two-stage virtual network embedding (VNE) algorithm with delay and placement constraints. Our solution revolves around a novel notion of similarity between virtual and physical nodes. To this end, taking advantage of Markov Reward theory, we define a set of metrics for each physical and virtual node which captures the amount of resources in a node neighborhood as well as the degree of proximity among nodes. By defining a notion of *similarity* between nodes we then simply map virtual nodes to the most similar physical node in the substrate network. We have thoroughly evaluated our algorithm through simulation. Our experiments show that MCRM achieves good performance results in terms of blocking probability and revenues for the InP, as well as a high and uniform utilization of resources, while satisfying the delay and placement requirements.

Keywords

Cloud computing; Markov Reward Process; Network virtualization; Quality of Service; Virtual Network Embedding (VNE)

1. INTRODUCTION

The base for the future Internet is characterized by the Infrastructure as a Service (IaaS) service model [11], producing a decoupling of the ISP (Internet Service Provider) into two novel players: the Infrastructure Provider (InP) and the Service Provider (SP). The InPs (e.g., Cloud providers), who own and manage the infrastructure, rent out parts of the network resources to the SPs (e.g., Cloud users), who in turn meet the end-users demands, making virtual networks (VNs) and providing end-to-end services [10, 11]. In this flexible multi-layer architecture, the InPs have the challenging task to manage the substrate network efficiently in order to maximize the number of mapped virtual network requests (VNRs) along with the revenue. In practice, the InPs have to determine, on-line, a subset of physical nodes and links with sufficient resources to host each VNR composed by a set of virtual nodes and links with a certain amount of resource attributes, e.g., computational capacity, link bandwidth.

This problem is referred to as Virtual Network Embedding (VNE) which is a well known NP-complete problem [1, 23]. In order to support on-line operations, many heuristics have been proposed in the literature (e.g., [25, 17, 23, 9, 8, 24, 20, 16]) under different settings and assumptions.

In this paper, we study the VNE problem under maximum latency QoS constraints. Our goal is to determine a set of node and link resources in a substrate network which satisfy end-to-end latency constraints between pair of nodes. This is motivated by the growing number of multimedia and other delay-sensitive applications (Cisco envisioned that about 90% of Internet traffic is related to delay-sensitive applications [14]). Our contributions are as follows:

- Building on our prior work on VNE [6, 5], we present a new algorithm, for the VNE problem with QoS constraints (e.g., latency), called MCRM (Markov Chain Reward Metrics). MCRM is inspired by [13, 8, 24] which devised so called *coordinated* two-stage VNE algorithms: in the first stage the algorithm embeds virtual nodes into physical (also called substrate) nodes; then, in the second stage, the algorithm embeds the logical links into physical paths between the nodes.

- MCRM accounts for computational capacity and link bandwidth requirements, delay requirements over the virtual links as well as the possibility of pinned nodes, that is, virtual nodes which must be assigned to specific physical nodes. The latter constraints may arise in several scenarios including the physical location of sensor nodes, specific hardware availability, etc. To the best of our knowledge, this is the first *coordinated* two-stage VNE algorithm to account for all these constraints in a single algorithm.

- We propose a novel node mapping strategy which revolves around the notion of similarity between virtual and physical nodes. To this end, we define a set of metrics for each physical and virtual node which captures the amount of available (demanded) resources in the neighborhood of a physical (virtual) node as well as their degree of proximity with respect to the other nodes. By properly defining a notion of *similarity* between nodes we then simply map virtual nodes to the most *similar* physical node in the substrate network.

- Differently from prior approaches, our metrics are based on the accumulated reward of suitable Markov Chains. The intuition is that the accumulated reward, with each node reward being properly defined as to characterize the metric under study, well captures the availability of nodes resources in a given neighborhood of a node as well as the topological structure of the graph in that neighborhood.
- We have evaluated our algorithm through simulation. The results show that our new solution is able to achieve a good performance level in terms of number of blocked requests, revenue gained by the InP and average resource utilization while coping effectively with the QoS, e.g., delay constraints.

The rest of the paper is organized as follows. In section 3 we formalize the VNE problem. The new mapping algorithm (MCRM), based on Markov Reward Processes, is presented in section 4. In section 5 we evaluate MCRM through simulation. Section 6 concludes the paper.

2. RELATED WORK

The VNE is a well known and widely investigated problem which has been studied under different settings and assumptions. Since the VNE problem is NP-complete [1, 23] most solutions focus on efficient heuristics (e.g., [25, 17, 23, 9, 8, 24, 20, 16]).

Most heuristics consists of two successive stages, where first all the virtual nodes and then the virtual links are mapped to the physical network. Following this approach, our work is inspired by a recently proposed new class of two-stage VNE algorithms, known as coordinated algorithms (e.g.: CO-VNE [13]). The principal feature of this new category of algorithms (e.g., [13, 8, 24]) is that the information concerning the links, required for the second stage (i.e., link mapping), is also considered in the first one (i.e., node mapping). In particular, these solutions take into account the topology, by associating with each node a metric which captures the amount of the resources (node computational capacity and network link bandwidth) in the neighborhood of a node. The embedding is then executed via a greedy approach, by ranking the virtual and substrate nodes according to this metric and mapping the highly ranked virtual nodes onto the matching highly ranked substrate ones. Noticeably, the majority of these works drew inspiration from the PageRank algorithm [7].

The PageRank algorithm relies on the assumption that the importance of a web page is function of the number and importance of the web pages that link to it: the greater the number of incoming links and the importance of the web pages from which these links originate, the more important the web page is. By interpreting the web surfing as a Random Walk, it turns out that the importance of a web page corresponds to the stationary probability of a Markov Chain induced by the web links themselves, where the pages represent the Markov Chain states and the links the transitions between states. Following the same approach, the aforementioned VNE algorithms associate with each node a metric which captures the relative "importance" of a node in terms of resources, which accounts for the local resources and those of the connected links and the nearby nodes. As for the PageRank algorithm, this metric can be computed as a stationary probability of a suitable Markov Chain.

As explained in the introduction, while we consider a similar approach, we follow a totally different direction in determining the metrics of interest as we consider the discounted cumulative reward of a suitable Markov Chain rather than the stationary probability. Since the cumulative reward can be regarded as the accrued reward of a random walk the proposed approach lends itself to an interesting physical interpretation of the proposed metrics which we believe better capture the relative importance of nodes in terms of available resources in their neighborhood.

Many QoS metrics have been taken into account in solving the VNE problem including energy efficiency [18], survivability and resiliency [19], see [11] for a survey. Constraints on the physical location of nodes are less common. Chowdhury et al. [9] consider a requested location constraint (e.g., using geographic coordinates) for every virtual node and a maximum distance threshold indicating how far from the demanded location can be placed each node. They solve an optimization problem using mixed integer programming.

Most of the few existing works on delay-aware VNE solve the problem using a mixed integer programming formulation, as done by Ivaturi and Wolf in [14]. They treat the VNE as a facility location problem and p-hub median problem, pointing to attain a good balance between substrate utilization and end-to-end delay. They introduce a delay tuning parameter and a specific metric to evaluate the delay savings. Their objective is the optimal solution that minimizes the delay, so it is not suitable for the online operation.

Similar to our approach, Behrouznia et al. in a recent work [3] propose a two-stage QoS-aware algorithm, where the key feature is the adapted computation of quality of the physical paths, introduced in [21]. The objective metrics are cost-effectiveness and compliance with delay, packet loss and node location constraints. In the first phase, the algorithm sorts the substrate nodes according to the metric of the node's available resources (available computational capacity of the node multiplied by the outgoing available bandwidth), then it maps the virtual nodes onto the substrate ones. In the second phase, in order to map each virtual link, the algorithm finds the K shortest paths between each couple of involved mapped virtual nodes (through K-Shortest-Paths algorithm). Then it computes the combined quality index of the links making up the paths that comply with the requested bandwidth and QoS constraints of the relative virtual link. The virtual link is then mapped on the highest quality path. Differently, despite the fact that we also consider VNRs with pinned nodes, we calculate instead a resource delay-aware metric for both the request and substrate network and a *proximity* metric, which properly combined to define a measure of *similarity*, allows also to reduce the length of the physical paths. However, as we showed in our previous work [5], this algorithm achieves lower performance compared to our previous solution. For this reason, we do not report again the results. We do not compare explicitly the performance of our new solution with our previous one, but employing the new solution allows to improve even more the previous results on comparable scenarios (e.g., VNRs with no pinned nodes).

Beck and Linnhoff-Popien in [2] discuss the communication delay in the context of VNE. They introduce a delay model based on queueing theory and consider latency constraints for every virtual link. They present optimization

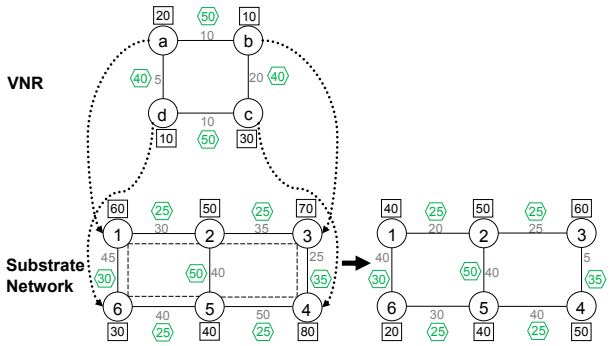


Figure 1: A VNR, the substrate network and a feasible mapping of the VNR with the deriving updated substrate network. Numbers inside rectangles near the nodes (circles), those near the edges (solid lines) and those inside green hexagons, represent the computational capacity, the link bandwidth and the link latency, respectively. Dotted lines and dashed lines show the node mapping and the consequent link mapping, respectively.

objectives for delay-aware embedding algorithms and introduce delay related evaluation metrics in order to evaluate the effectiveness of delay-aware algorithms. However, they do not devise their own algorithmic solution.

So far we have presented centralized solutions. Some approaches present distributed solutions. Shengquan et al. in [22] present a heuristic distributed online algorithm that decomposes the VN, and the resulting fragments are mapped making use of agents that communicate between themselves and with the central entity. Their goal is to reduce the bandwidth cost while keeping the average path delay within the constraint. The considered link delay varies proportionally according to the bandwidth use. The non-latency-aware algorithm used to compare their solution, reaches better results in terms of delay, even though both solutions meet the delay constraint.

3. VNE: PROBLEM DESCRIPTION

3.1 Substrate Network

We model the substrate network as an undirected graph, weighted on both nodes and edges, $G^s = (N^s, E^s, C^s, B^s, L^s)$, where N^s is the set of substrate nodes, E^s is the set of substrate links, C^s is the set of available CPU capacity associated with each node[1], B^s is the set of available bandwidth associated with each link and L^s is the set of the network delay associated with each link. We denote the set of substrate paths, without cycles, with PT^s. Figure 1 reports an example of substrate network where the numbers inside the rectangles denote the available node CPU capacities, those near the edges the link available bandwidth, and the numbers inside green hexagons the link latency.

3.2 Virtual Network Request

We model a virtual network request as an undirected graph, weighted on both nodes and edges, $G^v = (N^v, E^v, C^v, B^v, L^v, P^v, P^s)$, where N^v is the set of virtual nodes, E^v is the set of virtual links, C^v is the set of requested CPU capacity associated with each node, B^v is the set of requested bandwidth associated with each link, L^v is the set of the network delay constraints associated to each link, $P^v \subseteq N^v$ is the optional set of virtual pinned nodes and $P^s \subseteq N^s$ is the optional set of requested physical pinned nodes to embed the former ones. We will also use $VNR(t_a, t_d)$ to denote a VNR with arrival time t_a and lifetime t_d. Figure 1 shows a VNR made up of four nodes and four links with the same notation of the substrate network about the requested capacities and latencies. Notations are reported in Table 1.

[1]We only consider the computational capacity as node resource metric. The algorithm can be extended in case of multiple resources, *e.g.*, memory, I/O bandwidth

Table 1: Notations

Notations	Meaning
G^s	Substrate network
N^s	Set of substrate nodes
E^s	Set of substrate links
C^s	Set of available CPU capacities
B^s	Set of available bandwidth capacities
L^s	Set of latencies
PT^s	Set of substrate paths without cycles
G^v	Virtual network request
N^v	Set of virtual nodes
E^v	Set of virtual links
C^v	Set of CPU requests
B^v	Set of bandwidth requests
L^v	Set of latency constraints
P^v	Set of virtual pinned nodes
P^s	Set of requested physical pinned nodes
t_a	Arrival time of the VNR
t_d	Lifetime of the VNR

3.3 Virtual Network Mapping

Virtual network mapping $M : G^v \rightarrow G^s$ is the embedding of a VNR, G^v, in the physical network G^s. In particular, the set of virtual nodes is embedded into a subset of substrate nodes and the set of virtual links is embedded into a subset of substrate links, namely a subset of the substrate paths PT^s. This procedure is made up of two phases: node mapping and link mapping.

The node mapping phase provides a mapping function from a virtual node to a substrate one, $M_{node} : N^v \rightarrow N^s$, such that: $\forall n^v, m^v \in N^v$

$$M_{node}(n^v) = n^s \in N^s, \quad (1)$$

subject to:

- $M_{node}(n^v) = M_{node}(m^v)$ if and only if $m^v = n^v$;
- $C^v(n^v) \leq C^s(M_{node}(n^v))$.

Each virtual node is mapped into a distinct physical node provided that the latter has at least the same quantity of available resources. In Figure 1 the resulting node mapping for the VNR is $\{a \rightarrow 1, b \rightarrow 3, c \rightarrow 4, d \rightarrow 6\}$, graphically represented with dotted lines.

The link mapping provides a mapping function of a virtual link to a substrate path, with no cycles, consisting of one or more links, $M_{link} : E^v \rightarrow PT^s$, such that: $\forall e^v = (m^v n^v) \in$

E^v

$M_{link}(m^v n^v) =$
$pt^s(M_{node}(m^v), M_{node}(n^v)) \in PT^s(M_{node}(m^v), M_{node}(n^v)),$

subject to:

- $B^v(e^v) \leq \min_{e^s \in pt^s(M_{link}(e^v))} B^s(e^s);$
- $L^v(e^v) \geq \sum_{e^s \in pt^s(M_{link}(e^v))} L^s(e^s).$

Each substrate link in the path must possess at least the same quantity of resources of the virtual link. Furthermore, the overall latency of each physical path must comply with the requested virtual link's latency constraint. In Figure 1 the resulting link mapping for the VNR is $\{(ab) \rightarrow (12, 23), (bc) \rightarrow (34), (cd) \rightarrow (45, 56), (da) \rightarrow (61)\}$, denoted by dashed lines.

3.4 Objective

The goal of the InP is to maximize the revenue, while optimizing resource utilization. The SP, instead, wants to pay just for the agreed resource costs. Following previous approaches [23, 25, 9, 24, 13], we define the revenue $Rev(G^v, t_a, t_d)$ for the InP for accepting a VNR with lifetime t_d, at time t_a, as:

$$Rev(G^v, t_a, t_d) = \begin{cases} Rev_0(G^v, t_a, t_d) \cdot t_d, & \text{if accepted} \\ 0, & \text{otherwise} \end{cases}, \quad (2)$$

where $Rev_0(G^v, t_a, t_d)$ represents the revenue per time unit for $VNR(t_a, t_d)$ that can be defined as:

$$Rev_0(G^v, t_a, t_d) = \alpha_c \sum_{n^v \in N^v} C^v(n^v) + \alpha_b \sum_{e^v \in E^v} B^v(e^v), \quad (3)$$

where α_c is the unit price for the computational resources and α_b is the unit price for the bandwidth resources.

Generally, as highlighted in other works [23, 9, 8], the main objective for the InP is to maximize the long term time-average revenue:

$$Rev_{tot} = \lim_{T \to \infty} \frac{\sum_{i=1}^{N_T} Rev(G^{v,(i)}, t_a^{(i)}, t_d^{(i)})}{T}, \quad (4)$$

where N_T is the number of requests arrived within time T, and $Rev(G^{v,(i)}, t_a^{(i)}, t_d^{(i)})$ is the revenue gained from the i-th VNR with arrival time $t_a^{(i)}$ and lifetime $t_d^{(i)}$.

An additional important index, which has an impact on the former one, is the blocking ratio, namely the blocking probability:

$$BR = \lim_{T \to \infty} \frac{\sum_{j=1}^{NREJ_T} VNR_{rej}^{(j)}(t_a^{(j)}, t_d^{(j)})}{\sum_{i=1}^{N_T} VNR^{(i)}(t_a^{(i)}, t_d^{(i)})}, \quad (5)$$

where $NREJ_T$ is the total number of rejected VNRs within time T and $VNR_{rej}^{(j)}(t_a^{(j)}, t_d^{(j)})$ is the j-th rejected VNR.

Furthermore, as suggested in [2], we make use of the average path delay of each accepted VNR as a key metric whose meaning is how well a VNR performs after it has been mapped into the physical network.

4. MARKOV CHAIN REWARDS BASED LATENCY AWARE ALGORITHM

Similarly to previous approaches in the literature [8, 24, 13], our VNE algorithm (MCRM) comprises two distinct, sequential stages. In the first stage, the virtual pinned nodes are first assigned to the specified physical nodes; then, the virtual unpinned nodes are mapped to physical nodes with adequate resources. Then, in the second stage, links are mapped to physical paths with sufficient link capacity.

The main idea behind our node mapping strategy revolves around a novel notion of *similarity* between virtual and physical nodes. To this end, we define a set of metrics for each physical and virtual node which captures the amount of available (demanded) resources in the neighborhood of a physical (virtual) node as well as their degree of *proximity* with respect to the other nodes. By properly defining a notion of *similarity* between nodes we then simply map virtual nodes to the *most similar* physical node in the substrate network.

We define our metrics in subsection 4.1-4.2. We first present the MCRR-LA (Markov Chain Reward Resources-Latency Aware) metric. It is a latency aware metric based on Markov Chains with rewards model [12], which captures the amount of resources (computational capacity, network bandwidth) available in a node and its surrounding area coupled with the network latency between nodes. In subsection 4.2, we present the second type of metric, MCR-P (Markov Chain Reward-Proximity). This metric represents how close a node is with respect to a reference node. Then, in subsection 4.3, we define the node similarity metric. Finally, in subsection 4.4 we present the node and link mapping algorithms in detail.

4.1 MCRR-LA Metric

As observed in [13], a node ranking metric for the VNE problem should account for both local and neighboring nodes resources. Intuitively, the greater the amount of resources in terms of computing and network capacity in a node neighborhood, the more likely a virtual network can be embedded in that portion of the physical network. Building on this idea, in addition to computational and network resources which has been previously considered in the literature, we also consider the link latency in computing the resource aware metric by introducing a damping factor proportional to the link latency.

Following an approach similar to the one proposed by Zhang et al. in [24][2], we define the amount of available resources $CBL(n)$ of a node n by the product of the available CPU resource and the sum of the bandwidth resources, each multiplied by the normalized link latency as follows:

$$CBL(n) = C^s(n) \cdot \sum_{m \in N(n)} \left[B^s((n,m)) \cdot \left(\frac{max\_lat - L^s((n,m))}{max\_lat - min\_lat} \right) \right], \quad (6)$$

where $N(n)$ is the set of neighbors of node n. In (6), $L^s((n,m))$ is the latency over the physical link between nodes n and m, and $[min\_lat, max\_lat]$ is the range of latency values in the network. The latency term in CBL is

[2]The ranking metric can be similarly defined using alternative resource metrics, *e.g.*, [13].

thus normalized in the interval $[0,1]$, with 1 corresponding to links with minimum latency and 0 to links with maximum latency.

The node resources are then normalized as follows:

$$Res(n) = \frac{CBL(n)}{\sum_{m \in N^s} CBL(m)}. \quad (7)$$

We define the MCRR-LA ranking metric $V_\gamma(n)$ of node n, recursively, as a weighted sum of the local resources and the neighbouring nodes metric as follows

$$V_\gamma(n) = (1-\gamma)Res(n) + \gamma \sum_{m \in N(n)} \frac{Res(m)}{\sum_{h \in N(n)} Res(h)} V_\gamma(m), \quad (8)$$

with $\gamma \in [0,1)$ the relative weight of the neighbours metric. In (8), each neighbour contributes to the sum in proportion to its amount of resources with respect to the other adjacent nodes (represented by the factor $\frac{Res(m)}{\sum_{h \in N(n)} Res(h)}$). To rewrite (8) in a compact form, let $\mathbf{P}$ a $|N| \times |N|$ matrix defined as follows:

$$\mathbb{P}(n,n') = \begin{cases} \frac{Res(n')}{\sum_{h \in N(n)} Res(h)} & \text{if } (n,n') \in E^s \\ 0 & \text{otherwise} \end{cases}. \quad (9)$$

By construction, $\mathbf{P}$ is a stochastic matrix with all rows summing to 1. We can now conveniently rewrite (8) as:

$$V_\gamma(n) = (1-\gamma)Res(n) + \gamma \cdot \sum_{n' \in N} \mathbb{P}(n,n') V_\gamma(n'), \quad (10)$$

or in matrix form

$$\mathbf{V}_\gamma = (1-\gamma)\mathbf{Res} + \gamma \mathbf{P} \mathbf{V}_\gamma, \quad (11)$$

where $\mathbf{Res} = (Res(1), Res(2), \ldots, Res(|N|))^T$, and $\mathbf{V}_\gamma = (V_\gamma(1), V_\gamma(2), \ldots, V_\gamma(|N|))^T$. $\mathbf{V}_\gamma$ can be calculated as the unique solution of Eq.(11). Indeed, by observing that, since $\mathbf{P}$ is stochastic, $(\mathbf{I} - \gamma\mathbf{P})$, $0 \leq \gamma < 1$, is invertible, we readily obtain

$$\mathbf{V}_\gamma = (\mathbf{I} - \gamma\mathbf{P})^{-1}(1-\gamma)\mathbf{Res}. \quad (12)$$

The recursive equations (11) have an elegant physical interpretation. Indeed, (11) can be regarded as the Bellman equations [4] of the discounted cumulative reward, with discount factor γ, of the Markov Chain with transition probability matrix $\mathbf{P}$ over the set of nodes N, and rewards vector $\mathbf{Rew} = (Rew(1), Rew(2), \ldots, Rew(|N|))$, with $Rew(n) = (1-\gamma)Res(n)$, $n \in N$. In other words, the ranking metric of node n, $V_\gamma(n)$ corresponds to the expected discounted accumulated reward of a Markov Chain with transition probability $\mathbf{P}$, that is

$$V_\gamma(n) = \lim_{k \to \infty} \mathbb{E}_{\mathbf{P}} \left[\sum_{i=0}^{k} \gamma^i Rew(n_i) \right], \quad (13)$$

where $n_0, n_1, n_2, \ldots,$ denotes a sample path with initial state $n_0 = n$. This correspondence indicates that, for each node n, the node ranking $V_\gamma(n)$ is function of the amount of available resources and link latency in the neighborhood of n: the greater the amount of resources and the lower the link latency in a node neighborhood, the greater the value of V_γ. The discount factor γ is a measure of the size of the neighborhood taken into account to determine the node metric: $\gamma = 0$ only the local resources are taken into account, while as γ increases, larger and larger portion of the graph close to a node is accounted for in the metric.

Finally, it is worth noting that while metric $V_\gamma(n)$ has been so far implicitly applied only to the substrate network nodes, it can be applied to the virtual network nodes too. In this latter case, though, instead of the amount of available resources, $V_\gamma(n)$ represents the amount of resources *required* by a virtual node and its neighboring nodes. Hereafter, to avoid confusion, we will distinguish the two cases with a superscript: $V^s(n)$ will denote the MCRR-LA metric for substrate node $n \in N^s$ and $V^v(n)$ to denote the MCRR-LA metric for virtual node $n \in N^v$.

4.2 MCR-P Metric: Node Proximity

The MCRR-LA metric well captures the availability of computational resources as well as network bandwidth and link delay in nodes neighborhoods but it is not well suited to account for the presence of pinned nodes and the relative distance to them. To this end, we find convenient, we dare to say elegant, to complement MCRR-LA with additional Markov Reward Model based metrics which explicitly account for the presence - and position - of pinned nodes as well as for the number of links in the network paths to these nodes.

More specifically, for each node m (we will need the metric only for pinned and already placed nodes but for the sake of definition let us consider all nodes), we define the proximity metric $Z_{m,\gamma}(n)$, $n \in N$, recursively as

$$Z_{m,\gamma}(n) = Rew_m(n) + \gamma \cdot \sum_{n' \in N} \mathbb{P}_u(n,n') Z_{m,\gamma}(n'), \quad (14)$$

where the node n reward $Rew_m(n)$ is defined as

$$Rew_m(n) = \begin{cases} 1 & \text{if } n = m \\ 0 & \text{otherwise} \end{cases},$$

and transition matrix $\mathbf{P_u}$ defined as follows

$$\mathbb{P}_u(n,m) = \begin{cases} \frac{1}{degree(n)} & \text{if } (n,m) \in E^s \\ 0 & \text{otherwise} \end{cases}, \quad (15)$$

where $degree(n) = |N(n)|$ is number of adjacent links of node n. As before, we define this metric for both virtual and physical nodes, in case, we will distinguish the two cases with a proper superscript.

We can rewrite (14) in matrix form

$$\mathbf{Z}_{m,\gamma} = \mathbf{Rew_m} + \gamma \mathbf{P_u} \mathbf{Z}_{\mathbf{m},\gamma}. \quad (16)$$

$\mathbf{Z}_{\mathbf{m},\gamma}$ can be calculated as the unique solution of (16). Again, since $\mathbf{P_u}$ is stochastic, for $0 \leq \gamma < 1$, we readily obtain:

$$\mathbf{Z}_{m,\gamma} = (\mathbf{I} - \gamma \mathbf{P_u})^{-1} \mathbf{Rew_m}. \quad (17)$$

Following the same arguments above, $Z_{m,\gamma}(n)$ corresponds to the discounted accumulated reward of a Markov Chain with transition probability $\mathbf{P_u}$ and reward $\mathbf{Rew_m}$, that is,

$$Z_{m,\gamma}(n) = \lim_{k \to \infty} \mathbb{E}_{\mathbf{P_u}} \left[\sum_{i=0}^{k} \gamma^i \mathbb{1}_{\{n_i = m\}} \right], \quad (18)$$

where $n_0, n_1, n_2, \ldots,$ denotes a sample path with initial state $n_0 = n$ and $\mathbb{1}_{\{.\}}$ is the indicator function. This definition lends itself to a simple physical interpretation: $Z_{m,\gamma}(n)$ is reward accumulated by a random walk with initial state n

and uniform transition probability out of each node and unit reward associated to node m and zero reward elsewhere. By construction, $Z_{m,\gamma}(n)$ thus captures the proximity of node n to node m measured as the expected(discounted) number of visits to node m in a random walk. Intuitively, this metric captures the distance and the graph topology between the two nodes.

4.3 Similarity Measure for Node Mapping

Given the metrics $\mathbf{V}^{\mathbf{v}}_{\gamma}$, $\mathbf{Z}^{\mathbf{v}}_{\mathbf{m},\gamma}$, and $\mathbf{V}^{\mathbf{s}}_{\gamma}$, $\mathbf{Z}^{\mathbf{s}}_{\mathbf{m},\gamma}$ for the virtual and substrate networks, respectively, we can define the *similarity* metric between a virtual and a physical node. First, we normalize each metric:

$$\tilde{\mathbf{V}}_{\gamma} = \frac{\mathbf{V}_{\gamma}}{\max(\mathbf{V}_{\gamma})}, \tag{19}$$

$$\tilde{\mathbf{Z}}_{m,\gamma} = \frac{\mathbf{Z}_{m,\gamma}}{\max(\mathbf{Z}_{m,\gamma})}. \tag{20}$$

Then, given a set of virtual nodes $R^v \subseteq N^v$, and the corresponding set of physical nodes $R^s \subseteq N^s$, that is the set of physical nodes where they have been mapped, $R^s = \{n \in N^s | \exists n^v \in R^v : M_{node}(n^v) = n^s\}$, $R^s = M_{node}(R^v)$ for short, we define the *similarity* between a virtual node n^v and a substrate node n^s with respect the nodes R^v, $R^s = M_{node}(R^v)$ as:

$$D(n^v, n^s; R^v) = \sqrt{\begin{array}{l} w_{V_\gamma} \cdot (\tilde{V}^v_\gamma[n^v] - \tilde{V}^s_\gamma[n^s])^2 + \\ \sum_{m \in R^v} w_{Z_{m,\gamma}} \cdot (\tilde{Z}^v_{m,\gamma}[n^v] - \tilde{Z}^s_{M_{node}(m),\gamma}[n^s])^2 \end{array}}, \quad \forall n^v \in N^v, \forall n^s \in N^s,$$

where $w_{V_\gamma}, w_{Z_{m_1,\gamma}}, \ldots, w_{Z_{m_{|R^v|},\gamma}}$ are non negative weights which sum to 1, $w_{V_\gamma} + \sum_{m \in R^v} w_{Z_{m,\gamma}} = 1$. In this paper, for the sake of simplicity, we will assume the node proximity MCR-P weights to be equal, that is, $w_{Z_m} = \frac{1 - w_{V_\gamma}}{|R^v|}$, $m \in R^v$.

By definition, intuitively, the smaller $D(n^v, n^s; R^v)$, the more similar n^v and n^s are in terms of relative amount of resources and distance with respect to the nodes R^v and $R^s = M_{node}(R^v)$. For the sake of clarity, hereafter, we will refer to the set R^v and $R^s = M_{node}(R^v)$ as the *reference* nodes as they represent the nodes with respect to the proximity metrics are measured when computing the similarity metric D.

4.4 Full Mapping Algorithm

In this section we describe the VNE mapping algorithm, e.g., MCRM. Our VNE algorithm comprises two distinct, sequential stages, namely, the node mapping stage and the link mapping stage. When a VNR arrives, first of all virtual pinned nodes are mapped to the specified substrate nodes and then the remaining virtual unpinned nodes are mapped to suitable physical nodes. Successively, in the link mapping stage, virtual links are mapped to suitable substrate paths. In the following subsections we will describe the algorithms in details.

4.4.1 Node Mapping

Our node mapping algorithm is based on the premise that for each virtual node n^v, the best candidate for node placement is the most *similar* physical node, that is, the physical node $n^{s*}(n^v) \stackrel{\text{def}}{=} \arg\min_{n^s \in U^s} D(n^v, n^s; R^v)$ where U^s denotes the set of not yet assigned physical nodes.

