Information Sciences xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

System resource utilization analysis and prediction for cloud based applications under bursty workloads

Jianwei Yin [a], Xingjian Lu [a,*], Hanwei Chen [a], Xinkui Zhao [a], Neal N. Xiong [b,c]

[a] College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang 310027, China
[b] School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, Jiangxi 330013, China
[c] School of Computer Science, Colorado Technical University, Colorado Springs, Colorado 80907, USA

ARTICLE INFO

Article history:
Received 17 November 2013
Received in revised form 10 March 2014
Accepted 29 March 2014
Available online xxxx

Keywords:
Burstiness
Workload generation
Fined-grained prediction
Index of dispersion for count

ABSTRACT

Performance analysis and prediction need a solid understanding of the system workload. As a salient workload characteristic, burstiness has critical impact on resource provisioning and performance of cloud based applications. Thus performance analysis and prediction under bursty workloads are of crucial importance to cloud based applications. However, it is yet challenging for such analysis and prediction, since no accurate and effective bursty workload generator exists, as well as the fine-grained bursty workload analysis and prediction method. In this article, to deal with these challenges, a bursty workload generator has been proposed for Cloudstone (a cloud benchmark) based on 2-state Markovian Arrival Process (MAP2). Then based on this generator, a fine-grained performance analysis method, which can be used to predict the probability density function of CPU utilization, has been suggested for cloud based applications, to support better resource provisioning decision making and system performance optimization. Finally, extensive experiments are conducted in a Xen-based virtualized environment to evaluate the accuracy and effectiveness of the two methods. By comparing the actual value of Indices of Dispersion for Count with the target value deduced from MAP2 model, the experiments show the precision of our method is superior to existing works. By comparing the real and predicted system resource utilization under a variety of bursty workloads generated by the proposed generator, the experiments also demonstrate the effectiveness and accuracy of the proposed fine-grained system resource utilization prediction method.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Workload has become critical in the cloud journey. On one hand, workload analysis is an important step in determining what can run effectively in a cloud environment. It is important to understand the characteristics of the workloads to determine whether it's suitable to be delivered as a cloud service. On the other hand, for applications that have been moved to clouds, workload analysis is the premise of many major operations such as software optimization, profiling and performance evaluation. To analyze the performance of a cloud system, one needs a solid understanding of its workload.

* Corresponding author. Tel.: +86 15168470230.
E-mail addresses: zjuyjw@zju.edu.cn (J. Yin), zjulxj@zju.edu.cn (X. Lu), chw@zju.edu.cn (H. Chen), zhaoxinkui@zju.edu.cn (X. Zhao), xiongnaixue@gmail.com (N.N. Xiong).

http://dx.doi.org/10.1016/j.ins.2014.03.123
0020-0255/© 2014 Elsevier Inc. All rights reserved.

ARTICLE IN PRESS

Some characteristics such as burstiness (which means highly variable request arrival rate or service time) of workloads have critical impact on resource provisioning strategies and performance of cloud based applications. For example, flash-crowd service requests can cause resource allocation problems and seriously degrade cloud system performance; High variance in incoming traffic and service time distributions can collapse the system in few seconds [13]; Simultaneously launching jobs for different cloud based applications, which are no longer single-program-single-execution applications, during a short time period can immediately aggravate resource competitions and load unbalancing among computing sites [33]. So performance analysis and prediction under bursty workloads is significant to cloud based application performance optimization.

However, performance analysis and prediction for cloud based applications under bursty workloads are still challenging. Firstly, there is no accurate and effective generator for bursty workloads. Though burstiness, which has been observed in Ethernet LAN [22], Web applications [6], storage systems [29], grid systems [23] and cloud systems [33], is characterized by many mathematical methods, including Self-similarity [22,6], Peakedness [9], Peak-to-mean Ratio, Coefficient of Variation, and Indices of Dispersion for Count (IDC) [14,4], a few existing works can support its generation and later analysis. Geist [17], SWAT [20] and the method proposed in [24] were developed to provide mechanisms for burstiness injection, but the accuracy, controllability and effectiveness are still far away from being satisfactory. Geist, which focuses on the characteristics of bursty workload can only control the distribution and correlation of inter-arrival times for open systems. SWAT that was built on top of Httperf, can only introduce burstiness by using a highly variable session length and think time distribution in the session mode. By using IDC, average think time and population of clients as inputs of 2-state Markovian Arrival Process (MAP2), the workload generation method proposed in [24] can inject burstiness into arrival stream in a controllable manner, however, it is difficult for users to provide the 'magic' value of IDC in practice. And the request arrival rate in bursty state was approximated inappropriately so that it often resulted in low accuracy.

Secondly, traditional performance analysis and prediction methods often focus on resource utilization calculated by mean value, which is not accurate or beneficial enough to support optimal resource provisioning and scheduling decision making. This inaccurate analysis and prediction may cause more critical impacts on cloud system resource provisioning and scheduling particularly under bursty workloads, since burstiness often means high variability on request arrival rate or service time even if the workload seems to be stable from the view of mean value. Without detailed information, such as the distribution of system resource utilization, we cannot succeed to learn the real workload demands. Furthermore, due to huge number of users, various kinds of applications, and the constantly expanding scale of clouds, the probability and intension of burstiness will be definitely enhanced in clouds. Thus the fine-grained performance analysis and prediction under bursty workloads makes much sense to cloud based applications, for supporting performance optimization, better provisioning and scheduling decision making. For instance, the authors of [37] suggested that server consolidation should considerate the fine-grained probability density function (pdf), for reducing the risk of performance violation. Though a large number of approaches for performance analysis and prediction have been proposed, few of them considers the fine-grained performance analysis and prediction methods for cloud based applications under bursty workloads.

In order to deal with these challenges, we develop a synthetic bursty workload generator (MAP2_Generator) for Cloudstone [32] (a cloud benchmark) based on 2-state Markovian Arrival Process (MAP2) in this paper. The model of our bursty workload generator is simple and it is easy to use. With some plain parameters that can be straightforwardly derived from real system logs or provided by performance analysts, the proposed generator can produce workloads with different intension of burstiness as you specified. Then, based on this generator, a fine-grained performance analysis and prediction method under bursty workloads is suggested for cloud based applications. This method can be used to predict the pdf of CPU utilization, to support cloud performance optimization, better provisioning and scheduling decision making. Extensive experiments are conducted in a Xen-based virtualized environment, to evaluate the accuracy and effectiveness of the two methods. By comparing the actual value of IDC estimated from system logs with the target value deduced from MAP2 model, the experiments show our generation method is more accurate than existing works. By comparing the real and predicted system resource utilizations under a variety of bursty workloads generated by the proposed generator, the experiments also demonstrate the effectiveness and accuracy of the proposed fine-grained system resource utilization prediction method.

The remainder of this paper is organized as follows. Section 2 introduces the bursty workload generation method for cloud benchmark Cloudstone. Section 3 describes more details the fine-grained resource utilization prediction method for both of bursty and non-bursty workloads. Section 4 uses two groups of experiments to evaluate the accuracy and effectiveness of the bursty workload generation method and the fine-grained utilization prediction method. The extensive experiment results demonstrate the effectiveness and accuracy of the two methods. Section 5 describes the related work. And Section 6 gives the conclusions of this paper.

2. Bursty workload generation

Cloudstone consists of two main parts: the client, which is a workload driver developed based on Faban,[1] and the server, which is a typical Web 2.0 application Olio.[2] Since the generation tool for bursty workloads is not provided by Faban, we design and implement a bursty workload generator (MAP2_Generator) for Cloudstone based on MAP2. The main notations used in this paper are summarized in Table 1.

[1] http://java.net/projects/faban/.
[2] http://incubator.apache.org/olio/.

Table 1
Summary of notations.

Symbol	Meaning
λ_b	The request arrival rate for a client in bursty state
λ_n	The request arrival rate for a client in normal state
$\lambda_{N,Z}$	The request arrival rate when the population is N and the think time is Z
A	The discrete random variable that represent the number of arrivals in each sequential interval (i.e. one second) $\{a_1, a_2, \ldots, a_n\}$
C_b	The inter-arrival time in bursty state
C_n	The inter-arrival time in normal state
f	The frequency of burstiness
I	The index of dispersion for counts
k_b	The number of requests sent during a bursty period
k_n	The number of requests sent during a normal period
N	The number of clients (also called population) in a system
N_Z	The threshold of clients. When the population is bigger than this value for some think time Z, the system will become saturated
R_{avg}	The average system response time
R_b	The system response time in bursty state
R_n	The system response time in normal state
$R_{N,Z}$	The system response time when the population is N and the think time is Z
t_b	The average duration time in bursty state
t_n	The average duration time in normal state
U_{avg}	The average CPU utilization
X_b	The system throughput in bursty state
X_n	The system throughput in normal state
$X_{N,Z}$	The system throughput when the population is N and the think time is Z
X_{max}	The maximal throughput under certain system configuration
Z_{avg}	The average think time
Z_b	The think time in bursty state
Z_n	The think time in normal state
Z_N	The threshold of think time. When the think time is smaller than this value for some population N, the system will become saturated

2.1. Model definition

The burstiness of workloads can be caused by highly variable request arrival rate or service time. Since service time is determined by system characteristics that are not available to change, we mainly focus on the burstiness resulted by variable request arrival rate in this paper. The formal definition of bursty workload is as follows:

Definition 1 (*Bursty Workload*). The request arrival rate random variable $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ significantly deviates from the average arrival intensity (often not an exponential distribution), where $\alpha_i (i = 1, 2, \ldots, n)$ shows the number of arrivals in the fixed time interval t.

The most widely used index to denote the intension of burstiness is IDC [24]:

$$I = \lim_{t\to\infty} \frac{var(A)}{E(A)}, \tag{1}$$

where $var(A)$ and $E(A)$ represent the variance and the mean value of A respectively. When $I = 1,\ A$ follows an exponential distribution, i.e. there is no burstiness; when $I > 1$, the workload is bursty, and the larger the value of I, the higher the intension of burstiness.

