Online Optimization in the Non-Stationary Cloud: Change Point Detection for Resource Provisioning

(Invited Paper)

Jessica Maghakian, Joshua Comden, Zhenhua Liu
Department of Applied Mathematics & Statistics, Stony Brook University
100 Nicholls Road, Stony Brook, New York, 11794, USA
Email: {jessica.maghakian, joshua.comden, zhenhua.liu}@stonybrook.edu

***Abstract*—The rapid mainstream adoption of cloud computing and the corresponding spike in the energy usage of big data systems make the efficient management of cloud computing resources a more pressing issue than ever before. To this end, numerous online algorithms such as Receding Horizon Control and Online Balanced Descent have been designed. However it is difficult for cloud service providers to select the best control algorithm dynamically for resource provisioning when confronted with consumer resource demands that are notoriously unpredictable and volatile. Furthermore, it highly possible that it might not be the case for any one algorithm to consistently perform well over the months-long contract period. In this paper, we first exemplify the need to address non-stationarity in cloud computing by showcasing traces from MS Azure. We then develop a novel meta-algorithm that combines change point detection and online optimization. The new algorithm is shown to outperform existing solutions in real-world trace-driven simulations.**

I. Introduction

The usage of cloud computing services has skyrocketed in popularity among enterprises because of its ability to scale quickly in a cost effective manner. With local IT hosting, there are large overheads in the time needed to get a system up and running, as well as in the costs of purchasing equipment and hiring specialists to fix unexpected system failures. However with cloud computing, these issues are taken care of by the cloud service provider which leaves the subscribing enterprise with only usage costs that scale linearly with the size of the subscription. As a result, 83% of enterprise computing workloads will be hosted via cloud computing by 2020 [1]. Therefore, our main goal is to improve the performance of resource provisioning for cloud service providers.

Cloud service providers incur different types of costs in provisioning resources to their clients. *Operational costs* are those that directly depend on the amount of resources provided which can include electricity and cooling costs, amortized hardware costs, and service level agreement violation penalties. On the other hand, *switching costs* depend on the changes in provisioning decisions and can include extra wear and tear on the hardware induced by starting up and shutting down servers and other equipment [2]. Notably, switching costs considerably complicate the resource provisioning problem by coupling its decisions in time.

In addition to dealing with time-coupled decisions, cloud service providers must also account for consumer resource demands that are unpredictable and often volatile over the long service contract periods. Proposed prediction models for workload demands include a plethora of approaches such as Exponential Smoothing [3], [4], Markov Chains [5], AutoRegressive [6] and Autoregressive Moving Average [7], [8], Holt-Winter [9], Naive Bayes [7], Neural Networks [10] and Pattern Matching [11]. Because of the high uncertainty in demands and the necessity to satisfy them in real-time, resources must be provisioned in an *online* manner.

Online control for provisioning IT resources is an active area of research with many proposed algorithms that not only achieve strong theoretical results but also perform well in practice. The main objective for any online algorithm is to make cost-effective decisions despite uncertain knowledge of the future. Many popular algorithms rely heavily on predictions with variants on model predictive control [2], [8], [12]–[16]. Other algorithms avoid dependency on potentially noisy predictions by instead utilizing realistic models to make smart threshold triggered provisioning decisions [17]–[19] or employ gradient descent techniques [20], [21].

However, the diversity of consumer resource demand characteristics makes selecting an appropriate control algorithm non-trivial since each algorithmic approach has its strengths and weaknesses. For example, model predictive control can take advantage of accurate predictions but can perform poorly when the predictions inaccurate; whereas, online gradient descent is immune to inaccurate predictions and can obtain near optimal performance if the demand varies slowly over time.

There have been various meta-algorithms designed specifically for combining the decisions made by several control algorithms. These include: Follow the Leader (FTL) [22] which uses the actions from the best current performing algorithm; Weighted Majority Algorithm (WMA) [23] which uses a weighted average of the algorithms where the weights are updated at each round according to each of their performances in the previous round; and many other variants based off of them. They attempt to gradually learn which control algorithm is the best to rely on and have recently been proposed for cloud resource provisioning in [24] and show great potential if the switching costs between algorithms is taken into account.

