2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)

Hierarchical and Frequency-Aware Model Predictive Control for Bare-Metal Cloud Applications

Yukio Ogawa
Center for Multimedia Aided Education
Muroran Institute of Technology
Muroran, Hokkaido 050-8585 Japan
Email: y-ogawa@mmm.muroran-it.ac.jp

Go Hasegawa
Cybermedia Center
Osaka University
Toyonaka, Osaka 560-0043 Japan
Email: hasegawa@cmc.osaka-u.ac.jp

Masayuki Murata
Graduate school of Information Science
and Technology, Osaka University
Suita, Osaka 565-0871, Japan
Email: murata@ist.osaka-u.ac.jp

***Abstract*—*Bare-metal cloud* provides a dedicated set of physical machines (PMs) and enables both PMs and virtual machines (VMs) on the PMs to be scaled in/out dynamically. However, to increase efficiency of the resources and reduce violations of service level agreements (SLAs), resources need to be scaled quickly to adapt to workload changes, which results in high reconfiguration overhead, especially for the PMs. This paper proposes a hierarchical and frequency-aware auto-scaling based on *Model Predictive Control*, which enable us to achieve an optimal balance between resource efficiency and overhead. Moreover, when performing high-frequency resource control, the proposed technique improves the timing of reconfigurations for the PMs without increasing the number of them, while it increases the reallocations for the VMs to adjust the redundant capacity among the applications; this process improves the resource efficiency. Through trace-based numerical simulations, we demonstrate that when the control frequency is increased to 16 times per hour, the VM insufficiency causing SLA violations is reduced to a minimum of 0.1% per application without increasing the VM pool capacity.**

***Index Terms*—Bare-metal cloud, frequency-aware, auto-scaling, Model Predictive Control, resource reconfiguration**

I. Introduction

Bare-metal cloud offers infrastructure as a service (IaaS) in which a customer uses a dedicated set of physical servers (also called physical machines (PMs)) on a pay-per-use basis [1]. Existing on-premises types of deployment for business-critical applications, such as web-based applications like e-mail and collaboration [2], often use dedicated PM clusters to handle peak workload to avoid violating service level agreements (SLAs) and to satisfy manageability of software licenses and requirements for audit of compliance and security, which can cause the applications to become over-provisioned and under-utilized most of the time [3]. We suppose that an application provider rents such a PM cluster from a bare-metal cloud provider to improve resource efficiency, creates a VM pool on the cluster, and hosts business-critical applications on the pool without changing existing management policies. In this paper, we try to develop an optimal resource allocation mechanism for such applications in bear-metal cloud environments.

Elasticity is a key concept in cloud computing, and resource allocation mechanisms that embody it have been investigated for many years [4], [5]. When SLA violations are caused by time delays between detecting a workload change and completing corresponding reconfigurations of resources, proactive mechanisms are essential in that future workloads need to be known ahead of time. Previous studies have predicted future workloads with time series analysis using models like the auto-regressive integrated moving average (ARIMA) model and other techniques [4], [5], in which prediction accuracy is significantly affected by workload characteristics, training data sets, etc. For example, in the case of a sudden spike in workload, known as a flash crowd [6], it may be difficult to obtain an appropriate training data set, and any prediction techniques can cause inevitable errors. Hence, as an approach to reduce the impact of prediction errors, we increase the control frequency (i.e., the frequency of reconfiguration decisions) so that resource reconfiguration can adapt more quickly to workload changes.

In this paper, we describe a prediction-based proactive scaling of both PMs and virtual machines (VMs) as the computing resources in bare-metal cloud environments. In commercial clouds, on-demand VMs are ready to use within a few minutes [7], and users are charged on a per-second basis [8]. On the other hand, users of commercial bare-metal instances, i.e., PMs, are currently billed on a per-hour basis [1]. However, the authors in [9], [10] have investigated that the deployment of PMs can be completed within several minutes. Hence, it is technically possible that the billing and provisioning period required for PMs can also be reduced to a few minutes in future commercial clouds. These short provisioning periods of PMs and VMs enable a bare-metal cloud to be reconfigured at high frequency, which improves the resource efficiency, i.e., reduces the redundant resources leading to extra costs and the insufficient resources (caused by prediction errors) resulting in SLA violations. However, when this high-frequency reconfiguration is kept even if workload change is negligible, significant reconfiguration overhead occurs. We therefore adopted model predictive control (MPC) [11].

MPC is an adaptive control framework in which reconfiguration decisions are made at each control step by solving a problem to optimize the balance between resource efficiency and reconfiguration overhead using predictions of future workloads. We have explored an MPC-based technique for scaling bare-metal cloud applications and identified the following challenges from architectural and technical perspectives:

978-1-5386-5504-7/18/$31.00 ©2018 IEEE
DOI 10.1109/UCC.2018.00010

- We suppose that applications are hosted on the VMs, which are distributed across underlying PMs to guarantee application availability. In addition, a lead time is the time period between the initiation and the completion of a reconfiguration process, and PMs generally require a longer lead time than VMs. Thus, before we initiate the reconfiguration process of the VMs to allocate them to the applications, we need to initiate the process of the PMs with an estimation of the capacity required for the applications at the time these processes complete.
- Increasing the control frequency to improve the resource efficiency is required. However, reconfiguration overhead also increases significantly when a high-frequency reconfiguration continues regardless of workload change speed. This excessive reconfiguration should be suppressed, especially for the PMs, depending on the control frequency and the speed of workload changes.

To overcome these challenges, we propose a hierarchical control scheme and a frequency-aware control technique, and clarify their effect. Our main contributions are as follows.

- We apply a two-level hierarchical MPC to PMs and VMs for which different lead times required by them are given. This scheme ensures that the PM cluster is scaled to have the estimated capacity required for the applications, and that the cluster dynamically impose a capacity constraint on the VMs allocated to the applications.
- The MPC utilizes a weight factor to adjust the balance between resource efficiency and reconfiguration overhead. We assign different weight factors to the PMs and VMs depending on the control frequency. This prevents the PMs from being reconfigured excessively while enabling the VMs to be reconfigured repeatedly, when high-frequency control is needed to react to rapid changes in workload.
- When the resources are controlled at high frequency, the proposed technique adjusts the reconfiguration timing of PMs without increasing the number of reconfiguring actions, as well as increasing the reallocating actions of VMs to redistribute redundant capacity among the applications, which leads to the reduction of SLA violations without increasing the VM pool capacity.

In prediction-based proactive scaling, a certain amount of redundant resources should be prepared to avoid the SLA violations caused by prediction errors, but excessive redundancy results in extra cost. Through numerical evaluations using real world traces, we demonstrate that, when the control frequency is increased from 1 to 16 times per hour, the insufficient VMs causing SLA violations is reduced to a minimum of 0.1% per application while maintaining the PM cluster capacity. Moreover, the number of reconfiguration for PMs decreases to one-third at that time.

The rest of this paper is organized as follows. Section II discusses related work on MPC-based auto-scaling in cloud environments. Section III presents our framework for applying MPC-based auto-scaling to a bare-metal cloud environment. Section IV describes the proposed model of hierarchical and frequency-aware auto-scaling. Section V explains the algorithm to find optimal values of the model. Section VI evaluates the effectiveness of our auto-scaling technique. Finally, Section VII concludes the paper.

