Capacity Scaling Augmented With Unreliable Machine Learning Predictions

Daan Rutten  
Georgia Institute of Technology  
755 Ferst Dr NW, Atlanta, Georgia  
drutten@gatech.edu

Debankur Mukherjee  
Georgia Institute of Technology  
755 Ferst Dr NW, Atlanta, Georgia  
debankur@gatech.edu

ABSTRACT

Modern data centers suffer from immense power consumption. As a result, data center operators have heavily invested in capacity scaling solutions, which dynamically deactivate servers if the demand is low and activate them again when the workload increases. We analyze a continuous-time model for capacity scaling, where the goal is to minimize the weighted sum of flow-time, switching cost, and power consumption in an online fashion. We propose a novel algorithm, called the Adaptive Balanced Capacity Scaling (ABCS) algorithm, that has access to black-box machine learning predictions. ABCS aims to adapt to the predictions and is also robust against unpredictable surges in the workload. In particular, we prove that the ABCS algorithm is \((1 + \epsilon)\)-competitive if the predictions are accurate, and yet, it has a uniformly bounded competitive ratio even if the predictions are completely inaccurate.

Keywords

green energy efficiency, online algorithms, competitive analysis, speed scaling, competitive ratio

1. INTRODUCTION

As a result of their immense power consumption, data centers are constantly optimizing their servers for energy efficiency, pushing the hardware’s efficiency to nearly its limit. At this point, algorithmic improvements are critical in order to achieve substantial further gain [7]. A common practice for data centers has been to reserve significant excess service capacity in the form of idle servers, even though a typical idle server still consumes about 44% of its peak power consumption [7]. The recommendation from the U.S. Department of Energy [7], industry and the academic community is, therefore, to implement dynamic capacity scaling functionality based on the demand. Instead of physically toggling servers on or off, such dynamic scaling functionality is often implemented by carefully allocating a fraction of servers to lower priority services and quickly bringing them back at times of high demand; see [6] for a more detailed account.

The call for algorithmic solutions to capacity scaling has inspired a vibrant line of research over the last decade [1, 2, 4]. The problem fits into the framework of online algorithms. Here, the performance of an algorithm is captured in terms of the Competitive Ratio (CR), which is defined as the worst possible ratio between the cost incurred by the online algorithm and that by the offline optimum algorithm, which has accurate information about all future input variables. The key advantage of such strong performance guarantees lies in its robustness, that is, the algorithm safeguards against the worst-case scenario.

However, any of today’s modern large-scale systems has access to historical data, which, combined with Machine Learning (ML) algorithms, can reveal definitive patterns. In these cases, simply following the recommendations obtained from the ML predictions typically outperforms any competitive algorithm. However, besides empirical verification, the performance of such ML predictions is not guaranteed. In fact, repeated observations show that unexpected surges in the workload are not at all uncommon, which cause a significant adverse impact on the system performance.

These contrasting approaches reveal a gap between what we are able to prove and what is desirable in practice. Building on the methods introduced in [3], the current work aims to bridge this gap by incorporating ML predictions directly into the competitive analysis framework. In particular, we propose a novel low-complexity algorithm for capacity scaling, the Adaptive Balanced Capacity Scaling (ABCS) algorithm, which has access to a black-box predictor, lending predictions about future arrivals. Critically, not only is ABCS completely unaware of the prediction’s accuracy, we also refrain from making any statistical assumptions on the accuracy. The main challenge therefore is to design near-optimal algorithms which intelligently accept and reject the recommendations given by the ML predictor without knowing or learning their accuracy. It is worth emphasizing that this work is not concerned with how the ML predictions are obtained and uses them as a black box.

