Online Optimization in Cloud Resource Provisioning: Predictions, Regrets, and Algorithms

JOSHUA COMDEN, Stony Brook University, USA
SIJIE YAO, Stony Brook University, USA
NIANGJUN CHEN, Institute of High Performance Computing, Singapore
HAIPENG XING, Stony Brook University, USA
ZHENHUA LIU*, Stony Brook University, USA

Due to mainstream adoption of cloud computing and its rapidly increasing usage of energy, the efficient management of cloud computing resources has become an important issue. A key challenge in managing the resources lies in the volatility of their demand. While there have been a wide variety of online algorithms (e.g. Receding Horizon Control, Online Balanced Descent) designed, it is hard for cloud operators to pick the right algorithm. In particular, these algorithms vary greatly on their usage of predictions and performance guarantees. This paper aims at studying an automatic algorithm selection scheme in real time. To do this, we empirically study the prediction errors from real-world cloud computing traces. Results show that prediction errors are distinct from different prediction algorithms, across virtual machines, and over the time horizon. Based on these observations, we propose a simple prediction error model and prove upper bounds on the dynamic regret of several online algorithms. We then apply the empirical and theoretical results to create a simple online meta-algorithm that chooses the best algorithm on the fly. Numerical simulations demonstrate that the performance of the designed policy is close to that of the best algorithm in hindsight.

CCS Concepts: • **Social and professional topics → Computing equipment management**; • **Theory of computation → Online learning algorithms**; **Regret bounds**; • **Computer systems organization → Cloud computing**.

Additional Key Words and Phrases: online optimization; meta-algorithms; resource allocation

ACM Reference Format:
Joshua Comden, Sijie Yao, Niangjun Chen, Haipeng Xing, and Zhenhua Liu. 2019. Online Optimization in Cloud Resource Provisioning: Predictions, Regrets, and Algorithms. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 1, Article 16 (March 2019), 30 pages. https://doi.org/10.1145/3311087

*Corresponding Author

Authors' addresses: Joshua Comden, Stony Brook University, Department of Applied Mathematics & Statistics, 100 Nicolls Road, Stony Brook, NY, 11794, USA, joshua.comden@stonybrook.edu; Sijie Yao, Stony Brook University, Department of Applied Mathematics & Statistics, 100 Nicolls Road, Stony Brook, NY, 11794, USA, sijie.yao@stonybrook.edu; Niangjun Chen, Institute of High Performance Computing, 1 Fusionopolis Way, #16-16 Connexis North, 138632, Singapore, chennj@ihpc.a-star.edu.sg; Haipeng Xing, Stony Brook University, Department of Applied Mathematics & Statistics, 100 Nicolls Road, Stony Brook, NY, 11794, USA, haipeng.xing@stonybrook.edu; Zhenhua Liu, Stony Brook University, Department of Applied Mathematics & Statistics, 100 Nicolls Road, Stony Brook, NY, 11794, USA, zhenhua.liu@stonybrook.edu.

2476-1249/2019/3-ART16 $15.00
https://doi.org/10.1145/3311087

1 INTRODUCTION

The usage of cloud services has been increasing rapidly in the past decade and shows no signs of stopping. In fact, from 2012 to 2015 it represented 70% of the IT industry's revenue growth and is expected to still be 60% in 2020 [14]. One of the main reasons for an enterprise to make the decision to move from local IT hosting to the cloud is the ability to quickly scale up IT resources without large up-front costs and avoid the over-provisioning costs that come with building excess capacity [7]. However, this essentially transfers the IT resource provisioning problem from the enterprise to the cloud service provider. Therefore, providing *and* provisioning IT resources in a cost effective manner are major components of the cloud business model.

There are different types of costs that a cloud provider incurs in provisioning resources which have different characteristics. *Operational costs* come in the form of having a direct dependency on the amount of resources provided which include electricity and cooling costs, amortized hardware, and service level agreement violation penalties. Whereas, *switching costs* depend on the changes in provisioning decisions such as extra wear and tear on the hardware and startup/shutdown delay costs [38]. The inclusion of switching costs into resource provisioning decisions couples the decisions in time which means that control methods should be looking into the future well beyond the next decision.

Several different control methods are used in practice or have been proposed to cost-effectively provision IT resources. There are methods that do not use future information but instead build realistic models to make smart threshold triggered provisioning decisions [8, 34, 40] or employ gradient descent techniques [47, 53]. On the other extreme, other methods rely heavily on predictions of the future with variants on model predictive control [2, 5, 6, 33, 36, 44, 49, 50].

Due to the dependency of many control methods on having accurate predictions of the future to make good provisioning decisions, there has been a great deal of literature on prediction workload demand. Prediction models include Exponential Smoothing [30, 41], Markov chains [28], AutoRegressive [35], AutoRegressive Moving Average [34, 44], Holt-Winter [24], Naive Bayes [43], Neural Network [31], and Pattern Matching [16].

However, even with all of this literature on workload predictions and their utilization in control algorithms, the understanding of prediction error and how to handle it remains an important open issue and research challenge [1]. In this paper we aim to mend this gap by making the following contributions:

(1) Prediction error is modeled to aid in proving worst-case dynamic regret bounds for control algorithms. The model is simple, intuitive, and represents error in terms of the extra control cost from prediction inaccuracies. This allows prediction models to be compared and chosen based on potential control loss. (Section 3)
(2) Upper bounds on dynamic regret are proven for a variety of algorithms in terms of the prediction error model. In order to choose which algorithm to run without prediction error knowledge, a simple online meta-algorithm is designed to carefully select which algorithm to follow based on past performance. (Section 4)
(3) A detailed analysis of prediction accuracy is done for cloud computing by fitting real-world CPU utilization traces of Azure virtual machines to popular prediction models. We demonstrate that there is a large spectrum of prediction accuracy among virtual machines. (Section 5)
(4) Using real-world trace based simulations of CPU allocation for virtual machines, the proposed meta-algorithm is shown to outperform a popular algorithm selection policy and perform very closely to that of the best algorithm chosen in hindsight. (Section 6)

2 RELATED WORK

In order to make theoretical guarantees on cloud resource provisioning problems that include switching costs which couple the decisions in time, [36–38] abstracted the problem into a general framework called *Smoothed Online Convex Optimization (SOCO)*. These switching costs are assumed to satisfy the triangle inequality and give a *smoothing* effect between consecutive actions. In the Metrical Task Systems (MTS) [11–13] community they are known as the transition costs. In regards to having no predictions of the future, [3] proved that it is impossible to design an algorithm for SOCO that has *both* sublinear regret and a constant-competitive ratio. When predictions are available, Model Predictive Control (MPC) [27] also called Receding Horizon Control (RHC) from the control theory community was first thought as an excellent candidate algorithm to be used for SOCO since it uses the most up-to-date predictions when making the next decision. However, [36] proved that the competitive ratio of RHC applied to SOCO, although constant, does not improve with an increased number of predictions. This result is true even if the predictions have no error and increases as the switching cost increases. Unfortunately this means that even if RHC may be performing well, it could unexpectedly perform poorly. In response to the poor worst-case guarantees of RHC, [36] proposed a different algorithm called Averaging Fixed Horizon Control (AFHC) that incorporates out-dated and the most up-to-date predictions in making the next decision. This unintuitive approach makes AFHC less susceptible to changing actions too quickly when given new predictions which gives it a competitive ratio that decreases as the number of perfect predictions increases. For imperfect predictions with noisy errors, [17] showed that AFHC applied to online LASSO has a constant average-case competitive ratio and an average-case regret that is sublinear in the number of rounds. [18] generalized RHC and AFHC to an algorithm called Committed Horizon Control (CHC) and proved an upper bound on the competitive difference which is linear in terms of the time horizon and includes prediction errors.

One of the inspirations for our prediction error model came from that of Besbes et al. [9] where they used a variation budget to limit how much an adversary could change the environment from its current state. For their work, they also designed a meta-algorithm but it was based on partitioning the time horizon into batches and running an Online Convex Optimization algorithm with static regret guarantees to achieve dynamic regret guarantees. However, our main focus was on designing a meta-algorithm that could work with unknown prediction capabilities regardless of environmental conditions whereas they were focused on dealing with an environment that is non-stationary regardless of predictability.

3 PROBLEM FORMULATION

3.1 Model

Suppose that a cloud service provider needs to continually provision resources to meet a dynamically changing demand from its subscribers in an online manner over a time horizon T. At timeslot $t \in \{1, \ldots, T\}$, applications of a cloud service subscriber have requests on different resources like network, CPU, or memory. Let $y_t \in \mathbb{R}^n$ be the vector of demands at timeslot t where each dimension represents a type of resource requested; let $x_t \in \mathcal{X}$ be the vector of resources provisioned in the constrained provisioning action space $\mathcal{X} \subset \mathbb{R}^m$.

Note this model is general enough for services provided by geographically distributed resources, e.g, datacenters, where each dimension in y_t and x_t corresponds to the demand and supply of a resource at a given location. It can also be applied to an ensemble of virtual machines where each dimension in x_t corresponds to the supply of a resource for a particular individual virtual machine and each dimension in y_t corresponds to a total demand for a particular resource that must be

supplied among the ensemble. Cross-correlation among the elements of y_t in time can be used to represent the correlated demand among resources.

In provisioning x_t, the provider experiences two types of costs: (i) operational costs and (ii) switching costs.

Operational costs: such as the monetary cost of reserving and using virtual machines, amortized capital costs and energy expenditure to run local resources, as well as delay cost (revenue loss, penalty for SLA violations) when resources are under-provisioned. We model these costs by a convex function $f(x_t, y_t)$ of the demand and the provisioned resources. Through the vector y_t, the form of the function is general enough to capture a wide range of parameterized cost models for server power consumption, e.g., [4, 26, 51] as well as latency, e.g., [20, 36, 45]. For example in speed scaling, [51] uses the convex operational cost form $cx_t^\gamma + \frac{y_t}{x_t - y_t}$ for a M/GI/1 Processor Sharing queue where the scalar y_t is the processing load demand and the scalar x_t is the speed at which the processors are running. The first term represents the nonlinear function of speed on the average power consumption with the parameter $\gamma > 1$, while the second term represents the average response time. The parameter c balances the cost between power consumption and average response time.

Switching costs: such as wear and tear and delay from startup and shutdown of servers, as well as cost due to virtual machine migration and data transfer. Depending on the type of resource and risk aversion of the service provider, the cost of changing resource allocations can be equivalent to running them continually for a few minutes to several hours [10, 21, 48]. We model these costs with a general norm of the difference in the consecutive provisioning decisions $\alpha\|x_t - x_{t-1}\|$ where α represents the unit cost of change decisions.

The following optimization problem brings together these costs along with the actions that the cloud service provider needs to decide for minimizing cost:

$$\min_{\{x_1,\ldots,x_T\}\in\mathcal{X}} \quad \sum_{t=1}^{T} \left(f(x_t, y_t) + \alpha\|x_t - x_{t-1}\|\right) \tag{1}$$

where x_0 is the given starting point. Let the optimal solution be denoted as $\left(x_1^*, \ldots, x_T^*\right)$.

We make the mild assumption that the action space $\mathcal{X}$ is compact and convex and we define the diameter D of the action space such that $\|x_1 - x_2\| \leq D : \forall\{x_1, x_2\} \in \mathcal{X}$. This is motivated by the fact that cloud resources have capacity constraints. Additionally, we assume that $f(x, y)$ is G-Lipschitz continuous for any given demand y with respect to its provisioning decision x:

$$|f(x_1, y) - f(x_2, y)| \leq G\|x_1 - x_2\|_2 \quad \forall\{x_1, x_2\} \in \mathcal{X}, \forall y \tag{2}$$

which allows for the case when $f(x, y)$ is nondifferentiable.

3.2 Online Optimization

If the cloud service provider knows all of the demands $y_1, \ldots, y_T$ at the beginning, problem (1) can be solved efficiently. However, since the cloud service provider cannot know all demands beforehand, there is insufficient information to solve this optimization problem. Instead, the provider must solve an online version of the problem with the goal of approximating the optimal solution to the original Problem (1).

In practice, the cloud service provider tackles the lack of future information by making predictions on future demands. Denote $\hat{y}_{t|\tau}$ as the prediction of y_t using information up to $\tau - 1$ for $\tau \leq t$. Assuming a prediction window of w steps ahead, we can model the decisions and information flow of the cloud service provider in the following manner. At each timeslot $t \in \{1, \ldots, T\}$:

(1) The cloud service provider predicts the future demands $\hat{y}_{t|t},\ldots,\hat{y}_{t+w-1|t}$ and makes an allocation decision $x_t \in \mathcal{X}$.
(2) Subsequently, the true demand y_t is revealed and the cloud service provider incur the cost $f(x_t, y_t) + \alpha\|x_t - x_{t-1}\|$.

The cost of a particular online algorithm $\mathcal{A}$ under the sequence of resource demands $y_{1:T} := (y_1, \ldots, y_T)$ is therefore:

$$\text{cost}(\mathcal{A}, y_{1:T}) = \sum_{t=1}^{T} \left(f\left(x_t^{\mathcal{A}}, y_t\right) + \alpha \left\| x_t^{\mathcal{A}} - x_{t-1}^{\mathcal{A}} \right\| \right). \tag{3}$$

where $x_0^{\mathcal{A}} = x_0$ is assumed.