II. Related Work

MPC-based auto-scaling is a promising technique for proactive resource allocation in cloud environments. In the case of a single data center, the authors in [12], [13] adjusted the total cost required for allocating and reconfiguring servers and SLA violations. The authors in [14], [15] also minimized the sum of the energy and on-off costs of the servers while maintaining their capacity to meet an SLA. Moreover, the authors in [16], [17] minimized the total cost required for hosting servers and their power consumption and reconfiguration while satisfying a latency constraint over multiple data centers. Jiao et al. [18] also minimized the total cost of allocating and reconfiguring both servers and networks in two-tier data centers. De Matteis and Mencagli [19], [20] developed a strategy that controls a stream-based application in parallel distributed computing environments, and minimizes the total cost of resource allocation, reconfiguration, and penalty for service quality.

Since we target the bare-metal cloud that hosts the applications composed of VMs across PMs, we need a hierarchical resource allocation scheme. Gaggero and Caviglione [21] proposed an MPC-based predictive control to minimize the energy consumption by PMs and the migrations of VMs hosted on the PMs in a data center. However, they neglected the lead times for both machines because they supposed long, e.g., 1-hour, control interval. Kusic et al. [22] also proposed an MPC-based online provisioning control to maximize the profit, i.e., the difference between the response time gain and the total cost of power consumption and switching actions for both PMs and their hosting VMs in a cloud environment. Although they take into account different lead times for the PMs and VMs, they model the lead times as constants and do not discuss the influence of various lengths of them. By contrast, we evaluated the influence because a longer lead time causes larger prediction errors of future workloads, which affect the allocated resources capacity and SLA violations.

Furthermore, it is essential to increase control frequency to improve resources efficiency, for which the reconfiguration overhead is increased as well. Handling this relationship is a key focus in this paper, which was not within the scope of the previous studies.

III. Scaling Framework

This section presents an overview of a bare-metal cloud and our control framework. For convenience, Table I summarizes the main variables and parameters used in this paper.

1) System Architecture: Our target cloud environment consists of a dedicated PM cluster provided by a bare-metal cloud provider and a controller operated by an application provider, as depicted in Fig. 1. An application comprises a computing cluster, i.e., a set of VMs assigned from the VM pool on the PM cluster, and other components such as firewalls, load

TABLE I
LIST OF MAIN SYMBOLS AND NOTATION

Symbols for describing control inputs	
$u(t)$	Number of PMs invoked at time slot t to be additionally turned on/off
$v_i(t)$	Number of VMs invoked at time slot t to be additionally allocated/deallocated for application i
f	Control frequency (times per unit time)
h	Length of control and prediction horizons (in time slots)
$L_{\rm u}$	Lead time for a PM (in time slots)
$L_{\rm v}$	Lead time for a VM (in time slots) $(1 \leq L_{\rm v} \leq L_{\rm u})$
Symbols for describing states of resources	
$x(t)$	Number of PMs at time slot t
$x^{\min}(t)$	Minimum of $x(t)$
$x^{\max}(t)$	Maximum of $x(t)$
$y_i(t)$	Number of VMs being allocated for application i at time slot t
$y_i^{\min}(t)$	Minimum of $y_i(t)$
$y_i^{\max}(t)$	Maximum of $y_i(t)$
$C_{\rm u}$	Cost of renting a PM per unit time
$W_{\rm u}$	Cost of a reconfiguration action per PM
$W_{\rm v}$	Cost of a reconfiguration action per VM
N	Number of VMs per PM
Symbols for describing performance of application i $(i = 1, \cdots, M)$	
$\lambda_i(t)$	Request arrival rate at time slot t (= maximum arrival rate monitored during time slot t)
$\hat{\lambda}_i(t)$	Predicted value of $\lambda_i(t)$
μ_i	Processing capacity of a VM
r_0	Network latency
Q	Upper limit of response time
R	Permissible ratio of response time of more than Q
Others	
$x(t+l\|t)$	Value of variable x at time slot $t{+}l$ calculated with available information at time slot t

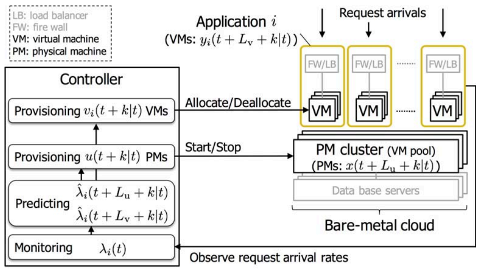

LB: load balancer
FW: fire wall
VM: virtual machine
PM: physical machine
Application i
(VMs: $y_i(t+L_{\rm v}+k|t)$)
Request arrivals
Controller
FW/LB
VM
Provisioning $v_i(t+k|t)$ VMs
Allocate/Deallocate
Provisioning $u(t+k|t)$ PMs
Start/Stop
PM cluster (VM pool)
(PMs: $x(t+L_{\rm u}+k|t)$)
Predicting
$\hat{\lambda}_i(t+L_{\rm u}+k|t)$
$\hat{\lambda}_i(t+L_{\rm v}+k|t)$
Data base servers
Bare-metal cloud
Monitoring $\lambda_i(t)$
Observe request arrival rates

Fig. 1. System architecture

balancers, and database servers, in which the PMs and VMs are scaled and other components are used statically. The controller has four functions of monitoring request arrivals to the applications, predicting future request arrivals, provisioning the PMs, and supplying VMs to the applications. It monitors request arrival rates to the applications at a fixed time interval. It also makes scaling decisions of the computing resources periodically at a fixed time interval called a time slot, and the current time slot is denoted by t. The control frequency f is the frequency of making scaling decisions, which is either equal to or lower than the monitoring frequency. The request arrival rate at t, denoted by $\lambda(t)$, is given by the maximum number of request arrival rates monitored during t to avoid SLA violations.

2) Lead Time: A lead time is defined as the time period between when a command is invoked to reconfigure a PM or VM and when the reconfiguration is completed; this is denoted by $L_{\rm u}$ and $L_{\rm v}$ for the PMs and VMs, respectively, and we assume $1 \leq L_{\rm v} \leq L_{\rm u}$. This lead time is composed of the time needed for environment setup and that for switching a resource. The environment setup comprises the configurations of, e.g., network and storage interfaces, server clusters, logging required for compliance and security, data copying, and procedures for authentication and payment, which correspond to the reconfiguration overhead. Switching a resource indicates the additional start-up/shutdown/reboot/migration of a PM or VM. We assume that most of the lead time is due to the environment setup, which can be very long, e.g., 1 hour, in the case of the PMs. Although the lead time is different among various environmental configurations and switching actions, we assume that the lead time does not depend on these factors for simplicity. In addition, we set the length of a time slot longer than the time for switching resources so that the resource switching at each time slot is completed within the time slot.

3) Control Steps: We adapt the MPC to scale the PMs and VMs, as depicted in Fig. 2. Control commands are invoked at each time slot along the *control horizon* of length h, i.e., the period between time slot t and time slot $t+h-1$. The commands are also completed at each time slot along the *prediction horizon* of the same length, i.e., the period between time slot $t+L_{\rm u}$ and time slot $t+L_{\rm u}+h-1$ for the PMs and that between time slot $t{+}L_{\rm v}$ and time slot $t{+}L_{\rm v}{+}h{-}1$ for the VMs. More precisely, at the current time slot t, the controller operates PMs and VMs through the following steps. Note that we denote a time slot on the control horizon as $t+k$, a time slot on the prediction horizon for the PMs as $t+l$, where $l = L_{\rm u} + k$, and a time slot on the prediction horizon for the VMs as $t+m$, where $m = L_{\rm v} + k$.