2. MODEL DESCRIPTION

We will use a canonical continuous-time dynamical system model that is used to analyze algorithms for energy efficiency; see for example [1, 2] for variations. Consider a system with a large number of homogeneous servers. A server is in either of two states: active or inactive. Let \( m(t) \) denote the number of active servers at time \( t \) and \( \omega, \beta \) and \( \theta \) be fixed non-negative parameters of the model. We will assume that workload arrives into the system according to an unknown and arbitrary rate \( \lambda(t) \) and gets processed at instantaneous rate \( m(t) \). We do not impose any restrictions on \( \lambda(\cdot) \). The system has a buffer of infinite capacity, where unprocessed workload waits while accumulating a waiting cost at rate \( \omega \). At any time, the system may decide to in-
crease or decrease \( m(t) \) in an online fashion. However, it pays a switching cost of \( \beta \) each time a server is activated. Also, active servers accumulate a power consumption cost at rate \( \theta \). The model is then

\[
\min_{m(0, T) \to \mathbb{R}_+} \text{Cost}^\lambda(m, T) := \omega \cdot \int_0^T q(s) \, ds + \beta \cdot \limsup_{\delta \downarrow 0} \sum_{i=0}^{[T/\delta]} [m(i \delta + \delta) - m(i \delta)]^+ + \theta \cdot \int_0^T m(s) \, ds
\]

s.t. \( q(t) = \int_0^t (\lambda(s) - m(s)) \mathbb{I}\{q(s) > 0 \text{ or } \lambda(s) \geq m(s)\} \, ds \)

where \( [x]^+ = \max(x, 0) \). To solve the optimization problem above, an algorithm needs to determine the function \( m(\cdot) \), given the parameters \( \omega, \beta, \theta \) and \( T \). Note that our goal is to investigate online algorithms, meaning that \( \lambda(\cdot) \) is revealed to the algorithm in an online fashion. In other words, at time \( t \), the algorithm must determine \( m(t) \) depending only on \( \lambda(s) \) for \( s \in [0, t] \). We will compare the total cost \( \text{Cost}^\lambda(m, T) \) for an online algorithm to that of the offline minimum defined as \( \text{Opt} := \inf_{m(0, T) \to \mathbb{R}_+} \text{Cost}^\lambda(m, T) \).

The model in (1) actually combines some well-studied state-of-the-art models [1, 2]. To see how it relates to the problem of capacity scaling, note that the objective function in (1) is a weighted sum of three common performance measures. The three metrics are:

(i) The flow-time. The flow-time is defined as the total time a task spends in the system and captures the response time of the system; see also [1]. The weight \( \omega \) could be determined based on loss of revenue or user dissatisfaction as a result of increased response time.

(ii) The switching cost. As in [2, 6], the parameter \( \beta \) can be viewed as the cost to increase the number of active servers. This may include for example, the cost to terminate a lower priority service and related migration costs.

(iii) The power consumption. The power consumption is proportional to the total time servers are in the active state. The weight \( \theta \) represents the cost of power. Also, the constraints in (1) model the dynamics of capacity scaling and \( q(\cdot) \) can be viewed as the queue length process or the remaining workload process. Note that (1) does not require \( q(t) \) or \( m(t) \) to be integer-valued. This is a fairly standard relaxation, since a service may typically request a fraction of the server’s capacity [6] and a single task is tiny; see for example [2, 4]. The system in (1) can also be interpreted as a fluid counterpart of a discrete system.

We further assume that the algorithm receives predictions about future workload through an ML oracle [3]. More precisely, at time \( t = 0 \), the ML oracle predicts an arrival rate function \( \hat{\lambda}(\cdot) \). We measure the accuracy of the predictions in terms of the mean absolute error (MAE) between the predicted arrival rate function \( \hat{\lambda} \) and the actual rate function \( \lambda \), which is commonly used in state-of-the-art ML algorithms.

**Definition 2.1.** Fix a finite time horizon \( T \) and arrival rate function \( \lambda(\cdot) \). For \( \eta > 0 \), we say that a prediction \( \hat{\lambda} \) is \( \eta \)-accurate for the instance if

\[
\|\hat{\lambda} - \lambda\|_{\text{MAE}} := \frac{1}{T} \int_0^T \left| \hat{\lambda}(t) - \lambda(t) \right| \, dt \leq \eta \cdot \text{Opt} / T.
\]

The definition of the prediction’s accuracy is intimately tied to the cost of the optimal solution. As already argued in [3], since \( \text{Opt} \) is a linear functional of \( \lambda \), normalizing the error by the cost of the optimal solution is necessary. This is because the definition should be invariant to scaling arguments. For example, if we double both \( \lambda(\cdot) \) and \( \hat{\lambda}(\cdot) \), then the prediction’s accuracy should still be the same.