On the other hand, cloud resource demands can have sudden unexpected shifts in its statistical characteristics. Fortunately,

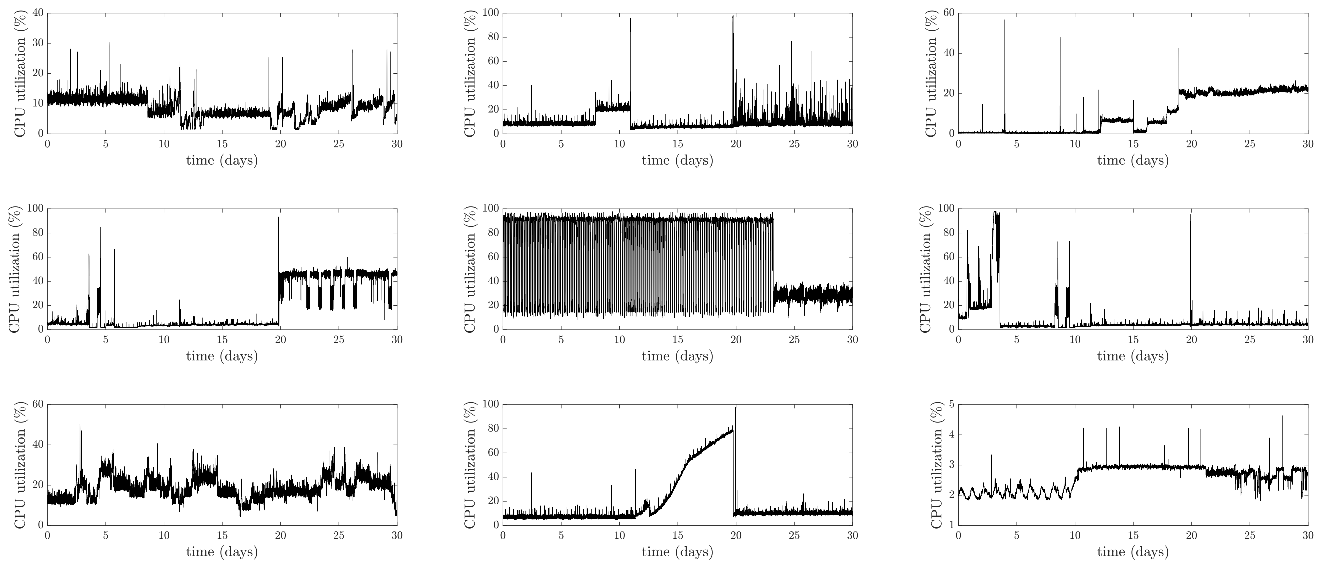


Fig. 1. Average CPU utilization for nine different Microsoft Azure virtual machines over a one month contract period.

detecting when these changes happen has a long rich history in the Statistics literature called Change Point Detection [25].

Therefore, this paper aims to improve algorithm selection meta-algorithms for cloud resource provisioning by:

1) Presenting a set of real world traces that capture the wide variety of non-stationarity behaviors present in resource demand in the cloud, specifically the sudden changes in demand patterns (Section II).
2) Designing a novel meta-algorithm that utilizes change-point detection to effectively select which algorithm to follow while taking into account previous performance (Section IV).
3) Evaluating the meta-algorithm's performance through real-world trace based simulations of CPU allocation of MS Azure virtual machines (Section V).

II. Motivation and Background

Change Point Detection

Consider a time series data stream $S = \{x_1, \dots, x_t, \dots\}$ that consists of an infinite sequence of elements x_t, where x_t is a d dimensional vector that arrives at time t. The time series data stream S is said to be stationary if its statistical properties are all constant over time. Change point detection is popular in application areas such as medical condition monitoring, climate change detection, speech recognition and image analysis [25]. In the case of cloud resource demands, unsupervised learning algorithms for change point detection are particularly attractive because they are capable of handling a wide variety of different situations without requiring computationally expensive training for each situation. Online unsupervised methods for change point detection include subspace, Bayesian, kernel-based, graph-based and clustering approaches. In particular, clustering methods do not have any restrictions on the time series data they permit.