Step 1: The controller determines the control action at the current time slot t for scaling the PMs.

Step 1.1: It predicts $\hat{\lambda}_i(t+l|t)$, the rate of request arrivals to application i $(i = 1, \cdots, M)$ at time slot $t+l$, along the prediction horizon for the PMs using available request arrival information at time slot t.

Step 1.2: It calculates $x(t{+}l|t)$, the number of PMs needed to be active at time slot $t+l$, along the prediction horizon for PMs. $x(t{+}l|t)$ is computed to ensure that an SLA, e.g., a constraint on response time performance, will be satisfied when application i receives the request arrival rate of $\hat{\lambda}_i(t+l|t)$ at time slot $t+l$.

Step 1.3: It determines $u(t+k|t)$, the number of additional PMs invoked at time slot $t+k$ so that $x(t+l|t)$ PMs will be active after $L_{\rm u}$ time slots, along the control horizon.

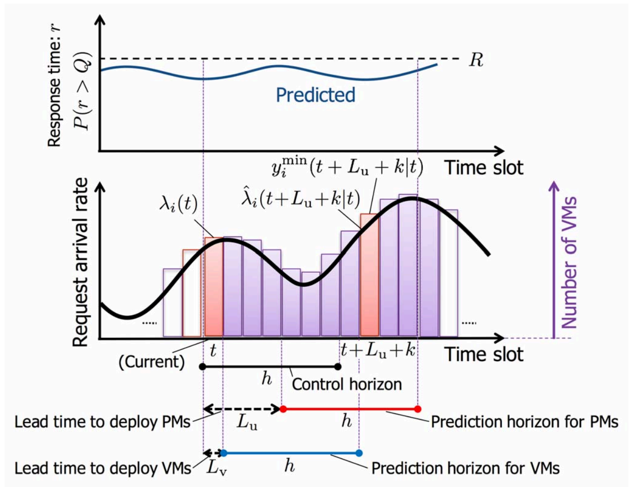


Fig. 2. Overview of model predictive control

Step 1.4: It issues only $u(t|t)$, the command of current time slot t for the PMs, which will have been completed at time slot $t + L_u$.

Step 2: The controller determines the control action at current time slot t for scaling the VMs of application i in a similar manner. This step is repeated for all the applications.

Step 2.1: It predicts $\hat{\lambda}_i(t + m|t)$ for the all applications.

Step 2.2: It calculates $y_i(t + m|t)$, the number of VMs being allocated to application i at $t+m$, using $\hat{\lambda}_i(t+m|t)$. It enables the VMs to make full use of the capacity of the PM cluster to reduce the risk of under-provisioning caused by the prediction errors.

Step 2.3: It computes $v_i(t + k|t)$, the number of additional VMs invoked at time slot $t + k$,

Step 2.4: It sends $v_i(t|t)$, the command at time slot t.

Step 3: It proceeds to the next time slot by replacing t with $t + 1$ and returns to Step 1.

IV. Model Formulation

This section presents a mathematical model for embodying the control steps explained in the previous section.

A. Problem Formulation

We explain the optimization problems that the controller solves to determine the control inputs.

1) Scaling of PMs: As mentioned in Step 1 in Section III, the controller solves the following problem at current time slot t to balance the redundant PMs and the reconfiguration overhead.

Objective: minimize

$$J_u = \left(\frac{C_u}{f}\right)^2 \sum_{l=L_u}^{L_u+h-1} \left(x(t+l|t) - x^{\min}(t+l|t)\right)^2 + W_u^2 \sum_{k=0}^{h-1} u(t+k|t)^2 \quad (1)$$

Subject to:

$$\begin{bmatrix} x(t+L_u|t) \\ \vdots \\ x(t+L_u+h-1|t) \end{bmatrix} = \begin{bmatrix} x(t+L_u-1|t-1) \\ \vdots \\ x(t+L_u-1|t-1) \end{bmatrix} + \begin{bmatrix} 1 & & 0 \\ \vdots & \ddots & \\ 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} u(t|t) \\ \vdots \\ u(t+h-1|t) \end{bmatrix}, \quad (2)$$

$$x(t+l|t) \geq x^{\min}(t+l|t) \quad (3)$$

$$(l = L_u + k;\ L_u = 1, 2, \cdots;\ k = 0, \cdots, h-1),$$

where $x^{\min}(t + l|t)$ is the minimum number of PMs required to keep an SLA at time slot $t + l$, which is explained in the next subsection.

The cost function J_u is a combination of the cost for holding redundant PMs along the prediction horizon and the cost for reconfiguring PMs along the control horizon. The reconfiguration cost is calculated on the basis of the total number of PMs being additionally activated/deactivated. The weight factors C_u/f and W_u represent the cost of renting a PM per time slot and the cost of a reconfiguration action per PM, respectively. As control frequency f becomes large, C_u/f reduces while W_u is not changed. This prevents the PMs from being overly reconfigured for small changes in workload when they are controlled at higher frequency.

Constraint (2) states that there are L_u time slots between the time slot at which a control command is sent to the PMs and that at which the PMs become ready to use. Constraint (3) ensures that the PMs should be provided with more than the minimal amount.

2) Scaling of VMs: As mentioned in Step 2 in Section III, the controller solves the following problem at current time slot t to balance the risk of VM insufficiency for the applications and the reconfiguration overhead, in a similar way to the scaling of the PMs.

Objective: minimize

$$\forall i\ (i = 1, \cdots, M),$$

$$J_{vi} = \left(\frac{C_u}{fN}\right)^2 \sum_{m=L_v}^{L_v+h-1} \left(y_i^{\max}(t+m|t) - y_i(t+m|t)\right)^2 + W_v^2 \sum_{k=0}^{h-1} v_i(t+k|t)^2 \quad (4)$$

Subject to:

$$\begin{bmatrix} y_i(t+L_v|t) \\ \vdots \\ y_i(t+L_v+h-1|t) \end{bmatrix} = \begin{bmatrix} y_i(t+L_v-1|t-1) \\ \vdots \\ y_i(t+L_v-1|t-1) \end{bmatrix} + \begin{bmatrix} 1 & & 0 \\ \vdots & \ddots & \\ 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} v_i(t|t) \\ \vdots \\ v_i(t+h-1|t) \end{bmatrix}, \quad (5)$$

$$y_i(t+m|t) \leq y_i^{\max}(t+m|t) \quad (6)$$

$$(m = L_v + k;\ L_v = 1, 2, \cdots;\ k = 0, \cdots, h-1),$$

where $y_i^{\max}(t+m|t)$ is the number of VMs available for application i at time slot $t+m$, which is explained in the next subsection, and Constraint (6) states that the number of VMs is equal to or less than $y_i^{\max}(t+m|t)$.

The cost function $J_{\mathrm{v}i}$ includes the cost proportional to the difference between the available and allocated VMs. The controller attempts to allocate VMs utilizing as much of the PM cluster's capacity as possible. Moreover, $^{C_{\mathrm{u}}}/_{fN}$ and W_{v} are the weight factors representing the cost of a VM per time slot and the cost of a reconfiguration action per VM, respectively, where N is the number of VMs per PM.