The algorithm works as follows. First, we map the pinned nodes to the specified physical nodes and initialize the two sets of reference nodes to the pinned nodes, that is, $R^v = P^v$ and $R^s = M_{node}(P^v)$. Then, we proceed iteratively by mapping the unpinned virtual nodes till all nodes are mapped. At the i-th iteration: 1) we determine the unpinned virtual node $n^v(i)$ which has the smallest similarity metric with respect to its best candidate physical node, that is, $n^v(i) = \arg\min_{n^v \in U^v} D(n_v, n^{s*}(n_v); R^v) = \arg\min_{n^v \in U^v} \min_{n^s \in U^s} D(n^v, n^s; R^v)$, where U^v denotes the set of not yet mapped virtual nodes; 2) we map the just selected virtual node $n^v(i)$ to to its best candidate physical node $n^{s*}(n^v(i))$, that is,

Algorithm 1 *NodeMapping*

Input : VNR $G^v(N^v, E^v, C^v, B^v, L^v, P^v, P^s)$, Substrate net $G^s(N^s, E^s, C^s, B^s, L^s)$, weight of $V^{v/s}_\gamma$ metrics w_{V_γ};
Output: Mapping of nodes M_{node};

$M_{node} \leftarrow \mathbf{0}$; $R^v \leftarrow \mathbf{0}$; $R^s \leftarrow \mathbf{0}$; $counter \leftarrow 0$;
$nodes\_mapping\_success \leftarrow false$;
\# physical node with max metric value
$sub\_max\_V\_node \leftarrow 0$;
\# virtual node with max metric value
$vnr\_max\_V\_node \leftarrow 0$;
$V^v_\gamma \leftarrow compute\_metric\_V(G^v)$;
$V^s_\gamma \leftarrow compute\_metric\_V(G^s)$;
\# if there are > 0 pinned nodes
if $|P^v| > 0$ **then**
  for $i \leftarrow 1, |P^v|$ **do**
    if $C^v(P^v[i]) \leq C^s(P^s[i])$ **then**
      \# map i-*th* virtual pinned node
      $M_{node}(P^v[i]) = P^s[i]$;
      $counter = counter + 1$;
      $R^v \leftarrow R^v \cup P^v[i]$; $R^s \leftarrow R^s \cup P^s[i]$;
    else
      break;
    end if
  end for
end if
\# if there are 0 pinned nodes
if $|P^v| = 0$ **then**
  $V^v_{\gamma sort} \leftarrow sort(V^v_\gamma)$; \# accord. to MCRR-LA
  $V^s_{\gamma sort} \leftarrow sort(V^s_\gamma)$; \#
  $vnr\_max\_V\_node \leftarrow find\_max\_node(N^v, V^v_{\gamma sort})$;
  for all *nodes* n^s *ordered according* $V^s_{\gamma sort}$ **do**
    if $C^v(vnr\_max\_V\_node) \leq C^s(n^s)$ **then**
      \# map $vnr\_max\_V\_node$ to n^s
      $M_{node}(vnr\_max\_V\_node) = n^s$;
      $counter = counter + 1$;
      \# add nodes to reference nodes vectors
      $P^v \leftarrow P^v \cup vnr\_max\_V\_node$;
      $P^s \leftarrow P^s \cup n^s$;
      $R^v \leftarrow P^v$; $R^s \leftarrow P^s$;
      break;
    end if
  end for
end if
\# if all pinned nodes successfully mapped
if $counter == |P^v|$ **then**
  $\tilde{V}^v_\gamma \leftarrow normalize(V^v_\gamma)$;
  $\tilde{V}^s_\gamma \leftarrow normalize(V^s_\gamma)$;
  $num\_pinned\_nodes \leftarrow |P^v|$;
  $ref\_node\_index \leftarrow 1$;

Algorithm 1 *NodeMapping* (*Part* 2)

```
46:     # for the remaining virtual nodes to map
47:     for x ← 1, (|N^v| − num_pinned_nodes) do
48:         w_{Z_{m,γ}} ← (1 − w_{V_γ})/|R^v|;
49:         for i ← ref_node_index, |R^v| do
50:             # metric vect. Z for reference node in pos. i
51:             Z^v_{R^v[i],γ} ← compute_metric_Z(G^v, R^v[i]);
52:         end for
53:         for i ← ref_node_index, |R^s| do
54:             # metric vect. Z for reference node in pos. i
55:             Z^s_{R^s[i],γ} ← compute_metric_Z(G^s, R^s[i]);
56:         end for
57:         for i ← ref_node_index, |R^v| do
58:             # normalize metric vector Z^v_{R^v[i],γ}
59:             Z̃^v_{R^v[i],γ} ← normalize(Z^v_{R^v[i],γ});
60:         end for
61:         for i ← ref_node_index, |R^s| do
62:             # normalize metric vector Z^s_{R^s[i],γ}
63:             Z̃^s_{R^s[i],γ} ← normalize(Z^s_{R^s[i],γ});
64:         end for
65:         # Compute the similarity measure matrix
66:         compute D;
67:         # Sort each row in ascending order
68:         D_sort = sort(D);
69:         min ← MAX_BIG_VALUE;
70:         virt_node ← 0; phys_node ← 0;
71:         # Find pair (n^v, n^s) with min D
72:         for all nodes n^v in seq. order ∉ R^v do
73:             for all unused nodes n^s ∉ R^s in the order of
                                                D_sort(n^v, :; R^v) do
74:                 if C^v(n^v) ≤ C^s(n^s) then
75:                     if D(n^v, n^s; R^v) ≤ min then
76:                         min ← D(n^v, n^s; R^v);
77:                         virt_node ← n^v; phys_node ← n^s;
78:                         break;
79:                     end if
80:                 end if
81:             end for
82:         end for
83:         # If a pair was found
84:         if min ≠ MAX_BIG_VALUE then
85:             # map virt_node to phys_node
86:             M_node(virt_node) = phys_node;
87:             counter = counter + 1;
88:             # virt./phys. nodes become fixed nodes
89:             R^v ← R^v ∪ virt_node; R^s ← R^s ∪ phys_node;
90:             ref_node_index ← |R^s|;
91:         else
92:             break;
93:         end if
94:     end for
95: end if
96: # if all the virtual nodes are successfully mapped
97: if counter == |N^v| then
98:     nodes_mapping_success ← true;
99:     update the CPU capacities C^s;
100: else
101:     mark the VNR as rejected;
102: end if
```

$M_{node}(n^v(i)) = n^{s*}(n^v(i))$; 3) we add $n^v(i)$ to the set of reference nodes R^v and correspondingly, we add $M_{node}(n^v(i))$ to $R^s = M_{node}(R^v)$, that is, $R^v = R^v \cup n^v(i)$ and $R^s = R^s \cup M_{node}(n^v(i))$.

The basic idea is that each time we map a virtual node, we add that virtual node (and the physical node to which it has been mapped) to the set of nodes with respect to we measure the proximity metric. Intuitively, as unpinned virtual nodes are mapped to physical nodes, we consider them as pinned nodes for the rest of the mapping phase[3].

A special case arises when there are no pinned nodes. In this case, in the first algorithm iteration we just compute the MCRR-LA metric $\mathbf{V}_\gamma$ and, in a greedy way, we map the virtual node with largest MCRR-LA metric $n^{v**} = \arg\max V^v_\gamma(n^v)$ to the physical node with the largest MCRR-LA metric $n^{s**} = \arg\max V^s_\gamma(n^s)$. The intuition is that, since there is no pinned node, we can freely decide in which area of the physical network we place the virtual network. In this scenario, a good solution is to place the virtual network where there are more available resources in the physical network, which is indeed captured by the MCRR-LA metric. Once the first node is mapped, we proceed to map the other unpinned nodes following the same iterative approach described above.

The node mapping is presented in Algorithm 1. We first compute resource latency-aware metric vectors (lines 7-8). If the VNR has one or more pinned node, the algorithm maps them to the requested physical pinned nodes and it adds them to the set of reference nodes, provided that substrate nodes have sufficient computing capacity to host them (lines 10-21), otherwise the entire mapping for this VNR will fail. If the VNR contains no pinned nodes, the algorithm sorts both virtual and physical nodes in non increasing order according to MCRR-LA metric and following the order determined by the metric, the virtual node with the largest metric value is mapped onto the first physical node which has adequate resources to host it (lines 23-39). If all the pinned nodes were successfully embedded, the MCRR-LA metric is normalized (lines 42-43) and the procedure continues with the iterative unpinned nodes mapping. Weights $w_{Z_{m,\gamma}}$, $m \in R^v$, are recomputed to reflect the increased number of reference nodes (line 48). Then, the MCR-P metrics are computed and normalized (lines 49-64).

The similarity metrics $D(n^v, n^s; R^v)$ are then computed (line 66), and then sorted in ascending order (line 68) to speed up the following greedy mapping policy. The algorithm determines the most similar unpinned virtual node-physical node pair, that is the pair with the smallest similarity metric $D(n^v, n^s; R^v)$ (lines 69-82), with the proviso that the candidate physical node must have enough resources to host the unpinned virtual node. If a suitable node pair was found, the node mapping is then carried out (lines 84-93) and the sets of reference nodes updated, otherwise the entire procedure fails. At the end of the iterative mapping procedure, if all nodes are successfully mapped, the VNR is accepted and the resources are allocated; otherwise the VNR is rejected (lines 97-102).

4.4.2 Link Mapping

In the second phase (Algorithm 2), the MCRM algorithm computes the shortest path (Dijkstra algorithm), in terms of latency, between each pair of substrate nodes. For the sake of efficiency, the substrate links without enough bandwidth are cut before executing the calculation of the shortest paths (lines 6-10). If it is impossible to find a path between two

[3]In an earlier version of the algorithm we just set $R^v = P^v$ throughout the entire node mapping phase. The resulting algorithm performance resulted always inferior to the version here presented.

physical nodes, meeting both bandwidth and latency constraints demanded by the currently processed virtual link, the heuristic is unable to determine a suitable assignment and the VNR is refused (lines 14-22).

Remark The second stage simply determines the shortest path among nodes, thus selecting the minimum latency path among them, irrespectively of the number of hops among nodes. In order to minimize resource consumption, the algorithm should also minimize the number of hops. We achieve this goal implicitly by using the proximity metrics MCR-P which basically rely on distance in terms of links. Indeed, properly assigning balanced weights to the resource latency-aware metric MCRR-LA and the proximity metrics MCR-P, making use of the similarity measure in the first stage, we determine as best candidates for node mapping not only the nodes with more resources in their vicinity, but also the nodes which are closer in terms of network hops.

Algorithm 2 *LinksMapping*

Input : VNR $G^v(N^v, E^v, C^v, B^v, L^v)$, Substrate net $G^s(N^s, E^s, C^s, B^s, L^s)$, mapping of nodes M_{node};
Output: Mapping of links M_{link};

1: $M_{link} \leftarrow \mathbf{0}$;
2: $links\_mapping\_success \leftarrow true$;
3: **for all** *virtual links* $e^v = (m^v n^v) \in E^v$ **do**
4: $G^s_{clone} \leftarrow G^s$; # clone the substrate network
5: # pre-cut links in G^s_{clone} without enough bandwidth
6: **for all** *physical links* $e^s \in E^s_{clone}$ **do**
7: **if** $B^s(e^s) < B^v(e^v)$ **then**
8: cut link $e^s = (m^s n^s) \in E^s_{clone}$;
9: **end if**
10: **end for**
11: compute a shortest path $pt^s(M_{node}(m^v), M_{node}(n^v))$ between the two nodes $M_{node}(m^v)$ and $M_{node}(n^v)$ in G^s_{clone}, in terms of latency;
12: # if a shortest path, in terms of latency, was found
13: # and it meets the requested latency constraint
14: **if** $|pt^s(M_{node}(m^v), M_{node}(n^v))| \neq 0$ **and** $pt^s(M_{node}(m^v), M_{node}(n^v)).tot\_lat \leq L^v(e^v)$ **then**
15: # mapping
16: $M_{link}(m^v n^v) \leftarrow pt^s(M_{node}(m^v), M_{node}(n^v))$;
17: update the bandwidth B^s of involved links in G^s;
18: **else**
19: $links\_mapping\_success \leftarrow false$;
20: mark the VNR as rejected;
21: **break**;
22: **end if**
23: **end for**
24: **if** $links\_mapping\_success = false$ **then**
25: undo the CPU and bandwidth updates to G^s;
26: **end if**

4.4.3 Computational Complexity

The node mapping phase cost is dominated by the cost of the inversion of the matrix $(\mathbf{I} - \gamma\mathbf{P})$ for the computation of the metric vector $\mathbf{V}^{\mathbf{s}}_{\gamma}$ which is $O(|N^s|^3)$ (observe that since $\mathbf{P_u}$ does not change over time, $(\mathbf{I} - \gamma\mathbf{P_u})^{-1}$ can be precomputed once for all greatly reducing the cost of computing the metric $\mathbf{Z}^{\mathbf{s}}_{\mathbf{m},\gamma}$). The link mapping requires for each virtual link the execution of the shortest-path algorithm between the two end-points with a cost $O(|E^v||N^s|^2)$ (or $O(|N^s|^3)$ if using the Floyd-Warshall algorithm to compute the all pairs shortest path). The overall cost is thus $O(|N^s|^3)$.

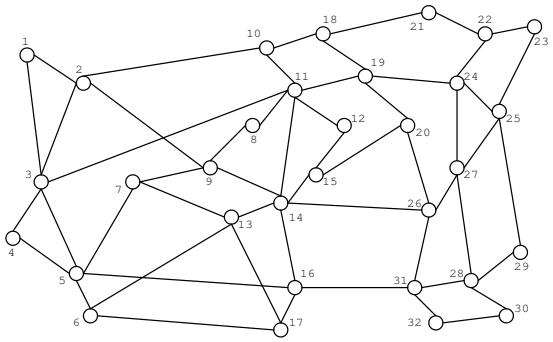


Figure 2: ANSNET network [15].

5. EXPERIMENTAL EVALUATION

We have extensively evaluated the performance of the MCRM algorithm through numerical evaluation. In this section, we report the results based on a realistic topology, the ANSNET topology (the Internet backbone built to replace the previous NSFNET backs in the early '90s which well captures the continental US geography and population distribution) which comprises a 32 node substrate network with 58 links, slightly modified as in [24], see Figure 2. We adopted the ANSNET topology to use a realistic topology rather a synthetic one, and for consistency with prior works on coordinated algorithms, including our previous papers on the subject [24, 6, 5].

We assume that the VNR arrival process obeys to a Poisson process with rate $\lambda = 5$. VNR lifetime is exponentially distributed with average lifetime T_{VNR} equal to 8, 12 and 16 time units, corresponding to a low, medium and high load scenario. Every VNR is a random graph with a number of nodes uniformly distributed in $\{4, \ldots, 10\}$ with link connectivity rate equal to $\frac{ln(|N^v|)}{|N^v-1|}$ to ensure that each node has $O(ln|N^v|)$ incident edges. We assume a VNR has either no pinned nodes (the VNR can be mapped anywhere in the substrate network) or two pinned nodes (which simulate the presence of two specific nodes to be included in the topology). The pinned nodes are chosen at random and assigned to two random nodes in the network which satisfy the delay constraints. The VNR CPU, bandwidth requirements are randomly drawn node per node and link by link in the sets $\{4, \ldots, 8\}$ and $\{2, \ldots, 5\}$, respectively. The substrate network physical resources in terms of CPU and bandwidth are normalized and all equal to the nominal value of 100. The substrate network links latency is proportional to the geographical distance between nodes. The basic VNR latency requirements over the links are randomly generated in the interval between the average and the maximum latency value of the substrate network links times a multiplicative factor δ. For the following results, every simulation experiment was 5000 time units long and was repeated 10 times. For the sake of clarity, in the figures we do not report the confidence intervals which were in general rather small (the width of the 95% confidence intervals of the different perfor-

mance indexes was always smaller than the 8% of the sample mean).

In the experiments, we compare different performance indexes, namely, the VNR blocking probability, the InP revenue, the average path delay between embedded VN nodes, along with the nodes and link utilization distribution for different traffic loads, topology characteristics and delay requirements as well as different algorithm parameters settings. In particular, we study the behavior of the algorithm as function of the similarity measure MCRR-LA weight w_{V_γ} (remember that we assume that all node proximity MCR-P weights have the same value, that is $w_{Z_m} = \frac{1-w_{V_\gamma}}{|R^v|}$, $m \in R^v$). For the discount factor γ, after an extensive parameter tuning, we set it to $\gamma = 1 - \frac{1}{diameter} \approx 0.83$, where $diameter = 6$ is the diameter of the considered ANSNET network. Intuitively, by regarding the discount factor γ as the step by step probability of continuing a random walk, we have that for $\gamma = 1 - \frac{1}{diameter}$, the average random walk length is exactly the substrate network diameter. In our experience, we always achieved the best algorithm performance for this value of γ.

Mapping Algorithm Running Time. We implemented the simulator and the MCRM algorithm in Java. For the experiments reported in this section, the average mapping algorithm execution time was 12.2ms on a 2.60 GHz Intel Core i7-4720HQ CPU platform with 16.0 GB of RAM.

Performance Indexes. In Figures 3-6, we plot the blocking probability, the revenue and the average path delay of our algorithm under different requests loads, varying the weight w_{V_γ} assigned to the MCRR-LA metric, for a number of pinned nodes equal to 0 and 2 and a delay requirement multiplicative factor δ equal to 1.5 and 3 (we remember that each VNR link latency requirement is randomly drawn in the interval $[\delta \cdot average\_lat, \delta \cdot max\_lat]$, where $average\_lat$ and max_lat are the average and maximum substrate network link latency, respectively). $\delta = 1.5$ and $\delta = 3$, thus correspond to tight, respectively, loose, delay bounds.

First of all, let us observe that the algorithm shows good general performance. As expected, the revenue but also the blocking probability increases at higher load and the blocking probability is significantly higher for stricter delay bound ($\delta = 1.5$) and when the VNRs have pinned nodes compared to the unpinned case. It is also important to note that the average VNR path delays only slightly increases with load. This shows that the MCRM algorithm offers consistent path delays behavior over a wide range of system utilization.

We now compare the behavior of the algorithm in the no pinned and pinned nodes case. In case of no pinned nodes, it is worth to remember that the algorithm uses the MCRR-LA ranking to locate where to place the first node (lines 23-39) which then constrains the placement of all the other nodes in the surrounding area. In this case, the performance of the algorithm basically depends on the ability of the MCRR-LA metric to locate resource rich/low latency areas where to place the first node (which constraints in that area the placement of the other VNR nodes). In this case, the impact of the similarity metric here plays a lesser role and indeed the algorithm performance is not significantly affected by the value of w_{V_γ}.

The algorithm behavior changes in case of VNRs with pinned nodes. As previously observed, performance is generally worse than in the unpinned node case; this is not unexpected since the presence of pinned nodes constrains the node placements. More interestingly, here the performance is noticeably affected by the similarity measures weights: when the delay constraints are tight ($\delta = 1.5$), the algorithm works best with $w_{V_\gamma} = 0$ while the opposite is true when delay constraints are less stringent ($\delta = 3$). This can be explained by observing that for VNRs with stringent path delays, the proximity metric MCR-P well captures the proximity/distance requirements from the pinned nodes during the placement process than the resource latency aware resource metric MCRR-LA which accounts for the computational and the network bandwidth resources but no so well the relative node distances. Instead, when VNR link delay is less stringent, we expect the reverse to be true since now node placement should be dictated by availability of resources rather than relative node distances.

Average Resources Utilization. In Figures 7-10, we plot the distribution of the average utilization of substrate nodes and links by our algorithm under high requests load as function of the weight w_{V_γ} for tight ($\delta = 1.5$) and loose ($\delta = 3$) delay requirements and no pinned and pinned nodes. The boxplots indicate the 25th, the 50th and the 75th percentile, along with the minimum and maximum value.

As expected, nodes and links utilization reflect the algorithm performance: utilization is higher when the blocking probability is lower, e.g., at high load, when VNRs have loose delay requirements and no pinned nodes the algorithm yields a blocking probability of $\approx 12\%$ which is reflected in average utilization of $\approx 90\%$, while higher blocking probabilities result in lower overall utilization. The same holds also for link utilization.

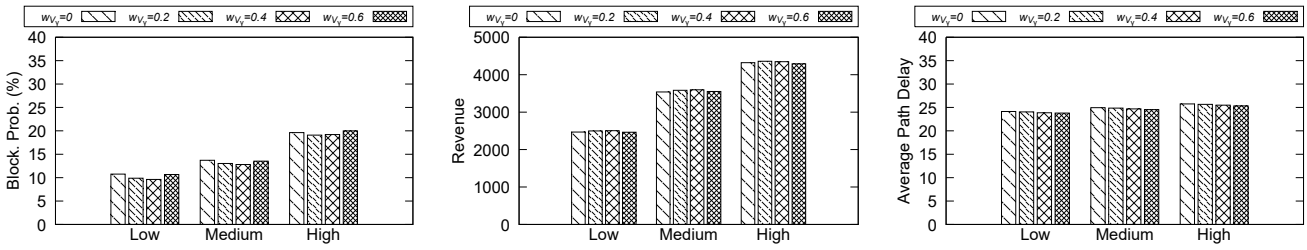
It is important to observe that node utilization is also quite uniform across the entire network. As observed in all our experiments, the latter property, in particular, is the key factor in reducing the blocking probability. Not surprisingly, the utilization variability increases for tighter delay bound and the presence of pinned nodes, as these reduce the algorithm degree of freedom in allocating resource which indeed is reflected in worse performance.

Link utilization exhibits a somewhat different behavior since we experience a larger spread with few underutilized links. This can be explained observing that because the link mapping is based on the Dijkstra shortest path algorithm, some redundant links of the ANSNET are unlikely to be used. In this case, we believe a different strategy could result into better balanced link utilization and will be subject of our future work.

6. CONCLUSIONS

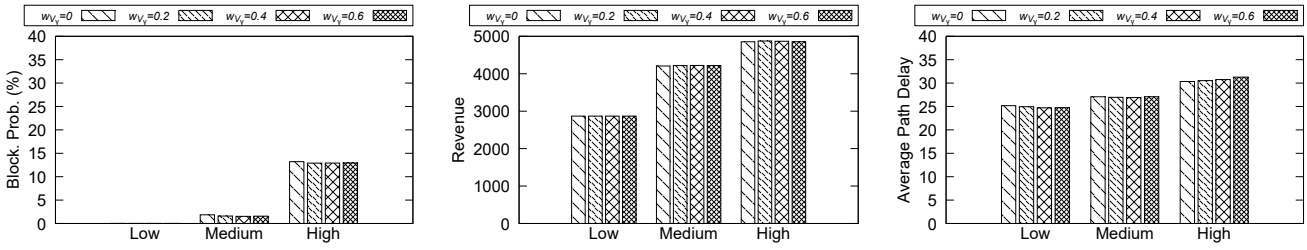
In this paper we have studied the Virtual Network Embedding problem. Differently from most approaches in the literature, which only consider node computational capacity and link bandwidth requirements, we have considered an extended version of the VNE problem which also accounts for delay requirements over the virtual links as well as the possibility of pinned nodes, that is, virtual nodes which must be assigned to specific physical nodes.

To this end, building on Markov Reward Theory, we have presented Markov Chain Reward Metrics (MCRM), a new algorithm for the VNE problem. MCRM consists of two steps: in the first step, MCRM maps the virtual nodes to the physical nodes using the notion of *similarity* between virtual and physical nodes; then, MCRM carries out the link mapping utilizing Dijkstra shortest path algorithm. The



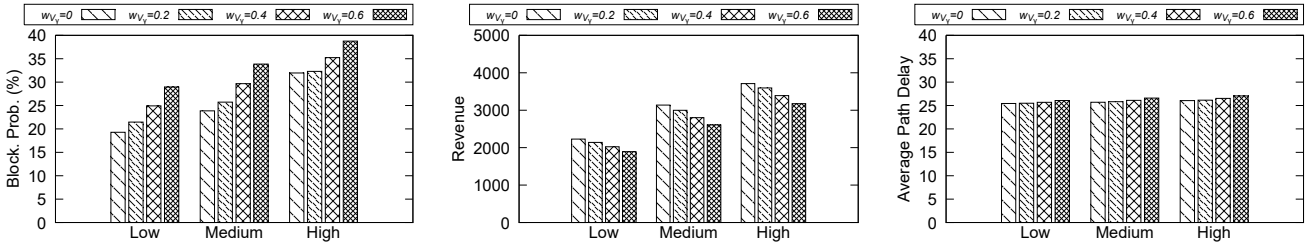
(a) Blocking Probability (b) Revenue (c) Average Path Delay

Figure 3: Performance under different loads: no pinned node, tight delay bound ($\delta = 1.5$).



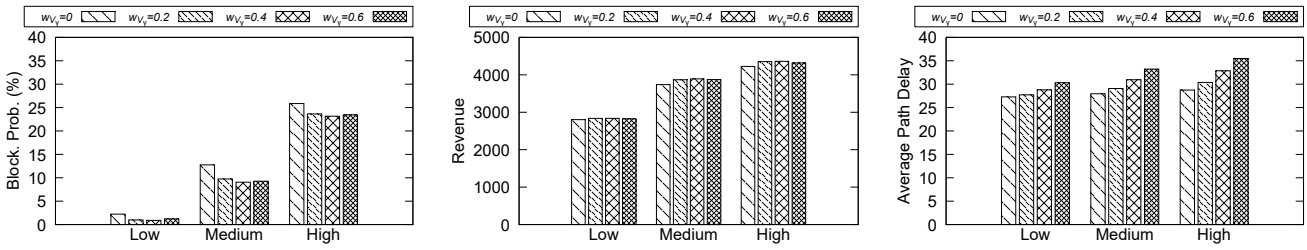
(a) Blocking Probability (b) Revenue (c) Average Path Delay

Figure 4: Performance under different loads: no pinned nodes, loose delay bound ($\delta = 3$).



(a) Blocking Probability (b) Revenue (c) Average Path Delay

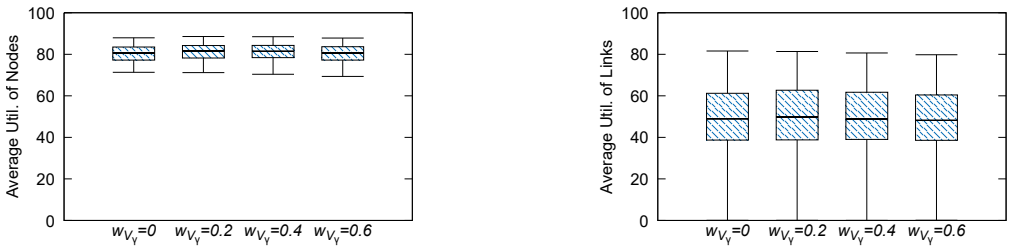
Figure 5: Performance under different loads: 2 pinned nodes, tight delay bound ($\delta = 1.5$).



(a) Blocking Probability (b) Revenue (c) Average Path Delay

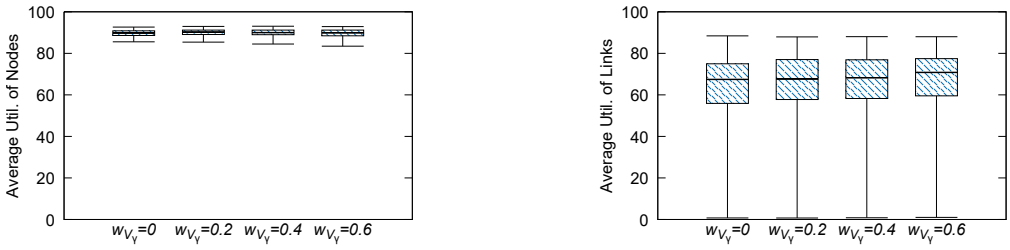
Figure 6: Performance under different loads: 2 pinned nodes, loose delay bound ($\delta = 3$).

key idea behind MCRM revolves around the definition of node metrics as the accumulated reward of suitable Markov Chains. In particular, for the VNE problem we have defined the following two key metrics: 1) MCRR-LA, the available embedding potential (requested amount of resources) of each substrate (virtual) node, that is aggregate resources/links delay in their respective neighborhood; and, 2) MCR-P, the level of *proximity* of nodes with respect to each other.



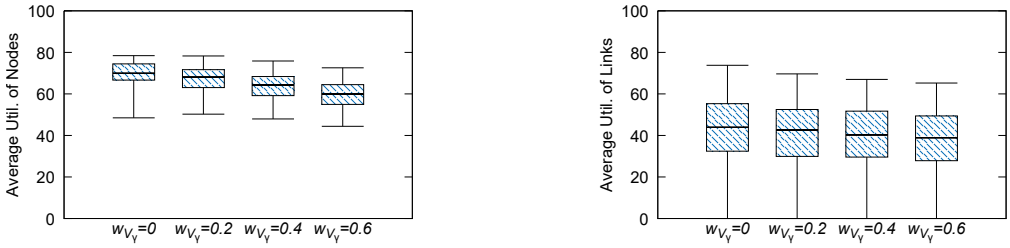
(a) Average Utilization of Nodes (b) Average Utilization of Links

Figure 7: Resource Utilization Distribution: no pinned nodes, stringent delay bound ($\delta = 1.5$).



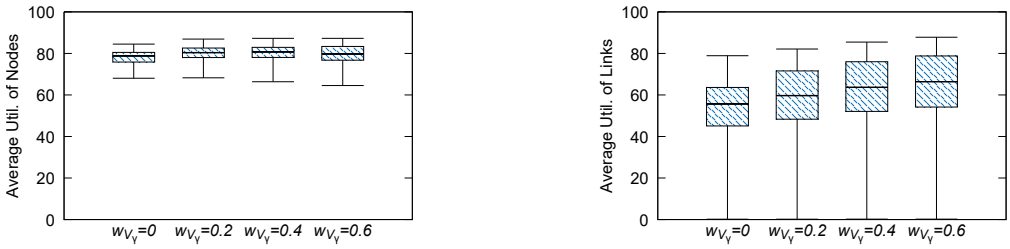
(a) Nodes Utilization (b) Link Utilization

Figure 8: Resource Utilization Distribution: no pinned nodes, loose delay bound ($\delta = 3$).