Non-stationary CPU Utilization

To motivate the need for change point detection, we show real-world CPU utilization traces for virtual machines from the the Microsoft Azure Public Dataset [9] which contains the CPU utilization of over two million virtual machines (VMs). For each VM, the minimum, maximum and average percent utilization is reported every five minutes. We identified a set of 241 candidate VM traces from examining 1003 of them that exhibited extremely non-stationary behavior. Nine of them are displayed in Figure 1. Notice that each of them has very visible and distinct sudden changes in its CPU utilization and have a time dependent mean. Thus, they are not stationary. These abrupt changes may represent transitions between states that underlie the generation of the data. This gives strong evidence that utilizing change point detection is necessary in making cost-effective resource provisioning decisions.

Online Algorithm Selection

Online algorithm selection has its origins in the well studied Prediction with Expert Advice problem from theoretical computer science [26] where a learner tries to make the best decision in deciding an asset portfolio from the advice of the several experts. The goal is to do as well as the best expert in hindsight. Many different algorithms have been developed for this problem; the most popular are Follow the Leader (FTL) [22] which uses the actions from the best current performing expert, and Weighted Majority Algorithm (WMA) [23] which uses a weighted average of the experts where the weights are updated at each round according to each of their performances in the previous round. [24] developed a version of the FTL that can account for switching costs between consecutive actions whereas the original problem does not include this type of cost.

III. Problem Formulation

Model

Consider a cloud service provider that needs to continually provision resources to meet a dynamically changing demand from its subscribers in an online manner over a time horizon T. At every time instance $t \in \{1, \ldots, T\}$, 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 time instance t where each dimension represents a type of resource requested, and let $x_t \in \mathcal{X}$ be the vector of resources provisioned in the constrained provisioning action space $\mathcal{X} \in \mathbb{R}^n$.

In provisioning x_t, the cloud service provider experiences operational costs and switching costs. Operational costs may include the monetary cost of reserving and using virtual machines, amortized capitol costs and energy expenditure in running local resources, as well as delay cost such as revenue loss and service level agreement violation penalties for under-provisioning resources. The operational costs are modeled by a convex function $f(x_t, y_t)$ of the demand y_t and provisioned resources x_t. This form of function is general enough to capture a wide range of parametrized cost models for server power consumption [27]–[29] and latency [30]–[32]. Switching costs account for wear and tear as well as delay from the startup and shutdown of servers, as well as cost due to virtual machine migration and data transfer. These costs are modeled with a general norm of the difference in consecutive provisioning decisions $\alpha\|x_t - x_{t-1}\|$, where α represents the cost per unit of changing decisions. The resulting optimization problem that the cloud service provider needs to solve is:

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

where x_0 is a given initial point. Let the optimal solution to (1) be denoted by $(x_1^*, \ldots, x_T^*)$.

Online Optimization

If the demands $y_1, \ldots, y_T$ are known a priori, the optimization problem (1) can be solved effectively. However in the context of cloud computing, this is not the case. Because of the insufficient available information at each time instance, the cloud service provider must solve an online version of (1) with the goal of approximating the optimal solution $(x_1^*, \ldots x_T^*)$.

In practice, a cloud service provider has access to and can exploit predictions of future demands. Let $\hat{y}_{t|\tau}$ be the prediction of the next w time instances of y_t from time instant τ. At each time instant $t \in \{1, \ldots T\}$:

1) The cloud service provider utilizes the prediction of future demands $\hat{y}_{t|t}, \ldots, \hat{y}_{t+w-1|t}$ to make an allocation decision $x_t \in \mathcal{X}$.
2) The true demand y_t is subsequently revealed and the cloud service provider incurs the cost $f(x_t, y_t) + \alpha\|x_t - x_{t-1}\|$.

Thus, the total cost of a given online algorithm $\mathcal{A}$, denoted by $C(\mathcal{A}, y_{1:T})$, is given by:

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

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

IV. Online Algorithms

Online Control Algorithms

Traditionally, the online learning community considers optimization problems in which the algorithm has no access to future information. Algorithms such as online gradient descent (OGD) [33] can achieve great success by utilizing only the gradient of the objective function at the previous time instant. OGD performs particularly well when the offline optimal choice x_t^* at each time instant t does not fluctuate significantly over the given time horizon.