B. Scaling Range

The lower bounds for the numbers of PMs and VMs at time slots t, denoted by $x^{\min}(t)$ and $y_i^{\min}(t)$, respectively, are determined to avoid SLA violations. The upper bound for the number of VMs, denoted by $y_i^{\max}(t)$, is determined to make full use of the PM's capacity. In addition, the upper bound for the number of PMs, denoted by $x^{\max}(t)$, is conveniently set to solve the optimization problem in the next section.

1) Minimum Resources: The controller determines the minimum resources at each time slot to have sufficient capacity to keep the SLA whose metric is defined in the following way. The controller computes $F(r)$, the cumulative distribution function of response time r, for each application and imposes a constraint on $P(r > Q)$, the ratio of the request arrivals whose response time r is more than upper limit Q:

$$P(r > Q) = 1 - F(Q) \ \leq \ R, \tag{7}$$

where R is a threshold. A well-known way to obtain the cumulative distribution function $F(r)$ is to apply the M/M/m queuing model [23]:

$$\begin{aligned} F(r) &= 1 - \pi e^{-(\mu y - \hat{\lambda})(r - r_0)} \quad (r \geq r_0), \\ \pi &= \frac{y\rho^y}{y!(y-\rho)} \left[\frac{y\rho^y}{y!(y-\rho)} + \sum_{c=0}^{y-1} \frac{\rho^c}{c!} \right]^{-1}, \ \rho = \frac{\hat{\lambda}}{\mu}, \end{aligned} \tag{8}$$

where r_0 is a constant network latency, μ is the processing capacity of a VM, and we omit '$_i(t{+}l|t)$' for brevity. The SLA constraint (7) depends on the number of VMs, $y_i(t+l|t)$, and the prediction of request arrival rate, $\hat{\lambda}_i(t+l|t)$, and therefore cannot be satisfied for actual request arrival rates. Note that we adopt the waiting time distribution, not the sojourn time distribution in (8) because the applications are supposed to respond quickly to a request without waiting for the request to be completed.

The minimum number of VMs satisfying the SLA constraint (7), $y_i^{\min}(t+l|t)$, is then computed by:

$$y_i^{\min}(t+l|t) = \underset{y_i(t+l|t) \in \mathbb{N}}{\arg\min} \left\{ R - P(r > Q) \middle| P(r > Q) \leq R \right\}. \tag{9}$$

The minimum number of PMs, $x^{\min}(t+l|t)$, in Objective (1) is therefore given by:

$$x^{\min}(t+l|t) = \left\lceil \frac{1}{N} \sum_{i=1}^{M} y_i^{\min}(t+l|t) \right\rceil. \tag{10}$$

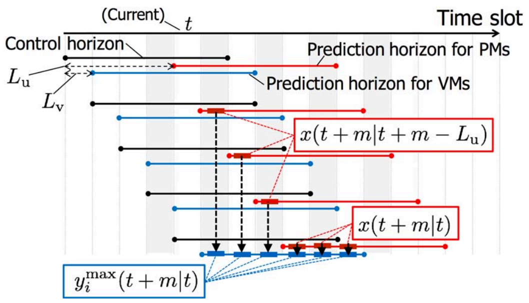


Fig. 3. Capacity constraint PMs impose on VMs

2) Maximum Resources: Any available space of redundant VMs in the PM cluster is divided among the applications proportionally by their VM amounts. The number of VMs available for application i, $y_i^{\max}(t+m|t)$, in Objective (4) is thus given by:

$$\begin{aligned} y_i^{\max}(t+m|t) &= round\left(N x^*(t+m) \frac{y_i^{\min}(t+m|t)}{\sum_{i=1}^{M} y_i^{\min}(t+m|t)} \right), \\ x^*(t+m) &= \begin{cases} x(t+m|t+m-L_{\mathrm{u}}) & (L_{\mathrm{v}} \leq m \leq L_{\mathrm{u}} - 1), \\ x(t+m|t) & (L_{\mathrm{u}} \leq m \leq h - 1), \end{cases} \end{aligned} \tag{11}$$

where $round$ means rounding a value to an integer. $x^*(t{+}m)$ is the capacity of the PM cluster, which has been determined at past time slot $t{+}m{-}L_{\mathrm{u}}$ if m is less than L_{u}, as depicted in Fig. 3. $y_i^{\max}(t+m|t)$ determined by using the past time slot might be less than $y_i^{\min}(t+m|t)$ calculated at current time slot t because of the prediction error of request arrival rate. If so, $y_i^{\max}(t+m|t)$ is replaced by $y_i^{\min}(t+m|t)$.

In addition, we define $x^{\max}(t{+}l|t)$, the maximum allowable number of PMs at time slot $t+l$, to form the envelop of minimum number of PMs over the prediction horizon, which is given by:

$$x^{\max}(t+l|t) = \max\left\{ x(t+L_{\mathrm{u}}-1|t-1), \max_{L_{\mathrm{u}} \leq l \leq L_{\mathrm{u}}+h-1} x^{\min}(t+l|t) \right\}. \tag{12}$$

C. Prediction of Request Arrivals

The controller calculates $\hat{\lambda}(t+l|t)$, l-time-slot-ahead prediction of request arrival rate at time slot t, with the ARIMA model [24]. Note that the subscript '$_i$' is omitted for brevity in this subsection. The series of a request arrival rate, $\lambda(t)$, is transformed into a stationary time series, $\Lambda(t)$, by applying the dth-order non-periodic differencing and the Dth-order periodic differencing:

$$\Lambda(t) = (1-B)^d (1-B^s)^D \lambda(t), \tag{13}$$

where B is the backward shift operator ($B\lambda(t) = \lambda(t-1)$), and s is the number of request arrival rates that make up a seasonal cycle. $\Lambda(t)$ is obtained by:

$$\Lambda(t) = \sum_{\alpha=1}^{p} \phi_\alpha B^\alpha \Lambda(t) + (1 + \sum_{\beta=1}^{q} \theta_\beta B^\beta)\epsilon(t), \tag{14}$$

where $\phi_\alpha, \theta_\beta$ are the parameters and $\epsilon(t)$ is white noise, i.e., $\epsilon(t) \sim N(0, \sigma^2)$. l-time-slot-ahead value $\Lambda(t+l)$ is then expressed as $\Lambda(t+l) = \sum_{j=0}^{\infty} \psi(j)\epsilon(t+l-j)$, where $\psi(j)$ is the parameter calculated from the past arrival rates and $\psi(0) = 1$ [24].

V. Resource Allocation Algorithm

The optimization problems described in Section IV-A are integer programming problems. Since $x(t+l|t)$ and $y_i(t+m|t)$ are non-negative integers, the optimal values of these variables can be found using dynamic programming, and subsequently, the optimal values of $u(t+k|t)$ and $v_i(t+k|t)$ are computed, respectively. We explain only the algorithm for determining the optimal value of $x(t+l|t)$ in this section because the optimal value of $y_i(t+l|t)$ can be determined as the same way.