### 3. RESULTS

Note that a trivial way to implement the predictions is to blindly trust the predictions, i.e., to let

\[
m \in \arg \min_{m(0, T) \to \mathbb{R}_+} \text{Cost}^\lambda(m, T).
\]

However, in this case, the performance decays drastically even for relatively small prediction errors. Indeed, if the actual arrival rate \( \lambda(\cdot) \) is higher than the predicted arrival rate \( \hat{\lambda}(\cdot) \) at the start, then the associated workload could stay in the queue forever and incur a significant waiting cost. Most algorithms proposed in the literature, such as the RHC and LCP algorithms from [2], follow the predictions blindly in a similar way and therefore are not competitive if the predictions turn out to be inaccurate. Hence, our goal here is to obtain an algorithm which follows the predictions most of the time, but ignores the predictions when appropriate.

We propose the new Adaptive Balanced Capacity Scaling (ABCS) algorithm which is parameterized by four non-negative numbers \( R_1 \geq r_1 \geq 0 \) and \( R_2 \geq r_2 \geq 0 \). Let \( \Delta \lambda(t) := (\lambda(t) - \hat{\lambda}(t))^+ \) for \( t \geq 0 \) and \( \Delta \lambda(t) = 0 \) for \( t < 0 \). Algorithm 1 below describes ABCS for any fixed choices of \( R_1, r_1, R_2, r_2 \).

**Algorithm 1: Adaptive Balanced Capacity Scaling**

Choose \( m(\cdot) \) such that at any time \( t \geq 0 \):

\[
\frac{dm(t)}{dt} = \frac{\hat{r}_1(t) \omega \cdot q(t) - \hat{r}_2(t) \theta \cdot m(t)}{\beta},
\]

where

\[
\hat{r}_1(t) = \begin{cases} R_1 & \text{if } m(t) - \bar{m}(t) > 0 \text{ or } q(t) \leq \bar{q}(t), \\ r_1 & \text{otherwise}, \end{cases}
\]

\[
\hat{r}_2(t) = \begin{cases} R_2 & \text{if } m(t) > \bar{m}(t) \text{ and } q(t) \leq \bar{q}(t), \\ r_2 & \text{if } m(t) \leq \bar{m}(t) \text{ or } q(t) > \bar{q}(t), \end{cases}
\]

and

\[
\bar{m}(t) = \bar{m}_1(t) + \bar{m}_2(t),
\]

\[
\bar{q}(t) = \int_0^t (\lambda(s) - \bar{m}(s)) \mathbb{I}\{q(s) > 0 \text{ or } \lambda(s) \geq \bar{m}(s)\} \, ds,
\]

where

\[
\bar{m}_1 \in \arg \min_{m(0, T) \to \mathbb{R}_+} \text{Cost}^{\lambda}(m, T),
\]

\[
\frac{d\bar{m}_2(t)}{dt} = \sqrt{\frac{\omega}{2\beta}} \left( \Delta \lambda(t) - \Delta \lambda \left( t - \sqrt{\frac{2\beta}{\omega}} \right) \right).
\]

We start by discussing the intuition behind ABCS. At each time \( t \), ABCS computes the derivative of the number of servers, i.e., how fast the system should increase or decrease the service capacity. The two functions \( \hat{r}_1(\cdot) \) and \( \hat{r}_2(\cdot) \)
control how fast the algorithm reacts, by increasing or decreasing the number of servers respectively. If the workload \( q(t) \) is non-zero, then the first term in the right-hand side of equation (4) increases the number of servers at rate \( \hat{r}_1(t) \). The second term is an "inertia term" which decreases the number of servers at rate \( \hat{r}_2(t) \). Note that if we integrate equation (4),

\[
\int_0^t \hat{r}_1(s) \omega \cdot q(s) \, ds = \int_0^t \beta \cdot \frac{dm(s)}{ds} \, ds + \int_0^t \hat{r}_2(t) \theta \cdot m(s) \, ds.
\]

This means that ABCS aims to carefully balance the flow-time with the switching cost plus the power consumption.