Despite the stringent assumptions of the online learning theory community, in practice predictions are available in a wide variety of applications. In particular, model predictive control takes advantage of access to future information. A popular algorithm paradigm, Receding Horizon Control (RHC) [2], at each time step optimizes its decision under the assumption that the predictions it uses have no error.

However, algorithms that heavily rely on predictions are sensitive to noisy or potentially corrupted information. A generalized variant of RHC, Committed Horizon Control (CHC) [34], uses the notion that predictions deteriorate the further they look into the future, and attempts to mitigate the impact of noisy predictions by averaging actions over several time steps.

Another recent algorithm, Online Balanced Descent (OBD) [35], instead utilizes the observation that the next step prediction usually is fine and utilizes only one step ahead predictions. The main idea of the algorithm 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.

Each of the described algorithms, OGD, RHC, CHC and OBD, has their own strengths and weakness. Choosing an optimal algorithm is non-trivial, which motivates the use of meta-algorithms for algorithm selection.

Online Algorithm Selection

Online algorithm selection has its origins in the well-known Prediction with Expert Advice problem in which there are k experts, and over a sequence of rounds the player has to choose which of the experts to take advice from [26]. A natural solution is the Follow the Leader (FTL) algorithm, which at each time instance t chooses to follow the advice of the expert with the best cumulative performance thus far.

Another popular algorithm is the Weighted Majority (WM) algorithm. At each timestep, the algorithm has a vector of weights $w_1, \ldots, w_k$ for how much weight is given to an expert. The better an expert, the higher its weight.

A recent meta-algorithm was developed specifically for the cloud resource provisioning environment [24]. It is a variant

Algorithm 1 Weighted Majority (WM)

```
1: Initialize selected algorithm, $\mathcal{A} \in \Gamma$.
2: for $t = 1, \ldots, T$ do
3:     Receive $x_t^{(i)}$ from every algorithm $i \in \Gamma$.
4:     Calculate and implement $x_t^{\mathcal{A}} = \frac{\sum_i w_t^{(i)} x_t^{(i)}}{\sum_i w_t^{(i)}}$.
5:     Update all weights by a factor of $\beta$ except algorithm
       $\arg\min_{i \in \Gamma} \left\{ f(x_t^{(i)}, y_t) + \alpha \| x_t^{(i)} - x_{t-1}^{(i)} \| \right\}$
6: end for
```

of FTL that also takes into account the switching cost. See Algorithm 3. The meta-algorithm only decides which control algorithm's actions it will implement periodically. While [24] has shown to perform well, they do not take into account the likely non-stationarity of the resource demands.

Algorithm 2 Periodic Follow the Closest Leader (PFCL)

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

Change Point Weighted Majority

The proposed Change Point Weighted Majority (CPWM) meta-algorithm builds on the traditional Weighted Majority algorithm by incorporating a simple change point detection algorithm similar to [36] which utilizes the change points by partitioning the time horizon into epochs in an online manner.

The change point detection algorithm relies on a clustering approach to detect potential shifts in an online manner. For each time instant t, the moving average $\hat{\mu}_t$ and moving standard deviation $\hat{\sigma}_t$ are calculated using a window of size w. If the point $(\hat{\mu}_t, \hat{\sigma}_t)$ is significantly far enough from the cluster of previous points $\{(\hat{\mu}_1, \hat{\sigma}_1), (\hat{\mu}_2, \hat{\sigma}_2), \ldots, (\hat{\mu}_{t-1}, \hat{\sigma}_{t-1})\}$, then the point y_t is flagged as a potential change point.

The weights for CPWM are updated similarly to WM except that the decisions of which algorithm performed the best is based on the accumulated cost during its current epoch instead of just the last time step.

Also, instead of weighting the implemented decision $x_t^{\mathcal{A}}$ based on all of the algorithms in Γ, it uses only a time dependent subset $\Gamma_t \subset \Gamma$ where Γ_t contains k_t control algorithms. The time dependent set Γ_t holds only the k_t algorithms that have the lowest accumulated cost since the beginning of the entire time horizon plus the cost of switching to its action. This is very similar to how the control algorithms are evaluated in PFCL, except here the best k_t are placed into a set for WM instead of just implementing the best.