Markovian Arrival Process has been successfully used to model the bursty workload. Considering the computational tractability, we use MAP2 to model the bursty workload generation in this paper. As shown in Fig. 1, one state of MAP2 represents the bursty request arrival process, while the other denotes the normal request arrival process. For these two states, the request arrival rate λ_b and λ_n are both exponential distributed. When a request is triggered, the underlying Markov chain has a probability P_{bn} or P_{nb} to transit from one state to the other. So it needs to provide a quadruple $\langle \lambda_b, \lambda_n, P_{bn}, P_{nb} \rangle$, to construct such a MAP2 model.

2.2. Parameterizing MAP2 Model

Similar to most web application benchmarks (such as TPC-W, and RUBiS), Cloudstone belongs to a kind of closed system, whose workload intension is determined by the number of users N and the think time Z. As we can never change the value of N in the run time, we only change the value of Z to mimic different burstiness intension of workloads. So parameterizing a MAP2 model transfers to how to use N and Z to construct the quadruple $\langle \lambda_b, \lambda_n, P_{bn}, P_{nb} \rangle$.

Definition 2 (*Inter-arrival Time*). Describes the time interval between any two requests.

Definition 3 (*Think Time*). Describes the time interval between receiving last response and sending a new request.

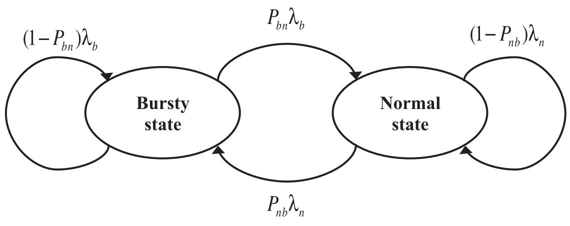


Fig. 1. Modeling bursty workload with MAP2.

First, we consider λ_b and λ_n. Assuming the time for sending and receiving request is negligible, the request arrival rate for a client can be defined as:

$$\lambda_{N,Z} = \frac{1}{C_{N,Z}} = \frac{1}{R_{N,Z}+Z}, \tag{2}$$

where $C_{N,Z}$ is the inter-arrival time, $R_{N,Z}$ is response time when the number of users is N and think time is Z.

According to (2), in order to get the value of λ_b and λ_n, the system response time for bursty and normal state R_b and R_n are needed. But it's impractical to derive $\lambda_{N,Z}$ by collecting all data of $\langle N, Z, R_{N,Z}\rangle$.

In addition, according to the response time law, we have:

$$X_{N,Z} = \frac{N}{R_{N,Z}+Z}. \tag{3}$$

So the $\lambda_{N,Z}$ can also be represented as:

$$\lambda_{N,Z} = \frac{X_{N,Z}}{N}. \tag{4}$$

For a closed system, the request arrival rate $\lambda_{N,Z}$ is inversely proportional to think time Z when the state is non-saturated. So together with (2) and (4), the request arrival rate can be approximated as:

$$\lambda_{N,Z} = \begin{cases} 1/Z & \text{if } N < N_Z \text{ and } Z > Z_N, \\ X_{max}/N & \text{if } N \geqslant N_Z \text{ or } Z \leqslant Z_N, \end{cases} \tag{5}$$

where N_Z and Z_N are thresholds for saturation, and X_{max} is the maximal throughput of system. If the system is not saturated ($N < N_z$ and $Z > Z_N$), the response time is negligible compared to the think time. So $\lambda_{N,Z}$ is approximated by $1/Z$. Otherwise, $\lambda_{N,Z}$ is set to X_{max}/N. Therefore, λ_b (λ_n) can be determined by testing if the pair value $\langle N, Z_b\rangle$ ($\langle N, Z_n\rangle$) can cause system saturated.

Next, we need to set value for P_{bn} and P_{nb}, which is also a nontrivial work. By analyzing the access logs in server side, we can have the total request arrival rates $N\lambda_b$ (N times of λ_b for a client), $N\lambda_n$, and the duration times for bursty (t_b) and normal (t_n) state as shown in Fig. 2. We define the frequency of burstiness as:

$$f = \frac{t_b}{t_b+t_n}. \tag{6}$$

Based on these data, we can calculate the average number of requests sent by a client during each state:

$$E(k_b) = N\lambda_b t_b, \quad E(k_n) = N\lambda_n t_n. \tag{7}$$

$E(k_b)$ and $E(k_b)$ can be interpreted as geometric distribution. That is, the underlying Markov chain can transit from bursty (normal) state to normal (bursty) state after sending Y_b (Y_n) requests. And the expected value of the random variables Y_b and Y_n can be defined as:

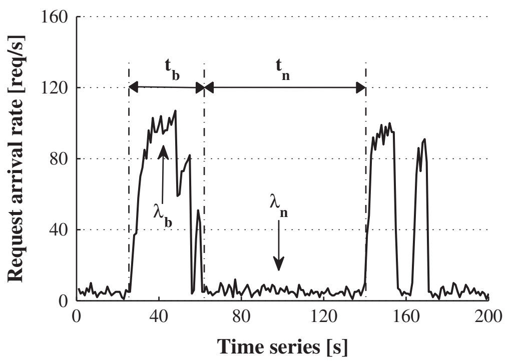


Fig. 2. Sample of request arrival rate.

$$E\left(Y_b = \frac{k_b}{N}\right) = \frac{1}{P_{bn}}, \quad E\left(Y_n = \frac{k_n}{N}\right) = \frac{1}{P_{nb}}. \tag{8}$$

Then the values of P_{bn} and P_{nb} can be determined with (7) and (8). Now, the two square matrices D_0 and D_1, which are used to define a MAP2 from a client's angle of view, are constructed as:

$$\mathbf{D_0} = \begin{bmatrix} -\lambda_b & 0 \\ 0 & -\lambda_n \end{bmatrix}, \quad \mathbf{D_1} = \begin{bmatrix} (1-P_{bn})\lambda_b & P_{bn}\lambda_b \\ P_{nb}\lambda_n & (1-P_{nb})\lambda_n \end{bmatrix}. \tag{9}$$

And the IDC and average request arrival rate can be calculated as [24]:

$$I = 1 + \frac{2P_{bn}P_{nb}(\lambda_b - \lambda_n)^2}{(P_{bn}\lambda_b + P_{nb}\lambda_n)^2(P_{bn} + P_{nb})}, \tag{10}$$

$$\lambda_{avg}^{-1} = \frac{P_{nb}\lambda_b^{-1} + P_{bn}\lambda_n^{-1}}{P_{bn} + P_{nb}}. \tag{11}$$

Notice that if we deduce the IDC from the server's perspective, we will have the exactly same value as (10). Also, from (1) and (11), we can get the average response time R_{avg} and think time Z_{avg}:

$$R_{avg} = \frac{P_{nb}R_b + P_{bn}R_n}{P_{bn} + P_{nb}} = \frac{f\lambda_b Z_b + (1-f)\lambda_n Z_n}{f\lambda_b + (1-f)\lambda_n}, \tag{12}$$

$$Z_{avg} = \frac{P_{nb}Z_b + P_{bn}Z_n}{P_{bn} + P_{nb}} = \frac{f\lambda_b R_b + (1-f)\lambda_n R_n}{f\lambda_b + (1-f)\lambda_n}. \tag{13}$$

According to Eq. (12) and (13), both R_{avg} and Z_{avg} are independent of t_b if f keeps constant, which will be verified in the experiment section.

2.3. Bursty workload generator based on MAP2

As the workload driver of Cloudstone, Faban also provides an extensible framework for workload driver development. Based on this framework, we develop a bursty workload generator called 'MAP2_Generator' by extending the basic class 'com.sun.faban.driver.engine.Cycle' in Faban driver framework.

The generator needs to be initialized by a six-tuple $\langle Z_b, Z_n, f, t_b, N, X_{max}\rangle$ and a threshold list file. The threshold list file contains some records of $\langle N_Z, Z_N\rangle$, which can be collected by running the original workload driver in the testbed. Algorithm 1 depicts the steps of initializing MAP2_Generator. First, λ_b and λ_n are determined using (5) (line 1–9). Second, P_{bn} and P_{nb} can be derived from λ_b, λ_n, f, and t_b based on (6)–(8) (line 10 and 11). At last, the initial state of MAP2 model is set to a normal state (line 12).

After initialization, the workload driver can use MAP2_Generator to generate a series of think time that can cause required intension of burstiness. Algorithm 2 will generate a think time based on current state of MAP2 (line 1–7). In each state, the think time is exponential distributed with $mean = Z_b \text{or} Z_n$. Meanwhile, the next state should be set based on a randomly generated probability (line 8–11). Using our method, the users can flexibly adjust the parameters $\langle Z_b, Z_n, f, t_b, N, X_{max}\rangle$ and see how it can impact system performance.

Algorithm 1. Initializing MAP2_Generator.

```
Input: $Z_b,\ Z_n,\ f,\ t_b,\ N,\ X_{max}$ and Threshold list file
Output: $P_{bn},\ P_{nb}$ and state
 1: $\lambda_b \leftarrow 1/Z_b,\ \lambda_n \leftarrow 1/Z_n$
 2: for each record $\langle N_Z, Z_N\rangle$ in Threshold list file do
 3:     if $N \geqslant N_Z$ and $Z_b \leqslant Z_N$ then
 4:         $\lambda_b \leftarrow X_{max}/N$
 5:     end if
 6:     if $N \geqslant N_Z$ and $Z_n \leqslant Z_N$ then
 7:         $\lambda_n \leftarrow X_{max}/N$
 8:     end if
 9: end for
10: $t_n \leftarrow t_b/f - t_b$
11: $P_{bn} \leftarrow 1/(\lambda_b t_b),\ P_{nb} \leftarrow 1/(\lambda_n t_n)$
12: state $\leftarrow$ 0 /* 0 for normal state and 1 for bursty state. */
```

Algorithm 2. Generating a think time.

```
Input: $Z_b,\ Z_n,\ P_{bn},\ P_{nb}$ and $state$
Output: $Z$ and $next\_state$
 1: if $state = 0$ then
 2:    $mean \leftarrow Z_n,\ P \leftarrow P_{nb}$
 3: else
 4:    $mean \leftarrow Z_b,\ P \leftarrow P_{bn}$
 5: end  if
 6: $r \leftarrow random(0,1)$ /*Randomly generate a value between 0 and 1*/
 7: $Z \leftarrow -mean * \ln r$
 8: $prob \leftarrow random(0,1)$
 9: if $prob < P$ then
10:    $next\_state \leftarrow (state + 1) \bmod 2$
11: end if
```

2.4. Estimating IDC from measurement

To validate whether the generated workload is consistent with the MAP2 model, we compare the IDC calculated from MAP2 model in client side (called target I) with the IDC estimated from the measured request arrival rate in server side (called actual I). The target I can be derived using (10). And the actual I should be estimated as [14]:

$$I = \lim_{n\to\infty} \frac{var(A)}{E(A)}\left[1 + 2\sum_{j=1}^{n-1}\left(1 - \frac{j}{n}\right)\rho_j\right] = \frac{var(A)}{E(A)}\left(1 + 2\sum_{j=1}^{\infty}\rho_j\right), \tag{14}$$

where $A = \{a_1, a_2, \ldots, a_n\}$ is a discrete random variable that represents the number of arrivals in each sequential interval (i.e. one second), $var(A)$ is the variance of A, $E(A)$ is the mean of A, and ρ_j is the autocorrelation coefficient of A at lag j.