(a) Nodes Utilization (b) Links Utilization

Figure 9: Resource Utilization Distribution: 2 pinned nodes, stringent delay bound ($\delta = 1.5$).



(a) Nodes Utilization (b) Links Utilization

Figure 10: Resource Utilization Distribution: 2 pinned nodes, loose delay bound ($\delta = 3$).

We have extensively evaluated MCRM and studied the impact of the different algorithm parameters on performance. Our experimental results indicate that our algorithm shows overall good performance in terms of low blocking probability, high revenues and balanced resources utilization.

As future work, we plan to improve the algorithm generalizing the notion of metrics to other performance characteristics, e.g., network resiliency, and to provide an imple-

mentation on a suitable platform.

7. REFERENCES

[1] D. G. Andersen. Theoretical approaches to node assignment. Unpublished Manuscript, Dec. 2002.

[2] M. T. Beck and C. Linnhoff-Popien. On delay-aware embedding of virtual networks. In *The sixth international conference on advances in future internet, AFIN*, 2014.

[3] S. Behrouznia, R. Dssouli, and M. El Barachi. A qos-based resource selection approach for virtual networks. In *International Conference on Computer and Information Science and Technology, 2015. CIST'15.*, May 2015.

[4] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[5] F. Bianchi and F. Lo Presti. A latency-aware reward model based greedy heuristic for the virtual network embedding problem. In *Proceedings of InfQ 2016 - New Frontiers in Quantitative Methods in Informatics (in conjunction with VALUETOOLS 2016)*, October 2016.

[6] F. Bianchi and F. Lo Presti. A markov reward model based greedy heuristic for the virtual network embedding problem. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*, 2016.

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, Apr. 1998.

[8] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang. Virtual network embedding through topology-aware node ranking. *SIGCOMM Comput. Commun. Rev.*, 41(2):38–47, Apr. 2011.

[9] N. Chowdhury, M. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791, April 2009.

[10] N. Feamster, L. Gao, and J. Rexford. How to Lease the Internet in Your Spare Time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, Jan. 2007.

[11] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: A survey. *Communications Surveys Tutorials, IEEE*, 15(4):1888–1906, Fourth 2013.

[12] R. G. Gallager. *Stochastic processes: theory for applications*. Cambridge University Press, Cambridge, 2013.

[13] L. Gong, Y. Wen, Z. Zhu, and T. Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *INFOCOM, 2014 Proceedings IEEE*, pages 1–9, April 2014.

[14] K. Ivaturi and T. Wolf. Mapping of delay-sensitive virtual networks. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 341–347, Feb 2014.

[15] Z. Li and J. J. Garcia-Luna-Aceves. Finding multi-constrained feasible paths by using depth-first search. *Wirel. Netw.*, 13(3):323–334, June 2007.

[16] J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, pages 81–88, New York, NY, USA, 2009. ACM.

[17] J. Lu and J. Turner. Efficient Mapping of Virtual Networks onto a Shared Substrate. Technical Report 2006-35, Washington University in St. Louis, 2006.

[18] L. Nonde, T. E. H. El-Gorashi, and J. M. H. Elmirghani. Energy efficient virtual network embedding for cloud networks. *Journal of Lightwave Technology*, 33(9):1828–1849, May 2015.

[19] M. R. Rahman and R. Boutaba. Svne: Survivable virtual network embedding algorithms for network virtualization. *IEEE Transactions on Network and Service Management*, 10(2):105–118, June 2013.

[20] A. Razzaq and M. Rathore. An approach towards resource efficient virtual network embedding. In *Evolving Internet (INTERNET), 2010 Second International Conference on*, pages 68–73, Sept 2010.

[21] J. Shamsi and M. Brockmeyer. Qosmap: Qos aware mapping of virtual networks for resiliency and efficiency. In *2007 IEEE Globecom Workshops*, pages 1–6, Nov 2007.

[22] L. Shengquan, W. Chunming, Z. Min, and J. Ming. An efficient virtual network embedding algorithm with delay constraints. In *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pages 1–6, June 2013.

[23] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, Mar. 2008.

[24] S. Zhang, Z. Qian, J. Wu, and S. Lu. An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 2408–2416, March 2012.

[25] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, April 2006.

Analyzing and shaping the lifetime and the performance of barrier coverage sensor networks

Lorenzo Donatiello
Computer Science and Engineering Department
University of Bologna
Bologna, Italy
lorenzo.donatiello@unibo.it

Gustavo Marfia
Department for Life Quality Studies
University of Bologna
Bologna, Italy
gustavo.marfia@unibo.it

ABSTRACT

In this work we model and provide the means to extend the lifetime of a barrier coverage sensor network deployed for target detection. We consider a scenario where sensors are randomly dropped on a bidimensional field in order to detect target traversals which occur in a stochastic way within a critical mission time. Once a target enters a sensor's detection area, the sensor transmits such information to a cluster head, in charge of receiving and retransmitting the messages received from the sensors deployed on the field. The contribution of this work is fourfold. We first identify the sensing nodes whose behavior is key to model the duration of sensing operations, assuming prior arrival and mobility models for target traversals. We then proceed, providing a heuristic estimation of the traffic received by the cluster head to quantify its energy requirements, resorting to specific lifetime definitions. We also evaluate the relationship between our probabilistic and heuristic models and the time until the barrier remains capable of detecting and reporting the traversal of any target to a sink, as obtained by simulation. Finally, we show how the lifetime of such network may be shaped, with the use of a sequential activation mechanism, for example to combat the traversals of adversaries exploiting the lifetime models obtained in this work.

Keywords

Surveillance sensor network; lifetime definition; stochastic analysis; lifetime extension; adversary detection.

1. INTRODUCTION

It is difficult to delimit the fields of application of sensor networks, as they have reached any type of context and site (e.g., body, underwater, vehicular, Internet of Things, etc.) [1,5,16,17]. Among the many challenges these networks pose, finding ways of characterizing and increasing their lifetime amounts to one of the main ones [2]: as nodes deplete their energy and shutdown, the sensing power of a network degrades, until no longer capable of fulfilling its scope.

Lifetime analysis is challenging, as no unique and widely accepted definition exists which well adapts to all possible application scenarios. In a context where the monitored variable is the surface temperature of a field, for example, an acceptable metric may amount to the lifetime of the longest lasting sensor, when all deployed sensors are expected to provide sample values which are very close to each other, no matter where positioned. In a barrier coverage situation, instead, the time of death of the last sensor covering a given traversal trajectory may coincide with the lifetime of the network, as beyond such time the network may no longer be capable of guaranteeing the detection of all possible crossings of the area of interest. In this work we adopt a perspective which is close to this second example, as we study the key variables which influence most the efficacy and duration of a surveillance system composed by a number of sensors randomly dropped to monitor the traversal of a given bidimensional area.

In particular, the surveillance system amounts to a clustered sensor network whose scope is to detect and report specific type of events: the breach of targets across an area of interest (e.g., herds of animals crossing a canyon, soldiers moving through a minecamp, etc.). As a target enters, moves across the area and reaches its exit, it enters the sensing region of different sensors. All those sensors that detect the presence of the target transmit a message to a cluster head, in charge of receiving and retransmitting the messages received by all sensors to a sink. In such scenario, our aim is to individuate and model the nodes whose lifetime matches the lifetime of the surveillance system, as determined by the following definition [10, 11]: *the lifetime of a surveillance system is assumed to be represented by the lifetime(s) of its most critical traversal path(s), the path(s) that first will be available to a target for an undetected traversal.*

To proceed with such analysis, we assume a sufficient number of nodes are initially deployed to guarantee the detection of a target at any inner point within the area of interest. In addition, we assume the availability of prior: (a) target arrival and mobility models and, (b) an energy consumption model per target detection and message transmission/reception. We then probabilistically model and analyze two classes of key sensing nodes: (a) the node(s) which exhibit the highest probability of dying first, and, (b) a specific subset (which will become clear in Section 4) of the node(s), located on the path(s) which traverse the node(s) determined at the previous point, with the highest probability of dying last. Resorting to such models, we provide an approximation of the probabilistic distribution function of the lifetime of a surveillance system. Such results are also utilized to heuristically estimate the amount of energy required by the cluster head, in order to guarantee the forwarding of any messages received during the active phase of the surveillance system. Finally, after corroborating the

values obtained from our models with those returned by simulations, we propose a simple approach that may be put to good use to shape the network lifetime through a sequential and disjoint activation of nodes, in order to combat any unwanted breach across those paths that first will result unmonitored as a consequence of battery depletion.

The remainder of this paper is structured as follows. Section 2 provides an overview of the state-of-the-art, from a point of view of the stochastic analysis of sensor network lifetime. In Section 3 we delineate the system scenario of interest, whereas in Section 4 we provide our models. An evaluation of our model is provided in Section 7, whereas Section 8 concludes this work.

2. RELATED WORK

Studies evaluating the lifetime model of wireless sensor networks may be found in a plethora of scientific contributions [6, 13, 14, 19, 24, 26]. However, only a few have analyzed the node and network lifetimes in a pure probabilistic way [4,7,18,23]. We here first provide an overview of of such works, then moving on to describe the approaches, which fall close to ours, which implement mechanisms to extend a barrier coverage network's lifetime.

[18] analyzes randomly deployed clustered networks, where: (a) an approximation for the complementary cumulative distribution function of cluster lifetime is found, (b) different sensing modes are considered, (c) network lifetime depends on the random location of sensors, the amount of energy required to transmit a data packet at different distances, the events' appearance time and location, and, (d) the network is assumed active as long as β out of N sensors are still alive [19]. Also [23] proposes an analysis providing: (a) a Markovian framework to characterize the energy consumption distribution during a given period of time, where energy consumption converges to a Normal distribution if the observation time is long, and, (b) node and network lifetime distributions. The contribution presented in [7] assumes an event may not always be detected by a sensor network, modeling the probability of detecting an event adopting an exponential negative decay law dependent on the distance from the interested node. A node is represented as an entity which can transmit a maximum number of packets, not considering the fact that energy resources are spent also while in idle mode. The main differences between our work and [7,18,23] are: (a) we introduce two new lifetime definitions, derived from the specific scenario of interest, (b) events are not modeled as instantaneous and punctiform, but characterized by the path traversing the monitored area, and, (c) when a target traverses the monitored area, all sensors whose sensing range cover the path are involved in the detection and reporting of the event.

The authors of [8] introduce the concept of local barrier coverage which is there put to good use to guarantee the detection of all movements whose trajectory is confined to a slice of the belt region of deployment. To demonstrate that local barrier coverage can be used to design localized algorithms, the authors develop a sleep-wakeup algorithm for maximizing the network lifetime, called Localized Barrier Coverage Protocol. In [15] the authors provide an optimal centralized solution to the sleep-wake-up schedule problem for the case when wireless sensor nodes are deployed to form an impenetrable barrier for detecting movements. More recently, [21] develop a distributed algorithm to divide a field of sensors monitoring an area into shifts in both horizontal and vertical directions. Each shift can ensure sensing one line of the monitored area such that no intruders can cross the line without being detected, while at the same time, the sensors in one shift have minimum sensing overlap to conserve the energy. At any time, k shifts of sensors are on in both directions. Compared to these works, our contribution differs as the proposed scheduling mechanism: (a) leverages on prior target arrival and mobility models, (b) provides a simple way to estimate the duration of detection operations based on such information, and, (c) may be exploited as a tool that may be put to good use to combat undetected adversary breaches, rather than as a scheme whose only scope is to maximize the barrier's lifetime.

Finally, a number of works have also concentrated on the definition and analysis of path finding algorithms (e.g., minimal exposure path), under different assumptions of prior information regarding the sensing field [22, 27]. Adopting the perspective of the sensor network designer, the authors of [20] instead proposed a mixture deployment of mobile and static sensors to dynamically change the topology and minimum exposure path of the whole network, in order to combat any intruder. Our work differs from all of these approaches, as it distinctively concentrates on a situation where traversals may occur from legitimate and non-legitimate actors, with the latter taking advantage of available information regarding the former. In addition, we here do not provide a strategy, but a tool (i.e., shaping the lifetime of nodes within the network) that may be put to good use to implement a desired strategy.

3. SYSTEM MODEL

We consider a bidimensional field A of arbitrary shape, where N sensors are randomly dropped to monitor the traversals of targets which may appear at random times. The N dropped nodes are supposed to be sufficient to cover every single point in A and to report the detection of a target to a fixed cluster head. The cluster head is also equipped with a limited amount of energy, which may be more than the one provided to single sensors. The following summarizes the system assumptions made to formulate our analysis.

A sensor node, in general, consumes energy for sensing, receiving, transmitting, data processing, sleeping and also during the idle mode when no data sensing, processing or exchange happens. Such model is here simplified, considering the following modes of operation as the ones necessary to determine the lifetime of a node for the scenario of interest: sensing, transmitting and idle. The idle state does not correspond to the sleep one, as a sensing node is capable of detecting an event during idle mode. We also assume that the energy consumed while active, monitoring for the occurrence of events, is much lower than the energy spent during sensing and transmitting phases. A similar model is also adopted for the cluster head, with the difference that the cluster head is solely devoted to receive and retransmit messages to a sink node. Hence, during reception and transmissions, we assume the cluster head consumes an amount of energy which is much higher than the amount of energy spent during idle mode, when scanning the channel for the reception of any transmission. Now, instead of energy, we will directly deal with the corresponding lifetime quantities. Instead of considering the total energy of a given node i, we will refer to its maximum lifetime, $l_{max,i}$, here defined as the

lifetime of node i while permanently staying in idle mode. For what concerns detection and transmission phases (or receival and retransmission at the cluster head), we assume i consumes a constant amount of lifetime equal to $l_{eh,i}$. As in previous works, events are assumed arriving to the monitored area according to a Poisson process. The Poisson distribution has been widely used in the literature to model events whose time/place of occurrence are random and independent from each other [3]. Unlike previous approaches, events do not popup at given points in space, but are detected by sensors as targets move along random trajectories within the area under observation. The shape of the monitored area may be random, depending on the application needs and on how the sensors spread when dropped. Our assumption is that it may always be possible to individuate an entry and an exit to the area. Different approaches may, then, be utilized to represent mobility within the area. A successful approach, adopted in literature, amounts to n-th order Markov Chains, $MC(n)$, where the probability of reaching a given state depends upon the n previously visited ones [12].

As anticipated, we consider a scenario where a sensor network organizes itself into clusters. With the adoption of a clustering scheme, nodes self-organize themselves into hierarchical layers. We assume three layers: (a) a bottom layer composed of sensing nodes in charge of detecting targets entering their sensing region and transmitting such information to the cluster head, (b) an intermediate one, made of a cluster head which receives and retransmits messages and, (c) a top one where a sink node collects all the information received by the cluster head. Assuming such type of organization, a time division multiple access (TDMA) scheme may be employed at the MAC layer, where the cluster head takes care of coordinating the time scheduling among the nodes. TDMA well fits our scenario of interest, as it is a common choice, at the MAC layer, for clustered WSNs [25]. Finally, such assumption also ensures the avoidance of collisions, allowing the modeling of intra-cluster transmissions as ideal.

4. LIFETIME MODEL

As anticipated, we consider two classes of sensing nodes to compute the lifetime of the surveillance system. The first class is very simple to identify: these are the node(s) that die first (exhaust their energy first) [9]. The second class, instead, is individuated as the subset of the node(s), which lying along the paths that contain the node(s) which are expected to die first, will die last [10]. Which subset we are referring to will become clear by the end of this Section.

Before proceeding, however, let us remind that in this paper we assume a generic sensing node i is equipped with an initial lifetime $l_{max,i}$, the time node i would live in case it was always idle. A sensing node i which, hence, utilizes part of its energy resources to detect targets and transmit messages, lives for an interval of time which is lower than $l_{max,i}$. Lifetime l_i may hence be written as: $l_i = l_{max,i} - l_{EH,i}$, where $l_{EH,i}$ amounts to the lifetime reserved and available for sensing and message transmission activities. We also assume all sensing nodes in the network are equipped with the same amount of energy at the beginning and that the amount of energy spent in correspondence of a sensing/message transmission activity is constant and equal for all nodes.

Now, the lifetime of a sensing node depends on how often it is involved in sensing/message transmission operations, i.e., on its probability of activation. The probability of activation, in turn, depends on the position of the sensing node within A and on the mobility model of targets. This means that depending on its position, but also on its sensing reach, a sensing node may be more often, or less, activated.

The individuation of the node that will die first, among all sensing nodes, can be performed individuating the position(s) with the highest activation probability, i.e., the highest probability of detecting a target: in fact, the sensors located in such position(s) will most probably exhaust their energy before any other.

The position(s) of the node(s) that will die first can also be put to good use to determine the sensor network lifetime according to the second approach of interest. In fact, with the individuation of the node(s) that will, with the highest chance, exhaust their energy first, it is also possible to individuate the path(s) that will, most probably, exhaust the energy of the monitoring sensors first, and their lifetime, as follows. Say, for example, only one position with the maximum activation probability value may be found in a specific area A. Denote such point as (x_M, y_M), with a corresponding activation probability equal to $p(x_M, y_M)$. Next, among all possible paths that traverse (x_M, y_M), individuate the path(s) characterized by the highest probability to be used to traverse the critical area (for simplicity, from now on we assume only one of such paths exist and indicate it as T). Such condition implies T is the path with the highest probability of exhausting its energy resources first, becoming viable for a target seeking an undetected traversal. Please note that the lifetime of T, and hence of the sensor network, may be analyzed modeling the behavior of the node exhibiting the lowest probability of activation: the lifetime of such node(s) amounts to the second lifetime metric considered in this work (the lifetime computation, according to this procedure, is presented in Algorithm 1). Clearly, resorting to such probability metrics concerning the critical path one can establish several other heuristics which may result useful to compute stochastic bounds of the variable of interest.

To visualize how the location of the two nodes of interest may be individuated, consider Figure 1. In particular, Figure 1b represents the activation probability values of a 1 x 1 square, where targets can enter from any point on the $y = 0$ segment, traverse the area walking along a straight line, exiting from any point on the $y = 1$ segment. According to Figure 1b, the node that is expected to die first, i.e., the one with the highest probability of activation, falls exactly at the center of the area. Now, the position of the second node of interest can be found as follows: of all paths which start at some point on the $y = 0$ segment and end on the $y = 1$ one, take the one exhibiting the maximum value of all the minimum activation probabilities. Following such rule, we find such path as the one crossing the area along the $x = 0.5$ segment (the activation probabilities along the $x = 0.5$ segment are plotted in Figure 1a). Finally, referring again to Figure 1b, we conclude saying that the first lifetime of interest may be computed modeling the behavior of a node located in $(0.5, 0.5)$ (the location with highest activation probability), whereas second lifetime metric can be obtained analyzing the node(s) located in $(0.5, 0)$ and $(0.5, 1)$. In fact, when the sensor(s) located in $(0.5, 0)$ and $(0.5, 1)$ exhaust their energy, it is probabilistically plausible

Data: Arrival and mobility model of targets entering, moving inside and exiting the monitored area A and lifetime consumption per event model.
Result: Probability the lifetime of the sensor network exceeds threshold value L.

Step 1: compute $p(x_i, y_i)$, $\forall i$, which amounts to the probability a target traverses the circular region centered in $(x_i, y_i) \in A$, within a radius of r (the sensing area of a sensor);

Step 2: find the set of locations $\{(x_M, y_M)\}$ s.t. $\{(x_M, y_M)\} = max_{(x_i,y_i)\in A} p(x_i, y_i)$. Such locations are those where the probability of activation is highest;

Step 3: Per each point in $\{(x_M, y_M)\}$, find the set feasible target paths $\{T_j\}$ which cross such points;

Step 4: Among all $\{(x_j, y_j)\} \in \{T_j\}$, find the location(s) $\{(x_m, y_m)\}$ exhibiting the maximum of all minimum activation probabilities: $\{(x_m, y_m)|p(x_m, y_m) = max\{min_{(x_j,y_j)\in T_j} p(x_j, y_j)\}$;

Step 5: Compute the probability the lifetime of the network exceeds threshold value as a function of the target arrival process, the node activation probability $p(x_m, y_m)$ and the event lifetime consumption model.

Algorithm 1: Lifetime computation procedure.

that the path connecting $(0.5, 0)$ to $(0.5, 1)$ will result free of active sensing devices. This is the most critical path in such scenario. Before concluding, please note that the cluster head amounts to an important bottleneck for the flow of information from the field of sensing nodes to the sink. Because of this, we will show how it is possible to put to good use the lifetime definitions provided in this Section to compute the amount of energy required by the cluster head. In the following we show how it is possible to obtain: (a) a stochastic model for sensing nodes lifetime according to the two aforementioned definitions, and, (b) a heuristic function which may be used to define the energy requirements of a cluster head. To do this, we first show how it may be possible to compute the activation probability of a node within a cluster and the move on to model the probability of activating a given number of nodes in correspondence of a traversal.

5. LIFETIME COMPUTATION

5.1 Node activation probability

From now on, for the sake of simplicity, we consider a situation where A is a square, of size X^2, with N sensor nodes located inside. N is big enough to guarantee that each point (x, y) in A is covered by at least one sensor, assuming each sensor can detect, according to a boolean model, a target moving within a radius r.

We want to compute the probability a generic node i, located in (x_i, y_i), is activated every time a target traverses A. Such probability may be computed searching, first of all, for the probability that i is activated when a single target traverses A. When a target traverses A, all the sensors that are within a distance of r units are activated, hence if i lies in a stripe determined by such quantity, i is activated. To find such probability hence, it is necessary to compute the probability that a random segment, describing the traversal of a target, falls within distance r from i.

Consider now the situation depicted in Figure 2. Assume a target may only enter from the bottom edge of the square and leave from the opposite one. The entrance and exit points are random, though. Such locations are chosen uniformly in $[0, X]$. Hence, according to the reference system adopted in Figure 2, the entrance point is given by $(x_1, 0)$ and the exit point by (x_2, X), where x_1 and x_2 are i.i.d. uniform random variables $U[0, X]$. The event that i is activated amounts to the event that a possible trajectory T intersects the sensing area of i. Such event is here indicated as $T \cap i$ and its probability is given by:

$$P(T \cap i) = \int_T P(T \cap i|T) \times P(T), \tag{1}$$

where $P(T \cap i|T)$ amounts to the probability of intersection, for a given trajectory T, and $P(T)$ to the probability of a given trajectory T. Let us start considering the problem of determining $P(T)$. Its value can be easily found considering the fact that $P(T) = P(x_1 \wedge x_2)$. Reminding that x_1 and x_2 are i.i.d. uniform random variables $U[0, X]$, we have that $P(T) = 1/X^2$. For what concerns $P(T \cap i|T)$, we have that:

$$P(T \cap i|T) = \begin{cases} 1 & \text{if the sensing area of } i \text{ and } T \text{ intersect,} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Now, it is possible to determine whether T and the sensing area of i intersect, for example, computing the distance between i's coordinates, (x_i, y_i) and trajectory T. Hence, may be written as $P(T \cap i|T) = \mathbb{1}_{\{d((x_i,y_i),T)\leq r\}}$. Equation 1 may hence be rewritten as:

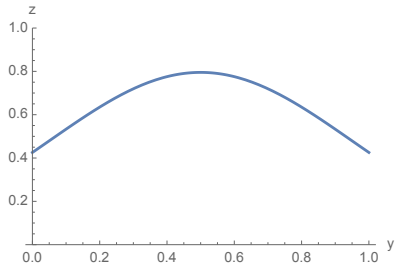
$$P(T \cap i) = \frac{1}{X^2} \times \int_0^X \int_0^X \mathbb{1}_{\{d((x_i,y_i),T)\leq r\}} dx_1 dx_2. \tag{3}$$

5.2 Number of detections per traversal

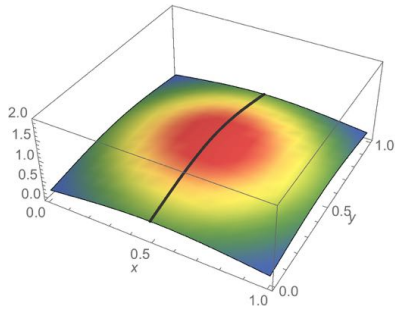
Computing the cluster head lifetime entails finding the number of events it will be handling. This quantity, in turn, depends on the number of targets traversing A and on the number of sensor nodes that are activated in correspondence of each single traversal. We here start computing the probability a given number of sensing nodes are activated in correspondence of a single traversal. Such function may be computed observing that, in correspondence of a traversal, all the nodes that fall within distance r will be activated. In essence, it is possible to approximate the area where all activated sensing nodes fall as a rectangle of dimensions $2 \times r$ by x, where $X \leq x \leq \sqrt{2} \times X$. With such assumption, we determine the probability a given number of sensing nodes falls in a rectangle of dimensions $2 \times r \times x$, given x. Such probability can be obtained resorting to a Binomial distribution, as our initial assumption is that nodes are dropped in a uniform way in A:

$$P(N' = n|x) = \binom{N}{n} \times \left(\frac{2 \times r \times x}{X^2}\right)^n \times \left(1 - \frac{2 \times r \times x}{X^2}\right)^{N-n}. \tag{4}$$

Now, Equation 4 may be put to good use the compute the probability of activating resorting to the law of total probability:



(a) Trajectory, among those crossing (x_{max}, y_{max}), which exhibits the maximum among the minimum probability of activation.



(b) Probability of activation field along with the trajectory that traverses (x_{max}, y_{max}) and exhibits the maximum among the minimum activation probabilities.

Figure 1: Activation probability example.

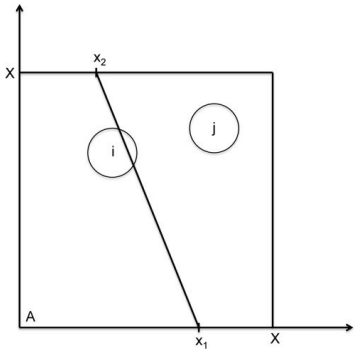


Figure 2: Computing the probability of activation of sensor i.

$$P(N' = n) = \int_X^{\sqrt{2}X} P(N' = n|x) \times P(x)dx, \qquad (5)$$

with $P(x)$ the probability a target traverses A walking along a path of length x. It is possible to show geometrically that $P(x) = 2 \times x \times (X - \sqrt{x^2 - X^2})/X^2)$, from which we obtain:

$$P(N' = n) = \int_X^{\sqrt{2}X} \binom{N}{n} \times \left(\frac{2 \times r \times x}{X^2}\right)^n \times \times \left(1 - \frac{2 \times r \times x}{X^2}\right)^{N-n} \times \frac{2 \times x \times (X - \sqrt{x^2 - X^2})}{X^2} dx, \qquad (6)$$

Equation that is here put to good use to compute a heuristic function which may return a significant value of the cluster head lifetime and consequent energy requirements.

5.3 Event lifetime consumption model

In this Subsection we proceed first providing a model which may be put to good use to compute the probability distribution function of the lifetime of any node in the cluster and then move on to provide the heuristic function which is utilized to estimate the energy requirements of the cluster head. Clearly, the proposed consumption model may be subject to modifications depending on the particular application scenario. The same may be said for the heuristic proposed to compute the energy required by the cluster head.

5.3.1 Sensing node lifetime

Assuming targets arrive in A according to a Poisson model with intensity γ, each single sensor i keeps seeing Poisson traffic with intensity $\gamma_i = p_i \times \gamma$, where p_i is obtained as in Equation 3.

Assuming a constant amount of lifetime $l_{eh,i}$ is lost at node i every time a target crosses its range, the problem amounts to a simple budgetary problem: an equal amount of energy is subtracted from the initial lifetime $l_{EH,i} = l_{max,i} - l_i$ available per each message detection and transmission procedure.

The probability of spending less than $l_{EH,i}$ units of lifetime assuming Z events are detected and handled is:

$$P(Z \times l_{eh,i} < l_{EH,i}|Z) = \begin{cases} 1 & \text{if } Z < l_{EH,i}/l_{eh,i}, \\ 0 & \text{if } Z \geq l_{EH,i}/l_{eh,i}. \end{cases} \qquad (7)$$

Using an alternative notation, $P(Z \times l_{eh,i} < l_{EH,i}|Z)$ amounts to the unit step function $u(l_{EH,i}/l_{eh,i} - Z)$. Taking now into account all possible arrivals, resorting to the total probability theorem, we can write:

$$F(l_{EH,i}, l_i) = \sum_{Z=0}^{\infty} F(l_{EH,i}, l_i|Z) \times e^{-\gamma_i l_i} \frac{(\gamma_i l_i)^Z}{Z!} = = e^{-\gamma_i l_i} \times \sum_{Z=0}^{\lfloor \frac{l_{EH,i}}{l_{eh,i}} \rfloor} \frac{(\gamma_i l_i)^Z}{Z!}. \qquad (8)$$

Equation 8, computed for $F(l_{max,i} - l_i, l_i)$, returns the probability of spending less than the amount of available lifetime for event handling, when the lifetime of node i is l_i. In other words, it provides us with the probability of i lasting at least l_i units of time.

5.3.2 Cluster head lifetime heuristic function

The lifetime $l_{EH,CH}$ spent by the cluster head for event handling may be obtained as the sum of all detections made by the cluster multiplied by the amount of energy spent per each relay operation. If Z traversals occur, the amount of battery lifetime required at the cluster head to manage all can be written as $l_{EH,CH} = l_{eh,CH} \times \sum_{i=1}^{Z} n_i$, where

n_i amounts to the number of nodes activated in correspondence of the i-th traversal and $l_{eh,CH}$ the constant amount of lifetime spent at the cluster head in correspondence of an event handling.

This said, the heuristic function that is proposed in this work to assess the amount of battery lifetime $\tilde{l}_{EH,CH}$ required by the cluster head for event handling is:

$$\tilde{l}_{EH,CH} = Z^{90\%} \times n^{90\%} \times l_{eh,CH}, \quad (9)$$

where $Z^{90\%}$ amounts to the 90-percentile of random variable representing the number of arrivals in A during the given mission time and $n^{90\%}$ the 90-percentile of the number of activations per traversal. In Section 7 we verify how the value of $l_{max,CH} - \tilde{l}_{EH,CH}$ relates to the cluster head lifetime value l_{CH} obtained by simulation.