First, we define the following equations to determine an optimal solution for Objective (1):

$$g(x(t+l|t)) = ({}^{C_\mathrm{u}}/_{f})^2 (x(t+l|t) - x^{\min}(t+l|t))^2 + W_\mathrm{u}^2 u(t+k|t)^2 \;\; (l = L_\mathrm{u} + k), \tag{15}$$

$$G_\kappa(\xi_\kappa) := \min \left\{ \sum_{l=L_\mathrm{u}}^{L_\mathrm{u}+\kappa} g(x(t+l|t)) \,\middle|\, \sum_{l=L_\mathrm{u}}^{L_\mathrm{u}+\kappa} x(t+l|t) = \xi_\kappa \right\}$$
$$(\kappa = 0, \cdots, h-1;\; \xi_\kappa = \xi_\kappa^{\min}, \cdots, \xi_\kappa^{\max}), \tag{16}$$

where $\xi_\kappa^{\min}$ and $\xi_\kappa^{\max}$ are the minimum and maximum numbers of PMs over the prediction horizon, respectively, which are given by:

$$\xi_\kappa^{\min} = \sum_{l=L_\mathrm{u}}^{L_\mathrm{u}+\kappa} x^{\min}(t+l|t), \tag{17}$$

$$\xi_\kappa^{\max} = \sum_{l=L_\mathrm{u}}^{L_\mathrm{u}+\kappa} x^{\max}(t+l|t). \tag{18}$$

The optimal values of Objective (1) are then obtained by:

$$\min J_\mathrm{u} = \min_{\xi_{h-1}^{\min} \le \xi_{h-1} \le \xi_{h-1}^{\max}} G_{h-1}(\xi_{h-1}), \tag{19}$$

which is calculated by the following recursive procedure:

$$G_0(\xi_0) = g(x(t+L_\mathrm{u}|t)),\; x(t+L_\mathrm{u}|t) = \xi_0$$
$$(\xi_0 = \xi_0^{\min}, \cdots, \xi_0^{\max}), \tag{20}$$

$$G_\kappa(\xi_\kappa) = \min_{x^{\min}(t+\nu|t) \le x(t+\nu|t) \le x^{\max}(t+\nu|t)} \Big\{ G_{\kappa-1}(\xi_\kappa - x(t+\nu|t)) + g(x(t+\nu|t)) \Big\}$$
$$(\nu = L_\mathrm{u} + \kappa;\; \xi_\kappa = \xi_\kappa^{\min}, \cdots, \xi_\kappa^{\max}). \tag{21}$$

VI. Experimental Evaluation

This section presents an experimental evaluation using real-world data traces. We suppose that the PMs are reconfigured at most once per hour while the VMs can be automatically reallocated and take several minutes to be usable in existing cloud environments. Starting from this assumption, our evaluation focuses on clarifying the effectiveness of the proposed technique when the control frequency is increased.

A. Experimental Setup

The parameter values we selected for the evaluations are listed in Table II, and additional explanations are described below:

1) Control inputs: We increased the control frequency f from 1 to 16 times per hour. This means that the length of a time slot ranges from 1 hour to 225 secs, in which the former is the length of the current billing interval [1] and the latter corresponds to the sum of the minimum time for provisioning a PM [9], [10] and that for a VM [7]. The length of control and prediction horizons is fixed to f time slots (i.e., 1 hour) because we assume that PMs are allowed to be reconfigured once an hour in existing clouds. The lead time L_u takes the values of f time slots and 1 time slot, while the lead time L_v takes only the value of 1 time slot because we assume that the reconfiguration of VMs has been already fully automated.

2) Resource states: The weight factors in Objectives (1) and (4) are defined in accordance with the ratio between resource cost and reconfiguration expense. A single PM cost C_u is in the range of \$0.517–\$4.747 per hour [1]; these values are exchanged on the yen basis at a rate of \$1 = ¥110. Moreover, we assume that the cost for reconfiguring a resource corresponds to the cost for employing technical staff members who manage and operate the resource. The personnel expense per staff member is in the range of ¥4700–¥8200 per hour, and a single staff member is supposed to manage up to 100 PMs or 1000 VMs [25] per hour. Thus, W_u is in the range of ¥47–¥82 per reconfiguration per PM, and W_v is in the range of ¥4.7–¥8.2 per reconfiguration per VM. Objective (1) is hence given ${}^{W_\mathrm{u}}/_{C_\mathrm{u}}$ ranging from 0.1 to 1.4. Although ${}^{W_\mathrm{v}}/_{C_\mathrm{u}}$ also ranges from 0.01 to 0.14, we give only the lower bound (${}^{W_\mathrm{v}}/_{C_\mathrm{u}}$ = 0.01) to Objective (4) because of the assumption about automated reconfiguration of the VMs.

3) Application performance: The monitoring interval of request arrival rates is set to the minimum length of a time slot (225 secs). To predict request arrival rate $\hat{\lambda}(t)$, we apply Box-Cox transformation [24] and set $d = 1$, $D = 1$, and $s = 168f$ (i.e., one-week seasonality) to (13), then identify the parameters in the ARIMA model (14) using the last two weeks of data [26]. In the SLA constraint (7), Q is set to 0.15 secs [27] and R is set to 0.05. Furthermore, μ_i is set to ${}^{1}/_{N}$ of the median of request arrival rates measured over the evaluation period (denoted by $\lambda_i^{\mathrm{median}}$) so that each application is scaled out to N VMs (i.e., a single PM) when it receives the median of request arrival rates.

TABLE II
PARAMETER VALUES USED FOR EVALUATION

f	1–16 times per hour
h	f time slots (= 1 hour)
L_u	1, f time slot(s) (= $1/f$, 1 hour(s))
L_v	1 time slot (= $1/f$ hours)
$W_\mathrm{u}/C_\mathrm{u}$	0.1–1.4
$W_\mathrm{v}/C_\mathrm{u}$	0.01
N	4 VMs per PMs
μ_i	$\lambda_i^\mathrm{median}/N$ ($\lambda_i^\mathrm{median}$: median of request arrival rates measured over the evaluation period)
r_0	0.02 secs (i.e., a supposed latency of a wide-area network)
Q	0.15 secs [27]
R	0.05

B. Trace data and prediction errors

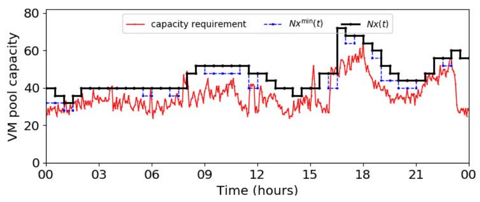
We suppose that three applications are hosted on the bare-metal cloud. We use HTTP traces from the following three real-world web applications. Each trace has a seven-week-long period and the starting time of each is set to time slot 1.

- *World Cup*: The 1998 FIFA World Cup website [28] (May 20, 1998 – Jul. 8, 1998, $\lambda_i^\mathrm{median} = 281$ requests/sec).
- *Campus*: A campus website of a university with about 30,000 students and staff members (Apr. 16, 2014 – Jun. 4, 2014, $\lambda_i^\mathrm{median} = 2.4$ requests/sec).
- *Video*: A video website to which access is traced on a gateway node of a Japan-wide backbone network (Sep. 30, 2015 – Nov 18, 2015, $\lambda_i^\mathrm{median} = 0.97$ requests/sec).