However, we may fail to get a A with infinite elements in practice. So we propose Algorithm 3 to estimate the IDC of measured request arrival rate. Using Matlab's sample autocorrelation function (autocorr), we can have a autocorrelation sequence with the confidence bounds as shown in Fig. 3. And the theoretical values of autocorrelation within bounds is effectively 0. So in Algorithm 3, only the first i autocorrelation values ($\rho_i \geqslant b$) are selected for IDC estimation (line 2–6).

Algorithm 3. Estimating IDC of measured request arrival rate.

```
Input: $A = \{a_1, a_2, \ldots, a_n\}$
Output: $I$ and $lag$
 1: $[\{\rho_1, \rho_2, \ldots, \rho_{n-1}\}, \pm b] \leftarrow autocorr(A, n-1)$ /* Generate a autocorrelation sequence from lag 1 to $n-1$. */
 2: for $i \leftarrow 1$ to $n-1$ do
 3:    if $\rho_i < b$ then
 4:       break
 5:    end if
 6: end for
 7: $lag \leftarrow i - 1$
 8: $sum \leftarrow 0$
 9: for $j \leftarrow 1$ to $lag$ do
10:    $sum \leftarrow sum + \rho_j$
11: end for
12: $I = \frac{var(A)}{E(A)}(1 + 2sum)$
```

3. Fine-grained system resource utilization prediction

In this section, we first developed a prediction method for probability density function $f(x)$ of system resource utilization under non-bursty workloads. Then, by combining this method and the parameters of MAP2 model, we proposed a method to predict the resource utilization probability density function $f_{MAP2}(x)$ for bursty workloads. For simplicity, we only take CPU as an example in this section, but the proposed methods can be applied to other kinds of system resources.

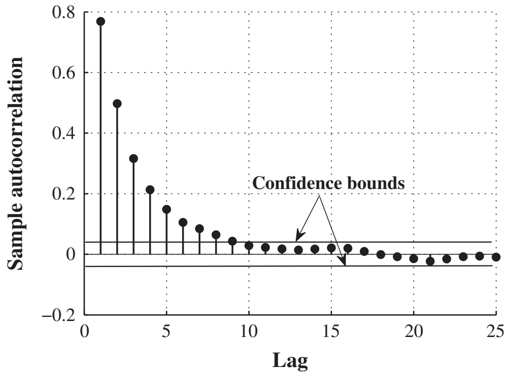


Fig. 3. Sample autocorrelation function.

3.1. CPU utilization prediction for non-bursty workloads

By analyzing the log files for lots of typical Web application benchmarks, we observed that the CPU utilization basically follows a normal distribution when the system is not saturated and there is no burstiness in the workload. But when the system is semi or full saturated, what the CPU utilization follows is not a normal distribution.

Definition 4 (*Non-saturated State*). In the observation period, there is no CPU utilization reaches 100%, i.e. $f(1)$ approaches to 0.

Definition 5 (*Partially-saturated State*). In the observation period, some CPU utilization reaches 100%, i.e. $f(1)$ is between 0 and 1.

Definition 6 (*Full-saturated State*). In the observation period, all the CPU utilization is 100%, i.e. $f(1)$ approaches to 1.

First, let's consider the case of Non-saturated State. Assuming the pdf of CPU as:

$$g(x) \sim N(\mu, \sigma^2), \tag{15}$$

where μ represents the average value of CPU utilization U_{avg}, and σ^2 is the variance. As the performance of closed system is determined by N and Z, we will use these two parameters to predict the parameters μ and σ for the pdf of CPU utilization.

Theorem 1. *The average CPU utilization U_{avg} is proportional to the value of $N/\lambda_{N,Z}$.*

Proof. From average resource utilization $U_{avg} = X_{N,Z}D$ ($X_{N,Z}$ represents the system throughput, D is the system service time), and $R_{N,Z} = N/X_{N,Z} - Z$ resulted by the response time law, we can derive:

$$U_{avg} = \frac{DN}{R_{N,Z} + Z}, \tag{16}$$

where $R_{N,Z}$ is the system response time and D is a constant.

Then according to (2), we can get:

$$U_{avg} = D\frac{N}{\lambda_{N,Z}}. \tag{17}$$

So U_{avg} is proportional to the value of $N/\lambda_{N,Z}$. $\square$

Thus, based on (16) and (17), we can use N and Z to predict the value of U_{avg} by using some liner regression methods. In the case of Non-saturated state, $\mu = U_{avg}$. In addition, by using the statical learning tool (such as Weka[3]), we observed the relation between μ and σ can be described as follows:

$$\sigma = p_1\mu^2 + p_2\mu + p_3, \tag{18}$$

where the value of p_1, p_2 and p_3 can be determined by polynomial regression methods. So the $g(x)$ can be determined by the N and Z.

Then, we will consider the case of Partially-saturated and Full-saturated state. If we directly use (17) and (18) to predict, we can get a normal distribution $g(x)$ (consists of S_2, S_3 and S_4), as described in Fig. 4. But, since CPU utilization is never

[3] http://www.cs.waikato.ac.nz/ml/weka/.

ARTICLE IN PRESS

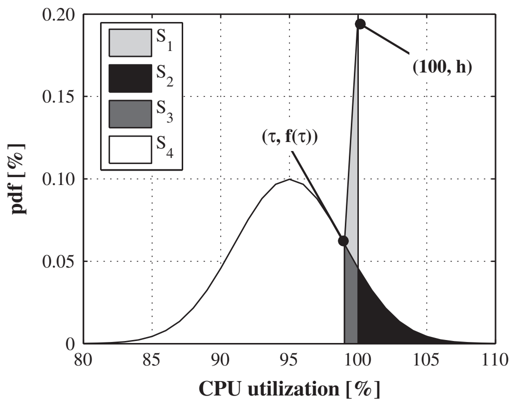


Fig. 4. The pdf of CPU utilization under semi saturation.

higher than 100%, we can use the keystone S_1 with the same proportion to replace S_2, i.e. the CPU utilization distribution consists of $S_1,\ S_3$ and S_4. Define τ as a threshold (such as 99%), i.e. the system is in the saturated state when CPU utilization is higher than this value. So the pdf of CPU utilization can be described as:

$$f(x) = \begin{cases} g(x), & \text{if } 0 \leqslant x \leqslant \tau \\ \frac{h-g(\tau)}{100-\tau}(x-\tau)+g(\tau), & \text{if } \tau < x \leqslant 100, \end{cases} \tag{19}$$

where h can be derived from following equations:

$$\begin{cases} 1 - S_4 & = S_1 + S_3, \\ S_4 & = \int_{x=1}^{\tau} g(x)dx, \\ S_1 + S_3 & = \frac{g(\tau)+h}{2}(100-\tau), \end{cases} \tag{20}$$

where $S_1,\ S_2,\ S_3$ and S_4 describes the proportion of each partition in Fig. 4 separately.

By using (19), we can model the pdf for the case of Non-saturated, Partially-saturated and Full-saturated states. And when the system is in the Non-saturated state,

$$\int_{x=0}^{\tau} f(x)dx = 1, \quad \int_{x=\tau}^{100} f(x)dx = 0, \tag{21}$$

when the case of Full-saturated state,

$$\int_{x=0}^{\tau} f(x)dx = 0, \quad \int_{x=\tau}^{100} f(x)dx = 1, \tag{22}$$

and when the system is in the state of Partially-saturated,

$$0 < \int_{x=0}^{\tau} f(x)dx < 1, \quad 0 < \int_{x=\tau}^{100} f(x)dx < 1. \tag{23}$$

3.2. CPU utilization prediction for bursty workloads

Since bursty workloads generated by MAP2 is a superposition of two non-bursty workloads with different $\langle Z, N\rangle$, the pdf of CPU utilization under bursty workloads can be regarded as a liner combination of two $f(x)$ with different expectations and variances, i.e. $f_b(x)$ and $f_n(x)$.

Theorem 2. *If the frequency of burstiness f keeps constant, CPU utilization μ is independent of the duration time for bursty state t_b.*

Proof. According to the definition of average request arrival rate, we have:

$$\lambda_{avg}^{-1} = \frac{P_{nb}\lambda_b^{-1} + P_{bn}\lambda_n^{-1}}{P_{nb} + P_{bn}}. \tag{24}$$

Combine with (2), we can drive the average system response time R_{avg} and average think time Z_{avg}:

$$R_{avg} = \frac{P_{nb}R_b + P_{bn}R_n}{P_{nb} + P_{bn}}, \qquad Z_{avg} = \frac{P_{nb}Z_b + P_{bn}Z_n}{P_{nb} + P_{bn}}. \tag{25}$$

Then based on (6)–(8), we can have:

$$R_{avg} = \frac{f\lambda_b R_b + (1-f)\lambda_n R_n}{f\lambda_b + (1-f)\lambda_n}, \tag{26}$$

$$Z_{avg} = \frac{f\lambda_b Z_b + (1-f)\lambda_n Z_n}{f\lambda_b + (1-f)\lambda_n}. \tag{27}$$

(26) and (27) show that if f keeps constant, R_{avg} and Z_{avg} is independent of t_b. Then, based on (16) and $\mu = U_{avg}$, we can see μ is independent of t_b. □

Since the bursty workloads generated by MAP2 can be determined by a six-tuple $\langle Z_b, Z_n, f, t_b, N, X_{max}\rangle$, we can use $\langle Z_b, N\rangle$ and $\langle Z_n, N\rangle$ to derive $\langle \mu_b, \sigma_b\rangle$ and $\langle \mu_n, \sigma_n\rangle$ separately, as described in previous subsection. Then $f_b(x)$ and $f_n(x)$ can be derived based on (15), (17)–(19). Additionally, X_{max} is a constant and t_b is independent of CPU utilization, so the distribution of CPU utilization under bursty workloads can be expressed as follows:

$$f(x) = f \cdot f_b(x) + (1-f)f_b(x). \tag{28}$$

4. Experiments

The testbed for our experiments is a cluster built by decades of Dell PowerEdge R710 servers. And we allocated six virtual machines (VMs) for Olio as shown in Fig. 5: two VMs for workload drivers, one for reverse proxy load balancer, one for application server, one for database, and one for Network File System (NFS). Each VM has 2 GB memory and 10 GB disk except that NFS has 80 GB disk. And the software configurations is listed in Table 2. With this topology, the bottleneck is the CPU utilization in application server as described in [32]. So only the CPU utilization metric in application server is shown in this paper for lack of space.