Within each epoch, number of algorithms k_t remains constant which is decided at the first time step of the epoch, i.e. when the change point detection algorithm determined that a new epoch has started. It chooses k_t that minimizes the cost of the previous time step's weighted decision and holds that constant for the rest of the epoch.

Algorithm 3 Change Point Weighted Majority (CPWM)

```
1: Initialize selected algorithm, $\mathcal{A} \in \Gamma$.
2: for $t = 1, \ldots, T$ do
3:     Receive $x_t^{(i)}$ from every algorithm $i \in \Gamma$.
4:     Calculate and implement $x_t^{\mathcal{A}} = \frac{\sum_{i \in \Gamma_t} w_t^{(i)} x_t^{(i)}}{\sum_{i \in \Gamma_t} w_t^{(i)}}$.
5:     if $y_t$ is a change point then
6:         $\tau \leftarrow t$
7:         Pick $k_t := \arg\min_{k \in \{|\Gamma|\}} \left\{ f\left( \frac{\sum_{i \in \Gamma_k} w_{t-1}^{(i)} x_{t-1}^{(i)}}{\sum_{i \in \Gamma_k} w_{t-1}^{(i)}}, y_{t-1} \right) \right.$
8:             $\left. + \alpha \| x_{t-1}^{\mathcal{A}} - \frac{\sum_{i \in \Gamma_k} w_{t-1}^{(i)} x_{t-1}^{(i)}}{\sum_{i \in \Gamma_k} w_{t-1}^{(i)}} \| \right\}$
9:     end if
10:    Update all weights by a factor of $\beta$ except algorithm
       $\arg\min_{i \in \Gamma_t} \{ C(i, y_{\tau:t}) \}$.
11:    $\Gamma_{t+1} := \emptyset$
12:    for $j = 1, \ldots, k_t$ do
13:        Select $\Gamma_{t+1} := \Gamma_{t+1} \cup \arg\min_{i \in \Gamma \setminus \Gamma_{t+1}} \left\{ C(i, y_{1:t-1}) \right.$
14:            $\left. + \alpha \| x_{t-1}^{\mathcal{A}} - x_{t-1}^{(i)} \| \right\}$
15:    end for
16: end for
```

V. Performance Evaluation

Simulation Setup

Suppose that at every time step t, once every 5 minutes, a cloud provider needs to decide how much CPU x_t to allocate to a given VM. The amount of CPU is expressed as a fraction of the 5 minute interval a CPU is devoted to the VM. The cost incurred to the cloud service provider for supplying CPU linearly increases with c as the per unit cost due to operational expenses such as electricity and cooling. From the perspective of the client, 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 incurs a cost in the form of lost revenue or service agreement penalties. This cost increases linearly with the size of the demand deficit by unit cost p. However, if the allocation is not less than the demand, then no cost is incurred. The provider also incurs switching cost α when changing CPU allocation decisions between time slots. Each VM is simulated to run for 30 days and changes its demand quantities y_t every 5 minutes based on the CPU usage traces in the Azure Public Dataset [9]. The starting point x_0 is set to zero. 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. The switching cost parameter α is set to \$0.036/(5-min-core)

TABLE I
META ALGORITHM PERFORMANCE ACROSS VMS

$\mathcal{A}$	Best Meta-Alg	Better than Offline Select
WM	10	0
PFTL	60	0
CPWM	171	145

which has a maximum cost equivalent to running the CPU for 6 hours.

For the algorithms which utilize predictions, we utilize the Naive prediction model that predicts the previous demand y_{t-1} for all future time instances. At each 5-min timeslot, each prediction model gives CPU demand predictions for the next 288 5-minute (1 day's worth) timeslots. The online algorithms available for the Meta-algorithm to choose from include the following: RHC utilizing naive predictions, individual versions of OBD utilizing predictions with 21 different stepsize settings, and individual versions of OGD with 21 different stepsize settings. The stepsizes used in OBD and OGD are $\eta = 2^k : \forall k \in \{-10, \dots, 10\}$. The discount rate β for CPWM was set to 2^{-2} which was found to perform best among the VMs from the set 2^{-i} for $i \in \{1, \dots, 10\}$.