To compare an online algorithm's performance to that of the offline optimal solution, we employ the worst-case performance metric *dynamic regret* constrained by the path length of the optimal solution.

Definition 1. *We define the L-constrained dynamic regret of online algorithm $\mathcal{A}$ to be:*

$$\mathcal{R}_T^{\mathcal{A}}(L) := \sup_{y_{1:T}} \{cost(\mathcal{A}, y_{1:T}) - cost(OPT_L, y_{1:T})\} \tag{4}$$

where OPT_L is the optimal solution of (1) with respect to an additional constraint on its path-length:

$$\sum_{t=1}^{T} \|x_t^* - x_{t-1}^*\| \leq L. \tag{5}$$

Note that by varying the value of L, Definition 1 smoothly interpolates between the notion of static regret [54] where $L = 0$ and the unconstrained dynamic regret [29, 32] widely studied in online convex optimization where $L = \infty$.

3.3 Prediction Error Model

Predictions are crucial to online decision making. However, the main challenge with incorporating realistic predictions into the analysis of online algorithms is the difficulty in quantifying the impact of prediction errors on an online algorithm's performance. Most design and analysis to date avoid this issue altogether in two ways. (i) They optimistically assume simple prediction errors such as a finite prediction window that contains perfect predictions [36, 38, 52] and more recent work with probabilistic prediction models which provides theoretical bounds on *expected* competitive performance [17, 18]. (ii) They pessimistically assume that no trustworthy predictions are available [29, 32, 46, 54]. This is similar to imposing no restrictions on the prediction errors where an adversary can make the predictions useless by always giving the decision maker completely wrong information. In this case, the online algorithm can only ignore predictions and decide the action solely based on past information.

However, predictions in cloud computing are neither perfect nor useless. As will be shown in Section 5, predictions can vary from being almost perfect to extremely inaccurate. As a result, the current dichotomy of analyzing algorithms based on either perfect or adversarial predictions tells us little about which algorithm(s) should be used in which environments.

To understand how to choose online algorithms under realistic predictions, we introduce a simple prediction error model that connects prediction errors to the potential performance loss. This is done by restricting the adversary's power in deciding how poor the predictions can be in terms of possible performance loss. Essentially, the adversary is given a budget B_T of error in the form of performance loss to allocate among the timeslots $\{1, \ldots, T\}$ any way it chooses.

Specifically, recall that $\hat{y}_{t|t}$ denotes the prediction of y_t at the beginning of time t before it is observed.

Definition 2. *The sequence of true demands and the corresponding predictions* $(y_1, \hat{y}_{1|1}), \ldots, (y_T, \hat{y}_{T|T})$ *satisfy a prediction budget of* B_T *if*

$$\sum_{t=1}^{T} \sup_{x \in \mathcal{X}} \left| f(x, y_t) - f(x, \hat{y}_{t|t}) \right| \leq B_T \tag{6}$$

This definition gives a deterministic upper bound on the loss in operation cost due to prediction errors, and is also flexible enough to allow adversarial allocations of errors over time. See Theorem 3 in Section 4 which demonstrates the direct impact errors in the form of a budget has on the performance upper bound of an online algorithm.

The prediction error model can be further extended to include multiple steps of prediction with error budgets. Specifically,

Definition 3. *The sequence of true demands and their k-step ahead predictions* $(y_k, \hat{y}_{k|1}), \ldots, (y_T, \hat{y}_{T|T-k+1})$ *satisfy a k-step prediction budget* $B_{k,T}$ *if*

$$\sum_{t=k}^{T} \sup_{x \in \mathcal{X}} \left| f(x, y_t) - f(x, \hat{y}_{t|t-k+1}) \right| \leq B_{k,T}. \tag{7}$$

To quantify online algorithms leveraging predictions up to w steps, let $\mathcal{Y}_{1:w,T}$ denote the set of sequences of true demands and their various k-step predictions that satisfy (7) for all $k \in \{1, \ldots, w\}$. Using the multi-step prediction error model, the dynamic regret can be further restricted to include the error budget in its definition.

Definition 4. *We define the L-constrained dynamic regret of online algorithm* $\mathcal{A}$ *with prediction error budgets* $B_{1,T}, \ldots, B_{w,T}$ *to be:*

$$\mathcal{R}_T^{\mathcal{A}}(B_{1:w,T}, L) := \sup_{\mathbf{y} \in \mathcal{Y}_{1:w,T}} \left\{ cost(\mathcal{A}, y_{1:T}) - cost(OPT_L, y_{1:T}) \right\}. \tag{8}$$

Here $\mathbf{y}$ is a specific instance in $\mathcal{Y}_{1:w,T}$ and L is the upper bound constraint on the dynamic offline optimal OPT_L solution's path length as defined by Equation (5). Like the single step prediction error budget, the multi-step prediction error budget can be used to prove upper bounds on dynamic regret for online algorithms which utilize multiple steps of prediction (See Theorem 4 in Section 4).

4 ONLINE ALGORITHMS

There exist many online algorithms which can be categorized by how much prediction they utilize. We state several of them and prove their dynamic regret upper bounds in the presence of prediction errors using the prediction error budget model. Afterwards, we give a simple yet practical online meta-algorithm to choose which algorithm to implement based on past performance.

4.1 No Predictions

A very popular algorithm is Online Gradient Descent (OGD) which chooses its next action by moving from the current action along the descent direction, i.e., the opposite direction of the gradient in the current step [54]. It is computationally easy and requires no memory since the only input to its decision comes from the previous timeslot.

The projected version of OGD is formally stated in Algorithm 1 where g_{t-1} is a (sub)gradient, $P_{\mathcal{X}}(\cdot)$ is the euclidean projection operator onto the action space $\mathcal{X}$ and η is the stepsize which determines how aggressively OGD moves along the descent direction.

Algorithm 1 Online Gradient Descent (OGD)

1: **for** $t = 1, \ldots, T$ **do**
2: Return $x_t = P_{\mathcal{X}}(x_{t-1} - \eta g_{t-1})$.
3: **end for**

Recall L is the largest path-length allowed for the optimal solution, i.e., $\sum_{t=1}^{T} \|x_t^* - x_{t-1}^*\| \leq L$. An upper bound on OGD's dynamic regret is stated in the following theorem for any given constant stepsize η.

Theorem 1. *OGD has the following upper bound on dynamic regret:*

$$\mathcal{R}_T^{OGD}(L) \leq \frac{2D^2\overline{\kappa}^2}{\eta} + L\left(\frac{D\overline{\kappa}}{\eta} - \alpha\right)^+ + \eta G\left(\frac{G}{2} + \frac{\alpha}{\underline{\kappa}}\right)T \tag{9}$$

where $\underline{\kappa}$ and $\overline{\kappa}$ are positive constants such that $\underline{\kappa}\|x\| \leq \|x\|_2 \leq \overline{\kappa}\|x\|$, and $(x)^+ := \max\{0, x\}$.

See Appendix A.1 for the proof.

The bound in Theorem 1 seems like on the order of $O(T)$. However, if we pick η by minimizing the RHS of (9), an $O(\sqrt{LT})$ upper bound on dynamic regret can be obtained.

Corollary 2. *If the stepsize η is set to $\sqrt{\frac{D\overline{\kappa}(2D\overline{\kappa}+L)}{G\left(\frac{G}{2}+\frac{\alpha}{\underline{\kappa}}\right)T}}$, then OGD has the following upper bound on dynamic regret:*

$$\mathcal{R}_T^{OGD}(L) \leq 2\sqrt{D\overline{\kappa}(2D\overline{\kappa} + L)G\left(\frac{G}{2} + \frac{\alpha}{\underline{\kappa}}\right)T}. \tag{10}$$

Proof. Plugging $\eta = \sqrt{\frac{D\overline{\kappa}(2D\overline{\kappa}+L)}{G\left(\frac{G}{2}+\frac{\alpha}{\underline{\kappa}}\right)T}}$ into Theorem 1 and noting that $\frac{D\overline{\kappa}}{\eta} \geq \left(\frac{D\overline{\kappa}}{\eta} - \alpha\right)^+$ gets the resultant after simplifying the terms. □

As simple and straightforward as OGD is, if the optimal path length L grows sublinearly with the time horizon T, OGD achieves a sublinear dynamic regret. See Section 6 for examples of this scenario. Clearly, if L is linear in T, the dynamic regret of OGD is linear. Since it does not make use of any future information, incorporating predictions should intuitively help.

4.2 Utilizing Single-step Predictions

With single-step predictions, we have a better, albeit imperfect, idea of what situation to expect this coming timeslot. One way to make use of this one step of prediction is Online Balanced Descent (OBD) recently introduced by Chen et al in 2018 [19]. The main idea of OBD is to project onto an appropriate sublevel set of the current predicted cost $f(x, \hat{y}_t)$. The sublevel set is chosen so that the operating cost and the switching cost are balanced. The details are stated in Algorithm 2.

Algorithm 2 Online Balanced Descent (OBD)

1: **for** $t = 1, \ldots, T$ **do**
2: Define $x(l) = P_{K_l}^{\Phi}(x_{t-1})$, increase l from $l = f_t(v_t)$, until $\|\nabla\Phi(x(l)) - \nabla\Phi(x_{t-1})\|_* = \eta\|\nabla f_t(x(l))\|_*$.
3: Return $x_t = x(l)$.
4: **end for**

At time t, let K_l be the l-sublevel set of $f_t(x, \hat{y}_t)$ that is feasible, i.e. $K_l = \{x \in \mathcal{X} | f_t(x, \hat{y}_t) \leq l\}$. Also, let Φ be an m-strongly convex function in terms of the norm $\|\cdot\|$ defined by the switching cost. For example, if the switching cost is defined by the L2 norm, we can pick $\Phi(x) = \frac{1}{2}\|x\|^2$. Define the projection operator onto the sublevel set with respect to Φ as

$$P^{\Phi}_{K_l}(x) = \arg\min_{z \in K_l} D_{\Phi}(x, z),$$

where $D_{\Phi}(x, y) = \Phi(x) - \Phi(y) - \langle \nabla\Phi(y), x - y\rangle$ is the Bregman divergence induced by Φ. Since Φ is m-strongly convex in $\|\cdot\|$, $D(x, y) \geq \frac{m}{2}\|x - y\|^2$.

With the prediction error budget B_T of the true demand and its one-step predictions sequence $(y_1, \hat{y}_{1|1}), \ldots, (y_T, \hat{y}_{T|T})$, we can bound the dynamic regret of OBD as the following:

Theorem 3. *OBD has the following upper bound on dynamic regret given one-step noisy predictions with error budget B_T:*

$$\mathcal{R}^{OBD}_T(B_T, L) \leq 2B_T + \alpha\sqrt{\frac{2GLT}{m}}. \tag{11}$$

See Appendix A.2 for the proof, we can see that, when both the prediction error budget B_T and the path-length L is sublinear, then OBD with one step prediction has sublinear dynamic regret.

4.3 Utilizing Multi-step Predictions

The most straightforward way to use predictions that look multiple steps ahead is to optimize the next action as if the predictions were true. This is the main intuition behind Receding Horizon Control (RHC) which has a long rich history of both theoretical and practical significance [15, 27, 42].

Specifically, suppose that there are predictions available up to w steps ahead starting from timeslot t. RHC uses them as input to the following optimization problem which is an estimation of its cost for the next w timeslots:

$$\min_{\{x_t, \ldots, x_{t+w-1}\} \in \mathcal{X}} \sum_{\tau=t}^{t+w-1} \left(f(x_\tau, \hat{y}_{\tau|t}) + \alpha\|x_\tau - x_{\tau-1}\|\right). \tag{12}$$

Define $X(x_{t-1}, \hat{y}_{t|t}, \ldots, \hat{y}_{t+w-1|t}) \in \mathcal{X}^w$ to be a solution to the w-step lookahead optimization (12). RHC implements only x_t from the solution which corresponds to $X_1(x_{t-1}, \hat{y}_{t|t}, \ldots, \hat{y}_{t+w-1|t})$ before moving to the next timeslot and repeating the procedure. The formal procedure is stated in Algorithm 3.

Algorithm 3 Receding Horizon Control (RHC)

1: **for** $t = 1, \ldots, T$ **do**
2: Solve (12) and return $x_t = X_1(x_{t-1}, \hat{y}_{t|t}, \ldots, \hat{y}_{t+w-1|t})$.
3: **end for**

Unfortunately however, [36] proved that RHC can have a competitive ratio that does not improve as the prediction window size increases, even if the predictions are perfect. Note that a constant competitive ratio is equivalent to a linear dynamic regret.

The work of [18] generalized RHC by providing a class of algorithms called Committed Horizon Control (CHC). It has a tunable a parameter $v \in \{1, \ldots, w\}$ called the commitment level which sets how much influence old decision trajectories have on the current decision. These old decision trajectories result from past solutions of (12) which used past predictions but were not implemented. The formal details are presented in Algorithm 4.