6. SEQUENTIAL ACTIVATIONS

Now, the analyzed scenario exhibits a situation where it is possible to identify areas where nodes deplete their energy resources at a faster rate than others, no matter how many nodes are deployed, as long as all are active implementing a barrier coverage functionality. In essence, it is plausible to expect areas where nodes will die first and hence which will be available for undetected traversals. The question, at this point, is: is it possible to extend the lifetime of the nodes that reside in such areas? Positive answers to such question have been provided in literature, and reviewed in Section 2, resorting to sleep-awake cycles.

We here to desire to move one step forward along this line, considering a situation where an adversary sits and waits until a path is available for an undetected traversal. In fact, a scenario like the one that has been so far described may be exploited by an someone who, for example, waits until the sensor field exhausts its energy resources as expected in those areas that are most often traversed by legitimate targets, to let non-legitimate targets sneak through without being detected. As a practical example, consider a canyon that is normally traversed by herds of animals, following some known mobility pattern: such harmless targets may be exploited by malicious intruders seeking an undetected traversal, simply sitting on one side of the monitored area until unmonitored paths are expected to be available. What we hence search here is, not simply a way of increasing the lifetime of the network, rather a strategy that may be followed to shape it as desired, within the given constraints, throughout the monitored region, in order to remove any predictive advantage to an adversary that is seeking for undetected traversals. In order to visualize such idea, consider the two contour fields plotted in Figure 3. In Figure 3a we show the activation probabilities that, for example, could be naturally induced by traversals. Clearly, in such situation, nodes which fall at the center of the field would be expected to last less than nodes located close to an edge, an adversary may choose to traverse the path connecting (0.5, 0) and (0.5, 1), when sufficient guarantees exist that such path is free of active sensors. A possible countermeasure could be put implemented as shown in Figure 3b: sensors which monitor the same areas are activated sequentially in the higher half of the square, so that targets entering from the bottom edge may find the path free up to a given point ($y \leq 0.5$), but unexpectedly monitored for $y > 0.5$.

Unlike the approaches taken in [8,15,21], where nodes are alternatively switched on and off in cycles to optimize the duration of the sensor field, our aim here is to provide a mechanism which may be employed to adapt, compatibly with the existing energy constraints, the lifetime of the detection capabilities of the network across different areas of a monitored field. For this reason, we here propose an approach where disjoint groups of nodes stay active as long as they can, but in non-overlapping periods of time. As a relevant example, consider Figure 4: nodes belonging to the same area are active in different time frames, in order to be able to guarantee detections during the desired mission time. In particular, referring to Figure 4, it is possible to imagine two non-overlapping areas exist, such as A_1 and A_2, where the nodes in the former area are activated more rarely than the nodes belonging to the latter. Nonetheless, one may which to achieve a homogeneous lifetime of the detection functionality in the two areas. Nodes in A_2 are hence activated in groups of smaller size than A_1, in order to prolong the monitoring time of A_2. Now, we here introduce a simple heuristic that can be put to good use to compute how many nodes should be active at the same time within a given area (assuming the groups chosen that area guarantee an adequate barrier coverage).

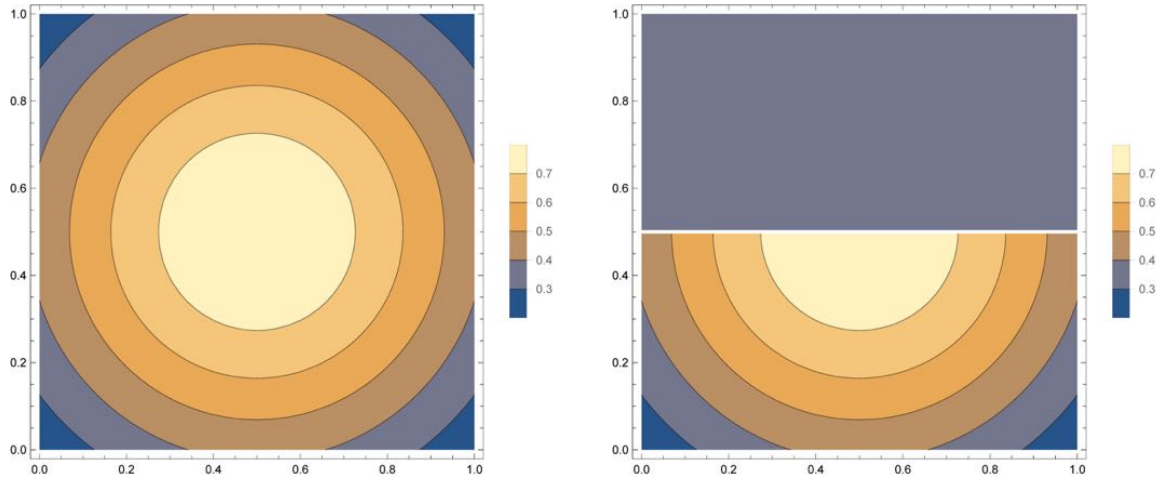
Assume N_i nodes fall in A_i and that the average lifetime of such nodes is l_i, considering a given mission time t. l_i is here obtained as the average of the lifetimes of the nodes that fall in A_i, assuming the lifetime of a node may be obtained from Equation 8. We now want to compute in how many subgroups g_i, nodes N_i should be divided to obtain an average lifetime equal to $\tilde{l}_i$, or, in other words, the number of nodes $N_{g,i}$ that should be active at the same time in A_i. $N_{g,i}$ is here obtained as $\lfloor N_i \times \tilde{l}_i / l_i \rfloor$ and $g_i = \lceil N_i / N_{g,i} \rceil$. Clearly, such numbers may not be adequate to provide proper coverage during each time interval, for this reason it is necessary to implement an iterative procedure as the one described in Algorithm 2.

Data: Area A_i, number of nodes N_i which fall in area A_i, desired average lifetime $\tilde{l}_i$.
Result: $N_{g,i}$ = number of nodes active during each period, g_i = number of groups of nodes
Step 0: compute l_i as average lifetime of the nodes that fall in A_i, where the lifetime of a node is obtained by applying Equation 8 for a given percentile value ;
Step 1: compute $N_{g,i} = \lfloor N_i \times \tilde{l}_i / l_i \rfloor$;
Step 2: compute $g_i = \lceil N_i / N_{g,i} \rceil$;
Step 3: Search for $N_{g,i}$ for each of the g_i groups, such that the nodes provide barrier coverage to A_i;
Step 4: If barrier coverage is not guaranteed by all g_i groups found at Step 3, $\tilde{l}_i = \tilde{l}_i - \delta$, where δ is a predefined value such that $\tilde{l}_i - \delta > l_i$, and return to Step 1;

Algorithm 2: Individuate the nodes that will be sequentially activated within a given area.

7. EVALUATION

We here validate ours models resorting to simulation results, produced with a custom simulator developed in Mathematica. We consider a square area A with $X = 6$ units.



(a) Contour plot of activation probabilities: all nodes are active at all times.

(b) Contour plot of activation probabilities: node activation times follow a predefined strategy.

Figure 3: Contour plot of activation probabilities on a 1×1 field.

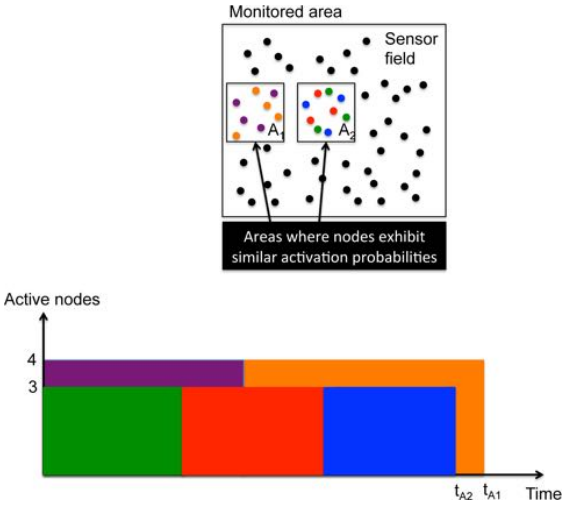


Figure 4: Sequential activation of nodes.

The situation is the one presented in Subsection 5.1: intruders have an equal probability of entering at any point of only one edge of square A and of exiting from any point of on the edge opposite to the entrance (no targets can enter from the remaining two edges). Parameter values are $N = 100$, $l_{max,i} = 100$, $l_{eh,i} = 5$, $\gamma = 1$ and $r = 0.5$ for all sensing nodes i.

In Figure 5 we plot all alive and dead nodes observed in 100 simulations within A, considering intermediate time steps before 80 units of time. As expected, the nodes that first stop operating are those closer to the center of the square. At time $t = 60$ the detection system is no longer reliable (Figure 5c).

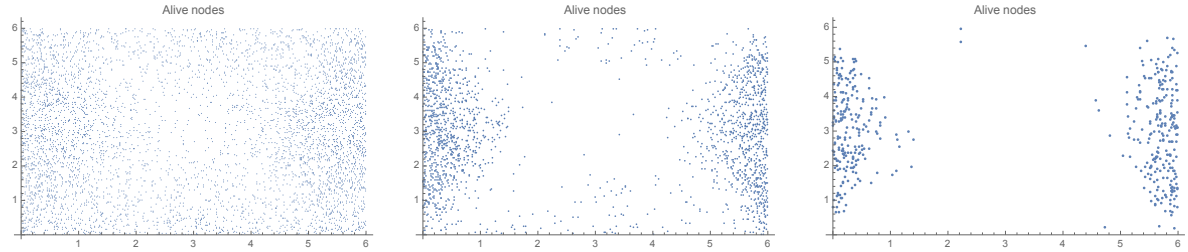
7.1 Sensing nodes lifetime

At this point we proceed computing the lifetime probability of two virtual nodes, one located in $(3, 0)$ and one in $(3, 3)$. In fact, in A, only one activation probability maximum exists, located at the center of the square ($p(3, 3) = 0.33$). Given the uniform condition on the entrance and exit edges, infinite trajectories T may be found, all crossing $(3, 3)$. All of them exhibit the minimum activation probability on the entrance and exit edges, which is the same for all. We hence consider, as second point of interest, $(3, 0)$. Equation 8 is plotted for $l_{max,i} = 100$, $l_{eh,i} = 5$, $\gamma = 1$ (hence $\gamma_{(3,0)} = 0.17$ and $\gamma_{(3,3)} = 0.33$), as a function of l_i in Figures 6a and 6b for positions $(3, 0)$ and $(3, 3)$, respectively. As l_i approaches the value of $l_{max,i}$ the probability of being capable of handling incoming events goes to zero. Please note both Figures 6a and 6b represent complementary cumulative distribution functions, as a function of l_i, as $l_{EH,i} = l_{max,i} - l_i$.

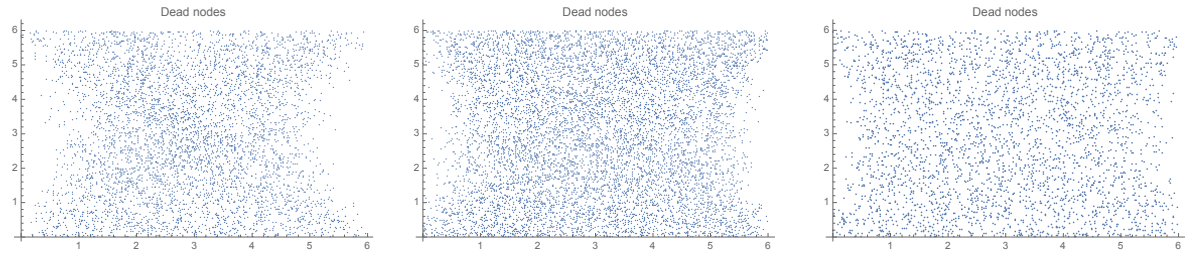
7.2 Cluster head energy requirements

To quantify the energy requirements of the cluster head, we first analyze the number of detections that occur in correspondence of one traversal resorting to Equation 6 and to our simulator. Figures 7a, 7b and 7c, in particular, show the empirical probability of being detected by a given number of nodes in corrispondence of a traversal: as the mission time increases, more nodes deplete their battery resources and the probability of not being detected by any sensing node increases. In Figure 7d we, instead, show that our model perfectly matches simulation results, in the case where sensing nodes are equipped with an infinite amount of energy.

We now assume battery lifetime the cluster head consumes in correspondence of an event is set to $l_{eh,CH} = 10$. The 10-percentile of lifetime of nodes located in $(3, 0)$ and $(3, 3)$ is instead obtained utilizing Equation 8: $l^{10\%}_{(3,0)} = 45$, whereas $l^{10\%}_{(3,3)} = 30$. Such values amount to two possible hypothetical mission times which we will assume from now on. Resorting to our heuristic function reported in Equation 9, for mission times 30 and 45, $\tilde{l}_{EH,CH}$ results equal to 8,418 and 12,233, respectively. We now aim at understanding how well such heuristic function works, utilizing fraction of such values as initial battery lifetime in 1,000 simulation runs for each row reported Tables 1 and 2, where $l^{10\%}_{CH}$ amount to the lifetime of the cluster head obtained as the 10-percentile of the 1,000 simulation results. Tables 1 and 2 report $l^{10\%}_{CH}$ values

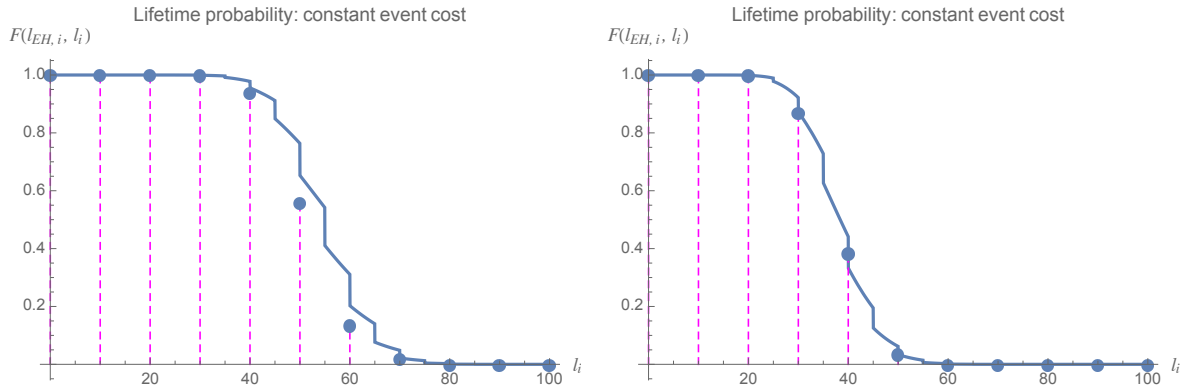


(a) Alive nodes, time = 20, mission time = 80. (b) Alive nodes, time = 40, mission time = 80. (c) Alive nodes, time = 60, mission time = 80.



(d) Dead nodes, time = 20, mission time = 80. (e) Dead nodes, time = 40, mission time = 80. (f) Dead nodes, time = 60, mission time = 80.

Figure 5: Spatial distribution of alive and dead nodes as a function time, before the end of mission time.



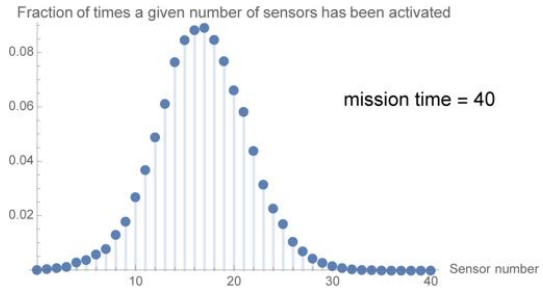
(a) Node located in $(3, 0)$. (b) Node located in $(3, 3)$.

Figure 6: Computing the lifetime of a sensor network: model (continuous line) vs simulation (dots).

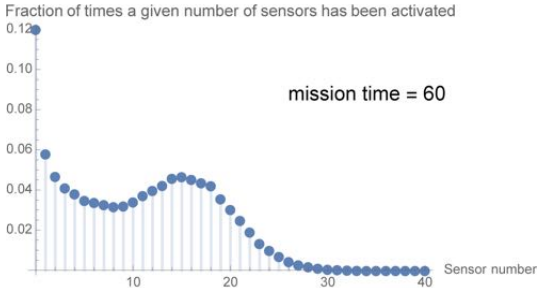
as higher fractions of the computed battery lifetime are employed. For mission time 30, 80% of the battery lifetime given by our heuristic is sufficient to probabilistically guarantee that in the 90% of cases the cluster head will also last at least 30 units of time. Something similar happens also when the mission time is equal to 45, thus confirming the fact that the heuristic rule provided in Equation 8 tends to return amounts of battery lifetime that are larger than those that are really necessary.

7.3 System lifetime: models vs. simulation

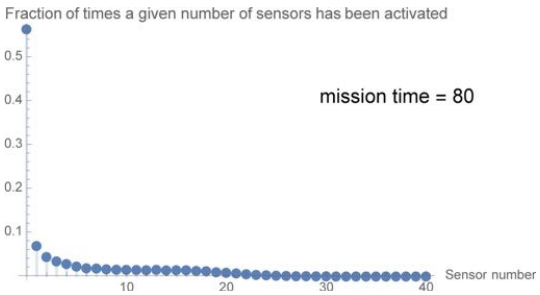
Our analysis, to this point, tells us that with an initial maximum battery lifetime $l_{max,i} = 100$ we may obtain a lifetime of 30 units of time, when taking as a reference node $(3, 3)$ and equipping the cluster head with an initial battery lifetime greater than $l_{max,CH} = 6,764$. If, instead, we take as a reference the sensing node located in $(3, 0)$, such lifetime grows to 45 units of time, when equipping the cluster head of a battery lifetime that is at least equal to $l_{max,CH} = 8,608$. The two different reference nodes return remarkably different lifetime values and cluster head battery lifetime requirements, thus of paramount importance is to understand which, of the considered nodes, may be considered a reliable indicator of the sensor system lifetime. To this aim, we designed a test that builds upon the results plotted in Figure 5. In essence, in our simulations, after an event is handled, we check whether a potential undetected path, i.e., an open path, has become available. However, to limit its computation time, we limit our search to all the feasible paths containing $(3, 3)$. When fixing the mission time to 30 units of time, we find that no open path becomes available. When the mission time amounts to 45, instead, we obtain a 10-percentile of 41.5 units of time (i.e., in the 90% of cases an open path will become available after 41.5) and a minimum value of 28.6. The statistical values of interest are plotted in Figure 8 for different mission times. Note that for mission times 45, 50 and 60 the maximum values amount to 45, 50, and 60 respectively, as simulations are



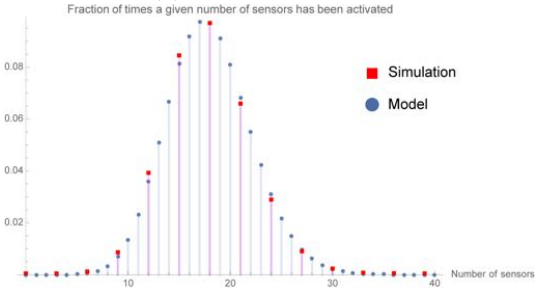
(a) Mission time = 40.



(b) Mission time = 60.



(c) Mission time = 80.



(d) Number of detections probability during an ideal traversal (all nodes always on): model vs simulation comparison.

Figure 7: Number of detections per traversal probability when sensor nodes stop working as they deplete their batteries (Figures 7a, 7b and 7c). Ideal situation, when nodes never die Figure 7d.

CH battery lifetime used in simulation (% of $\tilde{l}_{max,CH}$)	**10-percentile of CH lifetime $l_{CH}^{10\%}$ obtained running 1,000 simulations**
5,081 (60%)	22
5,923 (70%)	26
6,764 (80%)	30

Table 1: Cluster head 10-percentile lifetime obtained from 1,000 simulations for mission time = 30 as a function of different fractions of the initial battery lifetime value ($\tilde{l}_{max,CH} = l_{CH} + \tilde{l}_{EH,CH} = 30 + 8,418 = 8,448$).

interrupted at such times. Beyond mission time 60, instead, an undetected path opens up always before the completion of mission time. In this Figure, we may also observe that the median and 10-percentile values (49.2 and 37.7, respectively) grow up to 50, falling beyond this value. As the mission time increases, less lifetime resources are available for target detection and message transmission. For what, instead, concerns our original question, which sensing node appears as the best sensing system lifetime indicator, we observe that no open path becomes available when the mission time equals 30, whereas in the 90% of cases a path opens after 41.5 units of time when the mission time amounts to 45. To be thorough, we find that in the 82% of cases a breach is open at or after 45 units of time. In essence, the lifetime value obtained observing node $(3, 3)$ appears too low and conservative, whereas the lifetime value obtained observing $(3, 0)$ is slightly optimistic.

7.4 Shaping the network with sequential activations

In order to verify whether the proposed sequential activa-

CH battery lifetime used in simulation (% of $\tilde{l}_{max,CH}$)	**10-percentile of CH lifetime $l_{CH}^{10\%}$ obtained running 1,000 simulations**
6,161 (50%)	28
7,385 (60%)	37
8,608 (70%)	45

Table 2: Cluster head 10-percentile lifetime obtained from 1,000 simulations for mission time = 45, as a function of different fractions of the initial battery lifetime ($\tilde{l}_{max,CH} = l_{CH} + \tilde{l}_{EH,CH} = 45 + 12,233 = 12,278$).

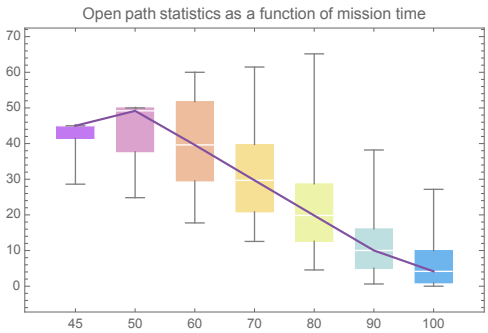


Figure 8: Maximum, 90-percentile, median, 10-percentile and minimum value obtained for the open path variable from 1,000 simulations in correspondence of different mission time values.

tion mechanism works, we again resort to simulations. We here refer to the same sensor field considered so far, shonw in Figure 9, where we highlight the two subareas, A_1 and A_2, which will be considered to provide a preliminary assessment of the efficacy of Algorithm 2. We here consider a mission time of 60 units of time, leaving the other parameters employed to model the network unaltered.

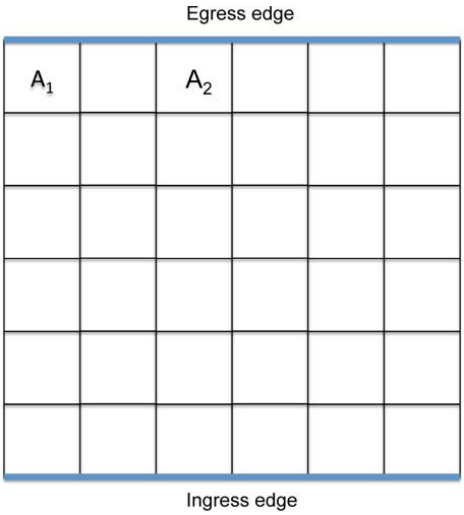


Figure 9: Sensor field.

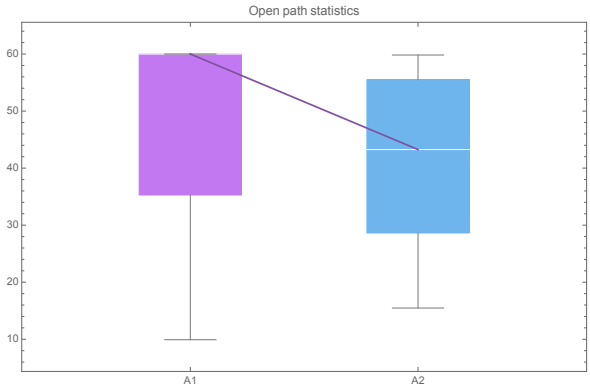


Figure 10: Maximum, 90-percentile, median, 10-percentile and minimum value obtained for the open path variable from 1,000 simulations for the two subareas of interest.

Figure 10 provides the open path statistics for the two subareas, when no sequential activation mechanism is employed. As expected, what occurs is simply the fact that breach detection capabilities last longer in A_1 than in A_2 (this is clearly no surprise). Resorting to Algorithm 2 we create two groups, a first one which is active during the first half of mission time, a second one which instead covers A_2 during the second half. Unlike what indicated in Algorithm 2, we implemented a relaxed version where we do not verify whether the two groups of nodes provide full coverage for the area of interest. Nodes, in fact, are here chosen randomly, as the number of nodes that are dropped in the area (1,000 sensors), and, the ratio between sensor nodes' detection range (0.5 units) and the subareas maximum dimension

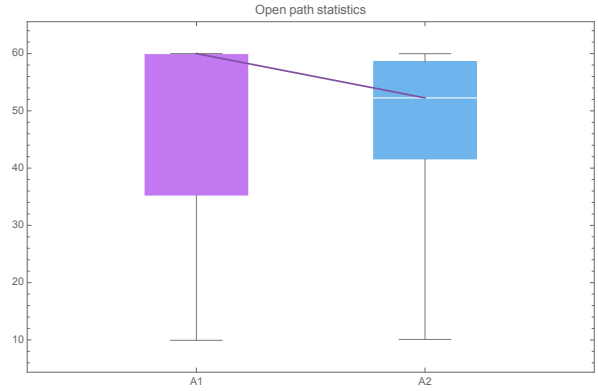


Figure 11: Maximum, 90-percentile, median, 10-percentile and minimum value obtained for the open path variable from 1,000 simulations for the two subareas of interest utilizing a sequential activation mechanism in A_2.

(1.44 units) indicate that with a small number of nodes full coverage can be obtained.

This said, in Figures 10 and 11 we provide the statistic obtained for the open path variable in case no sequential activation mechanism is used and a sequential activation mechanism is used, respectively. As expected, utilizing the proposed methodology, it is possible to extend the duration of the breach detection functionality of the network.

8. CONCLUSION

In this work we modeled the lifetime of a clustered sensor network deployed for target detection. Compared to other approaches, we here considered the influence that the target mobility model has on the duration of surveillance operations. The validity of the adopted approach has been proven confronting our models with simulation results. We have then proceeded analyzing the network from the point of view of an adversary, who might decide to exploit the paths that first become available for undetected traversals. To combat such phenomenon, we introduced a sequential activation mechanism that can be put to good use to shape the duration of the barrier coverage functionalities. Preliminary simulation results reveal that such functionality may be considered when designing a sensor network that should not be optimized in terms of duration, but in terms of its capability of presenting an unexpected monitoring scenario to a hostile intruder.

9. REFERENCES

[1] Akyildiz, I., Sankarasubramaniam, Y., and Cayirci, E. A survey on sensor networks. *IEEE Communications Magazine 40*, 8 (aug 2002), 102–114.

[2] Anastasi, G., Conti, M., Di Francesco, M., and Passarella, A. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks 7*, 3 (may 2009), 537–568.

[3] Aylor, J. System lifetime optimization for heterogeneous sensor networks with a hub-spoke technology. *IEEE Transactions on Mobile Computing 3*, 3 (jul 2004), 286–294.

[4] Bruneo, D., Distefano, S., Longo, F., Puliafito, A., and Scarpa, M. Evaluating wireless sensor node

longevity through Markovian techniques. *Computer Networks 56*, 2 (2012), 521–532.

[5] Butun, I., Morgera, S. D., and Sankar, R. A Survey of Intrusion Detection Systems in Wireless Sensor Networks. *IEEE Communications Surveys & Tutorials 16*, 1 (2014), 266–282.

[6] Cardei, M., Thai, M., Yingshu Li, Y., and Weili Wu, W. Energy-efficient target coverage in wireless sensor networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* (2005), vol. 3, IEEE, pp. 1976–1984.

[7] Chakraborty, A., Rout, R. R., Chakrabarti, A., and Ghosh, S. K. On Network Lifetime Expectancy With Realistic Sensing and Traffic Generation Model in Wireless Sensor Networks. *IEEE Sensors Journal 13*, 7 (jul 2013), 2771–2779.

[8] Chen, A., Kumar, S., and Lai, T. H. Designing localized algorithms for barrier coverage. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking - MobiCom '07* (New York, New York, USA, 2007), ACM Press, p. 63.

[9] Dietrich, I., and Dressler, F. On the lifetime of wireless sensor networks. *ACM Transactions on Sensor Networks 5*, 1 (2009), 1–39.

[10] Donatiello, L., and Marfia, G. Leveraging on Mobility Models for Sensor Network Lifetime Modeling. In *Proceedings of the 14th ACM International Symposium on Mobility Management and Wireless Access - MobiWac '16* (New York, New York, USA, 2016), ACM Press, pp. 155–162.

[11] Donatiello, L., and Marfia, G. Modeling and shaping the lifetime of target detection sensor networks. In *10th EAI International Conference on Performance Evaluation Methodologies and Tools* (2016).

[12] Gambs, S., Killijian, M.-O., and del Prado Cortez, M. N. Next place prediction using mobility Markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility - MPM '12* (New York, New York, USA, 2012), ACM Press, pp. 1–6.

[13] Jung, D., Teixeira, T., and Savvides, A. Sensor node lifetime analysis. *ACM Transactions on Sensor Networks 5*, 1 (feb 2009), 1–33.

[14] Kumar, S., Arora, A., and Lai, T. On the lifetime analysis of always-on wireless sensor network applications. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.* (2005), IEEE, pp. 186–188.

[15] Kumar, S., Lai, T. H., Posner, M. E., and Sinha, P. Maximizing the Lifetime of a Barrier of Wireless Sensors. *IEEE Transactions on Mobile Computing 9*, 8 (aug 2010), 1161–1172.

[16] Latré, B., Braem, B., Moerman, I., Blondia, C., and Demeester, P. A survey on wireless body area networks. *Wireless Networks 17*, 1 (nov 2010), 1–18.

[17] Mainetti, L., Patrono, L., and Vilei, A. Evolution of wireless sensor networks towards the Internet of Things: A survey. In *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on* (2011), IEEE, pp. 1–6.

[18] Noori, M., and Ardakani, M. Lifetime Analysis of Random Event-Driven Clustered Wireless Sensor Networks. *IEEE Transactions on Mobile Computing 10*, 10 (oct 2011), 1448–1458.

[19] Tian, D., and Georganas, N. D. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications - WSNA '02* (New York, New York, USA, 2002), ACM Press, p. 32.

[20] Tian, J., Wang, G., Yan, T., and Zhang, W. Detect smart intruders in sensor networks by creating network dynamics. *Computer Networks 62* (2014), 182–196.