We use three 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 (WM) [23] which makes a decision based on the weighted average among different algorithms. The discount rate β was set to 2^{-5} which was found to perform best among the VMs from the set 2^{-i} for $i \in \{1, \dots, 10\}$. The third was PFCL where the commitment duration to stick to a particular online algorithm is set to be 144 5-minute (1/2 day's worth) timeslots.

Results

The performances of the meta algorithms are shown in Table I as the number of times each meta-algorithm was the best when run with the 241 virtual machines traces. The proposed CPWM algorithm achieved the lowest cost among the 171 traces which amounts to 71% of them. The table also shows the number of times that each algorithm beat the single best offline selected control algorithm. The CPWM outperformed the best offline selected control algorithm for more than 60% of the trials, even though neither Weighted Majority nor Periodic Follow the Leader was able to break through that barrier.

Figure 2 displays one of the tested virtual machines to showcase how CPWM can outperform all others. The CPU utilization trace is shown in Figure 2a along with its marked change points and average utilization between each consecutive pair of change points. The accumulated cost over time for each meta-algorithm is shown in Figure 2b. Notice that all of them started out performing very similarly until around day 3 when there was many sudden shifts in the utilization. Afterwards, CPWM broke away more and more after each sudden change due to its ability to readjust to the new utilization patterns, whereas the other meta-algorithms could not.

VI. CONCLUSION

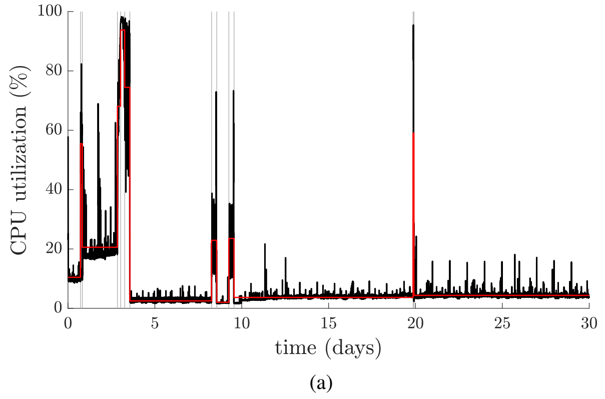
In this paper, we incorporate change point detection into control algorithm selection for cloud resource provisioning. We first demonstrate the need for change point detection by showcasing real-world virtual machine traces from the MS Azure Public Dataset [9]. Each displayed trace shows very sudden shifts in CPU usage patterns which need to be accounted for when choosing which control algorithm to run. We modify the widely used Weighted Majority Algorithm to utilize detected changes in resource demand so that it can adjust how it updates its weights. Simulated results on real-world traces show that this new meta-algorithm can beat existing solutions and can even beat the best single control algorithm chosen in hindsight.

ACKNOWLEDGMENTS

This research was partially funded by NSF grants CNS-1617698, CNS-1717588, CNS-1730128, and DGE-1633299.

REFERENCES

[1] T. Atobishi, S. Podruzsik, and S. Z. Gabor, "A review of the security challenges in the cloud computing," 2018.
[2] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 5, pp. 1378–1391, 2013.
[3] J. Huang, C. Li, and J. Yu, "Resource prediction based on double exponential smoothing in cloud computing," in *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*. IEEE, 2012, pp. 2056–2060.
[4] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan, "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, 2010, pp. 514–521.
[5] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems." *CNSM*, vol. 10, pp. 9–16, 2010.
[6] S. Khatua, A. Ghosh, and N. Mukherjee, "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, 2010, pp. 82–87.
[7] L. R. Moore, K. Bean, and T. Ellahi, "A coordinated reactive and predictive approach to cloud elasticity," 2013.
[8] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 500–507.
[9] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "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, 2017, pp. 153–167.
[10] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.
[11] E. Caron, F. Desprez, and A. Muresan, "Forecasting for cloud computing on-demand resources based on pattern matching," 2010.
[12] A. Al-Shishtawy and V. Vlassov, "Elastman: elasticity manager for elastic key-value stores in the cloud," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. ACM, 2013, p. 7.
[13] D. Ardagna, M. Ciavotta, and R. Lancellotti, "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, 2014, pp. 445–452.