Algorithm 4 Committed Horizon Control (CHC)

if $t = 1$ **then**
  for $k = 2, \ldots, v$ **do**
    Solve (12) starting from x_0 using $k-1$ steps of predictions.
    Set $(x_1^{(k)}, \ldots, x_{k-1}^{(k)}) = X(x_0, \hat{y}_{1|1}, \ldots, \hat{y}_{k-1|1})$.
  end for
end if
for $t = 1, \ldots, T$ **do**
  Set $k = 1 + (t-1) \mod v$.
  Solve (12) starting from $x_{t-1}^{(k)}$ using w steps of predictions.
  Set $(x_t^{(k)}, \ldots, x_{t+v-1}^{(k)}) = X_{1:v}(x_{t-1}^{(k)}, \hat{y}_{t|t}, \ldots, \hat{y}_{t+w-1|t})$.
  Return $x_t = \frac{1}{v}\sum_{j=1}^{v} x_t^{(j)}$.
end for

Note that RHC is CHC with $v = 1$ where no old decision trajectories are incorporated in the current decision as it just solves (12) and implements the first decision with no memory of its past solutions. An advantage to having old decision trajectories influence the current decision is that it can smooth the implemented decisions being made. However, this also incorporates extra loss from those predictions that were made looking further into the future.

The prediction error budget model (7) gives us enough structure on the prediction errors to prove the following upper bound on dynamic regret for CHC.

Theorem 4. *CHC has the following upper bound on dynamic regret:*

$$\mathcal{R}_T^{CHC}(B_{1:w,T}, L) \leq \frac{2}{v}\left(D\alpha T + \sum_{k=1}^{v} B_{k,T}\right). \tag{13}$$

See Appendix A.3 for the proof.

Theorem 4 shows us that the commitment level v has two opposing effects on the dynamic regret of CHC which depends directly on the effect that the lookahead has on the error budget. If the error budget does not increase quickly as the lookahead k increases, then increasing v will make the upper bound smaller. However, if the error budget increases rapidly as one lookahead further in time, then v should remain small.

On the other side, we provide a lower bound on CHC.

Theorem 5. *Assume $L \geq D$. For any v, CHC has the following lower bound on dynamic regret:*

$$\mathcal{R}_T^{CHC}(B_{1:w,T}, L) \geq \left(\frac{1}{w+1}T - 1\right)\alpha D. \tag{14}$$

See Appendix A.4 for the proof.

Notice that the RHS of (14) holds even when the algorithm has access to perfect predictions. The bad example used in proving the theorem shows us that in an application environment with an area of shallows cost, it is possible for CHC to get stuck at a single point in an area of shallow cost and incur a cost that is a constant amount more than that of the optimal solution at every timeslot. This can happen in a cloud computing environment when the cost of over-provisioning a resource for a short timescale is relatively low compared to its switching cost. This situation can cause CHC to constantly over-provision beyond the optimal level since that within its given prediction window it sees that switching to a lower level would be too expensive. Over a long time

Table 1. Upper Bounds on Dynamic Regret by Algorithm

Alg.	Dynamic Regret
OGD	$2\sqrt{D\overline{\kappa}(2D\overline{\kappa}+L)G\left(\frac{G}{2}+\frac{\alpha}{\underline{\kappa}}\right)T}$
OBD	$2B_T + \alpha\sqrt{\frac{2GLT}{m}}$
CHC	$\frac{2}{v}\left(D\alpha T + \sum_{k=1}^{v} B_{k,T}\right)$

Table 2. Dynamic Regret depending on the complexity of the error budget B_T and the optimal path length L.

Alg.	$B_{k,T} \in o(T)$ $L \in o(T)$	$B_{k,T} \in O(T)$ $L \in o(T)$	$B_{k,T} \in o(T)$ $L \in O(T)$	$B_{k,T} \in O(T)$ $L \in O(T)$
OGD	$o(T)$	$o(T)$	$O(T)$	$O(T)$
OBD	$o(T)$	$O(T)$	$O(T)$	$O(T)$
CHC	$O(T/v)$	$O(T)$	$O(T/v)$	$O(T)$

horizon, the over-provisioning costs would accumulate to becoming much greater than that of switching to a near optimal level earlier in time. However on the plus side, this theorem gives us the key to avoiding that scenario which is to utilize more predictions with a larger w.

Observe that the upper bound and lower bound just about meet when $v = w$ and there is zero error budget. Under those conditions, the bounds decrease with increasing w. Unsurprisingly, this means that if perfect predictions are available, they should all be used. However, when prediction error exists, the commitment level should be chosen carefully depending on how quickly the error budget $B_{k,T}$ increases with k while increasing the prediction window w is always advantageous with respect to the lower bound.

4.4 Algorithm Selection

With all of the available algorithms described above, the decision becomes which one(s) should be chosen at runtime and at what parameter settings. The proved upper bounds on dynamic regret for all the described algorithms are summarized in Table 1. Remember that CHC also includes RHC when $v = 1$. Note that even thought the stepsize setting η for OGD given in Corollary 2 minimizes the worst-case, it may not be the best setting to use in practice since it may rarely or never experience adversarial worst-cases.

Additionally, it is important to know under what conditions and for which algorithms that dynamic regret can be proved to be sublinear with respect to the time horizon T. This depends on the complexity that error budget $B_{k,T}$ and optimal path length L have on the dynamic regret. Table 2 shows the dynamic regret complexities for the described algorithms under different complexities of $B_{k,T}$ and L. OGD gives sublinear dynamic regret whenever the optimal path length L is sublinear. OBD can also obtain sublinear dynamic regret when both B_T and L are sublinear. When the error budgets are sublinear, CHC can lower its upper bound by increasing its commitment level v towards the prediction window size w. However when the error budgets are linear, the optimal v depends on the relationship of the steps of prediction k on $B_{k,T}$.

Nonetheless in practice, it may be difficult to know beforehand the relationship of the optimal solution's path length L to that of the time horizon T. Actually, even the time horizon itself is not usually known beforehand. Therefore, the decision of which algorithm to choose and its parameter setting should be decided in an online manner. In this direction, we design a simple meta-algorithm

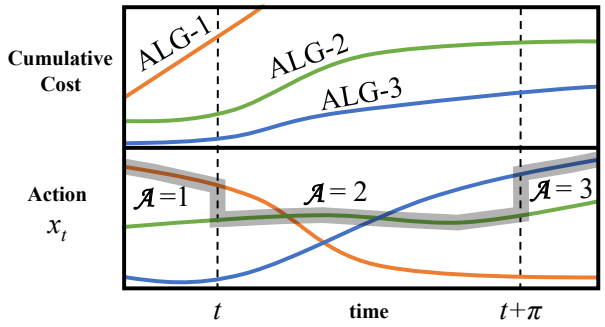


Fig. 1. Meta-algorithm example. Initially the meta-algorithm is on ALG-1 before time t, however it notices that ALG-1's cumulative cost is increasing rapidly. At t, it switches to ALG-2 and not ALG-3 because the switching cost associated with distance between ALG-1 and ALG-3 is too large. However at $t+\pi$, the distance between ALG-2 and ALG-3 allow the meta-algorithm to switch to ALG-3 which still has the lowest cumulative cost.

that selects which algorithm from a finite set Γ to implement and periodically updates its selection according to past performance. The technical details are given in Algorithm 5.

Algorithm 5 Meta-algorithm

1: Initialize selected algorithm, $\mathcal{A} \in \Gamma$.
2: **for** $t = 1, \ldots, T$ **do**
3: **if** $t \mod \pi = 0$ **then**
4: Select $\mathcal{A} := \arg\min_{i \in \Gamma} \left\{ \text{cost}(i, y_{1:t-1}) + \alpha \| x_{t-1}^{\mathcal{A}} - x_{t-1}^{(i)} \| \right\}$
5: **end if**
6: Receive $x_t^{(i)}$ from every algorithm $i \in \Gamma$.
7: Implement $x_t^{\mathcal{A}}$.
8: **end for**

It essentially learns which algorithm was the best in hindsight and switches to that while taking into account the cost of switching from its current algorithm trajectory to that of another. Specifically, it runs all algorithms in parallel and evaluates their costs according to their past decisions. At every π timeslots, it greedily chooses the algorithm which has the lowest current total incurred cost in addition to the cost of switching to that algorithm's decision trajectory. It then commits to that algorithm for the next π timeslots. See Figure 1 for an explanatory demonstration of what is happening at Line 4 of Algorithm 5. We show its practical performance in Section 6.

For some practical applications, it may very costly to run all possible algorithms under a large number of settings simultaneously due to the processing power constraints of the controller. In these cases, the upper bounds shown in Table 1 along with past observed values of their parameters can be used to appropriately select a small number of algorithms under limited settings for the meta-algorithm's set Γ.

5 VIRTUAL MACHINE PREDICTION

Before applying the online algorithms described in Section 4 to cloud resource provisioning, we need to first analyze the workload traces and generate predictions for the online algorithms. In this

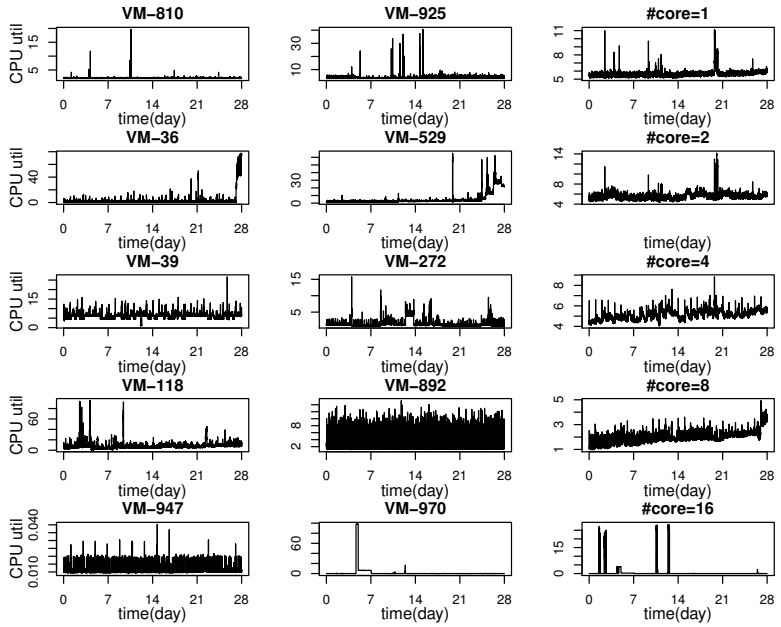


Fig. 2. Time series of VM CPU utilization with reserved number of cores 1, 2, 4, 8, and 16 (from Row 1 to Row 5, respectively).

section, we first study the MS Azure dataset in Section 5.1. Then we analyze the prediction errors from several methods in Section 5.2 and model the errors in Section 5.3.

5.1 The MS Azure Dataset

We study the CPU utilization of 1,003 long running virtual machines (VMs) from the Microsoft Azure Public Dataset [23], which contains over two million VMs that ran during a 30 day interval between the dates Azure from November 16, 2016 and February 16, 2017. The particular VM trace IDs used in this analysis are listed in [22]. For each VM, the utilization is measured every five minutes, so 288 data points are collected each day. In other words, this dataset contains 1,003 time series, each corresponding to the CPU utilization of a VM.

The VMs provision different numbers of CPU cores. In this dataset, VMs may reserve 1, 2, 4, 8, and 16 CPU cores due to the limited VM types provided in Azure. The numbers of VMs with cores 1, 2, 4, 8, and 16 are 557, 126, 193, 103, and 24, respectively.

To have a better idea on the time series and heterogeneity features of the data, we show time series of CPU utilization of VMs with cores 1, 2, 4, 8, and 16 in each row of Figure 2. On each row, the first two plots show the CPU utilization of two randomly selected VMs provisioning the corresponding number of CPU cores, and the third plot shows the time series of CPU utilization averaged across all VMs with the same number of cores.

Some interesting features of the data can be found in Figure 2. First, most series contain one or several big spikes in CPU utilization, and most spikes are not periodic.These spikes are important in cloud resource provisioning. If resources are under provisioned during these periods, large performance degradations may occur. Therefore, they can not be treated as outliers of the series like the traditional time series analysis. Second, all series show strong seasonal effects and are not stationary. Although the actual usage of CPU in each VM is different over time, the pattern of CPU

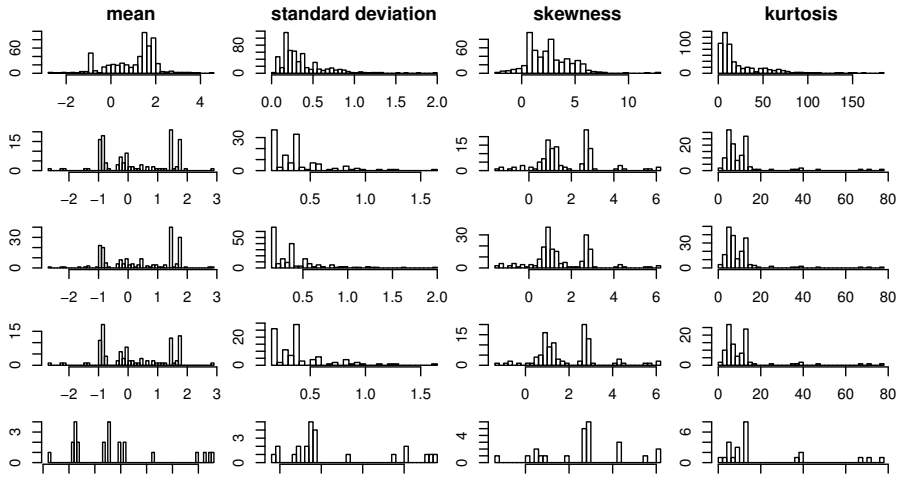


Fig. 3. Histograms of sample means, sample standard deviations, sample skewness, and sample kurtosis of CPU utilization of VMs with cores 1, 2, 4, 8, 16 (from Row 1 to Row 5, respectively). The reported sample statistics are calculated after taking the logarithm of the original trace values.

utilization in each day seems significant. Third, the patterns in the time series of CPU utilization for VMs with different number of cores seem different. For example, the spikes in VMs with 1 core or 16 cores are much stronger and irregular than those in VMs with 4 or 8 cores.