[21] Tian, J., Zhang, W., Wang, G., and Gao, X. 2D k-barrier duty-cycle scheduling for intruder detection in Wireless Sensor Networks. *Computer Communications 43* (2014), 31–42.

[22] Veltri, G., Huang, Q., Qu, G., and Potkonjak, M. Minimal and maximal exposure path algorithms for wireless embedded sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems - SenSys '03* (New York, New York, USA, 2003), ACM Press, p. 40.

[23] Wang, Y., Vuran, M. C., and Goddard, S. Stochastic Analysis of Energy Consumption in Wireless Sensor Networks. In *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)* (jun 2010), IEEE, pp. 1–9.

[24] Wu, K., Gao, Y., Li, F., and Xiao, Y. Lightweight Deployment-Aware Scheduling for Wireless Sensor Networks. *Mobile Networks and Applications 10*, 6 (dec 2005), 837–852.

[25] Younis, O., Krunz, M., and Ramasubramanian, S. Node clustering in wireless sensor networks: recent developments and deployment challenges. *IEEE Network 20*, 3 (may 2006), 20–25.

[26] Zhang, H., and Hou, J. On deriving the upper bound of α-lifetime for large sensor networks. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing - MobiHoc '04* (New York, New York, USA, may 2004), ACM Press, p. 121.

[27] Zhang, W., Li, M., Shu, W., and Wu, M.-Y. Smart Path-Finding with Local Information in a Sensory Field. Springer, Berlin, Heidelberg, 2006, pp. 119–130.

Characterization and Evaluation of Mobile CrowdSensing Performance and Energy Indicators

Riccardo Pinciroli
Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy.
riccardo.pinciroli@polimi.it

Salvatore Distefano
Università di Messina
Contrada di Dio, S. Agata, 98166 Messina, Italy.
sdistefano@unime.it
Kazan Federal University
35 Kremlyovskaya street, 420008 Kazan, Russia.
sdistefano@kpfu.ru

ABSTRACT

Mobile Crowdsensing (MCS) is a contribution-based paradigm involving mobiles in pervasive application deployment and operation, pushed by the evergrowing and widespread dissemination of personal devices. Nevertheless, MCS is still lacking of some key features to become a disruptive paradigm. Among others, control on performance and reliability, mainly due to the contribution churning. For mitigating the impact of churning, several policies such as redundancy, over-provisioning and checkpointing can be adopted but, to properly design and evaluate such policies, specific techniques and tools are required.

This paper attempts to fill this gap by proposing a new technique for the evaluation of relevant performance and energy figures of merit for MCS systems. It allows to get insights on them from three different perspectives: end users, contributors and service providers. Based on queuing networks (QN), the proposed technique relaxes the assumptions of existing solutions allowing a stochastic characterization of underlying phenomena through general, non exponential distributions. To cope with the contribution churning it extends the QN semantics of a service station with variable number of servers, implementing proper mechanisms to manage the memory issues thus arising in the underlying process. This way, a preliminary validation of the proposed QN model against an analytic one and an in depth investigation also considering checkpointing have been performed through a case study.

Keywords

Mobile Crowdsensing; queuing networks; G/G/x; performance; energy consumption; checkpointing.

1. INTRODUCTION

Internet of Things (IoT) is gaining more and more importance thanks to recent advancements in mobile, pervasive sensing and communication technologies. It aims at connecting several physical objects into a unique and larger space, the *cyberspace* [21, 14], opening up new application scenarios that may potentially involve billions devices and users, a big challenge to properly address. A promising attempt in this direction is Mobile CrowdSensing (MCS), which proposes to deploy and run IoT applications on mobiles by actively involving their owners in a volunteer-/crowd-based fashion [13]. This way, MCS allows to reach a large number of users, which may also act as contributors sharing their devices to mainly provide sensing facilities.

Several success stories confirm the effectiveness of the crowdsourcing approach in IoT contexts [7, 10, 23], thus attracting interests from both academic and business communities that are investing resources and efforts to implement middleware mechanisms [25, 29] and to investigate on a commercial exploitation for MCS. This imposes to revise the MCS paradigm, extending its scope to business contexts where service level agreements on functional and non functional properties have to be enforced to meet quality of service (QoS) requirements. Therefore, proper mechanisms and tools for supporting the design and the operation of MCS services are required. In particular, modeling and evaluation techniques dealing with MCS contribution dynamics issues such as churning (i.e., the random and unpredictable join and leave process characterizing contributors) must be specified.

These arrival and service processes, mainly identifying the time behavior of an MCS system, have to be properly characterized indeed, recurring, when possible, to stochastic models able to adequately represent their fluctuations. This way, the resulting model is often not memoryless and the corresponding stochastic process is non-Markovian, implying to take into account related memory issues in its analysis. This is particularly true in MCS systems where the service requests are usually split in simple tasks then assigned to contributor nodes, whose processing could be interrupted due to churning, and thus rescheduled. Sometimes, the results obtained by the partial processing of an interrupted task are not wasted, e.g., for purely sensing activities, or also in the case the MCS services implement checkpointing policies to limit the impact of rescheduling on QoS and performance. To take into account this aspect in modeling, i.e., to properly represent restart from the conditions before the interruptions or from checkpoints, specific memory preservation mechanisms are required.

State space based solutions have been mainly proposed in literature to address these issues [9, 8, 17, 16, 22]. Nevertheless, the main drawback of the state space models is the well-known *"state space explosion"* when dealing with large-complex problems, limiting their applicability, especially in

MCS-IoT contexts where the number of requests and contributors could easily reach thousands or even millions. Furthermore, none of them except [9] takes into account checkpointing policies and related issues, allowing anyway to represent contribution dynamics and churning, but not dealing with the memory problem.

A solution to this problem has been proposed in [24], where we provided a technique based on QNs for dealing with all such issues altogether. In particular we introduced a new service station policy copying with variable number of servers explicitly adopted in the MCS system model to represent the contribution dynamics. Memory policies have been also implemented in order to allow the representation of checkpoint-restart processing, also allowing to consider three different perspectives, i.e., contributors, service provider and end-users, in the evaluation through specific performance and energy metrics.

This paper extends our previous work [9, 24] by improving the overall technique and the model they proposed. New features, such as the request decomposition into tasks, have been considered, revising the original model by introducing new elements (i.e., fork-join). Furthermore, the evaluation part has been significantly extended by new measurements and insigths, reporting on the validation experiments against a stochastic model (a continuous time Markov chain, CTMC) and another QN model developed and evaluated by the JMT tool [3] in the exponential case.

Details are showed in the remainder of this paper, organized as follows: Section 2 provides an overview of MCS, explaining the problem at hand. A QN-based modeling technique for MCS systems is proposed in Section 3, then adopted in Section 4 for the evaluation of an example also exploited to validate the model in the exponential case. Related work is overviewed in Section 5, while final discussions, remarks and cues for future work close this paper in Section 6.

2. A CLOSE-UP VIEW OF MCS

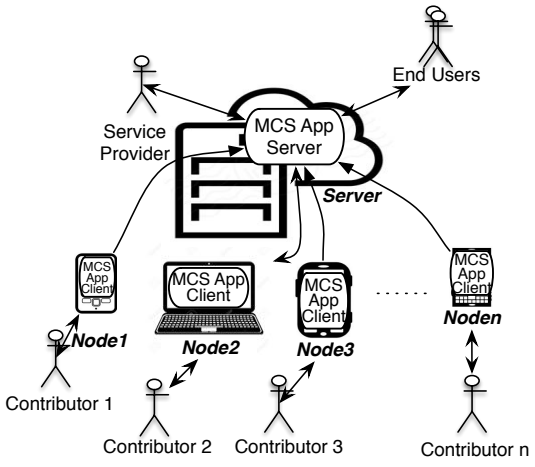


Figure 1: MCS application scenario.

MCS [13] is a computing paradigm where contributors share their mobile devices (e.g., smartphones, tablets, PDAs, laptops, etc.) with specific applications in order to gather and aggregate their data for further processing. From a high level perspective, the architecture of an MCS application mainly implements a client-server model as shown in Figure 1. The MCS application client deployed on each involved node (usually after it has been explicitly downloaded and installed by the contributor on his/her device) produces, (pre-)processes and sends data to the server that collects, aggregates, further processes (if required) and stores such data, providing information and results to the end-users. More specifically, three different stakeholders can be thus identified:

- *contributors*, i.e., the owners of the devices shared to build up the MCS sensing infrastructure;
- the *service provider*, which manages the whole system and application by gathering, aggregating and processing data and results from contributing nodes;
- *end-users*, which use the MCS application to access the services and the information it provides.

These roles are not mutually exclusive and, for example, contributors may also act as end-users and vice-versa.

In the MCS application processing, the main activities are decomposed into simpler *tasks* to be performed by the nodes independently. Each client may elaborate zero, one or many tasks, whose results are collected and recomposed by the application server. To prevent and minimize churning issues due to random and unpredictable joins and leaves of contributors, an active contributing node may periodically send partially processed data (i.e., checkpoints) to restart future elaborations on other available nodes in the case of its departure. Indeed, the application client can sometimes predict the leave of a contributor (e.g., in the case of low battery).

The MCS paradigm has been already successfully adopted in several applications such as OpenStreetmap [15], Waze [28], Nericell [23], as an effective solution for problems related to mobility [30, 10, 7], traffic monitoring [28, 23], public safety [2], Smart Cities [4] environment and pollution monitoring [30], emergency and crowd management [20], to mention but a few. Nevertheless, most of the potential of MCS is still untapped due to the limits of the contribution-based approach, often unreliable and unpredictable and therefore reflecting as high uncertainty in service fruition.

Thus, it is of strategic importance to focus on the overall MCS system rather than on a specific application, since the MCS application performance is strongly and mainly affected by the MCS system performance, reliability and energy related metrics, which are governed by the contribution dynamics. The MCS application client and server processing can be just considered as a constant bias compared to the latter.

Specifically, several metrics of interest can be identified from the different perspectives above identified, as reported in Table 1. A contributor is mainly interested in knowing the impact of contribution on his/her device or node, in terms of computing resource utilization (CPU, memory, storage) and battery charge. Instead, service provider is mainly interested in managing the resources, i.e., maximizing their utilization and reducing power consumption, while reducing the time for processing a task and increasing the system throughput. This also affects the response time for processing a request, which is the main metric of interest for the end-users.

PERSPECTIVE METRIC	*Contributor*	*Service Provider*	*End User*
Resource Utilization	Node	Server	-
Energy Consumption	Battery	Server	-
Response Time	-	Task	Service
Throughput	-	Task	

Table 1: MCS system metrics of interests.

A model or a modeling technique can provide a valid support to the design, deployment and assessment of the MCS applications. However, several aspects must be taken into account to properly represent an MCS system. As said above, the most challenging one is the fluctuation of the number of contributors, which strongly impacts on the MCS underlying infrastructure reflecting on the non functional properties of the applications. Thus, an MCS system model has to primarily take into account the contribution dynamics in the stochastic characterization of both arrival (join) and departure (leave) times of contributors, by using specific probability distributions. A stochastic characterization is anyway required for the whole system, including the end-user requests arrival process and service time. In general these are not Markovian, since their stochastic behavior is not memoryless. Furthermore, the model has also to deal with the complexity of an MCS system, for example, taking into account queueing effects as well as checkpointing policies.

3. MODELING MCS

Starting from the description of Section 2, it is evident the need of a modeling technique that allows to evaluate the metrics of interest identified in Table 1, while dealing with churning issues. This strongly suggests to address the problem in a hierarchical, layered way, separating contribution aspects and concerns from provisioning and fruition ones. To this purpose, based on the scenario of Figure 1, a two-layer modeling technique is proposed splitting the contribution and the processing subsystems. In the contribution subsystem, the contribution dynamics due to node fluctuations joining and leaving the MCS system is mainly represented. The processing subsystem is instead focused on the MCS application requests and tasks processing and therefore on the service dynamics. However the two subsystems are not independent since contributors mainly serve application requests and tasks. Indeed, when a contributor joins the contribution network, a server in the processing subsystem is turned on and can serve incoming requests. The server is turned off when the contributor associated with that server leaves the contribution network. It is important to remark that a connection between a server and a contributor lasts until the latter leaves the system. Then, the server in the processing network may be switched on by another contributor. In other words, at most one active contributor at a time can be associated with any idle server.

Another aspect to take into account in the MCS modeling is the complexity. In an MCS system the number of users and contributors is commonly in the order of thousands, which excludes state space-based models. Due to its well known capability in dealing with complexity we choose the QN formalism.

3.1 The contribution QN

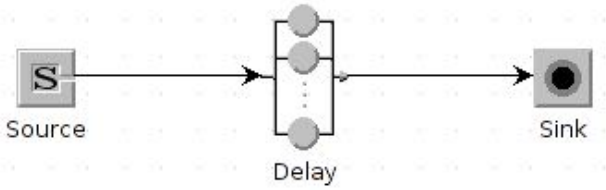


Figure 2: The contribution queuing network submodel.

The QN submodel representing the MCS contribution dynamics is shown in Figure 2. Contributors can join and leave the MCS system randomly and the time they spend in the system is the contribution time. Therefore, they are stochastically represented by general distributions. Thus, a job in the `Delay` station (i.e., a $G/G/\infty$ queue) represents a contributor node that is sharing its resources. The arrival rate to `Delay` and the time a job spends in it may be characterized by any general distribution. They represent the arrivals of contributors in the system and their contribution times (i.e., the time they are available to serve the end-users requests), respectively. As discussed above, the `Delay` station of this network is connected to the `Service Center` one in the processing QN. The number of servers in the processing submodel is equal to the number of jobs in the `Delay` station of the contribution QN.

3.2 The processing QN

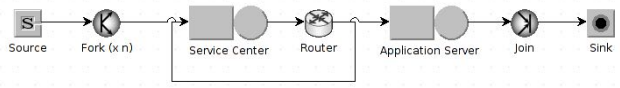


Figure 3: The processing queuing network submodel.

The processing submodel represents the end-user requests incoming to the MCS system, which have to be processed by the contributor nodes. Figure 3 shows the proposed QN submodel that is just a part of the model, the overall QN model is composed of the two submodels of Figures 2 and 3, strictly dependent. The task requests arrival process is characterized by a general distribution and does not depend on the contribution QN. It is modeled by the `Source` station. As well, the `Application Server` station does not depend on the contribution QN and its service time is generally distributed, thus resulting in a $G/G/1$ queue. This station models the commit of a request after it has been completely processed. As discussed above, an end-user request is decomposed into n tasks for processing, represented by the `Fork-Join` elements in the QN, thus splitting an end-user request before being served and recomposing it afterwards in the `Application Server`.

The processing in the `Service Center` station depends on the contribution QN. To represent this dependence, we refer to the `Service Center` as a $G/G/x$ queue with generally distributed arrival and service times and variable number of

servers $x \in [0, \infty)$ specifically representing the contribution churning dynamics. The number of servers x is associated with the number of contributors that are in the contribution system QN of Figure 2. To represent fluctuations on the number of servers for the `Service Center` station, each contributor arrival/departure turns on/off a server.

A task request is served by only one server at a time but, since it could be rescheduled due to churning, it may be served by different servers till it is completely processed. Indeed, a request exits from the `Service Center` after completion or if the i-th server processing the task is switched off, meaning that the associated contributor left the system. In the former case, the task processing results are sent to the `Application Server` by the `Router` to be aggregated with the other $n-1$ tasks of the same end-user request, and after the synchronization (through the `Join` station) further processed before leaving the system. In the latter case, the task request is rescheduled, thus sent back by the `Router` to the `Service Center`, waiting to be processed by an idle server.

Priority policies may be applied in the `Service Center` to prioritize dropped requests. Since the number of available servers may be lower than the number of requests that need to be served, only x requests can be served at the same time by the `Service Center`. This way, a task request that has been interrupted can be rescheduled for processing with a higher priority. The rescheduling has to also take into account possible memory policies and strategies adopted to reduce churning, such as the checkpointing one, thus implementing proper memory mechanisms to restart from checkpoints.

3.3 Measurements

Once the model has been specified, the performance and energy consumption metrics of interest must be identified on it in terms of QN measurements . Referring to Table 1, an end-user is interested in the average *Push time*, i.e., the time needed by the system to process an end-user request, that is equivalent to n task requests. Therefore, this is the time for processing the n task requests and their further elaboration by the application server after aggregation and, considering a generic end-user request i, it can be expressed as:

$$PushTime_i = max_{(j=1,..,n)}(R_i^j) \tag{1}$$

where n is the number of task requests that the system collects before returning some results to the end-user and R_i^j is the response time of the j-th task of the i-th end-user request, obtained by the processing system QN analysis. The average *Push time* is:

$$PushTime = \frac{\sum_{i=1}^{I_{max}} PushTime_i}{I_{max}} \tag{2}$$

where I_{max} is the number of end-user requests processed by the system.

The service provider may be interested in the average system response time R and throughput X. The former is obtained observing each end-user request processing, evaluating how long it spends in the system. The latter as $X = C/T$ where T is the observation time and C the number of completed end-users requests.

Finally, a contributor is interested in knowing the impact of contribution on his/her devices, quantified by the utilization U_C and the battery energy consumed by the MCS application during contribution. To this purpose, we evaluate U_C of each node as:

$$U_C = \frac{\sum_{i=0}^{x_{max}} ActivityTime_i}{\sum_{i=0}^{x_{max}} OnTime_i} \tag{3}$$

where $ActivityTime_i$ is the time spent by the server i in the `Service Center` serving a request, $OnTime_i$ is the total time that server i is available for the MCS application, either busy to serve a task request or idle, waiting for a task request to serve, and x_{max} is the maximum number of server in the `Service Center`. Assuming that the device is mainly used for computations, we evaluate the power consumption P_C of each node as shown in [12]:

$$P_C(U_C) = P_C^{idle} + (P_C^{max} - P_C^{idle})U_C \tag{4}$$

where P_C^{idle} is the power consumption when the device is idle, P_C^{max} is the power consumption of the device when it is fully utilized and U_C is the utilization of the device as obtained by eq. (3). Once the power consumption has been defined, the average energy consumption E_C can be specified as:

$$E_C = P_C \cdot S_C \tag{5}$$

where S_C is the average contributor service time, i.e., the overall time the contributor spends in the system, obtained by the contribution QN of Figure 2. We estimate the percentage of battery depleted by the contributor for sharing his/her resources with the MCS system as:

$$Battery\ consumption = \frac{E_C}{W_C} \tag{6}$$

where W_C is the battery capacity of the node.

4. VALIDATION AND EVALUATION

To explain in details the proposed technique, in this Section we analyze a case study on an MCS system. Therefore, we first discuss the parameters and configuration settings used for the analysis based on the values taken from existing literature when available. This way, we evaluate the model to demonstrate its validity through comparison against other models (analytic and QN), restricting the scope to the exponential case. The exponential assumption is then relaxed to perform further investigations on the MCS system performance and energy aspects from the different NCS stakeholder perspectives.

4.1 Parameters and Configurations

As above discussed in Section 3, each request submitted by the end-user is split into n simpler tasks then assigned and processed by contributor nodes. Once the n tasks have been processed, the provider application server aggregates the results and sends the obtained outcomes to the end-users. To take into account checkpointing-restart policies, we investigate MCS systems with and without memory strategies. In the former case, the checkpoint strategy lets the system keep memory of the task requests processing at restarting, while in the latter the processing restarts from scratch.

As stated above, the parameters used in the experiments have been taken from literature when possible. Specifically, the contributor and request arrivals are exponentially distributed, as well as the service time of the `Delay` and the `Application Server` stations. The service time of each

server in the `Service Center` station is characterized by a 3-stage Erlang, similarly to [9]. The average service time of the `Application Server` S_{AS} and the `Delay` S_C is the same in all experiments, i.e., 10 seconds and 30 minutes, respectively. The inter-arrival time of the two networks (i.e., A_C and A_R for contribution and processing QN, respectively) and the average service time of a server in the `Service Center` S_{SC} have been changed in the experiments: $A_C = \{1, 10, 20, 30\}$ minutes; $A_R = \{100, 200, 300\}$ minutes; $S_{SC} = \{5, 10, 15, 20, 25, 30, 35, 40\}$ minutes.

The strategy used in the `Service Center` to select the next request to be processed is a FIFO with priority. The priority of a request increases every time the request is sent from the server back to the queue. The job with the largest priority is the first to be served. If two or more jobs have identical priority, FIFO strategy is adopted. The number of task requests that must be committed before the system returns some results to end-user is $n = 10$ (i.e., the number of tasks generated by each end-user request).

4.2 Model Validation

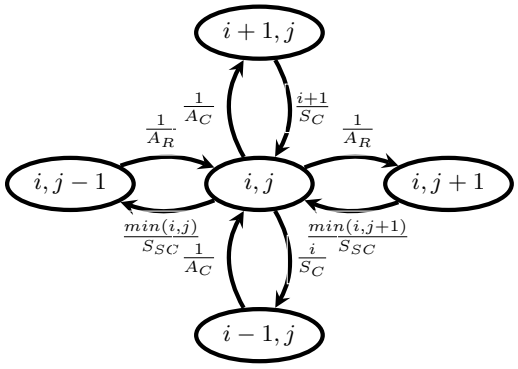


Figure 4: Part of the CTMC of the MCS system used for validation.

To validate the proposed model, we compare it against the CTMC of the MCS system shown in Figure 4, thus restricting the validation scope to the exponential case. Each state of this CTMC is characterized by a pair (i, j) where i and j are the number of contributors and requests in the MCS system, respectively. When the number of contributors changes, we can have a transition to state $(i-1, j)$ at rate i/S_C if a contributor leaves the system or to state $(i+1, j)$ at rate $1/A_C$ if a new one joins the MCS system. If a request is completed the state can move from (i, j) to $(i, j-1)$ with rate $min(i, j)/S_{SC}$ or to $(i, j+1)$ with rate $1/A_R$ if a request enters the `Service Center`.

To validate our QN model against the CTMC, all the QN time parameters must be approximated by exponential distributions. This way, the service time of each server in the `Service Center` is exponentially distributed with mean time equal to S_{SC} and the `Fork/Join` stations of the processing QN are neglected, thus the new A_R is given by the original one divided by the number of task requests n. Furthermore, for validation purposes, we evaluated only one case, characterized by $A_R = 20$ minutes and $A_C = 10$ minutes (i.e., 3 contributors into the system on average). We also truncated the state space, assuming that both the number of contributors and the number of requests cannot be larger than 20. This does not affect the results since the system we are considering for validation rarely has a number of contributors and requests larger than 20.

We consider two different discrete event simulation tools for the validation phase. OMNeT++ [27], an extensible, modular, component-based C++ simulation library and framework used for network simulations, and JMT [3], a comprehensive framework for performance evaluation of QN models. All the performance indexes have been estimated with 99% confidence intervals.

The results obtained by analyzing these models, mainly referred to the utilization, the system response time and the average number of requests in the `Service Center`, are shown in Figure 5. From Figures 5(a), 5(b) and 5(c) we can definitely argue that the performance of the MCS system obtained by the CTMC and the QN models are almost identical. Indeed, Figure 5(d) shows the relative error

$$\frac{|\Theta_{CTMC} - \Theta_{OMNeT++}|}{\Theta_{CTMC}} \qquad \Theta = \{U, R, N_{SC}\},$$

of the QN OMNeT++ model against the CTMC. The maximum relative error for all the considered indexes is always lower than 4%. In particular, the average relative errors for the utilization and the system response time are lower than 1%, while it is lower than 1.26% for the number of requests in the `Service Center`. Similar results can be obtained by the QN model evaluated through JMT.

4.3 Evaluation

After validation, the QN model has been used to perform parametric experiments by varying one between A_C and A_R, while keeping the other constant, and relaxing the exponential assumption. The MCS system has been analyzed varying A_C (or A_R) and setting $A_R = 200$ minutes (or $A_C = 20$ minutes). In the experiments also S_{SC} has been varied, assuming that the tasks allocated to the server may have different complexity and service demands, thus performing tests where 2 out of 3 parameters are variable.

With regard to the energy measurements, in the experiments we assume $P_C^{idle} = 0.268W$, $P_C^{max} = 1W$ as the idle and the maximum power consumption values [5] and $W_C = 7.77$ Watthour as the battery capacity for all the devices. The MCS system has been observed for $T = 7$ days.

In order to evaluate the QN model described in Section 3, considering the non-exponential parameters and the performance indexes there specified, we extended OMNeT++ implementing the dependency between the contribution and the processing submodels, and modifying the `Job` implementation to consider different memory strategies as discussed in Section 2. The results thus obtained are discussed in the following Sections, taking into account the different MCS stakeholders perspectives.

4.3.1 Contributor

Contributors are mainly interested in quantifying the impact of the contribution on their resources. It can be evaluated by the utilization and the battery consumption of their nodes. We analyze these quantities with eq. (3) and (6). Results are shown in Figure 6, where the average utilization per contributor as a function of the average service time of `Service Center` is depicted. In Figures 6(a) and 6(b) different inter-arrival time of the contributors are considered with requests arrival time of 200 minutes and $n = 10$,

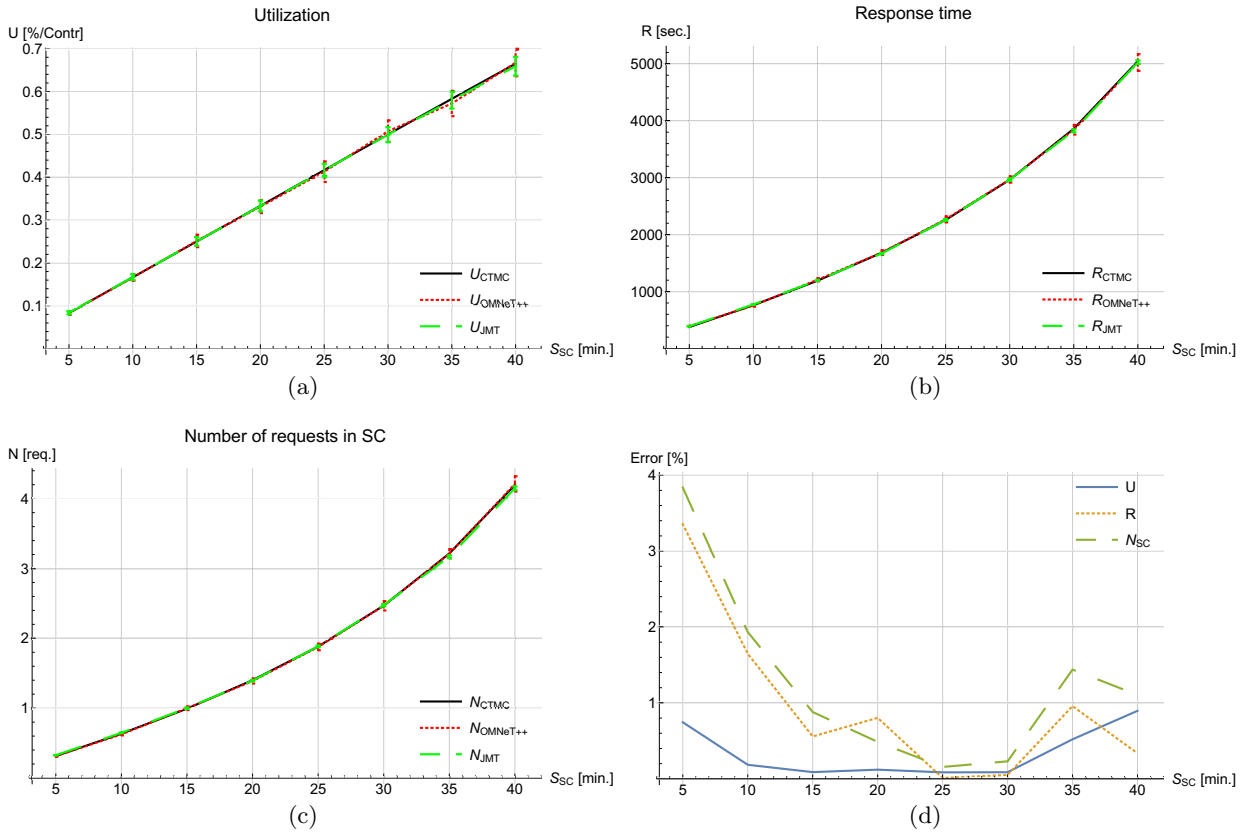


Figure 5: Performance indexes of the MCS system obtained by the CTMC and two QN simulations (OMNeT++ and JMT). Utilization (a), system response time (b), number of requests in `Service Center` (c) and relative errors of OMNeT++ QN results against CTMC ones (d).

comparing two strategies (i.e., checkpoint and no memory, respectively). As expected, the nodes utilization decreases with the contribution inter-arrival time since the total number of contributors (i.e., $N_C = S_C/A_C$) increases when the inter-arrival time A_C decreases, being the average contribution time constant, i.e., $S_C = 30$ minutes. Furthermore, if a checkpoint memory strategy is adopted, the utilization of each device is lower than the service without memory, since contributors have to work more in the latter case. In Figures 6(c) and 6(d) the inter-arrival time of contributors is set to 20 minutes, whereas the request arrival time varies. Also in this case the checkpoint contributor utilization is lower than the no memory one. A higher utilization is also observed when the request inter-arrival time is small since a larger number of requests is in the system.

The utilization is then used to investigate the power absorption through eq. (4), by which the energy consumption is derived. For this reason, energy consumption curves have the same trend of utilization ones and have not been plotted for the sake of space. However, considering the parameters given in Section 4, the battery depletion for a 30 minutes contribution is never greater than 7%, when $U = 100\%$.

4.3.2 *Service provider*

Several interesting metrics for the MCS service provider could be obtained using the QN model. One of them is the number of contributors in the system, since the number of end-users requests that can be served depends on it. Figure 7 shows the contribution dynamic for $A_C = 1$ minute and $S_C = 30$ minutes. The evolution of the contribution QN is represented only for a day in stationary conditions. In this case, the number of contributors is in the range between 14 and 47, and its average is 30.

Similarly, the system utilization is important to identify saturation conditions (i.e., when the utilization of each server is 100%) where the performance degrades and the requested QoS cannot be satisfied, driving to SLA violations. The system response time is plotted in Figure 8 on the service time, also considering different memory strategies. In Figures 8(a) and 8(b) the arrival time of end-user requests is set to 200 minutes and the contributor one varies. In the former, the checkpoint memory strategy is used, in the latter no memory strategies are adopted. It can be observed that the system response time is inversely proportional to the number of contributors in the system. Indeed, if enough contributors are into the system, the task requests are served without interruption and they are completed soon. Instead, when the number of the contributors is too low, each task request may spend several time waiting to be served. This behaviour worsens without checkpointing, since a task request must be processed from scratch if rescheduled.