The reason behind the precise choices of \( \hat{r}_1(t) \) and \( \hat{r}_2(t) \) will be clear later from the performance of the algorithm. From a high-level perspective, these are chosen to approach the behavior of the "predicted" number of servers \( \tilde{m}(t) \). Indeed, if ABCS has less than the predicted number of servers \( \tilde{m}(t) \), then it increases \( m(t) \) at the higher rate \( \tilde{R}_1 \) and decreases it at the lower rate \( \tilde{R}_2 \). Similarly, if ABCS has "sufficiently" more servers than the predicted number \( \tilde{m}(t) \), then it increases \( m(t) \) at the lower rate \( \tilde{R}_1 \) and decreases it at the higher rate \( \tilde{R}_2 \). The number of servers of ABCS therefore judiciously approaches the number of predicted servers. However, it does not blindly follow \( \tilde{m}(t) \) to protect against inaccurate predictions. For example, if the workload \( q(t) \) is significantly higher than the current number of servers \( m(t) \), then ABCS will always increase the number of servers at a non-zero rate. The predicted number of servers \( \tilde{m}(t) \) consists of two components, an offline component \( \tilde{m}_1 \) and an online component \( \tilde{m}_2 \). The offline component \( \tilde{m}_1 \) is determined upfront by the optimal number of servers to handle the predicted arrival rate \( \lambda \). The online component \( \tilde{m}_2 \) is determined in an online manner and reacts if the actual arrival rate is higher than the predicted arrival rate.

The ABCS algorithm is memoryless and computationally cheap. The derivative of the number of servers depends only on the current workload and number of servers, without requiring knowledge about the past workload, number of servers or arrival rate.

Our main result characterizes the performance of ABCS analytically, which is presented in Theorem 3.1 below. The detailed proofs of all of the results are given in [5].

**Theorem 3.1.** Let \( r \geq 1 \) be a hyperparameter, representing the confidence in the predictions. Let \( \text{CR}(\eta) \) denote the competitive ratio of ABCS (Algorithm 1) when \( \eta \) has access to an \( \eta \)-accurate prediction. If \( \tilde{R}_1 = \Theta(r) \), \( \tilde{R}_2 = \Theta(r^{-1}) \), then

\[
\text{CR}(\eta) \leq \min \left( 1 + \left( \frac{\sqrt{2} \omega \beta + \theta \eta}{1 + \Theta(r^{-1})} \right), \Theta(r^3) \right).
\]

Another interesting thing to note is that for \( r > 1 \), the competitive ratio in (5) is the minimum of two terms: the first term, which we call the Optimistic Competitive Ratio (OCR), is smaller when the prediction is accurate and the second term, which we call the Pessimistic Competitive Ratio (PCR), is smaller when the prediction is inaccurate. From the algorithm designer’s perspective, there is a clear trade-off between OCR and PCR, which is conveniently controlled by the confidence hyperparameter \( \eta \). It is important to note that ABCS provides performance guarantees for any fixed \( \eta \geq 1 \) irrespective of the model parameters or the accuracy of the predictions. However, the choice of \( \eta \) reflects the risk that the system designer is willing to take in the pessimistic case against the gain in the optimistic case. This trade-off, however, is not specific to our algorithm as the next theorem shows.

**Theorem 3.2.** Let \( A \) be any deterministic algorithm for the capacity scaling problem in (1), and \( \text{CR}(\eta) \) denote its competitive ratio when \( \eta \) has access to an \( \eta \)-accurate prediction. There exist choices of \( \omega, \beta, \) and \( \theta \) such that, for any \( \delta > 0 \), if \( \text{CR}(0) \leq 1 + \delta \), then

\[
\text{CR}(\frac{2}{\delta^2}) \geq \frac{1}{\delta^2}.
\]

To test the performance of our algorithm in practice, we implemented them on both a real-world dataset of DNS requests observed at a campus network and a set of artificial datasets, and the results appear to be promising.

4. REFERENCES