In this section, we present two experiments. One is used to validate the accuracy and efficiency of our bursty workload generation method and the other is used to analyze the proposed pdf prediction method for resource utilization under various kind of bursty workloads.

4.1. Analyzing bursty workload generation method

4.1.1. Accuracy validation

Firstly, We use 8 sets of inputs listed in Table 3 to generate bursty workloads. If not specified, the unit of time in Table 3 and below is second. As the bold face shows, for every two inputs, only one parameter is different. With these inputs, the target IDC I and average think time Z_{avg} can be calculated. Next, we use these I and Z_{avg} to generate bursty workloads with the method (labeled as Mi2009) suggested in [24] so that the accuracy of the two methods can be compared. For each experiment, the running time of Olio consists of a 5-minutes ramp-up, a 600-minutes run period, and a 2-minutes ramp-down, which is long enough for IDC estimation. Then, we collect the access logs of Nginx and estimate the actual IDC. At last, we compare the actual IDC generated by our method (labeled as MAP2_Generator) and the one generated by Mi2009 with the target I.

Fig. 6 shows the results of the accuracy validation experiments. From Fig. 6(a), we can see the actual IDC generated by our method is more closed to the target one. Moveover, the differences between target IDC and actual value of Mi2009's method increases as growing of IDC. But the difference between target IDC and ours is smaller and without evident variation when IDC increases. Fig. 6(b) depicts the lag values returned by Algorithm 3 in our method, which shows that the autocorrelation of random variable A tends to increase as IDC. Fig. 6(c) illustrates the relative error of I and Z_{avg}, which is defined as

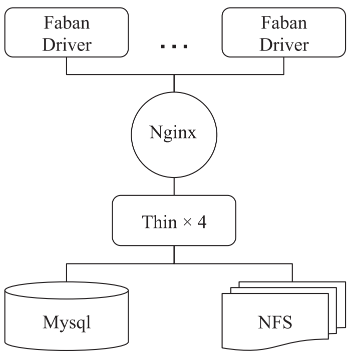


Fig. 5. Experiment setup.

ARTICLE IN PRESS

Table 2
Software configuration.

Type	Configuration
OS	Ubuntu 11.10 32 bit
VMM	Xen 3.1.2
JVM	jdk1.6.0_31
Driver	Faban-kit-121411
Load balancer	Nginx 1.0.5
App server	Ruby1.8, Rails 2.2, Thin 1.3
Database	Mysql 5.1.62
Monitor	Vmstat, Xenmon

Table 3
Experiment inputs for accuracy validation.

ID	Z_b	Z_n	f	t_b	bfI	Z_{avg}
T1	**0.5**	5	0.2	240	77	2.694
T2	**0.1**	5	0.2	240	132	2.156
T3	0.1	**5**	0.2	300	165	2.156
T4	0.1	**10**	0.2	300	256	2.741
T5	0.1	10	**0.1**	360	291	4.556
T6	0.1	10	**0.2**	360	307	2.741
T7	0.1	10	0.2	**420**	358	2.741
T8	0.1	10	0.2	**480**	409	2.741

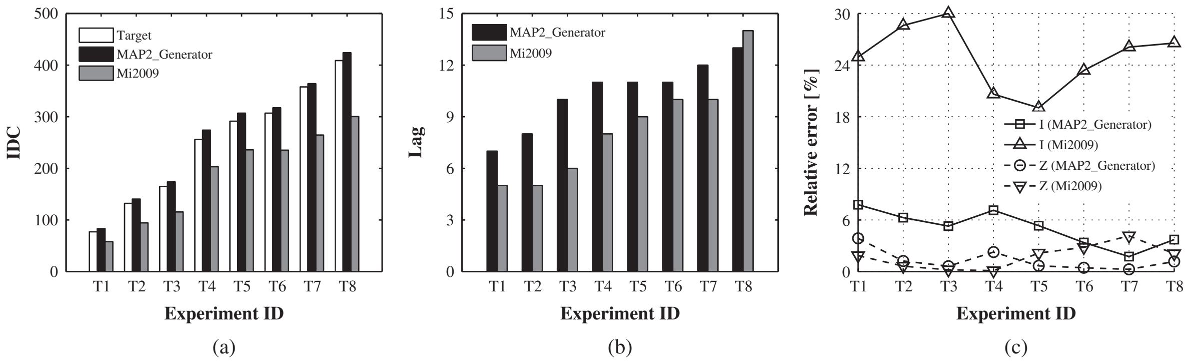


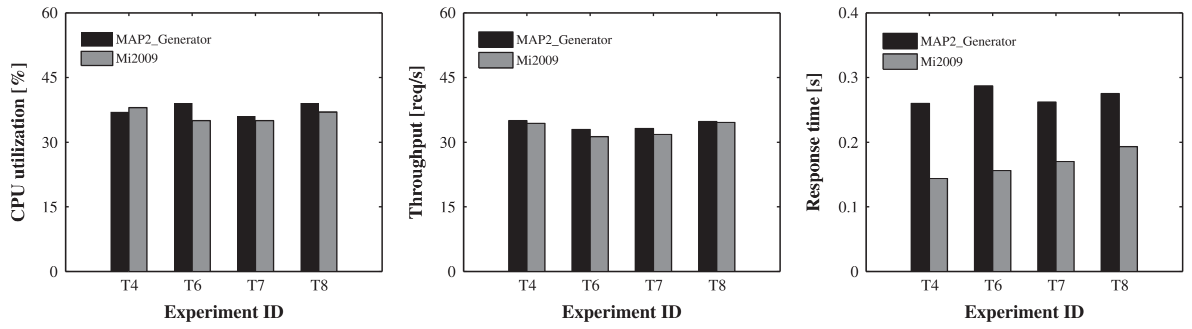
Fig. 6. Experiment results for accuracy validation.

$|1 - actual/target| \times 100\%$. From this figure we can see that for Mi2009's method, the relative error of I increases as growing of IDC when Z_{avg} is constant (see T4, T6, T7 and T8). Furthermore, the smaller the value of Z_{avg}, the larger the relative error of I (compare T1 and T2, T3 and T4, T5 and T6). The main reason can be explained as follows: the weight of bursty state (λ_b) increases as growing of IDC or decreasing of Z_{avg}, and for Mi2009's method, they use $1/Z_b$ to approximate the value of λ_b. Though this kind of approximation may be accurate when the system is in normal sate, but for bursty state, it is inaccurate. And this inaccuracy will be amplified when λ_b increases. So the relative error of I increases as IDC increasing or Z_{avg} decreasing. But for our method, the relative error of I is totally correlated with the relative error of Z_{avg}. That means the accuracy of I is dependent on the accuracy of Z_{avg}. In our method, the random function is used when generating the think time, so the deviation is inevitable. However, the relative error of I in our method is much lower than Mi2009's. All these relative errors are smaller than 10%, and they show no evident increasing trend as IDC increases.

4.1.2. Burstiness and performance

The purpose of generating synthetic bursty workload is to study its impact on system performance. [24] has shown that the system performance will deteriorate as I increases (Z_{avg} remains the same). Firstly, we plot the average CPU utilization, throughput and response time of application server in experiments T4, T6, T7, and T8, which have the same Z_{avg} and increased IDC, in Fig. 7. The CPU utilization and throughput are almost the same for both methods. Actually, the distributions of CPU are also almost the same, which are not shown for lack of space. But the response times are different (see Fig. 7(c)). For our method, the response times are longer and do not grow up as IDC. While for Mi2009's method, the response times increase as IDC. These differences result from two reasons. The first is that the values of actual IDC generated by our method are larger, which cause longer response times. And the second is the values of IDC are increased by different ways. As shown

ARTICLE IN PRESS



(a) Average CPU utilization. (b) System throughput. (c) System response time.

Fig. 7. System performance of T4, T6, T7, and T8.

in Table 3, we increase the values of IDC by enlarging the values of t_b. And (12) has already shown that the response time will not be effected by t_b if f is not changed. In contrast, Mi2009's method increase the values of IDC by adjusting P_{bn} and P_{nb} iteratively. Therefore, with our method, we can find and prove the first interesting phenomenon: *the system performance will not be effected by changing the value of IDC, if only the value of t_b is modified*. And for these cases, we can say that the intensions of burstiness are actually the same even the values of IDC are different. So Mi2009's method which only uses the IDC to denote and generate different intensions of burstiness is not accurate enough.

Next, we want to check if the same values of IDC may cause different performance. By carefully preparing the parameters, we construct some new sets of inputs for our method to generate bursty workloads for performance analysis as shown in Table 4. The T9 is a non-bursty workload ($I = 1$), which is used as a baseline. T10 and T11 have the same values of I, Z_{avg} and t_b. And T12 increases the value of I without changing Z_{avg} and t_b. Table 4 also shows some high-level (average) performance metrics of these four experiments including the system throughput X_{avg} (req/s), response time R_{avg} (s), CPU utilization U_{avg} of application server, allocation time per execution (ms/exc) and execution counts per second (exc/s). The last three metrics are collected by Xenmon. From the results, we find another two interesting phenomenons: First, *the same IDC values can cause different R_{avg}, U_{avg}, execution counts and allocation time (see T10 and T11)*; Second, *R_{avg} grows up as IDC increases, while the system resource utilization goes down (see T9, T10, and T12)*.