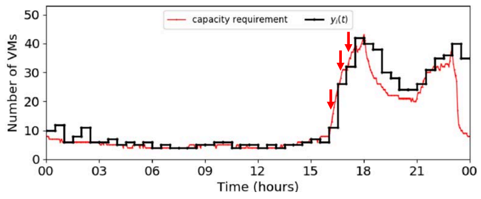
We use World Cup as an example of a website for individuals and Campus and Video as examples of websites for an organization or enterprise. Fig. 4 presents examples of the traces and corresponding resource allocations[1] over a 1-day period. Fig. 4a shows the total capacity requirement (calculated as $\sum_i y_i^\mathrm{min}(t)$ at $f = 16$ for actual request arrival rates) and the minimum and assigned capacity of the VM pool on the PM cluster (calculated as $Nx^\mathrm{min}(t)$ and $Nx(t)$, respectively). Figs. 4b, 4c and 4d also illustrate the required and allocated capacities for the applications, respectively.

Furthermore, Fig. 5 indicates that the prediction errors for f-time-slot-ahead (i.e., 1-hour-ahead) prediction (labeled by $L_\mathrm{u} = f$) and 1-time-slot-ahead prediction (labeled by $L_\mathrm{u} = 1$) in terms of the mean absolute error $\overline{|\lambda_i(t) - \hat{\lambda}_i(t)|}$ normalized by $\lambda_i^\mathrm{median}$. The prediction errors are mainly caused by a large spike lasting a few hours in the case of World Cup (see arrows in Fig. 4b) and by a fluctuation during several tens of minutes in the cases of Campus and Video (see arrows in Figs. 4c and 4d). The errors decrease as the control frequency f increases, i.e., the length of a time slot decreases, especially in the case of $L_\mathrm{u} = 1$, because a shorter interval reduces the workload changes within the interval, resulting in smaller prediction errors at the interval. This is especially true for the large spike lasting for hours resulting from a flash cloud event.

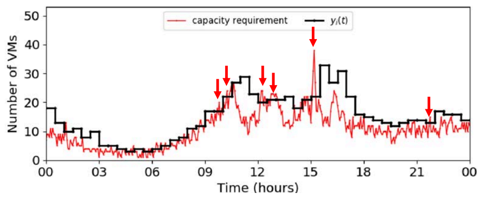
[1]The average computation time per time slot was less than 12.4 secs in our evaluation environment (CPU: 10 cores, 3.3 GHz, memory: 128 GB).



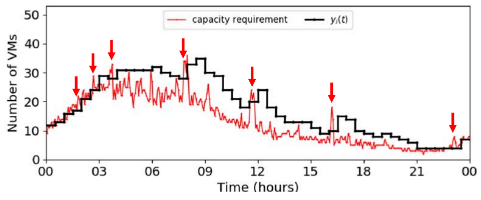
(a) Capacity of VM pool on PM cluster



(b) Number of VMs in World Cup



(c) Number of VMs in Campus



(d) Number of VMs in Video

Fig. 4. Examples of capacity requirements and allocated resources over a 1-day period ($f = 2$, $W_\mathrm{u}/C_\mathrm{u} = 0.7$, $L_\mathrm{u} = 1$, The arrows in Figs. 4b, 4c, and 4d indicate main time slots where SLA violations occur.)

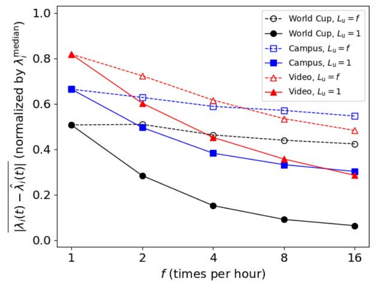
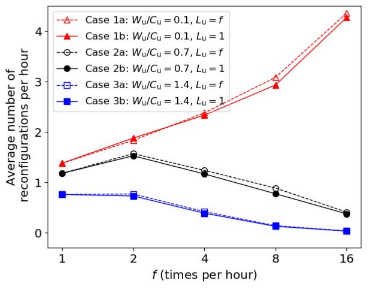


Fig. 5. Prediction errors

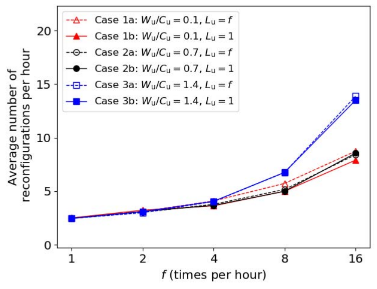
C. Effect of control frequency

Higher-frequency control can react faster to large and rapid changes. However, this leads to more reconfiguration actions. We thus evaluated the effect of our frequency-aware MPC on this context with the following three control options:

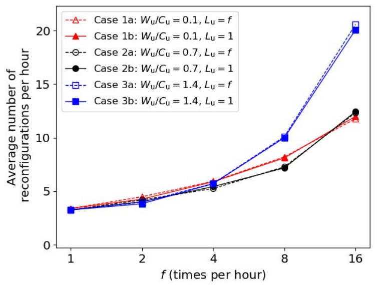
- Case 1: the lower weight for reconfiguration ($W_\mathrm{u}/C_\mathrm{u} = 0.1$)
- Case 2: the medium weight for reconfiguration ($W_\mathrm{u}/C_\mathrm{u} = 0.7$)
- Case 3: the upper weight for reconfiguration ($W_\mathrm{u}/C_\mathrm{u} = 1.4$)



(a) PMs

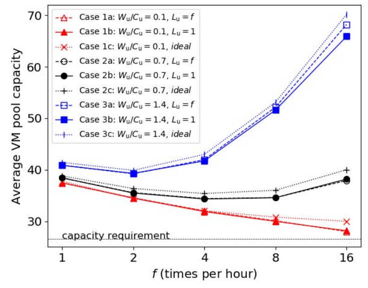


(b) VMs (smallest – World Cup)

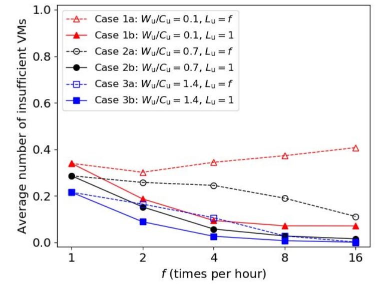


(c) VMs (largest – Campus)

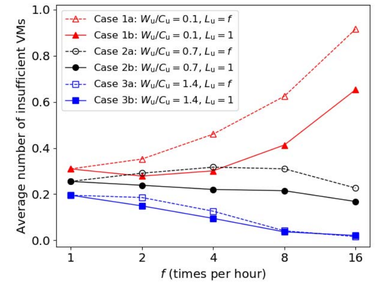
Fig. 6. Effect of control frequency on resource reconfiguration



(a) VM pool capacity



(b) Insufficient VMs (smallest – World Cup)



(c) Insufficient VMs (largest – Campus)

Fig. 7. Effect of control frequency on resource efficiency

Each of the cases was analyzed with (a) the upper bound of PM's lead time ($L_\mathrm{u} = f$) and (b) the lower bound of PM's lead time ($L_\mathrm{u} = 1$).