Figures 8(c) and 8(d) show the system response time for the requests when the contributor arrival time is 20 minutes and the request inter-arrival time may vary. Similar considerations to the previous case can be drawn. Indeed, the system response time is also in inverse proportion to the

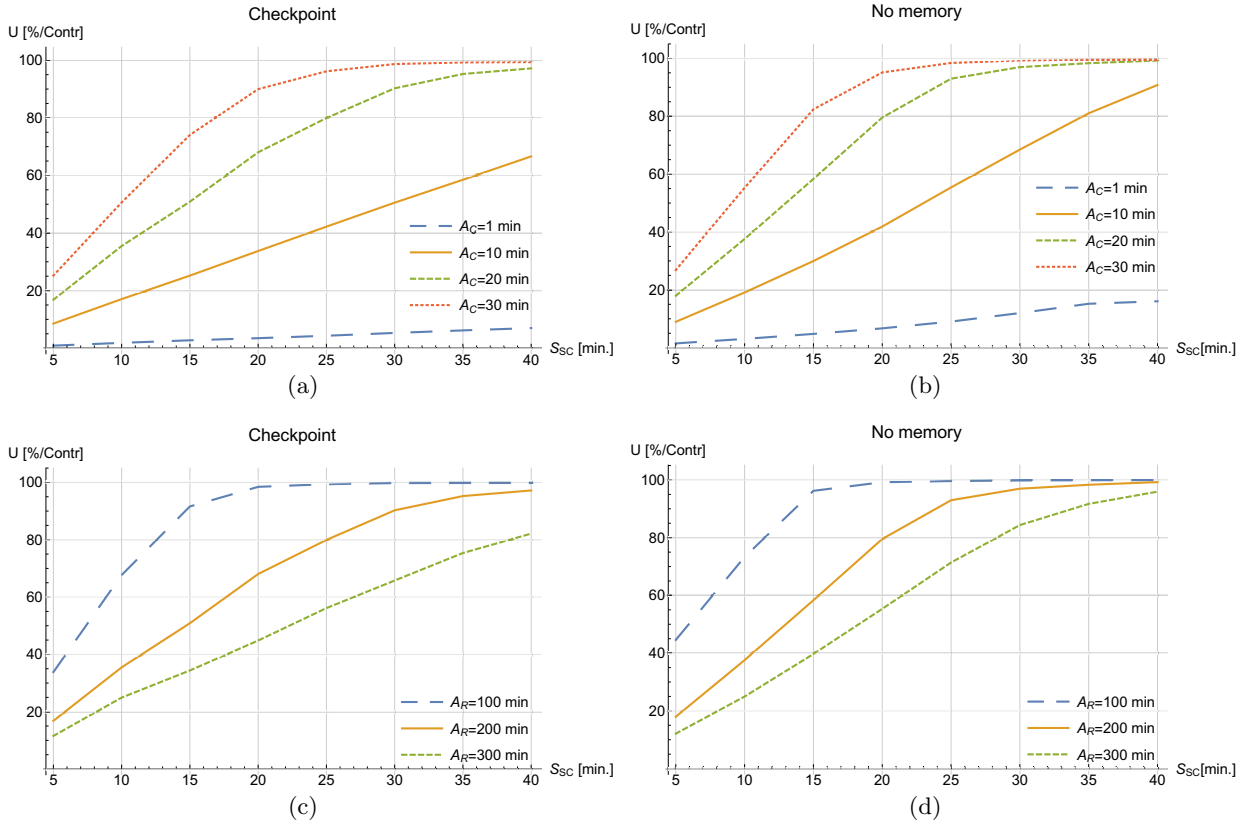


Figure 6: Average utilization per contributor, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d).

request arrival time.

To be sure to not incur in penalties, the provider is also usually interested in the system throughput, since the provisioning is often constrained by SLAs on this performance metric. The system throughput is shown in Figure 9 as a function of the service time, where the checkpoint memory strategy is compared to the no memory one also considering different inter-arrival time for contributors and requests. A larger number of contributors (i.e., smaller value of contributor arrival time) lets the system provide the largest throughput also when complex requests are in the system. Being an open QN model, if the system is not saturating, the average requests throughput is equal to the request arrival rate. The throughput has its maximum value when the tasks to be completed are simple, whereas it decreases when the requests are complex. Also in this case the checkpoint memory strategy positively impacts on the MCS system, ensuring larger throughput than without memory.

4.3.3 *End-user*

End-users are mainly interested in the push time, i.e., the time elapsed since the submission of the end-user request to the result feedback returned by the MCS system. In our example, the system has to process $n = 10$ task requests before returning the results to the user. The push time thus evaluated by the QN model, as specified by eq. (1) and (2), is shown in Figure 10 on the service time, comparing checkpoint and no memory strategies, considering different values of either contributors or requests inter-arrival time. In Figures 10(a) and 10(b) the contributors arrival time may vary. Considering requests inter-arrival time of 200 minutes, the push time behaviour is similar to the response time one, even if its absolute values are larger. The push time increases when the requests are complex (i.e., for larger service time values) or when the number of available contributors is low (i.e., for larger A_C values). It is worth noticing that the larger the number of requests in the system, the faster the push time increases. Indeed, requests with the same complexity can easily affect the system performance when several other requests are already in the system. In both the considered cases, checkpoint strategy provides the best results.

4.3.4 *Further results*

In this section we want to investigate on the likelihood of exponential and non exponential models by comparing the CTMC validation results against those obtained by a non-Markovian QN model characterizing by an Erlang distribution the `Service Center` service time and also including the `Fork/Join` stations to represent the end-user request decomposition. To this purpose, we compare the system response times of the two models with $A_C = 10$ minutes and $A_R^{nex} = 200$ minutes, where end-user requests are decomposed into $n = 10$ tasks (thus $A_R^{ex} = A_R^{nex}/n = 20$ minutes for the exponential model). For non-exponential model we considered both the case with checkpoint memory strategy

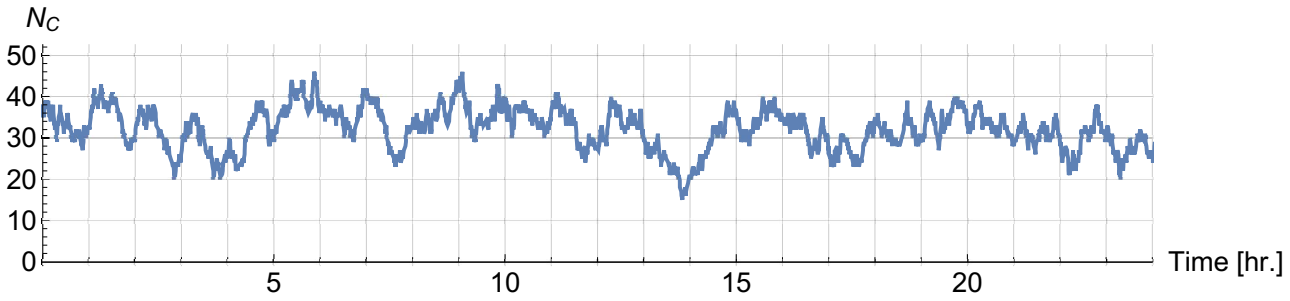


Figure 7: Number of contributors in the MCS system in the case of $A_C = 1$ minute and $S_C = 30$ minutes, for 24 hours of observation.

and the one without memory, to be compared with the exponential model that, being memoryless, cannot represent any memory policy. The results thus obtained, shown in Figure 11, clearly highlight that exponential models cannot be used as a valid approximation of non-exponential ones. The exponential values are closer to the ones obtained by the non exponential model when considering the checkpointing memory policy. This allows us to argue that dealing with non exponential distributions and related memory policies is mandatory to properly represent MCS systems.

5. RELATED WORK

Several works in literature deals with the MCS challenges above identified. With specific regard to performance and energy modeling and evaluation, it is important to investigate on the MCS application behaviours under different scenarios, dealing with complexity and churning issues. In this context, simulation is proposed for the evaluation of MCS system performance in [8], while Karaliopoulos et al. [18] used stochastic model to face the problem of user mobility, selecting users for crowdsensing campaign through an optimization problem. Stochastic models are also used to investigate how to maximize utility [17] and profit [16] in MCS applications. In these works, Lyapunov optimization is used to find the best solution for the considered problem. Liu et al. [19] adopted a CTMC to study the data collected by sensing vehicles, focusing on their spatial distribution. In [9, 22] MCS applications are modeled to analyze their QoS. In particular, the authors adopted an extended version of Stochastic Petri net formalism taking into account contributors and workload fluctuations in MCS system performance and reliability evaluation. The main problem of all such techniques lies on largeness and complexity, in terms of numbers of user requests and contributors. State space based models have intrinsic limitations in dealing with such an issue due to the well know state space explosion problem. Also simulation time could be significantly affected by these parameters, even if in a less critical way than the state space models.

In this paper we recurred to queueing theory to model MCS user and contributors arrival processes, since it allows to represent complex systems with both open and closed workloads. The main issue to address is on the contribution modeling, which is related to the queuing station server. In MCS this parameter is variable in time, due to churning issue, and in the QN domain it means that a technique able to deal with variable number of servers is required. An interesting solution could be provided by QN with breakdowns and repairs [26, 6, 11, 1], considering the departure and the arrival of a contributor as a breakdown and a repair event for a QN station server, respectively. In [26] only closed QN are considered, whereas [6, 11, 1] analyze the performability of an open QN. All these techniques adopts Markov processes to model the failure state of the systems, thus restricting the scope to exponentially distributed sojourn times for MCS contributors, that are considered as faulty/reparable servers. In our work, we proposed to deal with the contributor/server fluctuations representing them as job arrivals in the contribution QN (i.e., contributors) and as servers in the processing network. In this way, we relax the memoryless-exponential limitation on contributors/servers allowing to representation of any kind of distribution in the QN model.

Furthermore, to provide quantitative results from different perspectives (contributors, service provider and end-users), we also took into account energy related metrics on the nodes and servers. Conti et al. [8] identified this aspect as one of the key challenges to motivate participants in contributing. Indeed, since the battery capacity of these devices is often very limited, MCS applications should have a low impact on the battery depletion, letting users be almost unaware of their contributions to MCS. The proposed technique also allows to derive energy indexes from performance metrics, thus providing a multiple-view and a comprehensive analysis of an MCS system.

6. CONCLUSIONS AND FUTURE WORK

In this paper, a technique for evaluating MCS system performance and energy consumption properties is proposed. Through queuing network models, this technique allows to assess corresponding indicators from different perspectives including end-users, contributors and providers. In this way, the contribution dynamics issues due to the random and unexpected leave of contributors, i.e., churning, can be taken into account, as well as related policies for mitigating their effects on the MCS application performance such as the checkpoint memory strategy. To deal with churning dynamics the QN model has been extended to allow variable number of servers in the contribution service station, thus properly representing contributor join and leave fluctuations. The proposed technique therefore overcomes the limits of existing solutions, coping with non exponential distributions stochastically characterizing the contribution, the end-user request and the service time dynamics. To this purpose, memory mechanisms have been implemented providing a powerful mechanisms for the evaluation of advanced policies such as the checkpointing one, allowing to

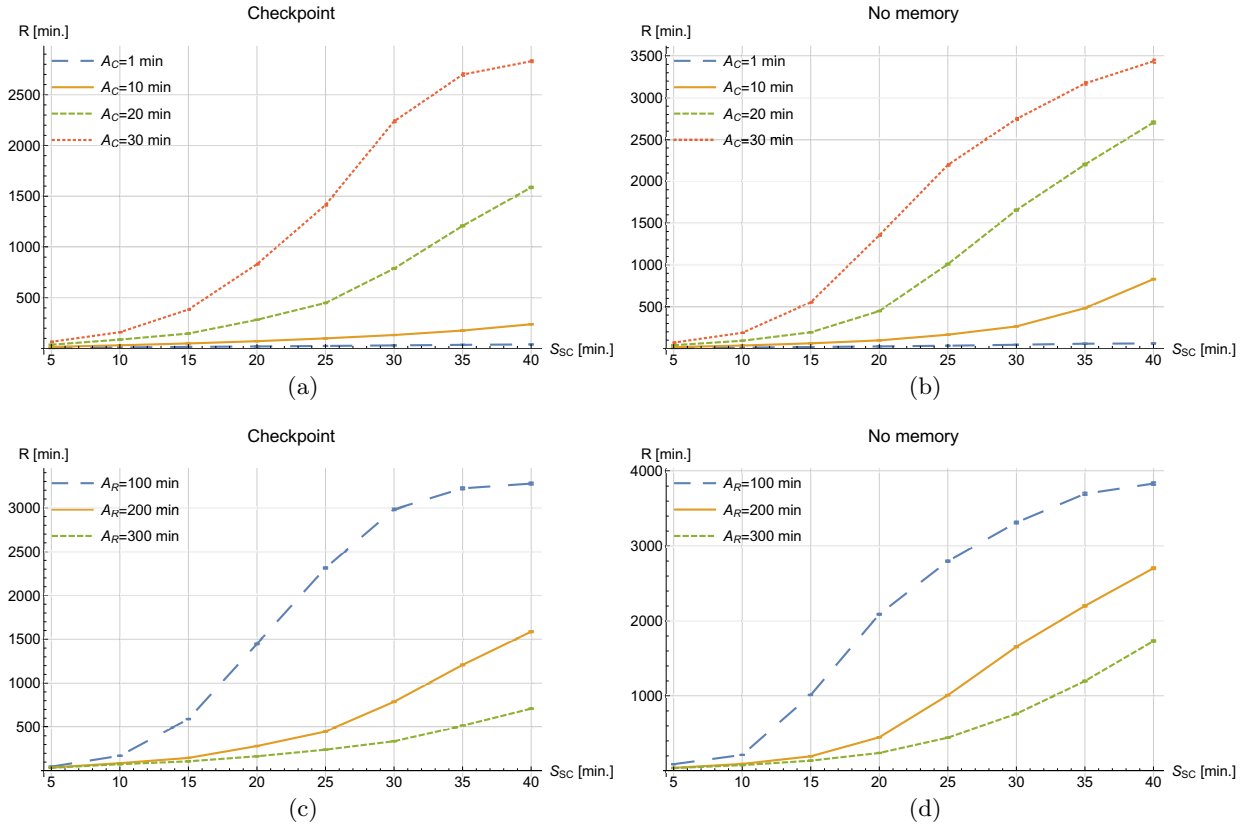


Figure 8: Average system response time for task requests, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d)

properly represent restarts from checkpoints in stateful, non-memoryless processes.

A case study has been discussed to explain by example how to use the proposed technique. In the experiments, our simulation-based implementation, extending OMNeT++, has been validated through a continuous time Markov chain, also comparing the results thus obtained against the ones computed by using the JMT tool to solve our QN model. Then, further evaluation allowed us to investigate the effect of checkpointing strategies on the performance and energy indicators of the MCS system taken into account from the contributor, end-user and provider perspectives. Eventually, the validity, the suitability and the effectiveness of the proposed technique have been demonstrated through a case study.

7. REFERENCES

[1] S. Andradóttir, H. Ayhan, and D. G. Down. Compensating for failures with flexible servers. *Operations Research*, 55(4):753–768, 2007.

[2] E. Aubry, T. Silverston, A. Lahmadi, and O. Festor. Crowdout: A mobile crowdsourcing service for road safety in digital cities. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 86–91, March 2014.

[3] M. Bertoli, G. Casale, and G. Serazzi. Jmt: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, 2009.

[4] G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, and R. Curtmola. Fostering participaction in smart cities: a geo-social crowdsensing platform. *Communications Magazine, IEEE*, 51(6):112–119, June 2013.

[5] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14. Boston, MA, 2010.

[6] R. Chakka, E. Ever, and O. Gemikonakli. Joint-state modeling for open queuing networks with breakdowns, repairs and finite buffers. In *2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 260–266. IEEE, 2007.

[7] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 481–490. ACM, 2012.

[8] M. Conti, C. Boldrini, S. S. Kanhere, E. Mingozzi, E. Pagani, P. M. Ruiz, and M. Younis. From manet to people-centric networking: milestones and open research challenges. *Computer Communications*, 71:1–21, 2015.

[9] S. Distefano, F. Longo, and M. Scarpa. Qos assessment of mobile crowdsensing services. *Journal of*

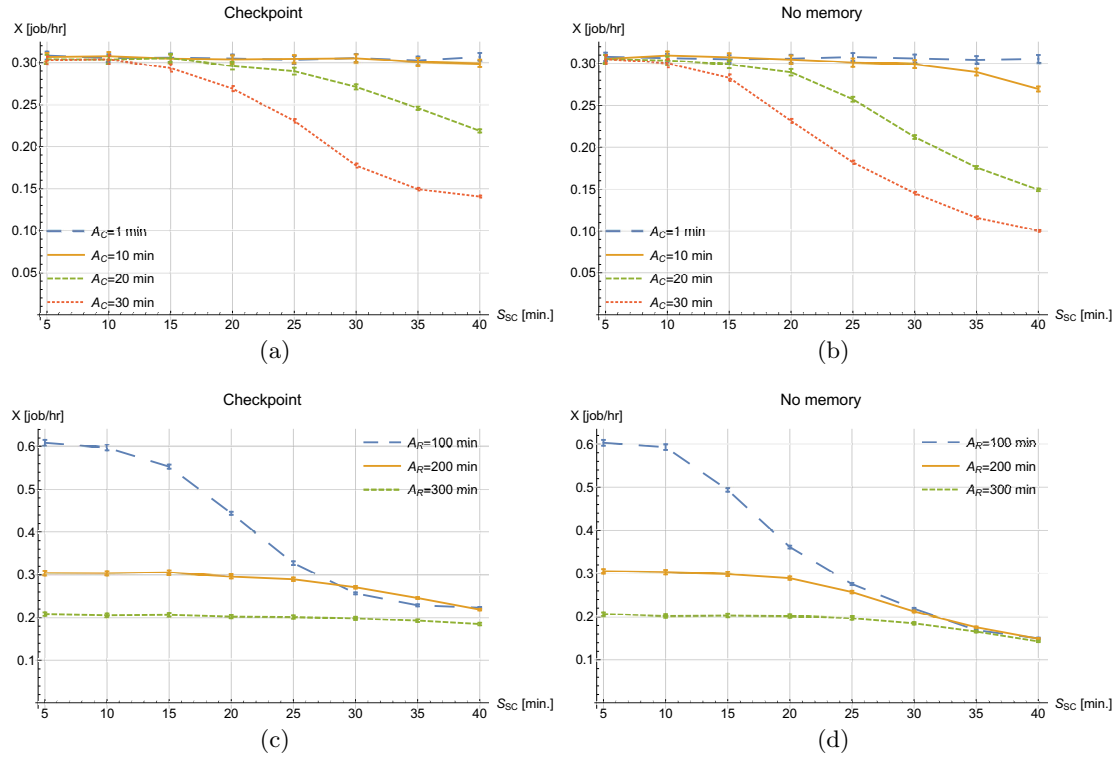


Figure 9: Average system throughput, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d).

Grid computing, 14, 2016.

[10] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):6, 2009.

[11] E. Ever. Fault-tolerant two-stage open queuing systems with server failures at both stages. *IEEE Communications Letters*, 18(9):1523–1526, 2014.

[12] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[13] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, 2011.

[14] B. Guo, D. Zhang, Z. Wang, Z. Yu, and X. Zhou. Opportunistic iot: Exploring the harmonious interaction between human and the internet of things. *Journal of Network and Computer Applications*, 36(6):1531–1539, 2013.

[15] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, Oct 2008.

[16] Y. Han and Y. Zhu. Profit-maximizing stochastic control for mobile crowd sensing platforms. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 145–153. IEEE, 2014.

[17] Y. Han, Y. Zhu, and J. Yu. Utility-maximizing data collection in crowd sensing: An optimal scheduling approach. In *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on*, pages 345–353. IEEE, 2015.

[18] M. Karaliopoulos, O. Telelis, and I. Koutsopoulos. User recruitment for mobile crowdsensing over opportunistic networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 2254–2262. IEEE, 2015.

[19] Y. Liu, J. Niu, and X. Liu. Comprehensive tempo-spatial data collection in crowd sensing using a heterogeneous sensing vehicle selection method. *Personal and Ubiquitous Computing*, 20(3):397–411, 2016.

[20] T. Ludwig, C. Reuter, T. Siebigteroth, and V. Pipek. Crowdmonitor: Mobile crowd sensing for assessing physical and digital activities of citizens during emergencies. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 4083–4092, New York, NY, USA, 2015. ACM.

[21] H.-D. Ma. Internet of things: Objectives and scientific challenges. *Journal of Computer science and Technology*, 26(6):919–924, 2011.

[22] G. Merlino, S. Arkoulis, S. Distefano, C. Papagianni, A. Puliafito, and S. Papavassiliou. Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Future Generation Computer Systems*, 56:623–639, 2016.

[23] P. Mohan, V. N. Padmanabhan, and R. Ramjee.

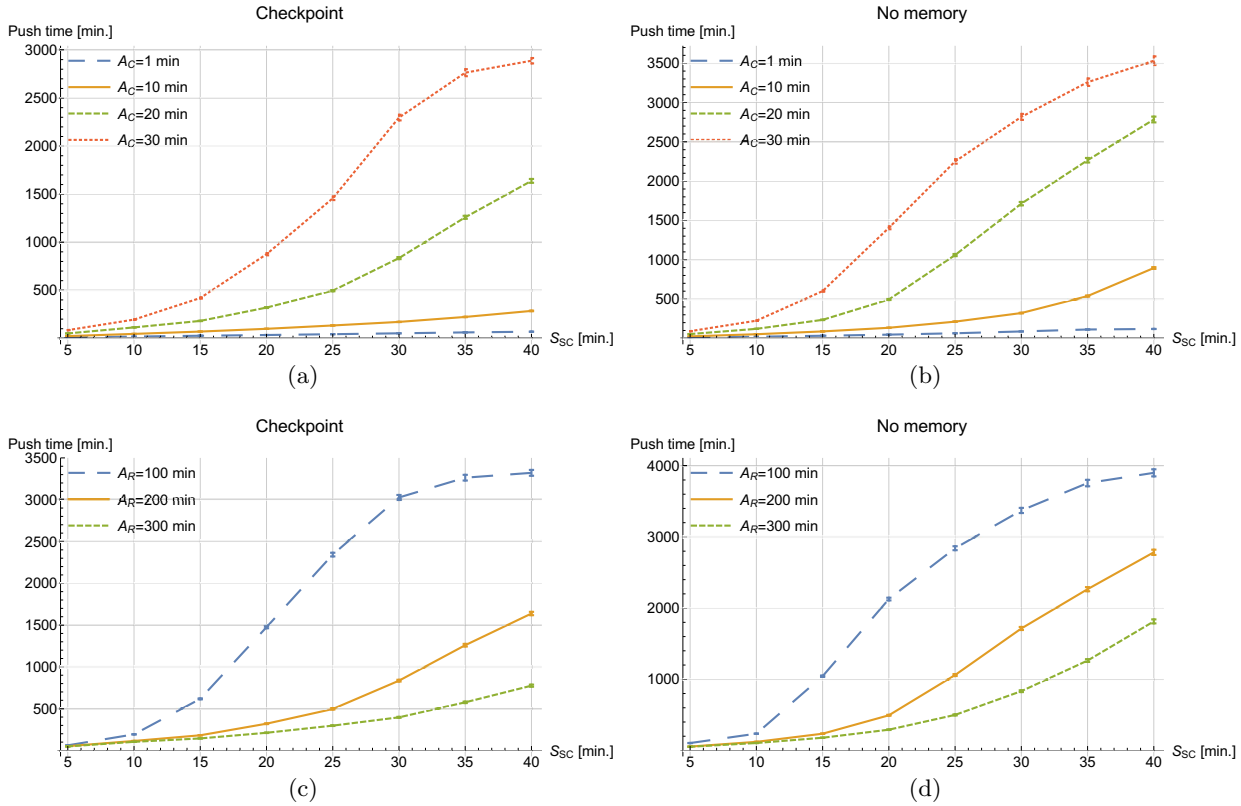


Figure 10: Average push time, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d).

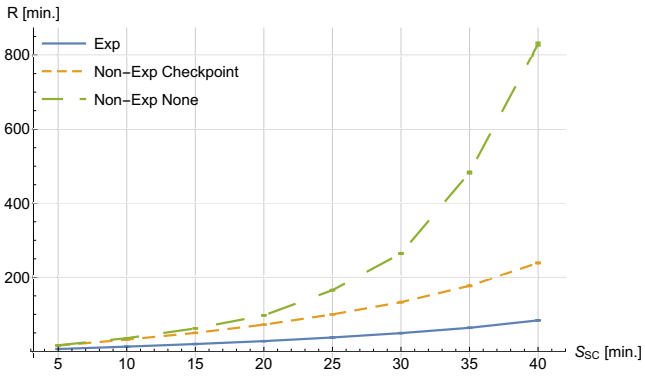


Figure 11: Comparison between the exponential model and the non-exponential one for system response time.

Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[24] R. Pinciroli and S. Distefano. Extending Queuing Networks to Assess Mobile CrowdSensing Application Performance. In *The sixth Workshop of the Italian group on Quantitative Methods in Informatics (InfQ2016) - Valuetools 2016*. ACM.

[25] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 337–350. ACM, 2012.

[26] C. Ramanjaneyulu and V. Sarma. Modeling server-unreliability in closed queuing-networks. *IEEE transactions on reliability*, 38(1):90–95, 1989.

[27] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[28] Waze Mobile. Waze website, 2006.

[29] Y. Xiao, P. Simoens, P. Pillai, K. Ha, and M. Satyanarayanan. Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, page 9. ACM, 2013.

[30] X. Yu, W. Zhang, L. Zhang, V. O. Li, J. Yuan, and I. You. Understanding urban dynamics based on pervasive sensing: An experimental study on traffic density and air pollution. *Mathematical and Computer Modelling*, 58(56):1328 – 1339, 2013. The Measurement of Undesirable Outputs: Models Development and Empirical Analyses and Advances in mobile, ubiquitous and cognitive computing.

Overcoming the lack of kinetic information in biochemical reactions networks

Niccolò Totis
University of Turin, Computer Science Department
Corso Svizzera 185
Turin, Italy
ntotis@di.unito.it

Laura Follia
University of Turin, Molecular Biotechnology and Health Sciences Department
Via Santena 5
Turin, Italy
laura.follia@unito.it

Chiara Riganti
University of Turin, Oncology Department
Via Santena 5/bis
Turin, Italy
chiara.riganti@unito.it

Francesco Novelli
University of Turin, Molecular Biotechnology and Health Sciences Department
Via Santena 5
Turin, Italy
franco.novelli@unito.it

Francesca Cordero
University of Turin, Computer Science Department
Corso Svizzera 185
Turin, Italy
fcordero@di.unito.it

Marco Beccuti
University of Turin, Computer Science Department
Corso Svizzera 185
Turin, Italy
beccuti@di.unito.it

ABSTRACT

A main aspect in computational modelling of biological systems is the determination of model structure and model parameters. Due to economical and technical reasons, only part of these details are well characterized, while the rest are unknown. To deal with this difficulty, many reverse engineering and parameter estimation methods have been proposed in the literature, however these methods often need an amount of experimental data not always available.

In this paper we propose an alternative approach, which overcomes model indetermination solving an Optimization Problem (OP) with an objective function that, similarly to Flux Balance Analysis, is derived from an empirical biological knowledge and does not require large amounts of data. The system behaviour is described by a set of Ordinary Differential Equations (ODE). Model indetermination is resolved selecting time-varying coefficients that maximize/minimize the objective function at each ODE integration step. Moreover, to facilitate the modelling phase we provide a graphical formalism, based on Petri Nets, which can be used to derive the corresponding ODEs and OP. Finally, the approach is illustrated on a case study focused on cancer metabolism.

1. INTRODUCTION

The inherent complexity of biological systems makes the study of their dynamics and behaviours a difficult issue. Within this context, computational modelling can help biologists/clinicians to identify which molecules and interactions have a crucial role on the global behaviour of the system. Many modelling approaches have been proposed in the literature. An appropriate choice among them depends on the specific biological question being addressed and on the available data. An extensive review of the most successful modelling approaches used for metabolic, and more in general for any biochemical network, is reported in [4], where the authors divide these approaches into three classes (i.e. *interaction-based*, *constraint-based* and *mechanism-based* approaches) depending on the used level of abstraction. For instance, *interaction-based* approaches do not consider any quantitative aspect, so that they allow only a topological analysis of the system. Instead, *constraint-based* approaches rely on the assumption of steady state and typically exploit Flux Balance Analysis (FBA) and its extensions [18] to study the system at equilibrium. *Mechanism-based* approaches describe interactions with high level of detail and are the most likely candidates to provide a complete understanding of biochemical dynamics.

In order to build a mechanism-based the modeller has to specify all its characteristics, like, for instance, the structure of the system, the mechanism of the events that produce changes in variables and the mathematical expressions used to represent them, with all their parameters defined. This considerable amount of preliminary information requires the modeller to acquire an extensive knowledge of the biological phenomena and to formulate all the assumptions on which the model relies. In this article we will focus on a particular type of biochemical systems, namely metabolic systems, and on modelling these systems with Ordinary Differential Equations (ODEs). Due to economical and technical reasons, the complete information needed to build the set of ODEs is rarely available, causing the modeller to face situations of indetermination. In this study we will consider in particular those metabolic models in which part of the reactions are fully characterized while the rest are not. Two groups of reactions are thus defined: determined reactions, having well characterized structure, mathematical expression and parametrization, and undetermined reactions, described just by their stoichiometry.

Issues of indetermination have been already deeply studied in literature and solutions have been proposed at different levels, depending on the type of missing information. Tech-

niques of reverse engineering (RE) and Parameter Estimation (PE), for instance, overcome this indetermination by means of available experimental data and optimization algorithms [4, 15, 8].
In our previous work [22] we proposed an alternative approach to deal with situations in which insufficient experimental data hamper the application of PE and RE methods. To overcome the lack of information concerning undetermined reactions we chose to exploit an empirical biological knowledge that helps us formulate some modelling assumptions and define an optimization problem (OP). Since the direct definition of the ODEs and of the OP may be difficult and may require advanced modelling skills, we facilitate this process providing a high level graphical formalism based on Petri Nets (PNs).