The time series of CPU utilizations are not log-normally distributed. Since utilization is positive, we consider the value after taking the logarithm. Figure 3 shows histograms of sample means, sample standard deviations, sample skewness, and sample kurtosis of the VM CPU utilization different cores. Two to four modes are observed in the sample means, and large dispersions are shown in the sample standard deviations. Besides, almost all series are positively skewed and most of their kurtosis are much larger than 3 (the kurtosis of a normal distribution), suggesting the logarithms series of CPU utilization are not normally distributed. These heterogeneous features indicate that it is difficult to fit a universal type of prediction models to the data.

5.2 Analysis of Empirical Prediction Error

We use four prediction methods for the dataset. The first one is kind of naive, which uses the last observation as the k-step-ahead prediction. The second model uses random forests (RF) regression, a widely used ensemble learning method. The third model use the seasonal exponential smoothing (SES) method to decompose the series into trend, seasonal and remainder components, and then extrapolates the trend effect as predictions. The fourth model is an extension of the SES model, which fits an ARMA(p, q) model to the de-seasoned series obtained in the SES model, we denote this as SARMA.

We obtain the out-of-sample predictions of the last three models in the following way. For the series of the ith VM, we use observations $\{y_1^i, \ldots, y_t^i\}$ as the training data and fit them in the RF and time series models to compute k-step-ahead predictions $\widehat{y}_{t+k|t}^i$, which is the prediction of the VM i's CPU utilization at time $t + k$ predicted at time t.

Table 3. MAE and MSDE of predictions for CPU utilization.

Step	MAE				MSDE			
	SES	SARMA	RF	Naive	SES	SARMA	RF	Naive
1	.709	.695	.650	.782	1.131	1.085	1.143	1.324
2	.765	.744	.730	.829	1.240	1.184	1.255	1.483
3	.795	.769	.768	.824	1.293	1.234	1.326	1.534
4	.825	.804	.806	.876	1.331	1.279	1.364	1.623
5	.847	.818	.838	.912	1.365	1.305	1.400	1.623

We use one-week trace as the test data to calculate the k-step-ahead prediction error, $E^i_{t,k} = |y^i_{t+k} - \widehat{y}^i_{t+k|t}|$. We consider two error metrics: the mean absolute error (MAE),

$$\text{MAE}_k = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=t_0}^{T} E^i_{t,k}, \tag{15}$$

and the mean standard deviation of errors (MSDE),

$$\text{MSDE}_k = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{T-1}\sum_{t=t_0}^{T}(E^i_{t,k} - \overline{E}^i_{t,k})^2}, \tag{16}$$

where N is the number of VMs (1,003 in the Azure dataset) and t_0 is the start of the test set. And $\overline{E}^i_{t,k}$ is the average value of $\{E^i_{t,k}\}$ with t from t_0 to T.

Table 3 summarizes the MAE and MSDE of different models for $k = 1, \ldots, 5$. We can see that both MAE and MSDE increase with larger prediction steps for all methods, and the naive prediction has the worst performance. RF has a smaller MAE, but the variance in error MSDE is larger than SES and SARMA. SARMA is slightly better than that of the SES model, both the SES and the SARMA models have smaller MSDE, indicating their predictions are more robust.

5.3 Error Budget Modeling

Based on the predictions obtained above, we consider the accumulated prediction errors of different methods over time. In order to find the relationship between steps of prediction and the error budget, we consider the averaged cumulative sums of k-step-ahead absolute prediction errors (CAPE) across all VMs,

$$\text{CAPE}_k(s) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=t_0}^{s} E^i_{t,k}. \tag{17}$$

Figure 4 and 5 show the CAPE of different models with $k = 1$ for $s = 0, \ldots, 2000$ for all VMs and 4 particular VMs, respectively. It is clear that the CAPE increases approximately linearly over time. Therefore, we consider the following error budget model in the rest of study,

$$B_{k,T} = b_k T \tag{18}$$

where b_k is expected to increase with k. Hence to investigate the property of b_k when different prediction step k is employed, we compare the CAPE_k of VM-118 in Figure 6 with $k = 1, 8$ and 16 for different prediction methods. And the results match our intuition that b_k should be an increasing sequence of k.

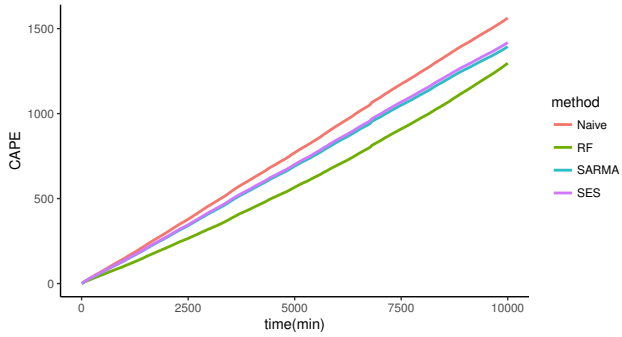


Fig. 4. CAPE(s) for different methods.

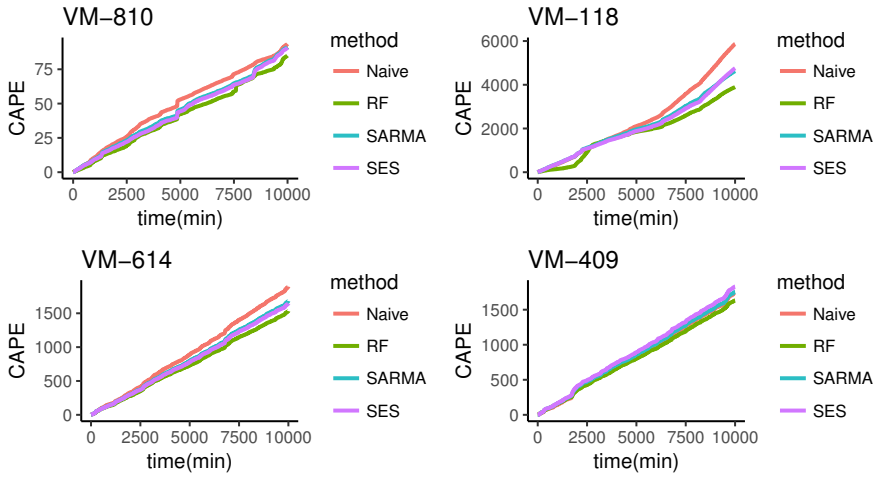


Fig. 5. Time v.s CAPE(s) for 4 VMs

6 PERFORMANCE EVALUATION

In order to measure the performance of our simple meta-algorithm in a cloud computing environment, we run simulations based on the Azure Public Dataset [23] described in detail in Section 5. We show that the meta-algorithm outperforms the popular Weighted Majority Algorithm [39] in almost all simulations and is close the that of the best picked algorithm in hindsight.

6.1 Setup

First we give a description of the setup followed by a detailed performance analysis. Suppose that at every 5-minute timeslot t, a cloud provider needs to decide how much CPU x_t to allocate to a particular VM. The amount of CPU is expressed as the fraction of the 5-minute timeslot a CPU is devoted to the VM. The VM can receive a maximum CPU of R virtual cores at each timeslot which is set at the initialization of the VM. The cost incurred to the provider for supplying CPU is expressed as an increasing linear relationship with c as the per unit cost due to operational expenses such as electricity and cooling. On the other side, the VM demands y_t CPU which is unknown before the allocation decision is made. If the demand y_t is larger than the allocation x_t, then the provider

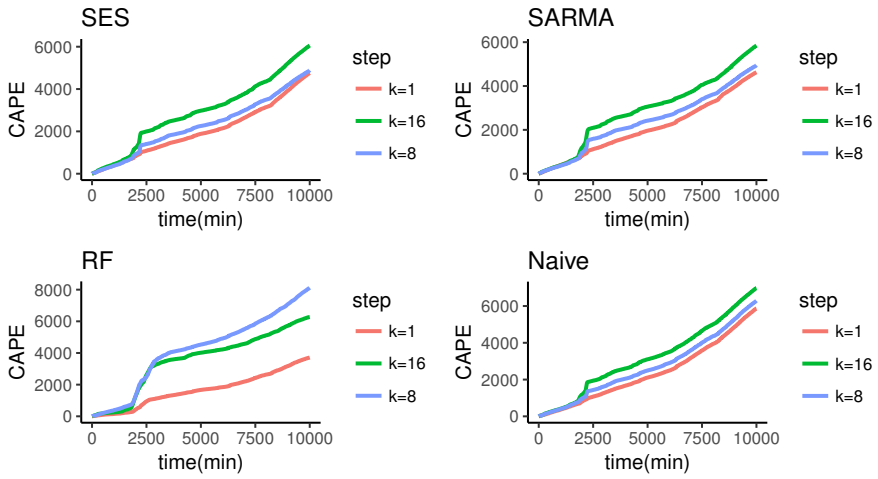


Fig. 6. Time v.s. CAPE(s) for VM-118 under different prediction models

incurs a cost in the form of lost revenue or service agreement violation penalties. This cost increases linearly with the size of the demand deficit by unit cost p. However, if the allocation is equal to or greater than the demand, no additional cost is incurred. Also, the provider incurs the switching cost α when changing CPU allocation decisions between timeslots. The offline optimization for the provider can be stated as the following:

$$\min_{x_1,\ldots,x_T} \quad \sum_{t=1}^{T} \left(c x_t + p\,(y_t - x_t)^+ + \alpha |x_t - x_{t-1}| \right) \tag{19}$$
$$\text{s.t.} \quad 0 \le x_t \le R \quad \forall t \in \{1,\ldots,T\}$$

where $(x)^+ := \max\{0, x\}$. Each VM is simulated to run for 1 week and changes its demand quantities y_t every 5 minutes based on the CPU usage traces in the Azure Public Dataset [23]. The particular VM trace IDs used in these simulations are listed in [22]. While CPU utilization traces are not necessarily the actual CPU demand but are instead produced by the demand under a particular resource allocation, they are strongly correlated with demand itself and so can still be used to demonstrate the performance of cloud resource provisioning algorithms. Specifically, the Utilization Law states that the utilization during a particular time interval is equal to the product of the average throughput and average demand [25], which is how the utilization traces are used here. The starting point x_0 is set to be the Naive prediction for y_1. The maximum CPU amount R depends on the VM and is in the set $\{1, 2, 4, 8, 16\}$. The operational cost coefficient c is set to \$0.0005/(5-min-core) which is sized according to the electricity consumption of a 1400 W server hosting 24 virtual cores at \$0.07/kWh. The insufficient allocation cost coefficient p is set to \$0.0035/(5-min-core) which is sized from the current pricing of an Azure Av2 Standard 8-core VM for \$0.333/hour. We vary the switching cost parameter α anywhere in the range of \$[0,0.144]/(5-min-core) which has a maximum cost equivalent to running the CPU for a day.

For the algorithms which utilize predictions, we train the SARIMA and Random Forest regression prediction models with three weeks of data to be used in the following week's predictions. At each 5-min timeslot, each prediction model gives CPU demand predictions for the next 288 5-minute (1 day's worth) timeslots. We also employ the simple Naive prediction model as a baseline.

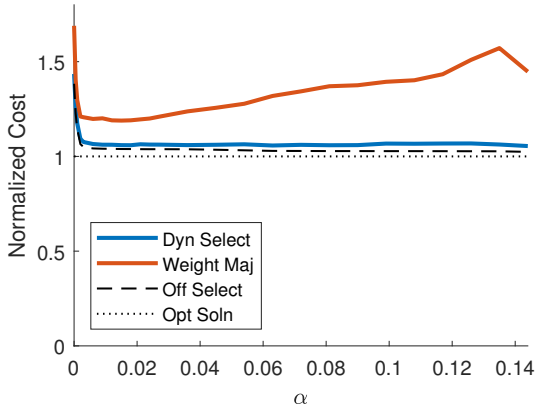


Fig. 7. Average cost incurred by the meta-algorithm (Dyn Select) among the 55 simulated VMs vs. switching cost coefficient α. The cost of each VM is individually normalized by that of its offline optimal solution before being averaged among the others. The meta-algorithm is compared against the Weighted Majority Algorithm and the Offline selection benchmark.

The online algorithms available for the Meta-algorithm to choose from include the following: individual versions of RHC utilizing the three different prediction models, individual versions of OBD utilizing the three different prediction models each with 21 different stepsize settings, individual versions of OGD with 21 different stepsize settings, and the two following allocation settings of $x_t = R$ and $x_t = x_0$ which simply hold constant the allocation at that level for the entire time horizon. The stepsizes used in OBD and OGD are $\eta = 2^k : \forall k \in \{-10, \ldots, 10\}$. The commitment duration for the meta-algorithm to stick to a particular online algorithm is set to be 144 5-minute (1/2 day's worth) timeslots.