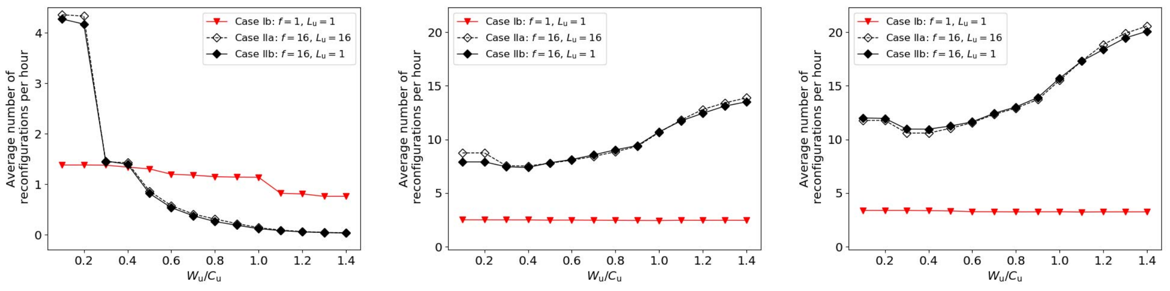
1) Reconfiguration: The average total number of reconfigured PMs per hour, i.e., $f\overline{|u(t)|}$, is shown in Fig. 6a, that of reconfigured VMs per hour (i.e., $f\overline{|v_i(t)|}$) in the least reconfigured application, i.e., World Cup is shown in Fig. 6b, and that of reconfigured VMs per hour in the most reconfigured application, i.e., Campus is shown in Fig. 6c. The total number of reconfigured PMs per hour increased from 1.4 at $f = 1$ to 4.4 at $f = 16$ in Case 1, while this increase was suppressed and the number decreased from 0.8 to 0.03 in Case 3. These results are slightly affected by the value of L_u. On the other hand, the total number of reconfigured VMs increases with f in all the applications because the VMs are easily reconfigured due to the minimum ${}^{W_\mathrm{v}}/_{C_\mathrm{u}}$. The total number becomes relatively large at $f = 8$ and $f = 16$ in Case 3 because the VM pool capacity becomes large at those times, as explained later.

In Case 2 and Case 3, as f increases, the total number of reconfigurations decreases for the PMs but increases for the VMs. This implies that the capacity of the VM pool is hold but the relocation of the VMs among the applications becomes more active with f.

2) Resource efficiency: We evaluated the number of redundant/insufficient VMs as metrics of resource efficiency. Fig. 7a shows average capacity of the VM pool on the PM cluster ($\overline{Nx(t)}$). It also indicates the average capacity requirement to clarify the redundant resources. Furthermore, Figs. 7b and 7c indicate the average number of insufficient VMs computed as the average shortage of the allocated VMs $y_i(t)$ from the capacity requirement for each application, where World Cup indicates the smallest insufficiency while Campus shows the largest. In Fig. 7a, all cases are additionally analyzed with no prediction errors, denoted by *ideal*, in which the results do not depend on the value of L_u.

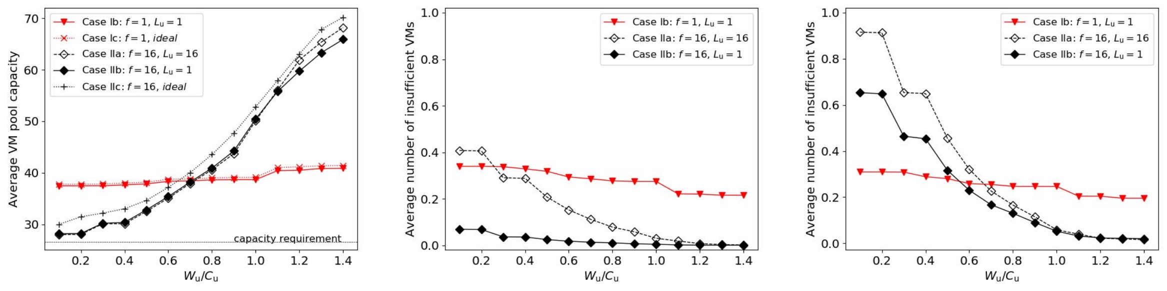
In Fig. 7a, when f increased from 1 to 16 in Case 1, the VM pool capacity was reduced from 37.5 to 28.1, which was very close to the capacity requirement 26.6. This is because high-frequency control with the light weight factor performs many reconfigurations, resulting in the reduction of resource redundancy. On the other hand, the VM pool capacity increased from 40.9 to 66.0 in Case 3 because fewer reconfigurations prevent the pool from shrinking as shown in Fig. 4a. These results are almost the same for all L_u and only slightly different from $ideal$. This indicates that the capacity of the PM cluster is little affected by the prediction errors.

Next, we explain the effect on SLA violations caused by resource insufficiency. World Cup had the fewest violations as shown in Fig. 7b because it had the smallest prediction errors among the three applications. When f increased from 1 to 16 in Case 1 with $L_\mathrm{u} = 1$, the number of insufficient VMs significantly improved from 0.34 to 0.07 because higher-frequency control can respond more quickly to a large spike like that in Fig. 4b, which makes the prediction errors small, leading to the reduction of insufficient VMs as well. In



(a) PMs (b) VMs (smallest – World Cup) (c) VMs (largest – Campus)

Fig. 8. Effect of reconfiguration weight factor on resource reconfiguration



(a) VM pool capacity (b) Insufficient VMs (smallest – World Cup) (c) Insufficient VMs (largest – Campus)

Fig. 9. Effect of reconfiguration weight factor on resource efficiency

contrast, the number of insufficient VMs became slightly worse from 0.34 to 0.41 in Case 1 with $L_u = f$ because the prediction errors do not significantly decrease with f and fewer redundant VMs are supplied at that time. On the other hand, in Case 3, the number of insufficient VMs improved from 0.22 to 0.001 for both $L_u = f$ and $L_u = 1$, owing to the significant increase of redundant VMs with f.

Campus experienced the most SLA violations (0.92 insufficient VMs) at $f = 16$ in Case 1 with $L_u = f$ as shown in Fig. 7c, which is caused by the largest prediction error and the smallest number of redundant VMs. The prediction error decreased with f in the case where $L_u = 1$ as shown in Fig. 5, but the number of insufficient VMs increased from 0.31 to 0.65 in Case 1 with $L_u = 1$ because of fewer redundant VMs. In contrast, the number of insufficient VMs improved from 0.20 to 0.02 VMs in Case 3 with both $L_u = f$ and $L_u = 1$ because many redundant VMs accommodates request arrival fluctuations like those shown in Figs. 4c and 4d.

3) Summary: High-frequency control in Case 1 performs many reconfigurations, which reduces the resource redundancy but causes many SLA violations, especially when the applications are controlled with $L_u = f$. Case 3 oppositely suppresses the SLA violations but provisions too many redundant VMs, and Case 2 lies in the middle of this trade-off relationship.

D. Effect of reconfiguration weight factor

Previous subsection explains the trade-off relationship between VM redundancy and insufficiency when controlled at high frequency. This subsection clarifies how much the weight factor W_u/C_u should weigh through the evaluations of the following two control options:

- Case I: low control frequency ($f = 1$)
- Case II: high control frequency ($f = 16$)

Each case was analyzed with (a) the upper bound of PM's lead time ($L_u = f$) and (b) the lower bound of PM's lead time ($L_u = 1$), but Case I with $L_u = f$ was omitted because it was equal to Case I with $L_u = 1$. We present the total reconfiguration numbers per hour and the resource efficiency as the function of W_u/C_u in Figs. 8 and 9 in the same manner as Figs. 6 and 7, respectively.