To explain these two phenomenons, we explore the fine-grained performance metrics as shown in Fig. 8. For the non-bursty workloads (T9), the probability mass function (pmf) of CPU utilization looks like normal distribution (Fig. 8(a)). And the variances of allocation time and execution counts are low (Fig. 8(b) and (c)). In addition, the pmfs of CPU utilization caused by bursty workloads all have two peaks, each of which corresponds one state of MAP2 (Fig. 8(d), (g) and (j)). For the bursty workloads with same values of IDC (T10 and T11), the pmfs of CPU utilization are totally different. This phenomenon can be explained by the great difference of allocation time and execution counts between T10 and T11. Though the average execution times of T10 (1313) and T11 (1358) is near, the variances of T11 is much bigger than that of T10. For allocation time, the average value (0.506 for T10 but 9.621 for T11) already has much difference and the variance of T11 is also bigger than T10. So in this case, we think the intension of burstiness for T11 is higher than T10 in a sense, and the U_{avg} and R_{avg} are different under the two workloads with same IDC values, which is the first interesting phenomenon.

The second interesting phenomenon describes that as the growing of the intension of burstiness (T9, T10 and T12), the average of response time increases (0.056, 0.076 and 0.222 for T9, T10 and T12 separately), but the average CPU utilization decreases (48.69%, 44.48% and 31.92% for T9, T10 and T12 separately). Intuitively, it seems impossible, since we often regard low utilization as low workload and further low response time. The factual observation from our experiments is totally different from what we thought before, why? In fact, as growing of the intension of burstiness, the proportion of CPU utilization around 100% becomes larger (comparing Fig. 8(g) with Fig. 8(j)). If we only consider the average performance metrics, such as CPU utilization here, we cannot find this partially saturated phenomenon, which is actually the reason for this surprising phenomenon. Furthermore, the allocation time and execution counts also confirm such phenomenon. The allocation time and execution counts both have higher and lower values corresponding to the two states of MAP2. When the intension of burstiness is high, the proportion of longer allocation time per execution (about 1000 ms/exc) is big (see Fig. 8(h) and

Table 4
Experiment inputs for burstiness and performance analysis.

ID	Z_b	Z_n	f	t_b	I	Z_{avg}	X_{avg}	R_{avg}	U_{avg}	Allocation	Execution/s
T9	–	–	–	–	1	4.556	21.59	0.056	48.69	0.379	1347
T10	2.0	10.00	0.314	480	89	4.556	21.90	0.076	44.48	0.506	1313
T11	0.5	5.55	0.050	480	89	4.556	22.26	0.129	42.81	9.621	1358
T12	0.1	10.00	0.100	480	388	4.556	20.91	0.222	31.92	37.996	822

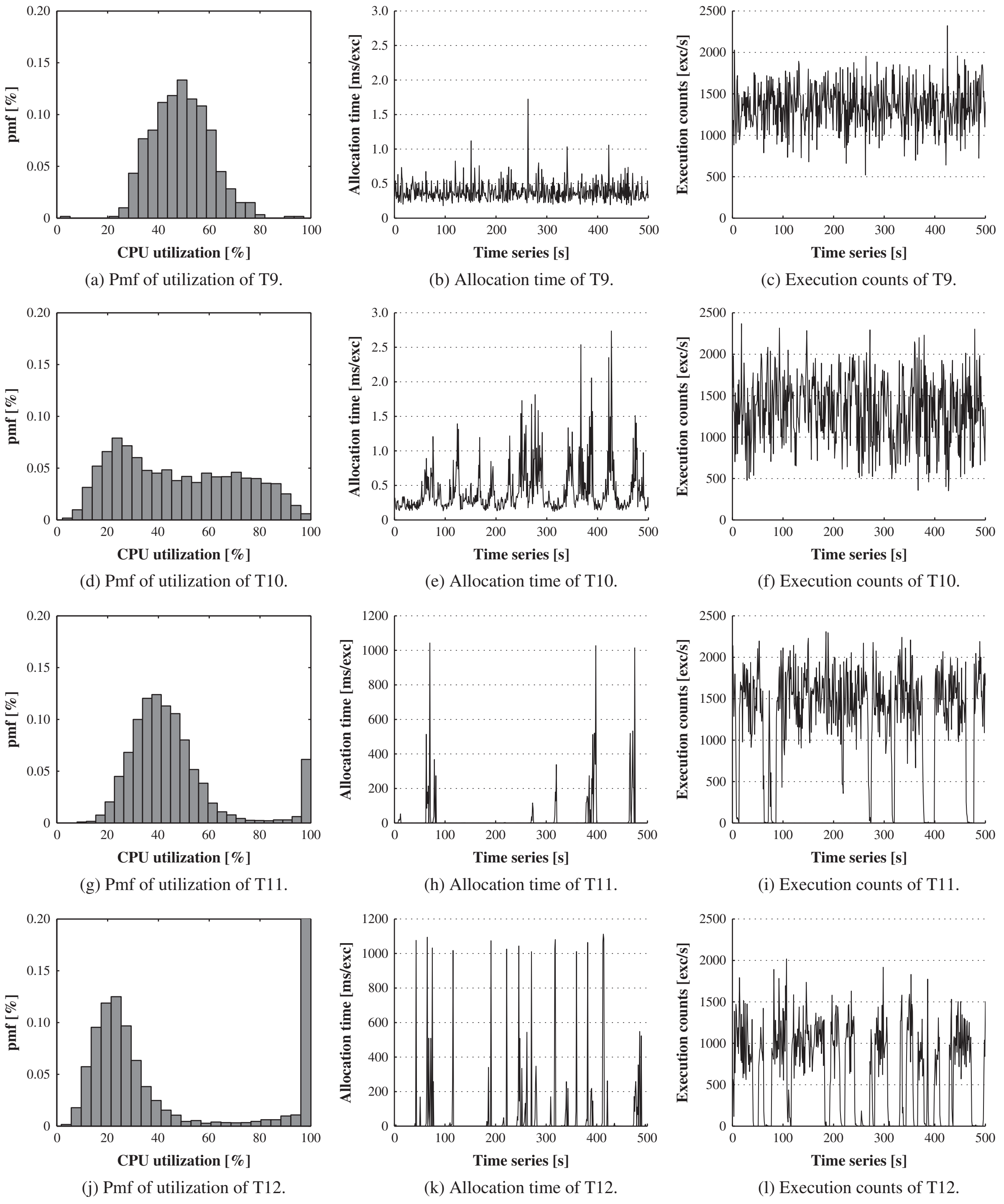


Fig. 8. Burstiness and system performance.

(k)). It shows that Xen VMM can automatically aggregate more CPU works in a execution period for bursty workloads, which will decrease the execution counts per second and reduce switch overhead. This strategy makes CPU more efficient, so from a macro point of view, the average CPU utilization decreases as growing of the burstiness of workloads. However, aggregating more CPU works in an execution period for bursty workloads keeps the I/O operations waiting for longer time, which will increase the response time. So R_{avg} grows up as the increasing of IDC, while the system resource utilization goes down.

Above experiment results particularly the interesting phenomenons show significance of fine-grained performance analysis. Performance evaluation based on mean value analysis often brings rough results or general trends. It is also motivated by the experiment results that we proposed the fine-grained resource utilization prediction method which is described before. It is worthwhile to note that the core idea of the proposed resource utilization prediction method also comes from above analysis of the experiment results. From the pmf of CPU utilization described in Fig. 8, we can see for non-bursty workloads, the pmf of CPU utilization looks like normal distribution, and for bursty workloads, there are two peaks, each of which corresponds to one state of MAP2. So we use this intuitive and accurate (demonstrated later) observation to model and predict the CPU distribution in our fine-grained performance analysis method. More details about the analysis of the proposed resource utilization prediction method are described below.

4.2. Analyzing system resource utilization prediction method

4.2.1. Effectiveness validation

In order to validate the effectiveness of our resource utilization prediction method, we construct six group of experiments (as Table 5 shows). The first group E1, which can be regarded as a baseline, describes the case of non-bursty workloads. From E2 to E5, we changed the values of Z_b and f to increase the intension of burstiness (I is increasing gradually). And E6 changes the value of I by increasing t_b. For comparability, the number of users N and the average think time Z_{avg} are the same for all the experiment groups.

Table 5 also shows the average experiment results, including the average system throughput X_{avg}, average system response time R_{avg} and average CPU utilization U_{avg}. We plot the probability distribution of CPU utilization for each experiment in Fig. 9. In this figure, the histogram describes the pmf of CPU utilization collected by Xenmon, while the curve denotes the continuous pdf derived by our prediction method ($\tau = 99\%$). Fig. 9(a) describes the distribution of CPU for non-bursty workloads, the prediction result is almost the same as the actual value, and both of them follow a normal distribution. Fig. 9(b)–(f) describes the distribution for the cases of bursty workloads. Since the value of Z_b is larger than others in E2 and E3, which brings smaller weight of bursty state (smaller λ_b), the system is in Partially-saturated state under these bursty workloads. So we can observe two peaks and a line near 100% in Figs. 9(b) and (c). Due to μ_b and μ_n resulted by Z_b and Z_n separately is near, the large number of switches between the two states result in large pmf for the CPU utilization between the two states in this partially saturated state. So the relative prediction error is big. However, as the decreasing of Z_b, the CPU utilization under bursty workloads tends to be Full-saturated. So there is only one peak (caused by Z_n) and a line (CPU utilization approaches to 100% under bursty workloads) in Fig. 9(d)–(f). In these cases, the predictions are more accurate. Because in these cases, the values of pmf for CPU utilization between the two states are approaching to zero, it's more accurate to predict a shape superposed by a normal distribution and a line than predicting the one superposed by two normal distributions and a line. Furthermore, the probability distribution is almost the same in Fig. 9(e) and (f), which again proves the conclusion of Theorem 2. Thus, Fig. 9 demonstrates the effectiveness of our fine-grained resource utilization prediction method, both in the case of non-bursty and bursty workloads.