In this article, we reviewed the approach presented in [22]. In detail we propose two new solution algorithms to speedup the solution process. Moreover, the performance of these new algorithms, in terms of accuracy and computational cost, are compared with the implementation proposed in [22] through a new set of experiments. The paper is organized as follows. Section 2 gives a general outlook on other methods currently used for similar purposes, while Section 3 provides the necessary background on PNs and OPs. In Section 4 our approach is introduced: a formal definition of our PN extension and the procedure to automatically translate a model described through such a high level formalism into the underlying ODE system and OP are provided. In Section 6 we conclude presenting some future works.

2. STATE OF THE ART

As mentioned in the introduction, situations of inadequate biochemical characterization are traditionally managed with **Reverse Engineering** (RE) or **Parameter Estimation** (PE) computational techniques. When the topology of the network is not known, several techniques of RE can be used. Besides topology, the parametrization of the model represents at present the main area of lacking information. The parameters required for a kinetic model include both the kinetic parameters and the initial conditions. Many parameters have been experimentally calculated with biochemical assays, and have been reported in literature, but many others are still unavailable. As some authors have pointed out, if we consider the parametrization of the mathematical expressions of enzyme activity: "for the foreseeable future, full availability of *true* rate equations for all enzymes is certainly an illusion" [3]. To cope with this situation, many PE techniques have proposed different approaches that aim to automatically infer model parameters from experimental data. In general, the inputs of PE methods include network topology, initial conditions and discrete-time time series of some biochemical species present in the model. When parameters and network topology are unknown, PE has to be embedded in the solution of a Reverse Engineering problem [4]. Instead, if parameters and initial conditions are unavailable, a joint state-kinetic constants estimation (see,e.g., [13]) has to be performed. PE performs a global optimization of a fitness function which evaluates the distance between the simulated solution and the experimental data points. Several alternative optimization algorithms can be found in literature, each adopting a specific approach to escape from local minima and to limit the computational complexity. The difficulty of PE lies in the availability of experimental data, which in many real cases are very scarce. In this paper we present an approach that can be used in situations where both part of the network structure and part of the parametrization are unknown, representing an alternative to a PE embedded in a RE process. Differently from these techniques, our method tries to escape a strict dependency from experimental data using an OP that relies on biological empirical knowledge. This OP exploits an objective function that is conceived to represent a specific biological behaviour or phenotype, as it will be discussed in Section 5.

The use of an objective function with a biological meaning has already been incorporated in **Flux Balance Analysis** and **Cybernetic Modelling** (CM). In most FBA studies the objective function describes an objective that we believe is physiologically relevant for the cell, like the maximisation of biomass. This is used to predict a physiological behaviour of the cell. The idea that a system, like a cell, seeks a biological objective is justified acknowledging the apparent goal-directedness of nature. Other computational approaches, like cybernetic modelling [25], have taken advantage of this concept, which has been formally defined with the word teleonomy. A teleonomic view of nature indeed considers that the genetic programs of all living organisms have been shaped by evolution in a way that highly performing phenotypes have been selected [25]. Both FBA and cybernetic modelling have proved that this perspective can be used in different ways to simulate real biological behaviours. However, referring to the modelling categories defined in the introduction, FBA pertains to the group of *constraint-based* models, which are able to offer only a representation of the system at steady state. In FBA, the assumption is that the period in which the concentrations of biochemical species fluctuate can be neglected, while primary relevance is attributed to the condition in which the system is at equilibrium, i.e. when concentrations do not change in time any more. A further improvement of FBA is represented by **dynamic Flux Balance Analysis** (dFBA), proposed by Mahadevan and coworkers [20]. Here the extracellular space is modelled dynamically while inside the cell the steady-state assumption holds. In dFBA an ODE system is built to describe all extracellular events while the distribution of intracellular fluxes is calculated via a static optimization algorithm (SOA) or a dynamic optimization algorithm (DOA). For reasons of similarity with our approach we will focus specifically on the SOA algorithm, in which a biological objective function, like cellular growth rate or biomass, is maximized at each integration step of the ODEs. It is worth highlighting that dFBA always considers the intracellular system at steady state, so it neglects all the dynamic fluctuations of internal metabolites. In the field of mechanism based models, the teleonomic perspective was adopted by cybernetic models [25], which have been presented in a long series of progressive refinements. Besides teleonomy, the CM approach also assumes that (i) the topology of the network is completely known, that (ii)metabolic enzymes are regulated at the gene expression level or via allosteric mechanisms, that (iii) some economic principles govern the way the cell implements these regulations, meaning that the cell, as if it were a rational investor, enhances those reactions that assure the highest yield in terms of the objective function. Finally CM assumes that (iv) accurate mathematical expressions with a complete list of kinetic parameters are

available. These assumptions limit the good predictions of cybernetic models only to situations in which all parameters are available or where sufficient experimental data allow PE methods to be employed.

3. BACKGROUND

In this section we first introduce a brief overview of the (Stochastic) PN (SPN) formalism highlighting how it is possible to derive from an SPN model an ODE system that can be used to study the system's behaviour. Subsequently, we will re-call the basis of the OPs and their solution techniques which will be used in the sequel of this paper.

Stochastic Petri Nets. PN and their extensions are graphical modelling formalisms which are becoming quite popular to build models of biological systems. The reason of this appeal is their capabilities of representing in a simple and intuitive manner many important features of these systems and thus of constructing models that are easy to understand also by non-mathematicians and non-computer scientists. PNs are bipartite directed graphs with two types of nodes called *places* and *transitions*. The *places*, graphically represented as circles, correspond to the system variables (e.g. enzymes and compounds), while the transitions, graphically represented as rectangles, encode the events (e.g. interactions among biochemical entities) which cause the system evolution. Arcs connecting places to transitions (and vice versa) express the relations between states and event occurrences. Places can contain tokens, graphically represented as black dots, that in the context of systems biology, often describe the number of molecules of the corresponding entities. An example of a PN model is shown in Fig. 2 which describes glycolysis in human red blood cells.

The state of a PN, called *marking*, is defined as the number of tokens in each place of the net. An example of marking for the PN in Fig. 2 is showed in the third column of the Table in Fig. 3 **.** The system evolution is given by the occurrence of enabled transitions, where a transition is enabled if each input place contains a number of tokens greater or equal than a given threshold defined by the multiplicity of the corresponding input arc. A transition occurrence, called *firing*, removes a fixed number of tokens from its input places and adds a fixed number of tokens to its output places. The multiplicities of the input/output arcs determine the number of tokens involved by transition firings. The set of all the markings, that a net can reach through transition firings from an initial marking, is called the *Reachability Set* (RS). Instead, the behaviour of the net is encoded by means of the *Reachability Graph* (RG), a directed graph whose nodes are the markings of the RS and whose arcs are tagged with the labels of the transitions that cause the corresponding marking changes.

Temporal specifications must be introduced to model and study the temporal dynamics of a PN. Several timed extensions have been proposed in the literature [19]. In this paper we focus on Stochastic PN (SPN) [16], in which exponentially distributed random delays (interpreted as durations of certain activities) are associated with transition firings. Thanks to this assumption the temporal behaviour of the system can be modelled with a random process governed by the so-called Chapman-Kolmogorov differential equations [5]. These equations correspond to the Master Chemical Equations [6] that are used to describe the behaviour of biological systems, thus making this formalism quite attractive for these types of applications. Specifically, the underlying stochastic process corresponds to a Continuous Time Markov Chain (CTMC) that can be represented as a graph isomorphic to the RG of the net. Formally an SPN can be defined as follows:

DEFINITION 1. *A stochastic Petri net system is a tuple*

$$\mathcal{N} = (P, T, I, O, \mathbf{m}_0, \lambda)$$

where:

- $P = \{p_i\}_{1 \leq i \leq n_p}$ *is a finite and non empty set of* places *of cardinality* n_p*;*
- $T = \{t_i\}_{1 \leq i \leq n_t}$ *is a finite, non empty set of* transitions *with cardinality* n_t *and such that* $P \cap T = \emptyset$*. All these transitions are Timed transitions which fire with a random delay characterized by a negative exponential probability distribution;*
- $I, O : P \times T \to \mathbb{N}$ *are the* input*,* output *functions that define the arcs of the net and that specify their multiplicities;*
- $\mathbf{m_0} : P \to \mathbb{N}$ *is a multiset on* P *representing the* initial marking*; the notation* $m_0(p_i)$ *specifies the initial marking of the place* p_i*;*
- $\lambda : T \to \mathbb{R}$ *gives the* firing intensities *of the transitions.*

The values assumed by the functions I and O can be collected in $n_p \times n_t$ matrices (which we still call I and O) and whose entries are $I(p_i, t_j)$ and $O(p_i, t_j)$, respectively. By $I(t)$ we denote the column of I corresponding to transition t (the same holds for O). The set of input places of transition t (i.e. the preset of t), denoted ${}^\bullet t$, and the set of output places of t (i.e. postset of t), denoted $t^\bullet$, are defined as follows: ${}^\bullet t = \{p \in P \mid I(p,t) \neq 0\}$, and $t^\bullet = \{p \in P \mid O(p,t) \neq 0\}$.

From SPN to ODE. It often happens that, in case of very complex models, the underlying CTMC can not be derived or/and solved due to the well-known state space explosion problem. To cope with this difficulty, whenever the stochasticity of the modelled system can be neglected (e.g. due to huge number of molecules), the so-called deterministic approach can be exploited, assuming that the behaviour of entities contained in a place of the net is described with an Ordinary Differential Equation (ODE) and that the whole model is specified with a system of ODEs, one for each place of the net. In the literature, different laws (e.g. Michaelis-Menten, Hill-equation, etc.) have been proposed to encode each reaction of the biological system into an ODE. Here we focus on the Mass Action (MA) law [23][1] in which the

[1]Observe that this choice does not affect the generality of our approach that can be applied independently of the assumed law.

ODEs describing the model have the following form:

$$\frac{dx_{p_i}(\nu)}{d\nu} = \sum_{j:O(p_i,t_j)\neq 0} O(p_i,t_j)\lambda(t_j) \prod_{h:I(p_h,t_j)\neq 0} x_{p_h}(\nu)^{I(p_h,t_j)} - \sum_{j:I(p_i,t_j)\neq 0} I(p_i,t_j)\lambda(t_j) \prod_{h:I(p_h,t_j)\neq 0} x_{p_h}(\nu)^{I(p_h,t_j)} \quad (1)$$

where $x_{p_i}(\nu)$ represents the amount of the entity in place p_i at time ν assuming that $x_{p_i}(0)$ is defined through the initial marking of the net so that $x_{p_i}(0) = m_0(p_i)$.

For instance, considering the PN model in Fig.2 the behaviour of place *GLC* is described by the following ODE equation assuming the MA law:

$$\frac{dx_{GLC}(\nu)}{d\nu} = +\lambda(K_{F_1}) \cdot x_{HK} \cdot x_{GLC} \cdot x_{ATP} - \lambda(K_{R_1}) \cdot x_{HK} \cdot x_{G6P} \cdot x_{ADP} \quad (2)$$

Optimization problem. In Mathematics, Computer Science, and Operations Research, optimization or mathematical programming consists of minimizing (or maximizing) a function by systematically choosing the values of its variables from a set of feasible possibilities properly exploiting analytical or numerical methods. In Systems Biology optimization is not a new concept since it has been already proposed to reconstruct gene regulatory networks, transcriptional regulatory networks, protein interaction networks, conditional specific sub-networks, and active pathways [11], and to perform FBA. Formally an optimization problem with inequality constrains can be defined as follows:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathcal{F}_{opt}(\mathbf{x}) \\ \text{subject to} \quad & \mathcal{G}_i(\mathbf{x}) \geq b_i, \quad 1 \leq i \leq l \\ & \mathcal{L}_i(\mathbf{x}) \leq c_j, \quad 1 \leq j \leq m \end{aligned}$$

where the vector $\mathbf{x} = (y_1, \ldots, y_n)$ is the *variable vector*, the function $\mathcal{F}_{opt} : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, the functions $\mathcal{G}_i(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ and $\mathcal{L}_i(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ are *inequality constraint functions*, and the constants $b_1, \ldots, b_l, c_1, \ldots, c_m$ are the *bounds* for the constraints. A vector $\mathbf{x}^\bullet$, called *optimal*, is the solution of the OP if, among all vectors that satisfy the constraints, it is that which yields the smallest (largest) value of the optimization function: $\forall \mathbf{z}$ s.t. $\mathcal{G}_1(\mathbf{z}) \geq b_1, \ldots, \mathcal{L}_1(\mathbf{z}) \leq c_m$ we have that $\mathcal{F}_{opt}(\mathbf{z}) \geq \mathcal{F}_{opt}(\mathbf{x}^\bullet)$.

We recall that an OP is called a *linear program* if the objective and constraint functions are linear and *non-linear* otherwise. As shown in the next sections of this paper, we will focus on non-linear programs in which constraints can be non-linear as well. To solve this type of OPs, several algorithms have been proposed in the literature, and the reader can find a complete survey of these methods in [12].

4. OUR APPROACH

Before describing our approach in details, we introduce the biological considerations which motivated our proposal.

First, in our method we assume that both the topology and the kinetic parameters of the network are specified with different level of detail, depending on the biochemical event we are observing. Since in biochemical systems events are generally represented as biochemical reactions, we decide to divide all the reactions into two classes: *determined* reactions and *undetermined* reactions.

We assume that for determined reactions the mechanism of their biochemical interaction has been already properly characterized, so that they can be modelled with mathematical expressions that contain the full lists of kinetic parameters.

Instead, regarding undetermined reactions, we assume that in this part of the network the topology is only partially known, so that while all the reagents, all the products and some modifiers are specified, we do not exclude that other modifiers could be present.

For instance, this assumption can be motivated by many studies that showed how many cells of different organisms own different isoforms of the same enzyme and that every isoform has a specific affinity for a group of molecular regulators [17]. Thus, in a specific cell, the resulting kinetic behaviour of a reaction is a direct consequence of the proportion at which these isoforms are present [17] [2]. In some situations this proportion may hardly be defined, and, even if it is known, the list of modifiers for secondary isoforms is not always clear.

Moreover, for specific conditions, like cancer, cells may present genetic mutations and quantitative alterations or isoform swithces that change the affinity between the enzymes and their regulators, thus affecting the overall enzymatic dynamics. Assuming the molecular interactions of an enzyme are unclear necessarily implies that its activity is not expressible with a fully-parametrized mathematical formulation.

Finally, as anticipated in Section 2, we assume that an objective function with a biological meaning can be profitably used to formulate an OP and to reproduce an actual biological behaviour. From our point of view, this biological objective can be interpreted in different ways. For instance, it can be related to its definition in FBA or CM and formulated similarly. In FBA, according to the teleonomic perspective, the objective function describes an objective that we believe is physiologically relevant for the cell, like the maximisation of biomass. This is used to predict a physiological behaviour of the cell. Inspired by other successful applications of FBA, also in our approach the objective function can express a non-physiological goal of the cell, like the maximisation of ATP production, and used to identify some property of the network, or, it may be used to impose an engineering (e.g. maximisation of the production of a particular amminoacid) or therapeutic (e.g. minimisation of some reaction fluxes vital for cancer cells) objective to a metabolic system. More in general, the objective function is intended to represent any relevant biological behaviour that has been experimentally measured or that has to be achieved. Our approach thus tries to investigate if and how this specific biological phenotype can be reproduced in a biochemical system where the topology and the parametrization are just partially known.

In details we propose a method that takes advantage of an iterative process of optimization: at each simulation step, an objective function with the aforementioned biological meaning allows us to estimate the activities of undetermined reactions, and thus to obtain a complete description of system behaviour.

In order to acheive this, we define a set of time-varying parameters for all undetermined transitions. Thus, while the kinetic parameters of determined transitions maintain their fixed values, only the parameters of T_us are allowed to vary. These are defined with the intent that their changing values

recapitulate the lack of information about the structure and the kinetic parameters of undetermined reactions.

To facilitate the construction of the model we propose a new graphical formalism based on PN, which allows to automatically translate the model into its mathematical representation, consisting of the ODEs system and the Optimization Problem (OP).

According to this we decide to present our approach firstly introducing this new graphical formalism, and then providing its automatic translation into a ODE system in which indeterminate transitions are tackled through an OP. For this purpose we use the model of Fig.2 as a "running example" that we comment in the rest of the paper to discuss the features of this new modelling formalism.

SPN with Indetermination. The formal definition of a new PN extension called Stochastic Petri Net with Indetermination (SPNI) is the following:

DEFINITION 2. *A stochastic Petri net with indetermination is a tuple*

$$\mathcal{N} = (P, T, I, O, \mathbf{m}_0, \lambda_n, \Lambda_u, \mathcal{F}^{\mathcal{N}}_{opt})$$

where:

- $T = T_n \cup T_u$ *is a finite, non-empty set of* timed transitions *with* $T_n \cap T_u = \emptyset$. T_n *is the set of* determined transitions, *while* T_u *is the set of* undetermined transitions.
- $\lambda_n : T_n \to \mathbb{R}$ *gives the* firing intensity *of* T_n *transitions.*
- $\Lambda_u : T_u \to \mathbb{R}^2$ *gives the range of variation of the flux of* T_u *transitions.*
- $\mathcal{F}^{\mathcal{N}}_{opt} : T \times P \to \mathbb{R}$ *is an* objective function *whose terms are represented by place markings and transition firing intensities.*

We use the notation $\Lambda^L_u(t)$ (resp. $\Lambda^U_u(t)$) to denote the lower (resp. upper) bound values of the interval in which the flux of a $t \in T_u$ can vary; $\Lambda_u(t)$ then represents a possible flux value of the (undetermined) transition t compatible with its specified lower and upper bounds.

From SPNI to ODE and OP.
Due to the indetermination associated with the T_u transitions, it is not possible to directly use Eq. 1 to represent the deterministic behavior of an SPNI model. We can however re-write Eq. 1 as follows:

$$\frac{dx_{p_i}(\nu)}{d\nu} = \sum_{j:O(p_i,t_j)\neq 0} O(p_i,t_j)\mathcal{M}_{t_j}(\nu) \prod_{h:I(p_h,t_j)\neq 0} x_{p_h}(\nu)^{I(p_h,t_j)} - \sum_{j:I(p_i,t_j)\neq 0} I(p_i,t_j)\mathcal{M}_{t_j}(\nu) \prod_{h:I(p_h,t_j)\neq 0} x_{p_h}(\nu)^{I(p_h,t_j)} \quad (3)$$

where $\mathcal{M}$ is a function defined in the following way:

$$\mathcal{M}_t(\nu) = \begin{cases} \lambda_n(t) & if \ \ t \in T_n \\ y_t(\nu) & otherwise \end{cases} \quad (4)$$

The parameter $y_t(\nu)$ encodes the indetermination associated with the undetermined transition t at time ν and must be properly estimated to solve the ODE system.

As we already pointed out, the undetermined transitions are part of the model either because their exact specification is not relevant with respect to the goals of the analysis carried on with the SPNI or because they are too difficult to identify in a precise manner. Independently of the context of the modelling experiment, it is usually the case that we want to minimize (or maximize) certain measures defined on the portion of the state of the system that is not directly affected by undetermined transitions. These measures, that may assume complex definitions, become the optimization functions that we use to study these models.

To cope with this problem we thus propose to exploit an optimization process in which the objective function depends on the solution of the ODE systems in Eq.3. In practice, the optimization process solves the ODE system for a specific time interval while, simultaneously, it uses the obtained solution to compute the objective function of the optimization problem. The maximum/minimum value of the objective function allows to identify the values of unknown parameters of undetermined reactions.

Given a SPNI model, the corresponding OP, whose solution will be used to estimate the firing intensity values of the T_us, is derived using the following definition.

DEFINITION 3. *The OP derived by the SPNI is a tuple*

$$\mathcal{O} = (\mathbf{y}_\nu, \mathcal{F}_{opt}, \mathcal{G}, \mathcal{L})$$

where:

- $\mathbf{y}_\nu$ *represents the optimizing values of undetermined transitions at time* ν, *i.e.* $\forall t \in T_u \Rightarrow y_t(\nu) \in \mathbf{y}_\nu$;
- $\mathcal{F}_{opt} = \mathcal{F}^{\mathcal{N}}_{opt}$;
- $\mathcal{G}$ *is defined by*

$$\forall t \in T_u \Rightarrow y_t(\nu) \prod_{h:I(p_h,t)\neq 0} x_{p_h}(\nu)^{I(p_h,t)} \geq \Lambda^L_u(t)$$

- $\mathcal{L}$ *is defined by*

$$\forall t \in T_u \Rightarrow y_t(\nu) \prod_{h:I(p_h,t_j)\neq 0} x_{p_h}(\nu)^{I(p_h,t_j)} \leq \Lambda^U_u(t)$$

For instance, considering the SPNI in Fig. 2, where the gray boxes highlight the transitions affected by indetermination, the vector $\mathbf{y}(\nu)$ has size six and represents the optimal values of the firing rates of transitions T_{uf1}, T_{ur1}, T_{uf3}, T_{ur3}, T_{uf12}, T_{ur12}. An example of objective function could be the maximization of the Lactate (LAC) as described in our case study in Section 5.

Moreover in our example
$\Lambda^L_u(T_{u_{f_1}}) = 1620 \cdot x_{HK} \cdot x_{GLC} \cdot x_{ATP}$
$\Lambda^U_u(T_{u_{f_1}}) = 2.592e+08 \cdot x_{HK} \cdot x_{GLC} \cdot x_{ATP}$,
with limit values chosen as explained in Section 5.

How to compute the model behavior. Let $\mathbf{x}(\nu)$ represent the behaviour of the model at time ν . The numerical integration of Eq. 3 provides the behaviour of the model at time $\nu+\tau$, in terms of the behaviour $\mathbf{x}(\nu)$ computed at time ν and of a set of parameters deriving from the structure of the SPNI (I, and O), the firing intensities of the definite transitions of the net (λ_n) and of the firing intensities of the undetermined transitions estimated at time ν and collectively represented as $y(\nu)$. The values of $\mathbf{x}(\nu+\tau)$ are thus the results of the evaluation of a function whose input

Algorithm 1 Algorithm to solve ODE system with Indetermination

```
1: function SolveODEI(ODEI, G, L, F_opt, y_t, τ, FinalTime)
2:     ν = 0.0;
3:     ODEI.Init(Value);
4:     while (ν ≤ FinalTime) do
5:         print(ν, Value);
6:         Res=SolveOP(y_t, Value, ODEI, ν + τ, G, L, F_opt);
7:         Value=Res.Value;
8:         y_t=Res.y_t;
9:         ν += τ;
10:    end while
11: end function
```

parameters are represented by a tuple $B(\nu) = (B, B_u(\nu))$ where $B = (I, O, \lambda_n)$ and $B_u(\nu) = (\mathbf{x}(\nu), \mathbf{y}(\nu))$ [2]. The integration step s identifies the time points $\nu_i = i * \tau$ where the evaluation of the model behaviour is of interest.

Role of the estimation phase of our method is that of finding a set $\mathbf{y}(\nu)$ that, being compatible with the constraints of the SPNI model (Λ_u), minimizes the objective function at time $\nu + s$. The optimization phase identifies a number K of initial conditions, that we denote with $B_u^{[k]}(\nu), k = 1, ..., K$, consisting of the behaviour of the model computed at time ν and of K random points within the space of firing intensities of the undetermined transitions identified by the constraints Λ. From each of these configurations the method numerically integrates the system of ODEs up to time $\nu + s$ to derive $\mathbf{y}(\nu)$. Letting $B^{[k]}(\nu) = \left(B, B_u^{[k]}(\nu)\right)$, with $B_u^{[k]}(\nu) = (\mathbf{x}(\nu), \mathbf{y}^{[k]}(\nu))$, the solutions obtained from the integration of the ODEs with parameters $B^{[k]}(\nu), k = 0, 1, ..., K$ and up to time $\nu + s$ are compared to identify the choice of $B^{[k]}(\nu)$ which provides the best evaluation of the objective function, thus identifying $\mathbf{y}^{[k]}(\nu + s) = \mathbf{y}(\nu)$. Crucial in this optimization step is that the numerical integration of the ODEs is performed with a method capable of identifying an integration step h small enough to allow a precise solution of the ODEs during these "tentative" evaluations that are used to select the firing intensities of T_us.

In general, this whole method is repeated for each time point ν_i starting from $\nu_0 = 0$. However, solving the OP for each value of ν_i can be excessively costly and we can thus reduce this computational effort by identifying a time interval ρ that is a multiple of τ and that determines the time points where the optimization is requested. By doing so, if we set $\rho = m \cdot \tau$, we assume that for $m - 1$ intermediate evaluation steps the values of $\mathbf{\Lambda}_u(\nu)$ (i.e. $\mathbf{y}(\nu)$) remain constant and an approximation is introduced.

Having discussed how to derive from an SPNI model (i) an ODE system with indetermination (see Eq.3), and (ii) an OP (see Def.3), we can devise an algorithm which combines them to derive the model behaviour.

The pseudo-code of this algorithm is shown in Alg. 1. It takes as input the ODE system with indetermination (i.e. *ODEI*), the OP (i.e. described by functions $\mathcal{G}$, $\mathcal{L}$ and $\mathcal{F}_{opt}$), the initial guess for the rate of undetermined transition (i.e. y_t), the step size used for the optimization schema (i.e. τ), and the final time (i.e. *FinalTime*). The output of the algorithm is represented by the values generated for each system entity at different time points (i.e. ν_i). In details, the method *Init()* at line 3 initializes the vector *Value* encoding the initial values assumed for all the entities of the model. Then, the code from line 7 to line 13 is repeated until the time horizon is not reached. In each iteration the function *print()* is called to print the current values of the system entities. Subsequently, the function *SolveOP()* solves the optimization and returns the new values of the system entities and of the rates of T_us (i.e. $Res.Value$ and $Res.y_t$ respectively). It takes as input an initial guess for the rate of T_us (i.e. y_t), the current values of the entities (i.e. *Value*), the final time in which the objective function will be evaluated (i.e. $\nu + \tau$), the ODE system (i.e. *ODE*), and a set of functions encoding the OP (i.e. $\mathcal{G}, \mathcal{L}$ and $\mathcal{F}_{opt}$). The functions $\mathcal{G}$ and $\mathcal{L}$ are used by the optimization solver to test if a new vector $\mathbf{y}$, randomly generated according to the parameter constrains, is a feasible solution. Indeed the functions $\mathcal{G}$ and $\mathcal{L}$ verify if $\mathbf{y}$ satisfies the inequality constraints. The function $\mathcal{F}_{opt}$ is instead called by the optimization solver to compute the value of the objective function associated with a feasible vector $\mathbf{y}$.

This function, takes as input the vector $\mathbf{y}$, the current values of the entities (i.e. *Value*), the ODE system (i.e. *ODE*), and the final time in which the objective function must be evaluated (i.e. $\nu + \tau$). First it computes the quantities values at $\nu + \tau$ assuming the missing rates to be equal to $\mathbf{y}$. Then, the computed values are used to evaluate the objective function, whose derived value is returned. When the optimization step is terminated the vector *Value* is updated with the new computed values.

Moreover, in Alg. 2 we report an extension of the previous pseudo-code in which the optimization solver could be executed less frequently, so not at each time step. Indeed, we propose to exploit a heuristic function to decide when the optimization phase must be performed. Hence, when the optimization solver is not called the previous value y_t are considered during the solution of ODE system (i.e. method *SolveODE()*). In Section 5 an example of such a heuristic function is discussed and some experimental results are presented.

Algorithm 2 Algorithm to solve ODE system with Indetermination integrated with the heuristic function

```
1: function SolveODEI(ODEI, G, L, F_opt, y_t, τ, FinalTime)
2:     ν = 0.0;
3:     ODEI.Init(Value);
4:     while (ν ≤ FinalTime) do
5:         print(ν, Value);
6:         if Heurist(Value,time) then
7:             Res=SolveOP(y_t, Value, ODEI, ν + τ, G, L, F_opt);
8:             Value=Res.Value;
9:             y_t=Res.y_t;
10:        else
11:            Value=ODE.SolveODE(Value, ν + τ, y_u);
12:        end if
13:        ν += τ;
14:    end while
15: end function
```

5. EXPERIMENTAL RESULTS AND DISCUSSION

Our implementation. To perform our experiments a prototype implementation of the proposed method integrated

[2] In the sequel of the paper we will indifferently use $\mathbf{y}_t(\nu)$ or $\Lambda_u(t, \nu)$ to represent the undetermined parameters of our models as provided by the optimization problem at time ν.

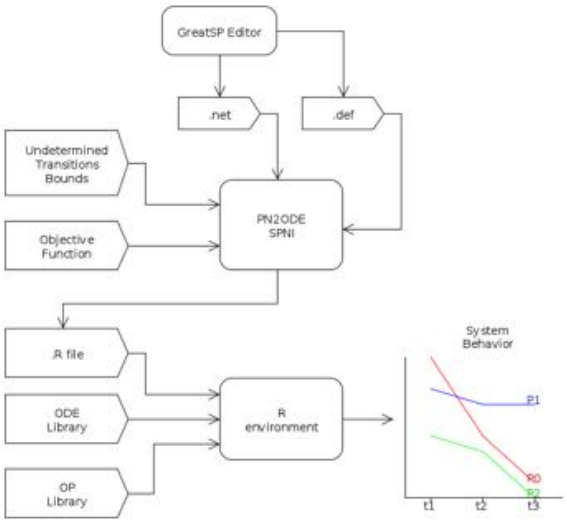


Figure 1: PN2ODE translation process.

in the *GreatSPN* framework [1] was developed. In detail, we extended the *GreatSPN* tool PN2ODE providing the automatic translation from SPNI to the ODEs system and OP system. A scheme representing this process is reported in Fig. 1. The generated ODEs system and OP are encoded in R language and saved into a file that is processed through the *R environment* to obtain the system behaviour. The R package *deSolve* [10] is used to solve ODEs integration, while package *GenSA* [21] is used to solve the OP.
The translation tool takes as inputs:

- the SPNI model, drawn with the GreatSPN GUI and saved into two files with extension .net and .def;
- One text file listing the undetermined bounded transitions;
- One text file containing the objective function.