1) Reconfiguration: The total number of reconfigured PMs per hour was kept around 1 at any value of W_u/C_u in Case I. In Case II, the PMs were less reconfigured than those in Case I when W_u/C_u was set to more than 0.4 as shown in Fig. 8a, while the relocation of VMs from one application to another was performed more frequently at the corresponding value of W_u/C_u as shown in Figs. 8b and 8c. For example, when we chose the middle value of W_u/C_u, i.e. 0.7, the total number of reconfigured PMs per hour was reduced from 1.2 to 0.4, while that for reconfigured VMs per hour was increased from 2.5 to 8.5 in World Cup and from 3.2 to 12.4 in Campus.

2) Resource efficiency: The VM pool capacity remained around 38 in Case I. Fewer redundant VMs in the pool were provisioned in Case II than those in Case I when W_u/C_u was less than 0.7, which was slightly affected by the prediction error caused by L_u, as shown in Fig. 9a. On the other hand, fewer insufficient VMs in World Cup were provisioned in Case II than those in Case I when W_u/C_u was more than 0.3 even if $L_u = 16$ as shown in Fig. 8b, and those in Campus improved when W_u/C_u was more than 0.7 even if $L_u = 16$ as shown in Fig. 8c. The trade-off relationship between VM redundancy and insufficiency is therefore balanced when W_u/C_u is given the value of around 0.7. For example, when we set W_u/C_u to 0.7, the number of insufficient VMs was reduced from 0.29 to 0.11–0.01 in World Cup and from 0.26 to 0.23–0.17 in Campus web, while maintaining VM pool capacity.

3) Summary: If the controller increases the control frequency f with the smallest weight factor W_u/C_u, it brings only excessive reconfigurations to the PMs. In contrast, when it increases f with an appropriate W_u/C_u, it suppresses the reconfigurations of PMs and improves the timings of the reconfigurations, as well as increases the reconfigurations of VMs to reallocate the redundant VMs among the applications; this process can lead to the reduction of the insufficient VMs causing SLA violations without increasing the redundant VMs. Our evaluations demonstrate that when f is increased from 1 to 16 times per hour with $W_u/C_u = 0.7$, the number of insufficient VMs is reduced to 0.23–0.01 VMs (2–0.1% of the allocated VMs) per application and the total number of reconfigured PMs is reduced to one-third, while maintaining VM pool capacity of about 38 VMs. Furthermore, the evaluations indicate that the high-frequency control is effective especially for large spikes lasting for hours as a result of flash cloud events.

VII. Conclusion

In this paper, we proposed a hierarchical and frequency-aware MPC for bare-metal cloud applications composed of VMs over PMs. When the control frequency is increased, the proposed technique improves the timing of reconfigurations for the PMs without increasing their reconfiguration overhead, as well as increases the reallocations of the VMs to adjust the redundant capacity among the applications, which leads to the reduction of SLA violations without increasing the resource redundancy level.

This paper focuses on clarifying the effect of higher control frequency by comparing the assumed existing cloud (i.e., a cloud composed of the PMs controlled once an hour and the VMs relocated easily). Moreover, this paper examines only a bare-metal cloud hosting a web application for individuals and that for an organization together. For future work, we plan to evaluate the proposed technique with various control options. We will also separately evaluate a bare-metal cloud for individuals and that for an organization, each of which includes different request arrival characteristics.

References

[1] International Business Machines Corporation, "Bare Metal Servers," https://www.ibm.com/cloud/bare-metal-servers, accessed Jul. 20, 2018.

[2] VMware, Inc., "Virtualizing Business Critical Applications," https://www.vmware.com/be/solutions/business-critical-apps.html, accessed Aug. 3, 2018.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb. 2009.

[4] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 430–447, Jun. 2017.

[5] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 73:1–73:33, Jul. 2018.

[6] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites," in *Proc. of WWW '02*, May 2002, pp. 293–304.

[7] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. of IEEE Cloud Computing '12*, Jun. 2012, pp. 423–430.

[8] Amazon Web Services, Inc., "Amazon EC2 Pricing," http://aws.amazon.com/ec2/pricing/, accessed Mar. 27, 2018.

[9] M. D. d. Assunção and L. Lefèvre, "Bare-metal reservation for cloud: an analysis of the trade off between reactivity and energy efficiency," *Cluster Comput.*, Aug. 2017.

[10] A. Srbu, C. Pop, C. erbnescu, and F. Pop, "Predicting provisioning and booting times in a Metal-as-a-service system," *Future Gener. Comput. Syst.*, vol. 72, pp. 180–192, Jul. 2017.

[11] J. M. Maciejowski, *Predictive control: with constraints*. Prentice Hall, Sep. 2000.

[12] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proc. of IEEE Cloud Computing '11*, Jul. 2011, pp. 500–507.

[13] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna, "Replica placement in cloud through simple stochastic model predictive control," in *Proc. of IEEE Cloud Computing '14*, Jun. 2014, pp. 80–87.

[14] T. Lu, M. Chen, and L. L. H. Andrew, "Simple and effective dynamic provisioning for power-proportional data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1161–1171, Jun. 2013.

[15] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1378–1391, Oct. 2013.

[16] J. Yao, X. Liu, W. He, and A. Rahman, "Dynamic control of electricity cost with power demand smoothing and peak shaving for distributed internet data centers," in *Proc. of IEEE ICDCS '12*, Jun. 2012, pp. 416–424.

[17] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 762–772, Dec. 2013.

[18] L. Jiao, A. M. Tulino, J. Llorca, Y. Jin, and A. Sala, "Smoothed online resource allocation in multi-tier distributed cloud networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2556–2570, Aug. 2017.

[19] T. De Matteis and G. Mencagli, "Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing," in *Proc. of ACM PPoPP '16*, Mar. 2016, pp. 13:1–13:12.

[20] G. Mencagli, "Adaptive model predictive control of autonomic distributed parallel computations with variable horizons and switching costs," *Concurrency and Computat.: Pract. and Exper.*, vol. 28, no. 7, pp. 2187–2212, May 2016.

[21] M. Gaggero and L. Caviglione, "Predictive control for energy-aware consolidation in cloud datacenters," *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 2, pp. 461–474, Mar. 2016.

[22] D. Kusic, N. Kandasamy, and G. Jiang, "Combined power and performance management of virtualized computing environments serving session-based workloads," *IEEE TNSM*, vol. 8, no. 3, pp. 245–258, Sep. 2011.

[23] R. Jain, *The Art Of Computer Systems Performance Analysis*. John Wiley & Sons, Apr. 1991.

[24] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer-Verlag New York, Inc., Apr. 2010.

[25] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[26] R. Hyndman and Y. Khandakar, "Automatic time series forecasting: The forecast package for R," *Journal of Statistical Software*, vol. 27, no. 3, pp. 1–22, Jul. 2008.

[27] N. Tolia, D. G. Andersen, and M. Satyanarayanan, "Quantifying interactive user experience on thin clients," *Computer*, vol. 39, no. 3, pp. 46–52, Mar. 2006.

[28] The Internet Traffic Archive, "1998 World Cup Web Site Access Logs," http://ita.ee.lbl.gov/html/contrib/WorldCup.html, accessed Apr. 4, 2018.