4.2.2. Accuracy analysis

From Fig. 9 we can see that different parameters result in different prediction accuracy. It is meaningless to evaluate the prediction accuracy on certain configuration and it is also impractical to evaluate our method by enumerating all the possible cases. So we show the average prediction accuracy resulted by different parameters in this section. The average prediction accuracy is modeled by the Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{[y_i - f(x_i)]^2}{n-1}}, \quad i = 1, 2, \ldots, n, \tag{29}$$

where y_i is the pdf calculated by the actual experiments and $f(x_i)$ is the prediction value.

First, we analyze the prediction accuracy under non-bursty workloads. Different think time often means different request arrival rate, which may bring different CPU utilizing patterns and further impact the prediction accuracy of our method. Here we use two experiment groups with different values of Z to analyze the prediction accuracy, one group with short think time (smaller than 1 seconds), the other with long think time (approximately 10 seconds). In most web application systems, the

Table 5
Experiment inputs for effectiveness validation.

ID	Z_b/s	Z_n/s	f	t_b/s	N	I	Z_{avg}/s	$X_{avg}/(req \cdot s^{-1})$	R_{avg}/s	$U_{avg}/\%$
E1	–	–	–	–	100	1	4	24.60	0.071	59
E2	**2.0**	10	**0.3914**	480	100	77	4	25.25	0.085	52
E3	**1.0**	10	**0.2378**	480	100	170	4	25.28	0.115	46
E4	**0.5**	10	**0.1703**	480	100	270	4	25.30	0.186	44
E5	**0.1**	10	**0.1228**	480	100	406	4	25.92	0.289	38
E6	0.1	10	0.1228	**600**	100	507	4	23.73	0.311	38

ARTICLE IN PRESS

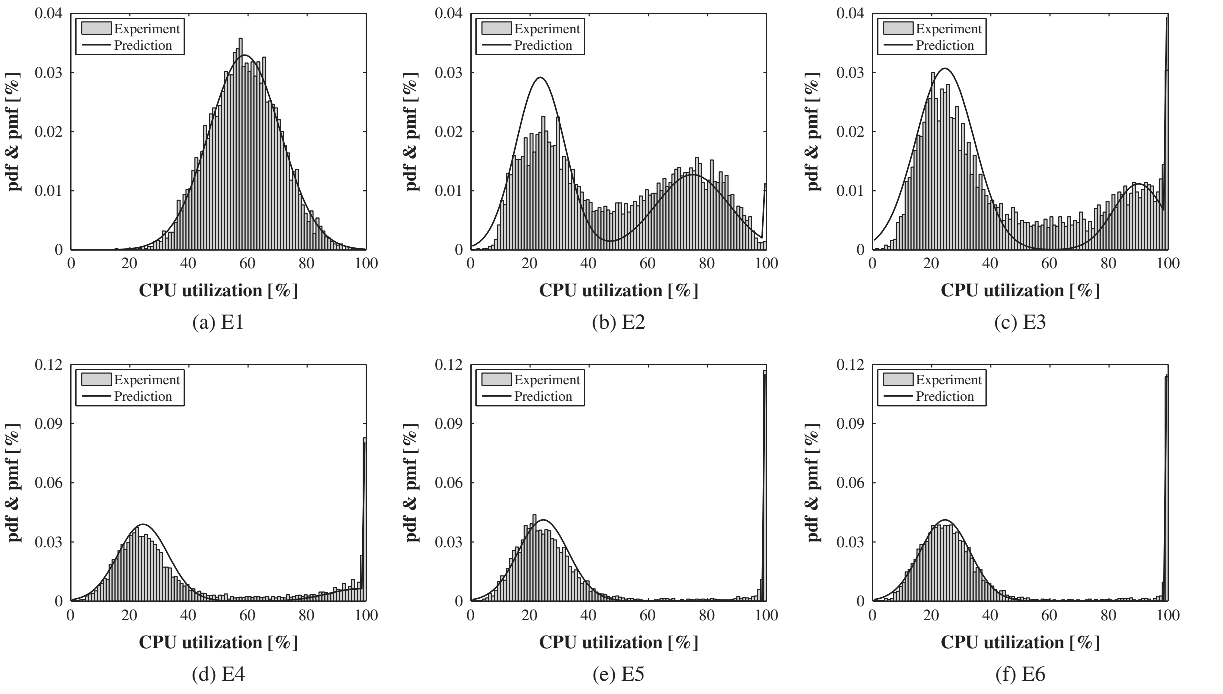


Fig. 9. The distribution of CPU utilization.

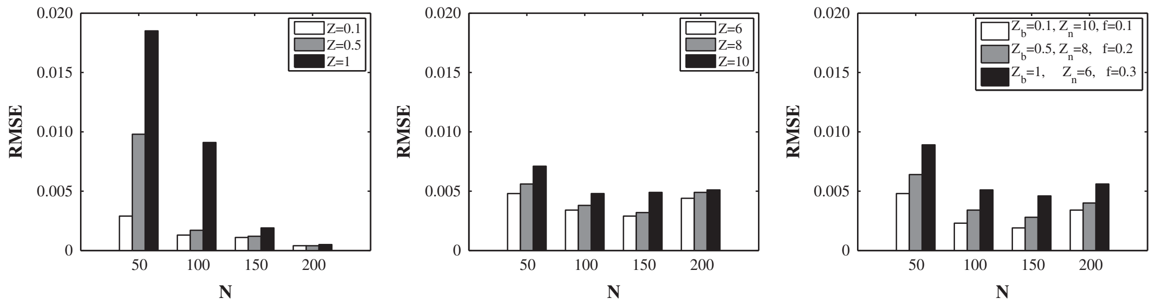
think time falls in this range. The results for both two groups are described in Fig. 10(a) and (b). Fig. 10(a) described the RMSE for the group of short think time. In this group, when the workload is low, i.e. the number of users N is small and the think time Z is large (such as $N = 50,\ Z = 0.5$ or 1 and $N = 100,\ Z = 0.5$), the system is in the state of Partially-saturated, so the prediction error is relatively large. However, when the workload is high, i.e. N is large and Z is small, the system is in Full-saturated state, and the prediction error is smallest. The reason can be explained as follows: when the system is in the state of Full-saturated, nearly all the values of CPU utilization are 100%, and the distribution of CPU utilization is a line at 100%, so the prediction error is smallest. However, if the system is Partially-saturated, the pmf of CPU utilization is neither a complete normal distribution nor a line at 100%. Instead, it is a superposition of a partial normal distribution and a line at 100% (due to space limitation, the pmf of CPU utilization is omitted here), so the prediction error is larger than that of the Full-saturated state. On the other hand, for the group of long think time, as described in Fig. 10(b), the prediction error is small and keeps little fluctuation, even under different values of N. The main reason is that these long think times bring slight workloads so that all the systems are in the state of Non-saturated. That means the pmf of CPU utilization follows a normal distribution in all these cases. So their prediction error is small and without much fluctuation.

Then, we will analyze the prediction accuracy of our method under bursty workloads, which are resulted by the combinations of long and short think times with different values of f. Since in the case of bursty workloads, the prediction function is a liner combination of the two pdf functions under non-bursty workloads, the prediction error is between long and short think time's. Fig. 10(c) confirms the conclusion. In all the cases with different values of Z and N, the prediction errors of pmf of CPU utilization described in Fig. 10(c) is between that of Fig. 10(a) and (b). Furthermore, it's important to note that the prediction errors of our method are acceptable (below 20%) in all the cases, including non-bursty or bursty workloads, with long or short think times, or with different number of users. So this experiment demonstrates the accuracy of our method.

4.2.3. Optimized resource configuration

As described before, the prediction of resource utilization often aims to achieve optimized resource configuration and then support better management or scheduling decision making. The proposed prediction method can be used in various of fields, including system performance tuning and optimization, application QoS assurance, cloud management (including virtual machine migration, consolidation, etc.) decision support, and so on. Here we use a case study to show what kinds of benefits our prediction method can gain. This case study is about how our prediction method can be used to find a optimal resource configuration for each virtual machine during the server consolidation process. From our prediction method, we can obtain the pdf of CPU utilization. Then we can easily get the average value U_{avg}, maximum value U_{max} or the percentile value of CPU U_{90th}, where U_{90th} is defined as:

$$U_{90th} = min\left\{\forall U \in [0,1] | \int_{x=0}^{U} f(x)dx \geqslant 90\%\right\}. \tag{30}$$



(a) Short think time, non-bursty workload.

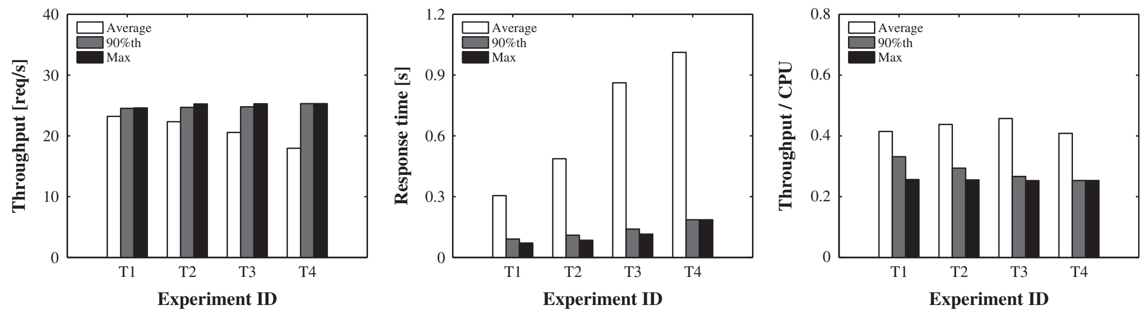
(b) Long thinkt ime, non-bursty workload.

(c) Bursty workload.

Fig. 10. Accuracy of prediction.

Table 6
CPU data resulted from prediction.

ID	U_{avg}/%	U_{90th}/%	U_{max}/%
E1	59	74	96
E2	52	84	99
E3	46	93	100
E4	44	100	100



(a) System throughput.

(b) System response time.

(c) Efficiency.

Fig. 11. System performance for different CPU configurations.