We use two baselines to evaluate our meta-algorithm. The first is an impractical one called "Offline Selection" which picks the best algorithm in hindsight. And the second is the widely known Weighted Majority Algorithm (WMA) [39] which makes a decision based on the weighted average among different algorithms. The weights are updated at each timeslot based on their performances. In our implementation of WMA, we discount the weights of all algorithms by β except the one which incurred the least cost in the previous timeslot. Afterwards, the weights are normalized before they are used to make the allocation decision. The discount rate β was set to 2^{-4} which was found to perform best among the VMs from the set 2^{-i} for $i \in \{1, \ldots, 10\}$.

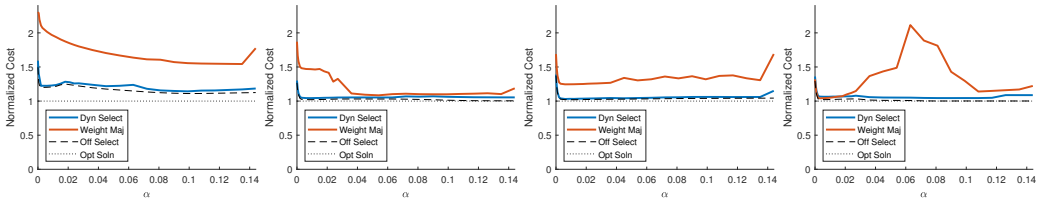
From the concrete structure of the objective function in (19), we can evaluate the error budget (7) as the following measurable quantity:

$$\begin{aligned} B_{k,T} &= \sum_{t=1}^{T} \max_{x \in [0,R]} \left| p\,(y_t - x)^+ - p\left(\hat{y}_{t|t-k+1} - x\right)^+ \right| \\ &= p \sum_{t=1}^{T} \left| y_t - \hat{y}_{t|t-k+1} \right| \end{aligned}$$

which is a scalar multiple of MAE_k defined in (15).

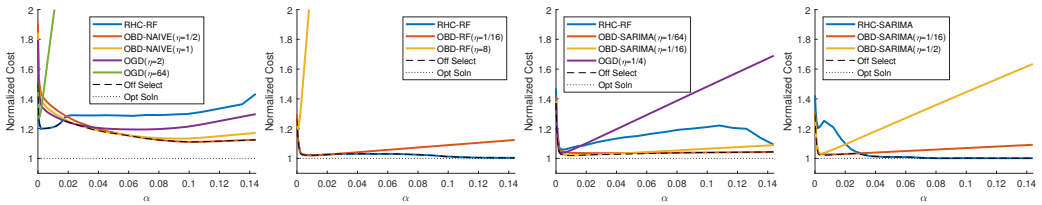
6.2 Results

This section mostly showcases simulations of four individual VMs that have high prediction error and large time variations in CPU demand which are commonly occurring situations in



(a) VM-118 (b) VM-562 (c) VM-646 (d) VM-721

Fig. 8. Cost incurred by the meta-algorithm (Dyn Select) vs. switching cost coefficient α under VMs: (a) 118, (b) 562, (c) 646, (d) 721. The costs are normalized by that of the offline optimal solution and compared against the Weighted Majority Algorithm and the Offline selection benchmark.



(a) VM-118 (b) VM-562 (c) VM-646 (d) VM-721

Fig. 9. Cost incurred by each individual algorithm vs. switching cost coefficient α under VMs: (a) 118, (b) 562, (c) 646, (d) 721. The costs are normalized by that of the offline optimal solution and compared against the Offline selection benchmark.

practice. Additionally, we run simulations derived from 55 randomly chosen VM traces out of the 1,003 studied in Section 5 to give an average performance. We study the performance of our meta-algorithm, the dependency of a particular VM and the size of switching cost on the specific algorithms' performances, and characterize the error budget and optimal solution's path length which connect back to the dynamic regret guarantees from Section 4.

6.2.1 Meta-algorithm performance. The performance of our simple meta-algorithm was compared against the "Offline" selection benchmark that picks the best single algorithm in hindsight and the popular Weighted Majority Algorithm. All of the meta-algorithms were run under constant parameter settings among the VMs. Figure 7 gives the incurred cost normalized by that of the optimal solution versus the switching cost coefficient α for the 55 VMs averaged together. The proposed meta-algorithm on average is consistently much closer to the offline selection benchmark than the Weighted Majority Algorithm. These observations are also found at the individual VM-level. Figure 8 gives the incurred cost normalized by that of the optimal solution versus the switching cost coefficient α for four individual VMs. In almost all situations, our meta-algorithm has a lower cost than that of the Weighted Majority Algorithm and very closely follows the performance of the Offline selection benchmark which is already relatively close to that of the optimal solution. There is a case where the Weighted Majority Algorithm does better than the meta-algorithm (See Figure 8(d)) under low switching costs. However, at higher switching costs the Weighted Majority Algorithm has a cost greater than twice the optimal cost while the meta-algorithm remains closer to that of the optimal solution. These average and individual results give evidence that the meta-algorithm's performance is robust to the different switching costs. Its robustness comes from its design as it

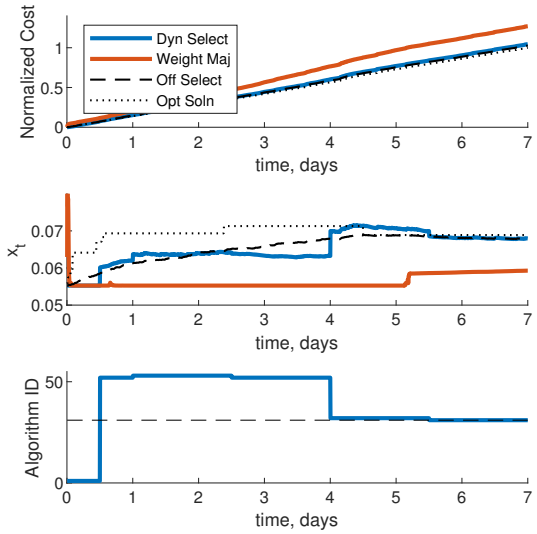


Fig. 10. Accumulated normalized cost vs. time (top), allocation decision trajectory (middle), algorithm decision trajectory (bottom) of the meta-algorithm (Dyn Select) with a switching cost coefficient of $\alpha = 0.036$ for VM-646. It is compared to the Weighted Majority Algorithm and the Offline selection benchmark. Relevant Algorithm IDs: 1 = $x_t = x_0$; {6,...,26} = OGD; {27,...,47} = OBD-SARIMA; {48,...,68} = OBD-RF.

does not just consider which algorithm is currently the best performing but also takes into account the extra cost it would incur by switching to a different one.

A sample allocation and algorithm decision trajectory along with its cost accumulation is given in Figure 10 of VM-646 with a switching cost coefficient of $\alpha = 0.036$. It shows that the meta-algorithm closely tracks the Offline selection benchmark (OBD-SARIMA, $\eta = 2^{-6}$) which tracks the offline optimal solution. It also does not incur a significant amount of cost during time it is further away from the optimal solution. However, the Weighted Majority Algorithm gets stuck with a majority of its weight on an algorithm that remains constant ($x_t = x_0$) or moves very slowly (OGD with extremely small stepsize $\eta = 2^{-10}$). This is due to the fact that it updates its weights based on the incurred cost at the previous timeslot. Algorithms that constantly move to better positions constantly incur switching costs and may never be counted as the "best" algorithm and thus always have their weights discounted.

6.2.2 Algorithm performance dependency on VM and α. Figure 9 gives the incurred cost normalized by that of the optimal solution versus the swithching cost coefficient α for the algorithm settings that perform the best within their type (RHC, OBD, OGD). The algorithm which performs the best among those tested has a strong dependency on both the VM it was run on and the switching cost parameter α. For example, RHC-RF is the best algorithm for VM-118 (Figure 9(a)) during low α but performs poorer during high α. Whereas, RHC is the best algorithm for VM-562 (Figure 9(b)) and VM-721 (Figure 9(d)) during high α. This is true even though VM-118 and VM-562 are using the same prediction model (Random Forest regression). Among all four VMs, it shows that the optimal stepsize η for OBD decreases as α increases. The same observation can also be seen for OGD on VM-118. This is in agreement with Corollary 2 for OGD and Theorem 3 for OBD.

Figure 11 shows the allocation decision trajectories of different algorithms for VM-646 with switching cost coefficient $\alpha = 0.036$. In this case, OBD and OGD better matches with the smoothness

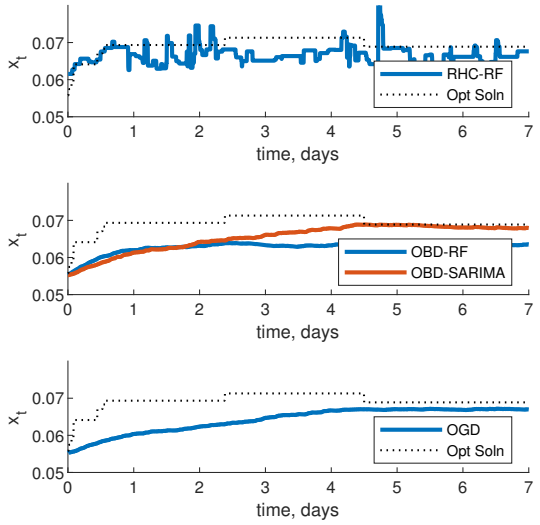


Fig. 11. Allocation decision trajectories of RHC (top), OBD (middle), and OGD (bottom) for VM-646 with $\alpha = 0.036$ compared to that of the optimal solution.

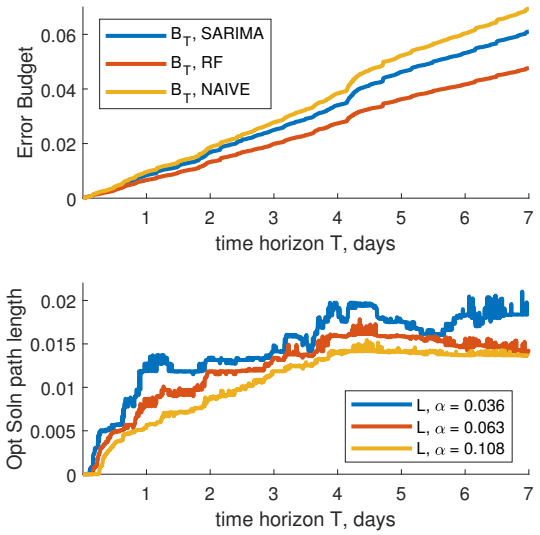


Fig. 12. Error budget (top) and optimal solution's path length (bottom) with respect to the time horizon T for VM-646.

of optimal solution's trajectory as compared to the jittery RHC trajectory. It can also be seen that OBD is more stable to different predictions compared to RHC as different predictions (SARIMA and RF) only has slight impact on the general direction that it moves.

6.2.3 Error Budget and Optimal Path Length. Figure 12 gives the error budget B_T and the optimal solution's path length as a function of the time horizon T for VM-646 with switching cost coefficient $\alpha = 0.036$. Specifically, this means that for each time horizon $T \in \{1, \ldots, 2016\}$ (one week's worth)

an optimal solution was found as if the problem was only defined from 1 to T. The difference in the error budget B_T explains why OBD-RF performed better than OBD-SARIMA in Figure 11.

For the optimal path length L, notice that it is not increasing monotonically with the time horizon T. In fact, there are intervals of time with consistent decline. Also, it seems to get increasingly sublinear with increasing α but it can have periods that trend linearly. For example, see $\alpha = 0.108$ from days 1 to 4 which increases linearly and is then followed by a slight linear decline. This complex nature of the optimal path length seems to suggest why the algorithm type (RHC, OBD, OGD) that is best can vary between VMs regardless on differences in error.

7 CONCLUSION

In this paper, we bridge the gap between prediction errors and online optimization algorithms. We provide a simple and intuitive prediction error model, which is used to derive theoretical upper bounds on the dynamic regret of popular online algorithms. Compared to existing literature, the novelty lies in the incorporation of switching costs, which couples decisions over time. In order to choose which algorithm to run without prediction error and optimal path length knowledge, we design a simple online meta-algorithm to carefully select which algorithm to follow based on past performance and the switching costs. Our real-world trace driven simulations highlight the proposed meta-algorithm outperforms a popular algorithm selection policy significantly and perform very closely to that of the best algorithm chosen in hindsight. Most prediction algorithms including those used in this paper are application agnostic, and developing computationally efficient, application specific prediction algorithms is our ongoing work. For future work, we plan to theoretically characterize the proposed meta-algorithm to explain its performance advantages over existing meta-algorithms as observed in Section 6. Also, exploring how predicting the optimal solution's path length could be incorporated into an algorithm selection policy is a very interesting direction. On the application side, we plan to evaluate the performance of the proposed meta-algorithm in an environment that provisions multiple resource types to each VM. Additionally, detailed experiments that run an application on VMs could be used to measure true resource demands that include feedback effects that result from the provisioning decisions.

8 ACKNOWLEDGMENTS

We would like to thank our reviewers and shepherd for their detailed comments and suggestions in making this paper more complete. This research was partially funded by NSF grants CNS-1617698, CNS-1717588, CNS-1730128, DGE-1633299, and DMS-1612501.