The translation is performed with the following pipeline:

- The program extrapolates from .net and .def files all the PN information, such as places, transitions, arcs, initial marking and firing rates. Unkown transition rates are marked as NA.
- The program verifies the correctness of the file containing undetermined bounded transitions. For each undetermined transition T_u two values $\Lambda_u^L(t)$ and $\Lambda_u^U(t)$ that bound its flux are required. Optionally, the starting point, from which the OP solver starts searching the optimal solution, can be specified. If no starting point is provided, then a default value is computed as half of the sum of $\Lambda_u^L(t)$ and $\Lambda_u^H(t)$.
- The objective function, stored in the .txt file, is processed through a lex and yacc parsing tool. It can be a generic expression whose terms are the places and the transitions of the net.
- The whole translation process is executed from the command line as follows: PN2ODE SPNI_file_name -M -P -T ./transitions_file -F ./obj_fun_file_name , where -M enables Mass Action policy, -P enables export format in R with the optimizer, while -T and -F are respectively used to specify the two text files containing the undetermined bounded transitions and the objective funtion.

Case study. The proposed approach is used to investigate the metabolic behaviour of cancer cells to illustrate its practical applicability. The model represents the glycolytic pathway in a generic human cell. It is inspired by the model presented in [9], which describes glycolysis in human red blood cells. Glycolysis is the most important and best studied intracellular metabolic pathway. In every cell of the human body, it leads to the consumption of Glucose (GLC) and a progressive production of Pyruvate (PYR) and energy, in the form of Adenosine Triphosphate (ATP). Then, in physiological conditions, in the presence of oxygen, PYR is metabolised by other pathways to generate the majority of the energy consumed by the cell. In absence of oxygen, PYR is converted to LAC without further energetic yields. The model is characterized by seventeen metabolic reactions, the related equations are reported in the first column of the Table in Fig. 3, and it can be graphically described by the SPN model in Fig. 2 where place names are chosen to recall the corresponding biological compounds. The first and the last transitions are included to reproduce the inflow of GLC and the outflow of LAC in and from the cell. All the other transitions describe forward and reverse reactions, catalized by specific metabolic enzymes.

Differently form normal cells, cancer cells exhibit an enhancement of glycolysis and production of LAC even in the presence of oxygen, a phenomenon known as Warburg Effect [7]. This phenomenon represents the central focus of our experiments. It has been recently shown that metabolic alterations seen in cancer cells are promoted by specific mixtures of isoforms of their metabolic enzymes. In particular, it seems that isoforms of **Hexokinase** (HK), **Phosphofructokinase** (PFK) and **Pyruvate Kinase** (PK) may play an eminent role [14]. Despite these discoveries, it is still complicated to characterize the *in vivo* kinetics of these isoforms. Conditioned by these constraints, we chose to set the reactions involving HK, PFK and PK as undetermined transitions, i.e deficient of a complete list of regulators and of a specific mathematical expression containing its kinetic parameters. Our approach is used here as an attempt to acquire a deeper understanding of cancer metabolic dynamics. The idea is to use an objective function that encodes the Warburg Effect, considering every type of cancer at every possible tumour stage. We decided to formalize it as the maximization of LAC production at every integration step. Thus, the optimization process searches the values of the firing intensities of all T_us that allow to maximize LAC. The fluxes of undetermined transitions $T_{u_{f_1}}$, $T_{u_{r_1}}$, $T_{u_{f_3}}$, $T_{u_{r_3}}$, $T_{u_{f_{12}}}$ and $T_{u_{r_{12}}}$ were allowed to vary in a wide range that agrees with the available biological knowledge. Specifically the boundary conditions were set as follows:

$$\Lambda_u^L(T_{u_{f_1}}) = 1620 \cdot x_{HK} \cdot x_{GLC} \cdot x_{ATP}$$
$$\Lambda_u^U(T_{u_{f_1}}) = 2.592e+08 \cdot x_{HK} \cdot x_{GLC} \cdot x_{ATP}$$
$$\Lambda_u^L(T_{u_{r_1}}) = 12.24 \cdot x_{HK} \cdot x_{G6P} \cdot x_{ADP}$$
$$\Lambda_u^U(T_{u_{r_1}}) = 1.9584e+06 \cdot x_{HK} \cdot x_{G6P} \cdot x_{ADP}$$

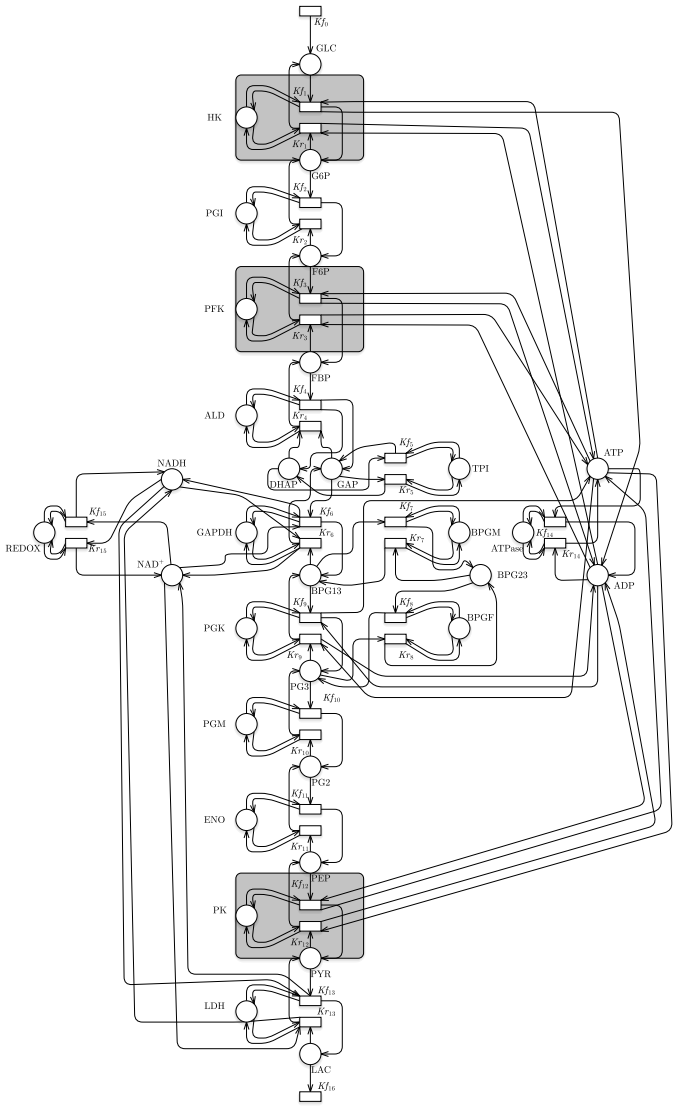


Figure 2: Case study: Glycolysis in *Homo Sapiens*.

$$\Lambda_u^L(T_{u_{f_3}}) = 2.5e+06 \cdot x_{PFK} \cdot x_{F6P} \cdot x_{ATP}$$
$$\Lambda_u^U(T_{u_{f_3}}) = 4e+11 \cdot x_{PFK} \cdot x_{F6P} \cdot x_{ATP}$$
$$\Lambda_u^L(T_{u_{r_3}}) = 211.864 \cdot x_{PFK} \cdot x_{FBP} \cdot x_{ADP}$$
$$\Lambda_u^U(T_{u_{r_3}}) = 3.38983e+07 \cdot x_{PFK} \cdot x_{FBP} \cdot x_{ADP}$$
$$\Lambda_u^L(T_{u_{f_{12}}}) = 1.29234 \cdot x_{PEP} \cdot x_{PK} \cdot x_{ADP}$$
$$\Lambda_u^U(T_{u_{f_{12}}}) = 206774 \cdot x_{PEP} \cdot x_{PK} \cdot x_{ADP}$$
$$\Lambda_u^L(T_{u_{r_{12}}}) = 0.0058511 \cdot x_{PK} \cdot x_{PYR} \cdot x_{ATP}$$
$$\Lambda_u^U(T_{u_{r_{12}}}) = 0.620322 \cdot x_{PK} \cdot x_{PYR} \cdot x_{ATP}$$

Fig. 4 (left) shows the evolution of LAC over time. The black line represents the results of the model of a normal cell, where all parameters are well characterized. The blue dashed line show the time evolution of LAC when uncertainty is applied. It can be noticed that this objective function is able to drive the system to accelerate LAC production. Even if the difference might not seem large enough to represent the Warburg Effect, we point out that these diagrams show the behaviour of our model for a very short time interval. Fig. 4 (right) shows the rapid accumulation of Fructose 6-Phosphate (F6P) in the normal cell model compared to a more balanced production-consumption dynamics in the cancer model. F6P, a high glycolitic intermediate, increases as a direct consequence of GLC degradation and is later processed by PFK. It is then significant to see which parameters the optimization solver independently chooses to tune in order to maximize its objective function. While parameter values of HK and PK did not vary markedly if compared to the normal cell model (data not shown), the highest difference regarded PFK. Many articles as [17] have demonstrated that PFK kinetics is highly non-linear and depends on many allosteric interactions. Our results seem to reinforce the Mulukutla's thesis [17] that the regulation of PFK activity has a crucial impact on the glycolitic flux and may be relevant to explain metabolic alterations in cancer.

With an additional set of experiments we studied if it was possible to reduce the computational costs of our approach while maintaining a good accuracy of the solution. With this intent, we progressively decreased the frequency at which the optimizator was invoked and then explored the performance of our algorithm. When the optimization process is not repeated at every integration step the time-varying parameters associated with T_us are then transformed into piecewise constant parameters. This can be motivated with the assumption that in the time interval between two consecutive optimization processes some fixed parameter values can well approximate the real behaviour of all T_us. The time interval that separates different optimization processes, in alternative called optimization step, was set to four different values: 1e-7, 5e-7, 1e-6, 5e-6 h. As reported in Table 5 the resulting computational times are compared. As expected, reducing the number of optimization processes allowed to significantly diminish the overall computational efforts of the algorithm. We then studied how these changes affected the dynamics of the system. Intriguingly, we found that some places, like LAC, as shown in Fig 5 (left), displayed little or no changes, while for other places, like PEP, the changes were much more relevant, as shown in Fig. 5 (right). We can then observe that for the more sensible places the reduction of computational time comes at the cost of the precision of the solution, which nevertheless maintains the capability to provide some qualitative information about the behaviour of the model.

Reactions	Rate Equations	Initial Marking
$\emptyset \xrightarrow{Kf0} GLC$	$Kf_0 = 6.48E+06$	$GLC_0 = 0$
$HK + GLC + ATP \underset{Kr1}{\overset{Kf1}{\rightleftarrows}} HK + G6P + ADP$	$Kf_1 = 6.48E+05, Kr_1 = 4.9E+03$	$HK_0 = 24$
$PGI + G6P + ATP \underset{Kr2}{\overset{Kf2}{\rightleftarrows}} PGI + F6P$	$Kf_2 = 1.15E+03, Kr_2 = 2,68E+03$	$G6P_0 = 0, PGI_0 = 218$
$PFK + F6P + ATP \underset{Kr3}{\overset{Kf3}{\rightleftarrows}} PFK + FBP + ADP$	$Kf_3 = 1E+09, Kr_3 = 8.47E+04$	$F6P_0 = 0, PFK_0 = 28$
$ALD + FBP \underset{Kr4}{\overset{Kf4}{\rightleftarrows}} ALD + DHAP + GAP$	$Kf_4 = 1.46E+02, Kr_4 = 1.18E+00$	$FBP_0 = 0, ALD_0 = 3,7E+03, DHAP_0 = 0$
$TPI + GAP \underset{Kr5}{\overset{Kf5}{\rightleftarrows}} TPI + DHAP$	$Kf_5 = 7.93E+00, Kr_5 = 4.53E+06$	$TPI_0 = 1.14E+04, GAP_0 = 0$
$GAPDH + GAP + NAD^+ \underset{Kr6}{\overset{Kf6}{\rightleftarrows}} GAPDH + BPG13 + NADH$	$Kf_6 = 1.42E+05, Kr_6 = 5.28E+06$	$GAPDH_0 = 7.6E+02, NAD^+_0 = 1E+03, NADH_0 = 0$
$BPGM + BPG13 \underset{Kr7}{\overset{Kf7}{\rightleftarrows}} BPGM + BPG23$	$Kf_7 = 1E+08, Kr_7 = 1E+05$	$BPGM_0 = 4.10E+02, BPG13_0 = 0$
$BPGF + BPG23 \underset{Kr8}{\overset{Kf8}{\rightleftarrows}} BPGF + PG3$	$Kf_8 = 6.84E+02, Kr_8 = 1E-09$	$BPGF_0 = 4.10E+02, BPG23_0 = 0$
$PGK + BPG13 + ADP \underset{Kr9}{\overset{Kf9}{\rightleftarrows}} PGK + PG3 + ATP$	$Kf_9 = 2.61E+04, Kr_9 = 1.45E+01$	$PGK_0 = 2.74E+02$
$PGM + PG3 \underset{Kr10}{\overset{Kf10}{\rightleftarrows}} PGM + PG2$	$Kf_{10} = 5.38E+01, Kr_{10} = 7.92E+00$	$PGM_0 = 4.10E+02, PG3_0 = 0$
$ENO + PG2 \underset{Kr11}{\overset{Kf11}{\rightleftarrows}} ENO + PEP$	$Kf_{11} = 5.82E+02, Kr_{11} = 3.44E+02$	$ENO_0 = 2.20E+02, PG2_0 = 0$
$PK + PEP + ADP \underset{Kr12}{\overset{Kf12}{\rightleftarrows}} PK + PYR + ATP$	$Kf_{12} = 5.17E+02, Kr_{12} = 5.17E-01$	$PEP_0 = 0, PK_0 = 6.9E+01,$
$LDH + PYR + NADH \underset{Kr13}{\overset{Kf13}{\rightleftarrows}} LDH + LAC + NAD^+$	$Kf_{13} = 1.04E+03, Kr_{13} = 2.34E+00$	$PYR_0 = 0, LDH_0 = 3.12E+02$
$ATPase + ATP \underset{Kr14}{\overset{Kf14}{\rightleftarrows}} ATPase + ADP$	$Kf_{14} = 9.74E-01, Kr_{14} = 9.74E+00$	$ATPase_0 = 1E+02, ATP_0 = 7E+02,$
$REDOX + NAD^+ \underset{Kr15}{\overset{Kf15}{\rightleftarrows}} REDOX + NADH$	$Kf_{15} = 9.74E-01, Kr_{15} = 9.74E-04$	$ADP_0 = 5E+02, REDOX_0 = 97$
$\emptyset \xrightarrow{Kf16} LAC$	$Kf_{16} = 1$	$LAC_0 = 0$

Figure 3: Table: Reactions, Equations and Initial marking of glycolysis in *Homo Sapiens*.

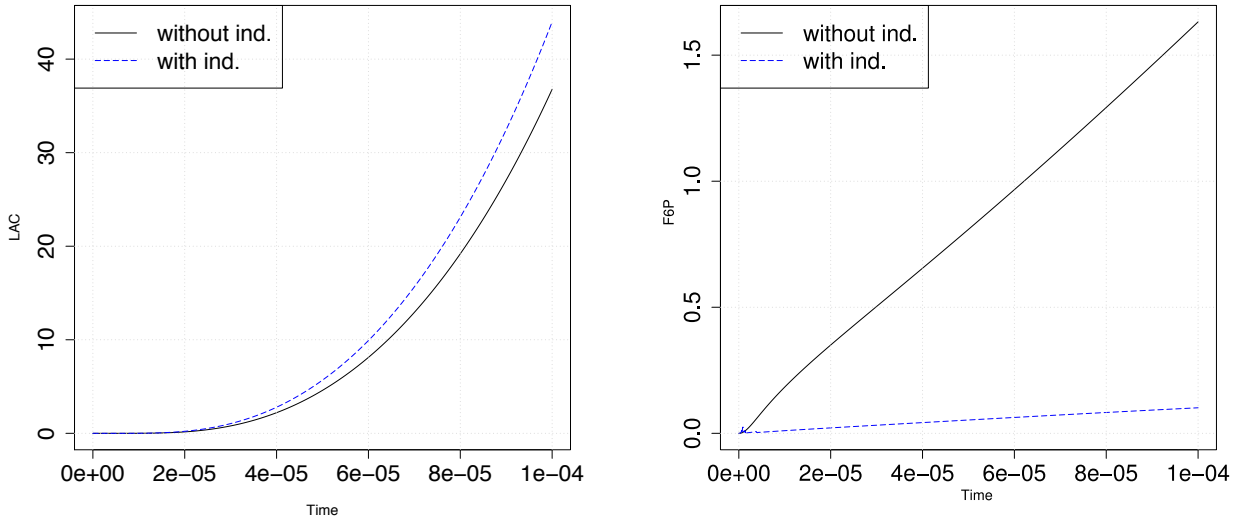


Figure 4: Behaviour of place *LAC* (left) shows that the model representing the metabolism of a cancer cell is able to reproduce a higher production of LAC over time. Significative differences affect the dynamic behaviour of place *F6P* (right), which is less likely to accumulate in the cancer model

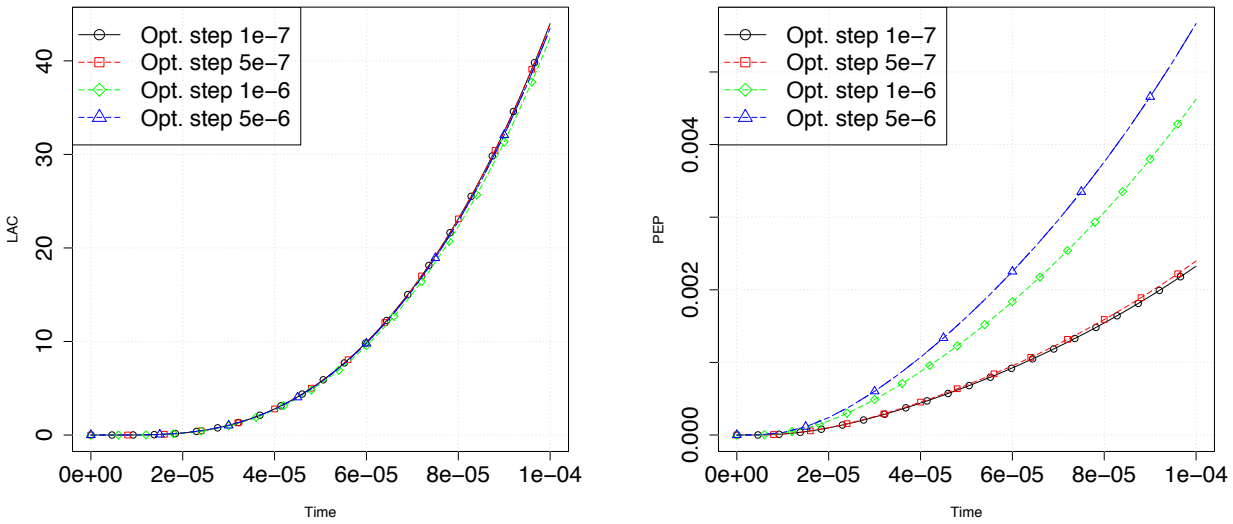


Figure 5: The figures show how, when we try to increase the time inbetween two optimization processes for some places, as for instance *LAC* (left), the behaviours are almost not affected, while for other, like *PEP* (right), the difference is non-trivial

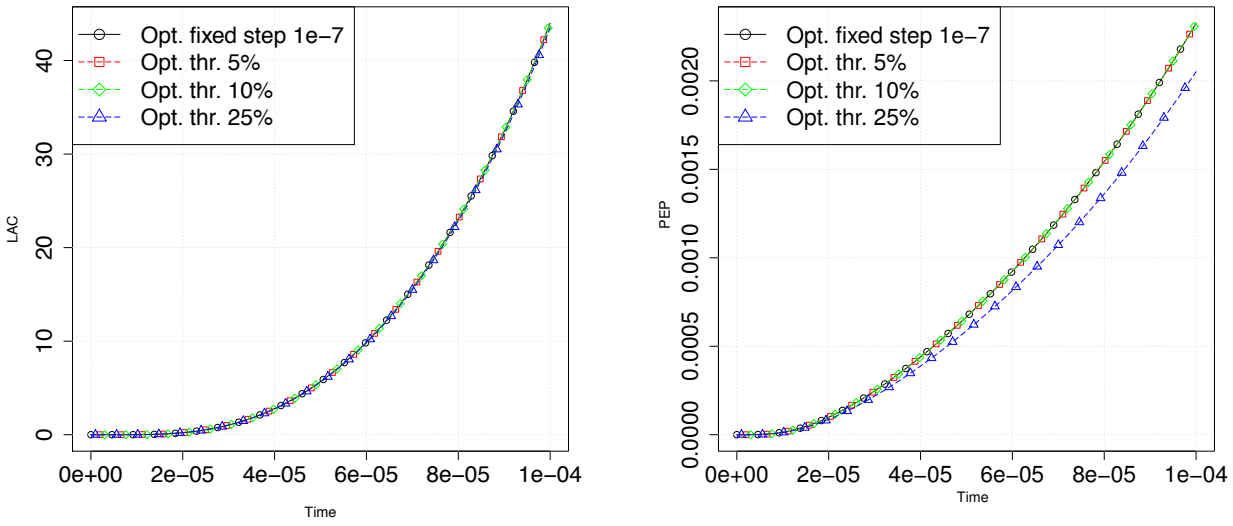


Figure 6: Behaviours of places *LAC* (left), *PEP* (right). The figures show that, when the algorithm with the heuristics is used and threshold values of 5 or 10% are chosen, the solution of the model is accurate, while the computation time is considerably reduced

ODE	Opt int 1e-7	Opt int 5e-7
0.732235s.	2441.878sec.	569.7389sec.
	Opt int 1e-6	**Opt int 5e-6**
	254.9282sec.	119.2946sec.

Table 1: Execution times for different optimization time intervals. All executions were performed on an INTEL i7 64bit 2.60GHz processor

In a further set of experiments we tested if the time step of the optimization processes could be adaptively tuned along the solution of the system. The rationale behind this choice is to invoke the optimization process just if it produces significant effects. In order to do it, we added in our algorithm an heuristics, presented in Section 4, that considers the relative change in the markings of input places of all T_us. When the relative change in one of these places crosses a user-defined threshold, the optimization process is executed. We studied how three different thresholds values (i.e. 5%, 10% and 25% of change) impacted both the computational time and the accuracy of the solution. Figure 6 displays the differences in the dynamic behaviours of PEP and LAC, when the different threshold values are considered. The execution times in the three cases are reported in Table 5, compared to the performance of the algorithm in the absence of the implemented heuristics. Data clearly show that, for instance when a 5% threshold value is used, no negative effects reflect on model solutions, while the computational cost is reduced of a factor greater that 10. Regarding the accuracy of the solution, also greater threshold values are satisfactory, with the advantage that they allow considerably higher time savings.

Opt int 1e-7	Thr=5%	Thr=10%	Thr=25%
2441.878sec.	195.0151sec.	135.4613sec.	82.67076sec.

Table 2: Execution times for different threshold values of relative change in place markings. All executions were performed on an INTEL i7 64bit 2.60GHz processor

Discussion. It is important to observe that our approach, moving forward from the strict teleonomic perspective adopted by dFBA and CM, allows to define objective functions that may mimic some experimentally observed behaviour. It is worth underlying that teleonomy has shown great utility to help dissect the metabolism of microorganisms. In the case of multicellular and more complex organisms like humans, however, the process of selection that justifies the teleonomic view is much more complex, and a clear goal-directedness of intracellular metabolism is not equally reasonable to consider. However, the LAC maximisation function, which we adopted in our case study of cancer glycolysis, besides reflecting the fact that a high production of LAC in cancer cells is a renown experimental finding, may also have a teleonomic value. In fact, it can be hypothesized that the process of evolution of cellular phenotypes observed in cancer, i.e. in the selection of aggressive cancer clones during its progression (current focus of extensive studies [24]), tends to select a goal-directed cellular phenotype characterized by LAC maximization.
Moreover our proposed approach, differently from dFBA and CM, does not assume the steady state of the intracellular metabolites, as in dFBA, and it does not require a complete knowledge of all kinetic parameters, as in CM. Finally, it is useful to highlight again that the lack of experimental data, which is very frequent in cancer cell scenarios, does not affect our approach as it does for techniques of RE and PE.

6. CONCLUSIONS AND FUTURE DIRECTIONS

In this work we proposed a new approach to deal with biochemical models with lacking kinetic information. As discussed in Section 5 it may be useful for different applications, although additional experiments will provide a further validation and will show its actual relevance.
For the future we intend to apply our approach to study different networks, either in microorganisms, in single human cells or in population of cells.

As for all OPs, the performances are highly dependent on the objective function, on the constraints that define the space of feasible solutions, and on the algorithm used to explore it. Depending on the specific model, objective functions with different biological meaning, as described in Section 4, will be tested. We will also investigate which available biological data can be effectively used to adapt the model to the specific situation under study. Either data obtained with transcriptomics, proteomics and metabolomics, as well as reaction fluxes and thermodynamic information are all possible candidates. These data will provide a fundamental support to define the initial conditions of the model, to inspire the choice of the objective function and to set the constraints for the optimization. Finally, depending on OP characteristics, the performances of different optimization algorithms will be tested.
In addition, for those cases in which stochastic behaviours are relevant, we plan to expand our approach with systems of stochastic differential equations.

7. ADDITIONAL AUTHORS

Gianfranco Balbo (University of Turin, Computer Science Department, Corso Svizzera 185 Turin Italy, email: *balbo@di.unito.it*)

8. REFERENCES

[1] J. Babar, M. Beccuti, S. Donatelli, and A. S. Miner. Greatspn enhanced with decision diagram data structures. In *Proceedings of 31st Int. Conf. of Applications and Theory of Petri Nets*, pages 308–317. IEEE Computer Society, June 2010.

[2] B. D. Bennett, E. H. Kimball, M. Gao, R. Osterhout, S. J. V. Dien, and J. D. Rabinowit. Absolute metabolite concentrations and implied enzyme active site occupancy in escherichia coli. *Nature Chemical Biology*, 5(8):593–599, 2009.

[3] S. Bulik, S. Grimbs, C. Huthmacher, J. Selbig, and H. G. Holzhutter. Kinetic hybrid models composed of mechanistic and simplified enzymatic rate laws - a promising method for speeding up the kinetic modelling of complex metabolic networks. *FEBS Journal*, 276(2):410–424, 2009.

[4] P. Cazzaniga, C. Daminani, D. Besozzi, R. Colombo, M. Nobile, D. Gaglio, D. Pescini, S. Molinari,

G. Mauri, L. Alberghina, and M. Vanoni. Computational strategies for a system-level understanding of metabolism. *Metabolites*, 4:1034–1087, 2014.

[5] W. Feller. *An Introduction to Probability Theory*, volume 1. Wiley, New York, NY, 1968.

[6] D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, 188(1-3):404–425, 1992.

[7] M. V. Heiden. Targeting cancer metabolism: a therapeutic window opens. *Nature Reviews. Drug discovery*, 10(9):671–684, 2011.

[8] D. M. Hendrickx, M. M. W. B. Hendriks, P. H. C. Eilers, A. K. Smilde, and H. C. J. Hoefsloot. Reverse engineering of metabolic networks, a critical assessment. *Molecular BioSystems*, 7:511–520, 2011.

[9] N. Jamshidi and B. O. Palsson. Mass action stoichiometric simulation models: Incorporating kinetics and regulatoin into stoichiometric models. *Biophysical Journal*, 98(2):175–185, 2010.

[10] S. a. Karline, c. Thomas, Petzoldt [aut, S. a. R. Woodrow, and odepack authors [cph]. Solvers for Initial Value Problems of Differential Equations (ODE, DAE, DDE). https://cran.r-project.org/web/packages/deSolve/deSolve.pdf, 2016.

[11] R.-S. W. L. Chen and X.-S. Zhang. *Biomolecular Networks: Methods and Applications in Systems Biology*. Wiley, New York, NY, 2009.

[12] K. Lange. *Optimization*. Springer, New York, NY, second edition, 2013.

[13] X. Liu and M. Niranjan. State and parameter estimation of the heat shock response system using kalman and particle filters. *Bioinformatics*, 28(11):1501–1507, 2012.

[14] U. E. Martinez-Outschoorn, M. Peiris-Pagès, R. G. Pestell, F. Sotgia, and M. P. Lisanti. Cancer metabolism: a therapeutic perspective. *Nature Reviews Clinical Oncology*, May 2016.

[15] C. G. Moles, P. Mendes, and J. R. Banga. Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Research*, 13(11):2467–2474, 2003.

[16] M. K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, 31(9):913–917, 1982.

[17] B. C. Mulukutla, A. Yongky, P. Daoutidis, and W.-S. Hu. Bistability in glycolysis pathway as a physiological switch in energy metabolism. *PLoS ONE*, 9(6):1–12, June 2014.

[18] J. D. Orth, I. Thiele, and B. Ø. Palsson. What is flux balance analysis? *Nature biotechnology*, 28:245–248, March 2010.

[19] L. Popova-Zeugmann. *Time and Petri nets*. Springer, Heidelberg, GE, 2013.

[20] M. Radhakrishnan, J. S. Edwards, and F. J. Doyle. Dynamic flux balance analysis of diauxic growth in escherichia coli. *Biophysical Journal 83.3 (2002)*, 83(3):1331–1340, 2002.

[21] G. Sylvain, X. Yang, S. Brian, H. Julia, and P. SA. R Functions for Generalized Simulated Annealing. https://cran.r-project.org/web/packages/GenSA/GenSA.pdf, 2016.

[22] N. Totis, M. Beccuti, F. Cordero, L. Follia, C. Riganti, F. Novelli, and G. Balbo. Dealing with indetermination in biochemical networks. *In Proceedings of ACM Int. Workshop. of Italian Group on Quantitative Methods in Informatics (InfQ2016), Taormina, Italy, October 25 2016.*

[23] E. O. Voit. *Computational analysis of biochemical systems : a practical guide for biochemists and molecular biologists*. Cambridge University Press, Cambridge, New York, 2000.

[24] L. R. Yates and P. J. Campbell. Evolution of the cancer genome. *Nat Rev Genet*, 13:795–806, 2012.

[25] J. D. Young. Learning from the steersman: A natural history of cybernetic models. *Industrial & Engineering Chemistry Research*, 54(42):10162–10169, 2015.