Table 6 describes the three kinds of CPU values derived by our prediction method, for experiment group E1 to E4. Then, we use them to set the Cap parameter (the maximum amount of CPU that a domain can be able to consume) for Xen virtual machine, and the performance results are showed in Fig. 11. Fig. 11(a) and (b) describe the system throughput and response time separately, and Fig. 11(c) shows the efficiency, which is defined as the ratio between the throughput and the Cap value. From Fig. 11(c), we can see the efficiency of the configuration by U_{avg} is higher than U_{90th} and U_{max}, but the response time of U_{avg} is much higher than U_{90th} and U_{max}, and the throughput of U_{avg} is also lower than U_{90th} and U_{max}, which means U_{avg} has the worst performance. Furthermore, with the increasing of the intension of burstiness, U_{avg} also shows worse and worse performance (decreasing throughput and increasing response time). For the configuration by U_{max}, it shows high throughput and low response time, but its efficiency is much lower than the configuration by U_{avg}. So U_{avg} and U_{max} often means two extreme values for this kind of configuration. However, the configuration by U_{90th} makes a balance between the two extreme values. As Fig. 11 shows, the throughput and response time of the configuration by U_{90th} is nearly the same as that by U_{max}, and the efficiency is higher than that by U_{max} particularly for the case of low intension of burstiness. Furthermore, with the increasing of the intension of burstiness, U_{90th} can still ensure high throughput and short response time, just like U_{max}. Therefore, as we can find that simply using the value of U_{avg} or U_{max} to set the Cap parameters during server consolidation generally cannot get a optimal solution. Instead, using U_{90th} or similar percentile values can adjust the value of CPU to find a balance between system throughput and response time, even in the cases of different burstiness intensions. It is the resource utilization pdf prediction method that can be used to calculate these percentile values. And it is also one of the most important motivations why we focus on the fine-grained resource utilization prediction.

In addition, from the experiment results, we know there should be enough reserved resources when consolidating the servers under bursty workloads, even the average utilization is not high (but the fine-grained utilization distribution may show high fluctuation). Otherwise, the performance could be easily deteriorated. [37] suggested that for satisfying QoS for computing intensive applications, the average CPU utilization should be kept no higher than 70%. This suggestion is also verified by our experiment, as the experiment result of group E1 shows, when the average utilization U_{avg} is about 59%, the maximum utilization U_{max} has already approached to 100%. If we decrease the Cap value of CPU again, the system may enter the partially-saturated state, and the performance may be deteriorated. In fact, this deterioration will be even worse when the workloads are busty (like E2 to E6 shows). So the proposed fine-grained resource utilization prediction method is significant to optimized resource configuration, for achieving better upper management decision making.

5. Related work

Cloud computing has become popular and attracted considerable research attention due to the maturity of related technologies such as network devices, software applications and hardware capacities [16]. But only a small portion of the work done so far has addressed performance issues [19]. In this section, we review related work from three aspects: performance analysis and prediction of clouds, burstiness modeling and analysis, and workload generation.

5.1. Performance analysis and prediction of clouds

The approaches of performance analysis and prediction often include analytical models (Petri Net, Markov model, Queueing Network model, etc.), simulation, benchmarks, machine learning and statistical learning methods, etc. Analytical models and simulations have been widely used to model and predict performance for a variety of systems, including monolithic operating systems [31], I/O subsystems [12], disk arrays [35] and static web services [7]. Standard benchmarks have also been widely used to evaluate the performance for many kinds of systems, e.g., for cloud computing, Napper et. al. examined the performance of Linpack benchmarks on different Amazon EC2 instance types [28]. Recently, performance prediction methods based on machine learning or statistical learning gain more and more attention, especially in the cloud computing community. The work [8] introduced a new learning-based solution for portable database workload performance prediction in the cloud. Several researchers utilized statistical techniques to build probabilistic models of workload and performance, for achieving better data-driven performance analysis and prediction. In [11], the authors developed a relationship model between SQL query statements and system performance based on kernel canonical correlation analysis (KCCA), in order to predict the performance of a modern parallel data base. Our method for performance analysis and prediction can be regarded as a combination of above methods. It used some statistical learning techniques (such as statistical learning tool Weka), Markov Arrival Model (for bursty workload generation), and some simulations to analyze and predict system performance. Our method is also evaluated in the cloud benchmark Cloudstone.

Moreover, none of above methods considered performance prediction under bursty workloads or fine-grained prediction for distribution of system resource utilization. Our performance analysis and prediction method can be used to evaluate system performance under bursty workloads, which have been observed in various kinds of systems and may cause critical impacts on performance of cloud based applications. More importantly, the proposed performance analysis and prediction method is fine-grained. It can be used to predict the pdf distribution of CPU utilization for both non-bursty and bursty workloads. Compared with traditional prediction methods based on mean value analysis, the fine-grained prediction results can be used as basis for better decision making and performance optimization, such as guiding server consolidation process for large-scale cloud computing platforms and cloud service level planning under burstiness [41]. Though studies have tried to generate multiple predictions [26,36], they are still point value predictions and it is difficult to determine which point would be correct. Gunho Lee proposed a method in his doctoral dissertation to predict the probability distribution of a MapReduce job completion time [21]. It enables a user to predict the distribution and to make sophisticated decisions that are difficult when using a point-value prediction. We also use the distribution prediction technique to evaluate cloud system performance. But we pay more attention on main characteristics of workloads, i.e. considering the case of bursty workloads when predicting such distributions. As far as we know, we are the first to provide fine-grained distribution predictions for bursty workloads.

5.2. Burstiness modeling and analysis

As a key characteristic of workload, burstiness has been first observed in networks [6], and then has been reported in a variety of settings [6,29,23,33]. Many mathematical methods have been proposed to characterize the intension of burstiness, including Self-similarity [22,6], Peakedness [9], Peak-to-mean Ratio, Coefficient of Variation, and Indices of Dispersion [14,4]. Self-similarity and Peakedness functions often focus on server-side, Indices of Dispersion, which characterizes the behavior in client-side, is more appropriate for workload generation, especially for closed system models. The work [4] showed that using Indices of Dispersion the accuracy of model prediction can be increased up to 30% compared with the model only parameterized by mean service demands. Thus, we used Indices of Dispersion for Count (one specific kind of Indices of Dispersion) to characterize intensions of burstiness and generate bursty workloads in this work. Though the indicator function for burstiness intension is same as [4], the main concerns of our method and [4] are different. In [4], they mainly focused on index of dispersion of the service process at a server. But our method aimed to capture the burstiness behavior of request arrival process by counting the number of requests in each interval. Furthermore, the used models on how to generate workloads satisfying required burstiness intension (specified by the value of Indices of Dispersion) are also different in the two methods.

In addition to burstiness characterization and modeling, the impact of burstiness on system performance has been examined and reported in [10,25]. According to the reports, burstiness often results in unpredictability of system performance. So there has been a recent spur of research activity in analysis, management and scheduling under bursty workloads. The work [39] developed a novel analytical model for communication networks in the presence of the Spatio-Temporal bursty traffic. [5] defined and studied a new class of capacity planning models called MAP queueing networks to describe and predict the performance of complex systems operating under bursty workloads. [30] presented RainMon, a novel end-to-end approach for mining bursty timeseries monitoring data designed to handle its size and unique characteristics. Compared with these works, ours is different in target and scope: we mainly focused on fine-grained system resource utilization analysis and prediction for cloud based applications under bursty workloads. But [39] aimed to model and analyze performance of communication networks; [5] paid more attention on capacity planning; [30] intended to provide tools for busty timeseries monitoring data analysis. In addition, the work [33] proposed an adaptive resource allocation algorithms to balance bursty workloads across all computing sites. [3] presented an approach for adaptive rate allocation in distributed stream processing systems under bursty workloads. [1] developed an autonomous elasticity controller that changes the number of virtual machines based on both monitored load changes and predictions of future system performance. For this kind of works, our bursty workload generator can be used to help them evaluate their schedulers' performance under bursty workloads. And then the fine-grained resource utilization prediction results can be used as complementary data to support their analysis and scheduling decision making.

5.3. Workload generation

Workload awareness is the premise and basis for system performance analysis and prediction. Workload, particularly the network traffic often shows non-linear, non-stationary and complex dynamical characteristics [40]. Though workload characterization, which aims to derive a model able to show, capture, and reproduce the behavior of the workload by using some quantitative parameters and functions, has been studied for nearly four decades, it's still challenging to characterize the workload accurately due to the increasingly complex system and unpredictable user behavior. Even after accurately characterizing the workload, it is more difficult than it seems to generate realistic workloads, e.g. [15] found that Web workload generation is more difficult than it seems. Too many features of workload generation infrastructure, such as software and hardware bottlenecks, can result in unrealistic workloads being generated or inaccurate measurements being recorded [15]. In the literature, researchers have described and evaluated many techniques for generating synthetic block-level I/O workloads [38], file-level workloads [34] and application-level I/O workloads [18]. There are also serval traditional Web workload generators such as Surge [2] and Httperf [27]. Surge can create a trace of HTTP requests with controlled distributions of many aspects including file size, request size and think time, etc. Httperf is a flexible and extensible HTTP workload generator. It provides a flexible facility for generating various HTTP workloads and for measuring server performance.

However, none of these generators can support bursty workload generation. In order to fill this gap, researchers developed two generators: Geist [17] and SWAT [20]. Geist, which focused on characteristics of bursty traffic, can only control the distribution and correlation of inter-arrival times for open systems. In addition, Geist used self-similarity, which means there is burstiness over all or a wide range of time-scales, to denote the intension of burstiness. Instead, we were more concerned about the burstiness on smaller request-level time-scale and we used IDC to characterize the burstiness intension. SWAT that was built on top of Httperf introduces burstiness by using a highly variable session length and think time distribution in the session mode. But there is no intuitive parameter for SWAT to explicitly specify the intension of burstiness, such as Peakedness and IDC. Moreover, none of SWAT and Geist provide a way to generate controllable and accurate burstiness intension. [24] was most similar to our work on bursty workload generation. The approach of [24] can be used to generate bursty workloads for closed systems by using IDC, average think time and population of clients as inputs of MAP2. But in practice, it is difficult for users to provide the 'magic' value of IDC. And the request arrival rate in bursty state was approximated inappropriately so that it often resulted in low accuracy. The model of our generator is simpler and easier to use. With some intelligible parameters that can be straightforwardly derived from real system logs or provided by performance analysts, our method can generate workloads with different intension of burstiness as specified.