REFERENCES

[1] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. 2018. Elasticity in cloud computing: state of the art and research challenges. *IEEE Transactions on Services Computing* 11, 2 (2018), 430–447.

[2] Ahmad Al-Shishtawy and Vladimir Vlassov. 2013. Elastman: elasticity manager for elastic key-value stores in the cloud. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. ACM, 7.

[3] Lachlan Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. 2013. A tale of two metrics: Simultaneous bounds on competitiveness and regret. In *Conference on Learning Theory*. 741–763.

[4] Lachlan LH Andrew, Minghong Lin, and Adam Wierman. 2010. Optimality, fairness, and robustness in speed scaling designs. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 38. ACM, 37–48.

[5] Danilo Ardagna, Michele Ciavotta, and Riccardo Lancellotti. 2014. A receding horizon approach for the runtime management of iaas cloud systems. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2014 16th International Symposium on*. IEEE, 445–452.

[6] Adnan Ashraf, Benjamin Byholm, and Ivan Porres. 2012. CRAMP: Cost-efficient resource allocation for multiple web applications with proactive scaling. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 581–586.

[7] Lee Badger, Tim Grance, Robert Patt-Corner, Jeff Voas, et al. 2012. Cloud computing synopsis and recommendations. *NIST special publication* 800 (2012), 146.

[8] Jeff Barr. 2018. New AWS Auto Scaling – Unified Scaling For Your Cloud Applications. https://aws.amazon.com/blogs/aws/aws-auto-scaling-unified-scaling-for-your-cloud-applications/.

[9] Omar Besbes, Yonatan Gur, and Assaf Zeevi. 2015. Non-stationary stochastic optimization. *Operations research* 63, 5 (2015), 1227–1244.

[10] Peter Bodik, Michael Paul Armbrust, Kevin Canini, Armando Fox, Michael Jordan, and David A Patterson. 2008. A case for adaptive datacenters to conserve energy and improve reliability. *University of California at Berkeley, Tech. Rep. UCB/EECS-2008-127* (2008).

[11] Allan Borodin and Ran El-Yaniv. 2005. *Online computation and competitive analysis*. cambridge university press.

[12] A Borodin, N Linial, and M Saks. 1987. An optimal online algorithm for metrical task systems. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 373–382.

[13] Allan Borodin, Nathan Linial, and Michael E Saks. 1992. An optimal on-line algorithm for metrical task system. *Journal of the ACM (JACM)* 39, 4 (1992), 745–763.

[14] Mark Brinda and Michael Heric. 2017. The Changing Faces of the Cloud. *Bain Company* (2017).

[15] Eduardo F Camacho and Carlos Bordons Alba. 2013. *Model predictive control*. Springer Science & Business Media.

[16] Eddy Caron, Frédéric Desprez, and Adrian Muresan. 2010. Forecasting for Cloud computing on-demand resources based on pattern matching. (2010).

[17] Niangjun Chen, Anish Agarwal, Adam Wierman, Siddharth Barman, and Lachlan LH Andrew. 2015. Online convex optimization using predictions. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 43. ACM, 191–204.

[18] Niangjun Chen, Joshua Comden, Zhenhua Liu, Anshul Gandhi, and Adam Wierman. 2016. Using predictions in online optimization: Looking forward with an eye on the past. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*. ACM, 193–206.

[19] Niangjun Chen, Gautam Goel, and Adam Wierman. 2018. Smoothed Online Convex Optimization in High Dimensions via Online Balanced Descent. In *Proceedings of the 31st Conference On Learning Theory (Proceedings of Machine Learning Research)*, Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet (Eds.), Vol. 75. PMLR, 1574–1594. http://proceedings.mlr.press/v75/chen18b.html

[20] Niangjun Chen, Xiaoqi Ren, Shaolei Ren, and Adam Wierman. 2015. Greening multi-tenant data center demand response. *Performance Evaluation* 91 (2015), 229–254.

[21] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 273–286.

[22] Joshua Comden, Sijie Yao, Niangjun Chen, Haipeng Xing, and Zhenhua Liu. 2019. Online Optimization in Cloud Resource Provisioning: Predictions, Regrets, and Algorithms: Virtual Machine ID Dataset. https://doi.org/10.5281/zenodo.2555195

[23] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 153–167.

[24] Ariel da Silva Dias, Luis HV Nakamura, Julio C Estrella, Regina HC Santana, and Marcos J Santana. 2014. Providing IaaS resources automatically through prediction and monitoring approaches. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*. IEEE, 1–7.

[25] Peter J Denning and Jeffrey P Buzen. 1978. The operational analysis of queueing network models. *ACM Computing Surveys (CSUR)* 10, 3 (1978), 225–261.

[26] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH computer architecture news*, Vol. 35. ACM, 13–23.

[27] Carlos E Garcia, David M Prett, and Manfred Morari. 1989. Model predictive control: theory and practice—a survey. *Automatica* 25, 3 (1989), 335–348.

[28] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. *CNSM* 10 (2010), 9–16.

[29] Eric C Hall and Rebecca M Willett. 2015. Online convex optimization in dynamic environments. *IEEE Journal of Selected Topics in Signal Processing* 9, 4 (2015), 647–662.

[30] Jinhui Huang, Chunlin Li, and Jie Yu. 2012. Resource prediction based on double exponential smoothing in cloud computing. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*. IEEE, 2056–2060.

[31] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. 2012. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems* 28, 1 (2012), 155–162.

[32] Ali Jadbabaie, Alexander Rakhlin, Shahin Shahrampour, and Karthik Sridharan. 2015. Online optimization: Competing with dynamic comparators. In *Artificial Intelligence and Statistics*. 398–406.
[33] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2009. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*. ACM, 117–126.
[34] Chuanqi Kan. 2016. DoCloud: An elastic cloud platform for Web applications based on Docker. In *Advanced Communication Technology (ICACT), 2016 18th International Conference on*. IEEE, 478–483.
[35] Sunirmal Khatua, Anirban Ghosh, and Nandini Mukherjee. 2010. Optimizing the utilization of virtual resources in Cloud environment. In *Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS), 2010 IEEE International Conference on*. IEEE, 82–87.
[36] Minghong Lin, Zhenhua Liu, Adam Wierman, and Lachlan LH Andrew. 2012. Online algorithms for geographical load balancing. In *Green Computing Conference (IGCC), 2012 International*. IEEE, 1–10.
[37] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. [n. d.]. Dynamic right-sizing for power-proportional data centers. In *2011 Proceedings IEEE INFOCOM*.
[38] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. 2013. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)* 21, 5 (2013), 1378–1391.
[39] Nick Littlestone and Manfred K Warmuth. 1994. The weighted majority algorithm. *Information and computation* 108, 2 (1994), 212–261.
[40] Paul Marshall, Kate Keahey, and Tim Freeman. 2010. Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 43–52.
[41] Haibo Mi, Huaimin Wang, Gang Yin, Yangfan Zhou, Dianxi Shi, and Lin Yuan. 2010. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In *Services Computing (SCC), 2010 IEEE International Conference on*. IEEE, 514–521.
[42] Hanna Michalska and David Q Mayne. 1993. Robust receding horizon control of constrained nonlinear systems. *IEEE transactions on automatic control* 38, 11 (1993), 1623–1633.
[43] Laura R Moore, Kathryn Bean, and Tariq Ellahi. 2013. A coordinated reactive and predictive approach to cloud elasticity. (2013).
[44] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 500–507.
[45] Virag Shah and Gustavo de Veciana. 2014. Performance evaluation and asymptotics for content delivery networks. In *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2607–2615.
[46] Shahin Shahrampour and Ali Jadbabaie. 2018. Distributed online optimization in dynamic environments using mirror descent. *IEEE Trans. Automat. Control* 63, 3 (2018), 714–725.
[47] Yue Tan and Cathy H Xia. 2015. An adaptive learning approach for efficient resource provisioning in cloud services. *ACM Sigmetrics Performance Evaluation Review* 42, 4 (2015), 3–11.
[48] Eno Thereska, Austin Donnelly, and Dushyanth Narayanan. 2009. Sierra: a power-proportional, distributed storage system. *Microsoft Research Ltd., Tech. Rep. MSR-TR-2009* 153 (2009).
[49] Lixi Wang, Jing Xu, Ming Zhao, and José Fortes. 2011. Adaptive virtual resource management with fuzzy model predictive control. In *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 191–192.
[50] Lixi Wang, Jing Xu, Ming Zhao, Yicheng Tu, and Jose AB Fortes. 2011. Fuzzy modeling based resource management for virtualized database systems. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*. IEEE, 32–42.
[51] Adam Wierman, Lachlan LH Andrew, and Ao Tang. 2009. Power-aware speed scaling in processor sharing systems. *IEEE INFOCOM, 2009* (2009), 2007–2015.
[52] Shaoquan Zhang, Longbo Huang, Minghua Chen, and Xin Liu. 2017. Proactive Serving Decreases User Delay Exponentially: The Light-Tailed Service Time Case. *IEEE/ACM Trans. Netw.* 25, 2 (April 2017), 708–723.
[53] Xiaoxi Zhang, Chuan Wu, Zongpeng Li, and Francis CM Lau. 2017. Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.
[54] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 928–936.

A PROOFS

A.1 Theorem 1

Proof. Let $z_{t+1} := x_t - \eta g_t$; thus, $x_{t+1} = P_{\mathcal{X}}(z_{t+1})$. After subtracting off the optimal decision x_t^*, it becomes $z_{t+1} - x_t^* = x_t - \eta g_t - x_t^*$. Squaring both sides we have:

$$\begin{aligned}\|z_{t+1} - x_t^*\|_2^2 &= \|x_t - \eta g_t - x_t^*\|_2^2 \\ &= \|x_t - x_t^*\|_2^2 - 2\eta\langle g_t, x_t - x_t^*\rangle + \eta^2\|g_t\|_2^2\end{aligned}$$

Since $\mathcal{X}$ is convex and $P_{\mathcal{X}}$ is the projection operator, then $\forall z \in \mathbb{R}^m$ and $\forall x \in \mathcal{X}$, then $\|P_{\mathcal{X}}(z) - x\|_2^2 \leq \|z - x\|_2^2$. Using that fact in the above equation, we have

$$\|x_{t+1} - x_t^*\|_2^2 \leq \|x_t - x_t^*\|_2^2 - 2\eta\langle g_t, x_t - x_t^*\rangle + \eta^2\|g_t\|_2^2$$

and after rearranging the terms, it leads to

$$\langle g_t, x_t - x_t^*\rangle \leq \frac{1}{2\eta}\left(\|x_t - x_t^*\|_2^2 - \|x_{t+1} - x_t^*\|_2^2\right) + \frac{\eta}{2}\|g_t\|_2^2$$

Since $f(x_t, y_t)$ is convex in x, then $f(x, y_t) - f(x_t, y_t) \geq \langle g_t, x - x_t\rangle : \forall x \in \mathcal{X}$ and we have that:

$$\begin{aligned}f(x_t, y_t) - f(x_t^*, y_t) &\leq \langle g_t, x_t - x_t^*\rangle \\ &\leq \frac{1}{2\eta}\left(\|x_t - x_t^*\|_2^2 - \|x_{t+1} - x_t^*\|_2^2\right) + \frac{\eta}{2}\|g_t\|_2^2. \end{aligned} \tag{20}$$

Let $R_T(L)$ be defined in the following way:

$$\begin{aligned}R_T(L) &:= \sum_{t=1}^{T}\left(f(x_t, y_t) + \alpha\|x_t - x_{t-1}\|\right) - \sum_{t=1}^{T}\left(f(x_t^*, y_t) + \alpha\|x_t^* - x_{t-1}^*\|\right) \\ &= \sum_{t=1}^{T}\left(f(x_t, y_t) - f(x_t^*, y_t)\right) + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right) \\ &\leq \sum_{t=1}^{T}\left(\frac{1}{2\eta}\left(\|x_t - x_t^*\|_2^2 - \|x_{t+1} - x_t^*\|_2^2\right) + \frac{\eta}{2}\|g_t\|_2^2\right) + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right) \\ &= \frac{1}{2\eta}\sum_{t=1}^{T}\left(\|x_t\|_2^2 - \|x_{t+1}\|_2^2 + 2\langle x_{t+1} - x_t, x_t^*\rangle + \frac{\eta}{2}\|g_t\|_2^2\right) + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right) \\ &= \frac{1}{2\eta}\left(\|x_1\|_2^2 - \|x_{T+1}\|_2^2\right) + \frac{1}{\eta}\sum_{t=1}^{T}\langle x_{t+1} - x_t, x_t^*\rangle + \frac{\eta}{2}\sum_{t=1}^{T}\|g_t\|_2^2 \\ &\quad + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right) \\ &= \frac{1}{2\eta}\left(\|x_1\|_2^2 - \|x_{T+1}\|_2^2 - 2\langle x_1, x_1^*\rangle + 2\langle x_{T+1}, x_T^*\rangle\right) + \frac{1}{\eta}\sum_{t=2}^{T}\langle x_t, x_{t-1}^* - x_t^*\rangle + \frac{\eta}{2}\sum_{t=1}^{T}\|g_t\|_2^2 \\ &\quad + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right). \end{aligned} \tag{21}$$

The first inequality comes from (20). The third equality comes from expanding out the squared terms. The forth equality comes from canceling out the telescoping terms. The fifth equality rearranges the inner product to be a subtraction of x_t^* instead of x_t.