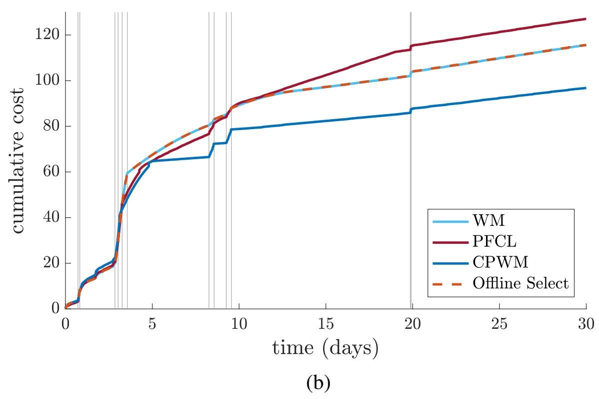


Fig. 2. (a) CPU utilization for a single virtual machine over a one month contract period with change points denoted by grey vertical lines and the average utilization within each epoch by the red horizontal lines; (b) Cost accumulations over time for each of the meta-algorithms and the best offline selected control algorithm along with the change points from (a) denoted by grey vertical lines.

[14] A. Ashraf, B. Byholm, and I. Porres, "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, 2012, pp. 581–586.

[15] E. Kalyvianaki, T. Charalambous, and S. Hand, "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, 2009, pp. 117–126.

[16] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A. Fortes, "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, 2011, pp. 32–42.

[17] J. Barr, "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/, 2018.

[18] C. Kan, "Docloud: An elastic cloud platform for web applications based on docker," in *Advanced Communication Technology (ICACT), 2016 18th International Conference on*. IEEE, 2016, pp. 478–483.

[19] P. Marshall, K. Keahey, and T. Freeman, "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, 2010, pp. 43–52.

[20] Y. Tan and C. H. Xia, "An adaptive learning approach for efficient resource provisioning in cloud services," *ACM Sigmetrics Performance Evaluation Review*, vol. 42, no. 4, pp. 3–11, 2015.

[21] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[22] A. Kalai and S. Vempala, "Efficient algorithms for online decision problems," *Journal of Computer and System Sciences*, vol. 71, no. 3, pp. 291–307, 2005.

[23] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Information and computation*, vol. 108, no. 2, pp. 212–261, 1994.

[24] J. Comden, S. Yao, N. Chen, H. Xing, and Z. Liu, "Online optimization in cloud resource provisioning: Predictions, regrets, and algorithms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, p. 30, 2019.

[25] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection," *Knowledge and information systems*, vol. 51, no. 2, pp. 339–367, 2017.

[26] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth, "How to use expert advice," *Journal of the ACM (JACM)*, vol. 44, no. 3, pp. 427–485, 1997.

[27] L. L. Andrew, M. Lin, and A. Wierman, "Optimality, fairness, and robustness in speed scaling designs," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1. ACM, 2010, pp. 37–48.

[28] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH computer architecture news*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[29] A. Wierman, L. L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," *IEEE INFOCOM, 2009*, pp. 2007–2015, 2009.

[30] N. Chen, X. Ren, S. Ren, and A. Wierman, "Greening multi-tenant data center demand response," *Performance Evaluation*, vol. 91, pp. 229–254, 2015.

[31] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *2011 Proceedings IEEE INFOCOM*.

[32] V. Shah and G. de Veciana, "Performance evaluation and asymptotics for content delivery networks," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 2607–2615.

[33] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 928–936.

[34] N. Chen, J. Comden, Z. Liu, A. Gandhi, and A. Wierman, "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, 2016, pp. 193–206.

[35] N. Chen, G. Goel, and A. Wierman, "Smoothed online convex optimization in high dimensions via online balanced descent," in *Proceedings of the 31st Conference On Learning Theory*, ser. Proceedings of Machine Learning Research, S. Bubeck, V. Perchet, and P. Rigollet, Eds., vol. 75. PMLR, 06–09 Jul 2018, pp. 1574–1594. [Online]. Available: http://proceedings.mlr.press/v75/chen18b.html

[36] D.-H. Tran, "Automated change detection and reactive clustering in multivariate streaming data," *arXiv preprint arXiv:1311.0505*, 2013.