Let D_2 be the quantity such that $\|x_1 - x_2\|_2 \leq D_2 : \forall\{x_1, x_2\} \in \mathcal{X}$. Without loss of generality, assume that $0 \in \mathcal{X}$ which means that $\forall\{x_1, x_2\} \in \mathcal{X}$, both $\|x_1\|_2 \leq D_2$ and $\|x_2\|_2 \leq D_2$. From the Cauchy–Schwarz inequality we have that $\langle x_1, x_2\rangle \leq \|x_1\|_2\|x_2\|_2 \leq D_2^2$. Also, since $\|x_1 - x_2\|_2 \leq D_2$,

$$\begin{aligned}
\|x_1 - x_2\|_2^2 &\leq D_2^2 \\
\|x_1\|_2^2 + \|x_2\|_2^2 - 2\langle x_1, x_2\rangle &\leq D_2^2 \\
-2\langle x_1, x_2\rangle &\leq D_2^2 \\
\langle x_1, x_2\rangle &\geq -\frac{D_2^2}{2}
\end{aligned}$$

Therefore, we have that $-\frac{D_2^2}{2} \leq \langle x_1, x_2\rangle \leq D_2^2$. Apply this to the first terms of (21), we have:

$$\begin{aligned}
R_T(L) &\leq \frac{2D_2^2}{\eta} + \frac{1}{\eta}\sum_{t=2}^{T}\langle x_t, x_{t-1}^* - x_t^*\rangle + \frac{\eta}{2}\sum_{t=1}^{T}\|g_t\|_2^2 + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right) \\
&\leq \frac{2D_2^2}{\eta} + \frac{D_2}{\eta}\sum_{t=2}^{T}\|x_{t-1}^* - x_t^*\|_2 + \frac{\eta}{2}\sum_{t=1}^{T}\|g_t\|_2^2 + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right) \\
&\leq \frac{2D^2\overline{\kappa}^2}{\eta} + \frac{D\overline{\kappa}}{\eta}\sum_{t=2}^{T}\|x_{t-1}^* - x_t^*\| + \frac{\eta}{2}\sum_{t=1}^{T}\|g_t\|_2^2 + \alpha\sum_{t=1}^{T}\left(\|x_t - x_{t-1}\| - \|x_t^* - x_{t-1}^*\|\right) \\
&\leq \frac{2D^2\overline{\kappa}^2}{\eta} + \left(\frac{D\overline{\kappa}}{\eta} - \alpha\right)\sum_{t=1}^{T}\|x_t^* - x_{t-1}^*\| + \frac{\eta}{2}\sum_{t=1}^{T}\|g_t\|_2^2 + \alpha\sum_{t=1}^{T}\|x_t - x_{t-1}\| \qquad (22)
\end{aligned}$$

The second inequality applies the Cauchy–Schwarz inequality. The third inequality comes from the fact that $\|x\|_2 \leq \overline{\kappa}\|x\| : \forall x$ and that the diameter of the action space D is defined by the norm $\|\cdot\|$. The forth inequality adds $\frac{D\overline{\kappa}}{\eta}\|x_1^* - x_0\|$ and combines the terms with traveling distance of the optimal solution.

As mentioned before, since we have that $\|P_{\mathcal{X}}(z) - x\|_2^2 \leq \|z - x\|_2^2$, and $\underline{\kappa}\|x\| \leq \|x\|_2$,

$$\begin{aligned}
\|x_t - x_{t-1}\| &\leq \frac{1}{\underline{\kappa}}\|x_t - x_{t-1}\|_2 \\
&= \frac{1}{\underline{\kappa}}\|P_{\mathcal{X}}(x_{t-1} - \eta g_{t-1}) - x_{t-1}\|_2 \\
&\leq \frac{\eta}{\underline{\kappa}}\|g_{t-1}\|_2 \qquad (23)
\end{aligned}$$

Applying this to (22) we have:

$$\begin{aligned}
R_T(L) &\leq \frac{2D^2\overline{\kappa}^2}{\eta} + \left(\frac{D\overline{\kappa}}{\eta} - \alpha\right)\sum_{t=1}^{T}\|x_t^* - x_{t-1}^*\| + \frac{\eta}{2}\sum_{t=1}^{T}\|g_t\|_2^2 + \frac{\eta\alpha}{\underline{\kappa}}\sum_{t=1}^{T}\|g_{t-1}\|_2 \\
&\leq \frac{2D^2\overline{\kappa}^2}{\eta} + \left(\frac{D\overline{\kappa}}{\eta} - \alpha\right)\sum_{t=1}^{T}\|x_t^* - x_{t-1}^*\| + \frac{\eta}{2}\sum_{t=1}^{T}\left(\|g_t\|_2^2 + \frac{2\alpha}{\underline{\kappa}}\|g_t\|_2\right)
\end{aligned}$$

Without loss of generality let $g_0 := 0$. The second inequality adds $\frac{\eta\alpha}{\underline{\kappa}}\|g_T\|_2$ and combines summations.

Now taking the definition of dynamic regret (4) we have

$$\begin{aligned}\mathcal{R}_T^{\mathrm{OGD}}(L) &= \sup_{y_{1:T}} \{R_T(L)\} \\ &\leq \frac{2D^2\overline{\kappa}^2}{\eta} + L\left(\frac{D\overline{\kappa}}{\eta} - \alpha\right)^+ + \eta G\left(\frac{G}{2} + \frac{\alpha}{\underline{\kappa}}\right)T \end{aligned} \tag{24}$$

where $(x)^+ := \max\{0, x\}$ is used to prevent the coefficient in front of the first summation from being negative and since $\sum_{t=1}^{T} \|x_t^* - x_{t-1}^*\| \leq L$ and $\|g_t\|_2 \leq G$ which is from the fact that the Lipschitz condition bounds the dual norm of the subgradient. □

A.2 Theorem 3

Proof. Define $R_T(B_T, L)$ as the dynamic regret without yet taking the supremum:

$$\begin{aligned} R_T(B_T, L) &:= \sum_{t=1}^{T} \left(f(x_t, y_t) + \alpha\|x_t - x_{t-1}\|\right) - \sum_{t=1}^{T} \left(f(x_t^*, y_t) + \alpha\|x_t^* - x_{t-1}^*\|\right) \\ &= \sum_{t=1}^{T} \left(f(x_t, \hat{y}_{t|t}) + \alpha\|x_t - x_{t-1}\|\right) - \sum_{t=1}^{T} \left(f(x_t^*, \hat{y}_{t|t}) + \alpha\|x_t^* - x_{t-1}^*\|\right) \\ &\quad + \sum_{t=1}^{T} \left(f(x_t, y_t) - f(x_t, \hat{y}_{t|t})\right) + \sum_{t=1}^{T} \left(f(x_t^*, \hat{y}_{t|t}) - f(x_t^*, y_t)\right) \\ &\leq \frac{GL}{\eta} + \frac{T\alpha^2\eta}{2m} + 2B_T \\ &= \alpha\sqrt{\frac{2GLT}{m}} + 2B_T \qquad \left(\text{Set } \eta = \sqrt{\frac{2GLm}{T\alpha^2}}\right) \end{aligned}$$

The second equality comes from adding $0 = f(x_t, \hat{y}_{t|t}) - f(x_t, \hat{y}_{t|t})$ and $0 = f(x_t^*, \hat{y}_{t|t}) - f(x_t^*, \hat{y}_{t|t})$ and swapping the true for the predicted cost functions. The inequality comes first from applying Theorem 10 from [19] on the first two summations where the prediction is considered to be error-free. On the last two summations separately, each has the absolute value taken and then the supremum with respect to the each individual timeslot's x_t or x_t^* followed by applying the error budget definition (6). The last equality comes by setting η to be the specific value in the bracket to the right. Taking the supremum on $\mathbf{y} \in \mathcal{Y}_{1:1,T}$ as defined by (8) gets the resultant. □

A.3 Theorem 4

Proof. Let $x_{t_1:t_2}$ denote the sequence of vectors $x_{t_1}, \ldots, x_{t_2}$. Let the cost of an algorithm during the sequence of timeslots $\{t_1, \ldots, t_2\}$ with boundary conditions x_S, x_E and cost function parameters $y_{t_1:t_2}$ be

$$H_{t_1,t_2}(x; y, x_S, x_E) = \alpha\left\|x_{t_1} - x_S\right\| + \sum_{\tau=t_1}^{t_2} f(x_\tau, y_\tau) + \sum_{\tau=t_1+1}^{t_2} \alpha\left\|x_\tau - x_{\tau-1}\right\| + \alpha\left\|x_E - x_{t_2}\right\|.$$

If x_E is omitted, then $x_E := x_{t_2}$ (and thus $\left\|x_E - x_{t_2}\right\| = 0$). If x_S is omitted, then $x_S = x_{t_1-1}$. Therefore, $H_{t_1,t_2}(x; y)$ depends only on $x_{t_1-1}, \ldots, x_{t_2}$.

We first analyze the dynamic regret for a single trajectory $x_{1:T}^{(k)}$ in Algorithm 4 before they are averaged together. Let M_k be the number of times that $(1 + (t-1) \mod v) = k$ is true for this particular k and let $\left(t^{(k),1}, \ldots, t^{(k),M_k}\right)$ be the timeslots for which it is true.

From the optimality of $x^{(k)}_{t^{(k),i}:t^{(k),i}+v-1}$ for its given predictions, we have that it minimizes $H_{t^{(k),i},t^{(k),i}+v-1}\left(x^{(k)};\hat{y}_{\cdot|t^{(k),i}},x^{(k)}_{t^{(k),i}-1},x^{(k)}_{E,v}\right)$ where $x^{(k)}_{E,v}:=x^{(k)}_{t^{(k),i}+v}$ if $v<w$ and $x^{(k)}_{E,v}:=x^{(k)}_{t^{(k),i}+w-1}$ if $v=w$. Therefore,

$$\begin{aligned}
&\alpha\left\|x^{(k)}_{t^{(k),i}}-x^{(k)}_{t^{(k),i}-1}\right\|+\sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1}f\left(x^{(k)}_{\tau},\hat{y}_{\tau|t^{(k),i}}\right)+\sum_{\tau=t^{(k),i}+1}^{t^{(k),i}+v-1}\alpha\left\|x^{(k)}_{\tau}-x^{(k)}_{\tau-1}\right\|+\alpha\left\|x^{(k)}_{E,v}-x^{(k)}_{t^{(k),i}+v-1}\right\|\\
&\leq\alpha\left\|x^{*}_{t^{(k),i}}-x^{(k)}_{t^{(k),i}-1}\right\|+\sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1}f\left(x^{*}_{\tau},y_{\tau|t^{(k),i}}\right)+\sum_{\tau=t^{(k),i}+1}^{t^{(k),i}+v-1}\alpha\left\|x^{*}_{\tau}-x^{*}_{\tau-1}\right\|+\alpha\left\|x^{(k)}_{E,v}-x^{*}_{t^{(k),i}+v-1}\right\|
\end{aligned}\tag{25}$$

We construct a sequence of T-tuples $(\xi^{(k),1},...,\xi^{(k),M_k})$ consecutively so that $\xi^{(k),i}_{\tau}=x^{(k)}_{\tau}$ for $\tau\in\{1,...,t^{(k),i}+v-1\}$ and $\xi^{(k),i}_{\tau}=x^{*}_{\tau}$ for $\tau\in\{t^{(k),i}+v,...,T\}$. Thus, $\xi^{(k),M_k}=x^{(k)}_{1:T}$. We also define another T-tuple $\xi^{(k),0}:=x^{*}_{1:T}$. The sequence is constructed by the following process. At each timeslot $t^{(k),i}$, we replace the associated actions in $\xi^{(k),i-1}$ with $x^{(k)}_{t^{(k),i}:t^{(k),i}+v-1}$ to get:

$$\xi^{(k),i}:=\left(\xi^{(k),i-1}_{1},...,\xi^{(k),i-1}_{t^{(k),i}-1},x^{(k)}_{t^{(k),i}},...,x^{(k)}_{t^{(k),i}-1+v},\xi^{(k),i-1}_{t^{(k),i}+v},...,\xi^{(k),i-1}_{T}\right).$$

By examining the terms in $\xi^{(k),i-1}$ and $\xi^{(k),i}$, we have

$$\begin{aligned}
&H_{1,T}(\xi^{(k),i};y)-H_{1,T}(\xi^{(k),i-1};y)\\
&=\sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1}\left(f\left(x^{(k)}_{\tau},y_{\tau}\right)-f\left(x^{*}_{\tau}\,y_{\tau}\right)\right)+\alpha\sum_{\tau=t^{(k),i}+1}^{t^{(k),i}+v-1}\left(\left\|x^{(k)}_{\tau}-x^{(k)}_{\tau-1}\right\|-\left\|x^{*}_{\tau}-x^{*}_{\tau-1}\right\|\right)\\
&\quad+\alpha\left\|x^{*}_{t^{(k),i}+v}-x^{(k)}_{t^{(k),i}+v-1}\right\|-\alpha\left\|x^{*}_{t^{(k),i}+v}-x^{*}_{t^{(k),i}+v-1}\right\|+\alpha\left\|x^{(k)}_{t^{(k),i}}-x^{(k)}_{t^{(k),i}-1}\right\|\\
&\quad-\alpha\left\|x^{*}_{t^{(k),i}}-x^{(k)}_{t^{(k),i}-1}\right\|\\
&\leq\sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1}\left(f\left(x^{(k)}_{\tau},y_{\tau}\right)-f\left(x^{(k)}_{\tau},\hat{y}_{\tau|t^{(k),i}}\right)\right)+\sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1}\left(f\left(x^{*}_{\tau},\hat{y}_{\tau|t^{(k),i}}\right)-f\left(x^{*}_{\tau},y_{\tau}\right)\right)\\
&\quad+\alpha\left\|x^{*}_{t^{(k),i}+v}-x^{(k)}_{t^{(k),i}+v-1}\right\|-\alpha\left\|x^{*}_{t^{(k),i}+v}-x^{*}_{t^{(k),i}+v-1}\right\|+\alpha\left\|x^{(k)}_{E,v}-x^{*}_{t^{(k),i}+v-1}\right\|\\
&\quad-\alpha\left\|x^{(k)}_{E,v}-x^{(k)}_{t^{(k),i}+v-1}\right\|\\
&\leq\sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1}\left(f\left(x^{(k)}_{\tau},y_{\tau}\right)-f\left(x^{(k)}_{\tau},\hat{y}_{\tau|t^{(k),i}}\right)\right)+\sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1}\left(f\left(x^{*}_{\tau},\hat{y}_{\tau|t^{(k),i}}\right)-f\left(x^{*}_{\tau},y_{\tau}\right)\right)\\
&\quad+2\alpha\left\|x^{(k)}_{t^{(k),i}+v-1}-x^{*}_{t^{(k),i}+v-1}\right\|
\end{aligned}\tag{26}$$

where the first inequality comes from applying (25), and the second inequality comes from applying the triangle inequality.

Summing up the final inequality from $i = 1$ to $i = M_k$ and noting that $\xi^{(k),0} = x^*$ and $\xi^{(k),M_k} = x^{(k)}$, we have

$$\begin{aligned}\operatorname{cost}\left(x^{(k)}_{1:T}, y_{1:T}\right) \leq \operatorname{cost}\left(\mathrm{OPT}_L, y_{1:T}\right) &+ \sum_{i=1}^{M_k} \sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1} \left(f\left(x^{(k)}_\tau, y_\tau\right) - f\left(x^{(k)}_\tau, \hat{y}_{\tau|t^{(k),i}}\right)\right) \\ &+ \sum_{i=1}^{M_k} \sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1} \left(f\left(x^*_\tau, \hat{y}_{\tau|t^{(k),i}}\right) - f\left(x^*_\tau, y_\tau\right)\right) + 2\alpha \sum_{i=1}^{M_k} \left\| x^{(k)}_{t^{(k),i}+v-1} - x^*_{t^{(k),i}+v-1} \right\| \end{aligned} \tag{27}$$

Since x^{CHC}_t is determined by averaging $x^{(1)}_t, \ldots, x^{(v)}_t$ in Algorithm 4 and (3) is convex, then Jensen's inequality can be applied to get

$$\begin{aligned}\operatorname{cost}\left(\mathrm{CHC}, y_{1:T}\right) &\leq \frac{1}{v} \sum_{k=1}^{v} \operatorname{cost}\left(x^{(k)}_{1:T}, y_{1:T}\right) \\ &= \operatorname{cost}(\mathrm{OPT}_L, y_{1:T}) + \frac{1}{v} \sum_{k=1}^{v} \sum_{i=1}^{M_k} \sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1} \left(f\left(x^{(k)}_\tau, y_\tau\right) - f\left(x^{(k)}_\tau, \hat{y}_{\tau|t^{(k),i}}\right)\right) \\ &\quad + \frac{1}{v} \sum_{k=1}^{v} \sum_{i=1}^{M_k} \sum_{\tau=t^{(k),i}}^{t^{(k),i}+v-1} \left(f\left(x^*_\tau, \hat{y}_{\tau|t^{(k),i}}\right) - f\left(x^*_\tau, y_\tau\right)\right) \\ &\quad + \frac{2\alpha}{v} \sum_{k=1}^{v} \sum_{i=1}^{M_k} \left\| x^{(k)}_{t^{(k),i}+v-1} - x^*_{t^{(k),i}+v-1} \right\| \\ &= \operatorname{cost}\left(\mathrm{OPT}_L, y_{1:T}\right) \\ &\quad + \frac{1}{v} \sum_{t=1}^{T} \sum_{j=1}^{v} \left(f\left(x^{(1+(t-j) \bmod v)}_t, y_{t|1+t-j}\right) - f\left(x^{(1+(t-j) \bmod v)}_t, \hat{y}_{t|1+t-j}\right)\right) \\ &\quad + \frac{1}{v} \sum_{t=1}^{T} \sum_{j=1}^{v} \left(f\left(x^*_t, \hat{y}_{t|1+t-j}\right) - f\left(x^*_t, y_{t|1+t-j}\right)\right) \\ &\quad + \frac{2\alpha}{v} \sum_{t=1}^{T} \left\| x^{(1+(t-v) \bmod v)}_t - x^*_t \right\| \\ &\leq \operatorname{cost}\left(\mathrm{OPT}_L, y_{1:T}\right) \\ &\quad + \frac{1}{v} \sum_{t=1}^{T} \sum_{j=1}^{v} \left| f\left(x^{(1+(t-j) \bmod v)}_t, y_{t|1+t-j}\right) - f\left(x^{(1+(t-j) \bmod v)}_t, \hat{y}_{t|1+t-j}\right) \right| \\ &\quad + \frac{1}{v} \sum_{t=1}^{T} \sum_{j=1}^{v} \left| f\left(x^*_t, \hat{y}_{t|1+t-j}\right) - f\left(x^*_t, y_{t|1+t-j}\right) \right| \\ &\quad + \frac{2}{v} \sum_{t=1}^{T} \left\| x^{(1+(t-v) \bmod v)}_t - x^*_t \right\| \end{aligned}$$

$$\leq \text{cost}\left(\text{OPT}_L, y_{1:T}\right) + \frac{2}{v}\sum_{j=1}^{v} B_{j,T} + \frac{2\alpha}{v}\sum_{t=1}^{T}\left\|x_t^{(1+(t-v) \bmod v)} - x_t^*\right\|$$

$$\leq \text{cost}\left(\text{OPT}_L, y_{1:T}\right) + \frac{2}{v}\sum_{j=1}^{v} B_{j,T} + \frac{2D\alpha T}{v} \tag{28}$$

The second equality is the first equality with the indexes renamed. This comes from that every timeslot $t \in \{1, ..., T\}$ is counted exactly once by $t^{(k),i}$ in the summation $\sum_{k=1}^{v}\sum_{i=1}^{M_k}$, and that $1 + t - j \in \left\{t^{(1+(t-j) \bmod v),1}, \ldots, t^{(1+(t-j) \bmod v),M_{(1+(t-j) \bmod v)}}\right\}$ for all $t \in \{1, ..., T\}$ and $j \in \{1, ..., v\}$. The second inequality comes from applying the error budget (7) j steps ahead. The last inequality comes from the fact that $\left\|x_t^{(k)} - x_t^*\right\| \leq D$. Subtracting off the $\text{cost}\left(\text{OPT}_L, y_{1:T}\right)$ off of both sides, gives the dynamic regret. □

A.4 Theorem 5

Proof. We analyze a single trajectory $x_{1:T}^{(k)}$ in Algorithm 4 before they are averaged together. Let M_k be the number of times that $(1 + (t - 1) \mod v) = k$ is true for this particular k and let $\left(t^{(k),1}, ..., t^{(k),M_k}\right)$ be the timeslots for which it is true. Also, suppose that $\mathcal{X} = [0, D]$, $x_0 = 0$, $f(x_t, y_t) = \frac{\alpha}{w+1}|x_t - y_t|$, $y_t = D : \forall t \in \{1, \ldots, T\}$, the norm is the absolute value, and CHC receives the true cost functions as its predictions for the next w timeslots into the future. At timeslot $t^{(k),i}$, CHC solves the following problem:

$$\min_{\{x_t, \ldots, x_{t-1+w}\} \in [0,D]} \sum_{\tau=t}^{t-1+w}\left(\frac{\alpha}{w+1}|x_\tau - D| + \alpha|x_\tau - x_{\tau-1}|\right). \tag{29}$$

From Lemma 6, all decisions in its decision trajectory will be equal to $x_{t-1}^{(k)}$. Since trajectories $x_{1:T}^{(k)} : \forall k \in \{1, \ldots, v\}$ initialize at $x_0 = 0$, then all decisions in its trajectory will remain at $x_0 = 0$. Therefore, after averaging we have that $x_t^{\text{CHC}} = 0 : \forall t \in \{1, \ldots, T\}$. Another possible solution is to go to D and remain there, i.e. $\tilde{x}_t = D : \forall t \in \{1, \ldots, T\}$.

This gives us the following lower bound on dynamic regret:

$$\begin{aligned}
\mathcal{R}_T^{\text{CHC}}(0, L) &\geq \sum_{t=1}^{T}\left(\frac{\alpha}{w+1}\left|x_t^{\text{CHC}} - D\right| + \alpha\left|x_t^{\text{CHC}} - x_{t-1}^{\text{CHC}}\right|\right) - \sum_{t=1}^{T}\left(\frac{\alpha}{w+1}\left|x_t^* - D\right| + \alpha\left|x_t^* - x_{t-1}^*\right|\right) \\
&\geq \sum_{t=1}^{T}\left(\frac{\alpha}{w+1}\left|x_t^{\text{CHC}} - D\right| + \alpha\left|x_t^{\text{CHC}} - x_{t-1}^{\text{CHC}}\right|\right) - \sum_{t=1}^{T}\left(\frac{\alpha}{w+1}\left|\tilde{x}_t - D\right| + \alpha\left|\tilde{x}_t - \tilde{x}_{t-1}\right|\right) \\
&= \sum_{t=1}^{T}\frac{\alpha}{w+1}D - \alpha D \\
&= \frac{\alpha}{w+1}DT - \alpha D.
\end{aligned} \tag{30}$$

The first inequality comes from applying a specific set of cost functions. The second inequality comes from $\tilde{x}$ being no better than the optimal solution. The third equality comes from applying the values $x_t := x_t^{\text{CHC}} = 0$ and $\tilde{x}_t$ for all $t \in \{1, \ldots, T\}$. Rearranging the last expression gives the resultant. □

Lemma 6. *The optimal solution to* (29) *is* $x_\tau = x_{t-1} : \forall \tau \in \{t, \ldots, t - 1 + w\}$.

Proof. We can rewrite Problem (29) into the following dynamic programming structure, $\forall \tau \in \{t, \ldots, t-1+w\}$:

$$h_\tau(x_{\tau-1}) := \min_{x_\tau \in [0,D]} \left\{ \frac{\alpha}{w+1}|x_\tau - D| + \alpha|x_\tau - x_{\tau-1}| + h_{\tau+1}(x_\tau) \right\} \tag{31}$$

where $h_{t+w}(\cdot) := 0$.

We will solve the dynamic program by induction. Let the induction hypothesis for its optimal solution be, $\forall i \in \{1, \ldots, w\}$:

$$x^*_{t+w-i} = x_{t+w-i-1}$$
$$h_{t+w-i}(x_{t+w-i-1}) = \frac{\alpha i}{w+1}|x_{t+w-i-1} - D|$$

where x^*_τ is the optimal solution to $h_\tau(x_{\tau-1})$.

The base case starting at $i = 1$, gives the following problem:

$$h_{t+w-1}(x_{t+w-2}) = \min_{x_{t+w-1} \in [0,D]} \left\{ \frac{\alpha}{w+1}|x_{t+w-1} - D| + \alpha|x_{t+w-1} - x_{t+w-2}| \right\}.$$

In order for x^*_{t+w-1} to be optimal, it must satisfy the first-order stationary condition which is that 0 is in its subgradient. Since $w \geq 1$, then only $x_{t+w-1} = x_{t+w-2}$ allows 0 to be in its subgradient which makes it optimal. Also, when $x_{t+w-1} = x_{t+w-2}$, then

$$h_{t+w-1}(x_{t+w-2}) = \frac{\alpha}{w+1}|x_{t+w-2} - D|.$$

Therefore, the induction hypothesis is true for the base case.

For any $i \in \{1, \ldots, w\}$ and the applying induction hypothesis at $i-1$ to $h_{t+w-i+1}(x_{t+w-i})$, we have that:

$$h_{t+w-i}(x_{t+w-i-1}) = \min_{x_{t+w-i} \in [0,D]} \left\{ \frac{\alpha i}{w+1}|x_{t+w-i} - D| + \alpha|x_{t+w-i} - x_{t+w-i-1}| \right\}.$$

As before, the only value that makes x_{t+w-i} optimal by allowing 0 to be in the subgradient is at $x_{t+w-i} = x_{t+w-i-1}$. At $x_{t+w-i} = x_{t+w-i-1}$, then

$$h_{t+w-i}(x_{t+w-i-1}) = \frac{\alpha i}{w+1}|x_{t+w-i} - D|.$$

This finishes the proof of induction. The lemma's statement is implied by the induction hypothesis. □

Received November 2018; revised December 2018; accepted January 2019