6. Conclusions

The study of could computing has attracted considerable research attention in Internet system community recent years, both in industry and academia. However, due to the constantly expanding scale and unpredictability of cloud systems, only a small portion of the work done so far has addressed performance issues. Performance analysis and prediction, which is an important step for system performance optimization, needs a solid understanding of its workload. Burstiness is a salient characteristic of workload and has been observed in various kinds of systems, e.g. cloud computing systems. It has been reported to have critical impact on resource provisioning and performance of cloud, since bursty workload often results in unpredictability of system performance. Though various of techniques and tools have been developed for synthetic workload generation, few of them can easily generate bursty workloads for cloud based applications. Moreover, no fine-grained performance analysis and prediction method exists so far for bursty workloads. So it is important yet challenging to analyze and predict performance for cloud based applications under bursty workloads. In this paper, a bursty workload generator is developed for cloud benchmark Cloudstone based on 2-state Markovian Arrival Process. And then based on this generator, a fine-grained performance analysis method is proposed for cloud based applications. This method can be used to predict the probability density function of system resource utilization such as CPU utilization, to support better resource provisioning and scheduling decision making, or system performance optimization. Finally, extensive experiments are conducted in a real Xen-based test bed to evaluate the two methods. The results show the effectiveness and accuracy of our methods.

In this work, we mainly focus on bursty workload generation and fine-grained CPU utilization prediction for cloud based applications under such bursty workloads. Many further works on these two aspects can be done in the future. On bursty workload generation, the generation accuracy and efficiency, especially for the large-scale production systems, need to be evaluated in more detail. Appropriate techniques are also needed to improve the accuracy and efficiency of generation. In addition, intelligent management and scheduling under bursty workloads, for both cloud based applications and the cloud platform itself, is one of the interesting works on this topic. Performance analysis and prediction is the premise and basis for system resource provisioning and scheduling decision making. The proposed work is just our first step in this direction. Our future work on this topic includes optimizing cloud based applications' performance by using fine-grained system performance predictions, and the targeted adjustments of system configuration. On the other hand, we intend to improve the efficiency of cloud platforms. After knowing more details (by fine-grained analysis and prediction) of all the applications running on the platform, we can do more intelligent consolidations among these applications to reduce the operation costs and improve the resource utilization efficiency assuring the QoS requirements for each application.

Acknowledgements

This work was supported by National Natural Science Foundation of China under Grant (No. 61272129), National High-Tech Research Program of China (No. 2013AA01A213), NewCentury Excellent Talents Program by Ministry of Education of China (No. NCET-12-0491), Zhejiang Provincial Natural Science Foundation of China (LR13F020002), and Science and technology Program of Zhejiang Province (No. 2012C01037-1).

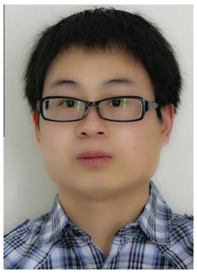
References

[1] A. Ali-Eldin, M. Kihl, J. Tordsson, E. Elmroth, Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control, in: Proceedings of the 3rd Workshop on Scientific Cloud Computing Date, ACM, 2012, pp. 31–40.
[2] P. Barford, M. Crovella, Generating representative web workloads for network and server performance evaluation, SIGMETRICS Perform. Eval. Rev. 26 (1998) 151–160.
[3] I. Boutsis, V. Kalogeraki, Radar: adaptive rate allocation in distributed stream processing systems under bursty workloads, in: IEEE 31st Symposium on Reliable Distributed Systems (SRDS), IEEE, 2012, pp. 285–290.
[4] G. Casale, N. Mi, L. Cherkasova, E. Smirni, How to parameterize models with bursty workloads, ACM SIGMETRICS Perform. Eval. Rev. 36 (2008) 38–44.
[5] G. Casale, N. Mi, E. Smirni, Model-driven system capacity planning under workload burstiness, IEEE Trans. Comput. 59 (2010) 66–80.
[6] M.E. Crovella, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, IEEE/ACM Trans. Netw. 5 (1997) 835–846.
[7] R.P. Doyle, J.S. Chase, O.M. Asad, W. Jin, A. Vahdat, Model-based resource provisioning in a web service utility, in: USENIX Symposium on Internet Technologies and Systems, vol. 4, 2003, pp. 5–5.
[8] J. Duggan, Y. Chi, H. Hacigumus, S. Zhu, U. Cetintemel, Packing light: portable workload performance prediction for the cloud, in: IEEE 29th International Conference on Data Engineering Workshops (ICDEW), IEEE, 2013, pp. 258–265.
[9] A.E. Eckberg, Approximations for bursty (and smoothed) arrival queueing delays based on generalized peakedness, in: Proceedings of the 11th International Teletraffic Congress, 1985, pp. 331–335.
[10] A. Erramilli, O. Narayan, W. Willinger, Experimental queueing analysis with long-range dependent packet traffic, IEEE/ACM Trans. Netw. (TON) 4 (1996) 209–223.
[11] A. Ganapathi, H. Kuno, U. Dayal, J.L. Wiener, A. Fox, M. Jordan, D. Patterson, Predicting multiple metrics for queries: better decisions enabled by machine learning, in: IEEE 25th International Conference on Data Engineering, ICDE'09, IEEE, 2009, pp. 592–603.
[12] G.R. Ganger, Y.N. Patt, Using system-level models to evaluate i/o subsystem designs, IEEE Trans. Comput. 47 (1998) 667–678.
[13] K. Gilly, C. Juiz, N. Thomas, R. Puigjaner, Adaptive admission control algorithm in a QoS-aware web system, Inf. Sci. 199 (2012) 58–77.
[14] R. Gusella, Characterizing the variability of arrival processes with indices of dispersion, IEEE J. Sel. Areas Commun. 9 (1990) 203–211.
[15] R. Hashemian, D. Krishnamurthy, M. Arlitt, Web workload generation challenges–an empirical investigation, Softw.: Pract. Exper. 42 (2012) 629–647.
[16] C.H. Hsu, K.D. Slagter, S.C. Chen, Y.C. Chung, Optimizing energy consumption with task consolidation in clouds, Inf. Sci. 258 (2014) 452–462.
[17] K. Kant, V. Tewari, R. Iyer, Geist: Generator of e-commerce and internet server traffic, in: Proc. of Int. Symposium on Performance Analysis of Systems and Software, 2001.
[18] W.I. Kao, R.K. Iyer, A user-oriented synthetic workload generator, in: Proceedings of the 12th International Conference on Distributed Computing Systems, IEEE, 1992, pp. 270–277.
[19] H. Khazaei, J. Misic, V.B. Misic, Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems, IEEE Trans. Parall. Distrib. Syst. 23 (2012) 936–943.
[20] D. Krishnamurthy, J.A. Rolia, S. Majumdar, A synthetic workload generation technique for stress testing session-based systems, IEEE Trans. Softw. Eng. 32 (2006) 868–882.
[21] G. Lee, Resource Allocation and Scheduling in Heterogeneous Cloud Environments, Ph.D. Thesis, University of California, Berkeley, 2012.
[22] W.E. Leland, M.S. Taqqu, W. Willinger, D.V. Wilson, On the self-similar nature of ethernet traffic (extended version), IEEE/ACM Trans. Netw. 2 (1994) 1–15.
[23] H. Li, M. Muskulus, Analysis and modeling of job arrivals in a production grid, SIGMETRICS Perform. Eval. Rev. 34 (2007) 59–70.
[24] N. Mi, G. Casale, L. Cherkasova, E. Smirni, Injecting realistic burstiness to a traditional client–server benchmark, in: Proceedings of the 6th International Conference on Autonomic Computing, ACM, New York, NY, USA, 2009, pp. 149–158.

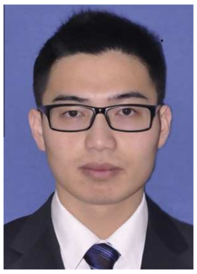
ARTICLE IN PRESS

[25] N. Mi, Q. Zhang, A. Riska, E. Smirni, E. Riedel, Performance impacts of autocorrelated flows in multi-tiered systems, Perform. Eval. 64 (2007) 1082–1101.
[26] K. Morton, M. Balazinska, D. Grossman, ParaTimer: a progress indicator for MapReduce DAGs, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM, 2010, pp. 507–518.
[27] D. Mosberger, T. Jin, httperf: a tool for measuring web server performance, SIGMETRICS Perform. Eval. Rev. 26 (1998) 31–37.
[28] J. Napper, P. Bientinesi, Can cloud computing reach the top500?, in: Proceedings of the Combined Workshops on UnConventional high Performance Computing Workshop Plus Memory Access Workshop, ACM, 2009, pp. 17–20.
[29] A. Riska, E. Riedel, Disk drive level workload characterization, in: Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 2006, pp. 9–9.
[30] I. Shafer, K. Ren, V.N. Boddeti, Y. Abe, G.R. Ganger, C. Faloutsos, Rainmon: an integrated approach to mining bursty timeseries monitoring data, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2012, pp. 1158–1166.
[31] K. Shen, M. Zhong, C. Li, I/o system performance debugging using model-driven anomaly characterization, in: FAST, vol. 5, 2005, pp. 23–23.
[32] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, D. Patterson, Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0, in: Proc. of CCA, 2008.
[33] J. Tai, J. Zhang, J. Li, W. Meleis, N. Mi, Ara: Adaptive resource allocation for cloud computing environments under bursty workloads, in: 30th IEEE International Performance Computing and Communications Conference (IPCCC), 2011, pp. 1–8.
[34] C.A. Thekkath, J. Wilkes, E.D. Lazowska, Techniques for file system simulation, Softw.: Pract. Exper. 24 (1994) 981–999.
[35] M. Uysal, G.A. Alvarez, A. Merchant, A modular, analytical throughput model for modern disk arrays, in: Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE, 2001, pp. 183–192.
[36] A. Verma, L. Cherkasova, R.H. Campbell, SLO-driven right-sizing and resource provisioning of MapReduce jobs, Proc. LADIS 11 (2011).
[37] A. Verma, G. Dasgupta, T.K. Nayak, P. De, R. Kothari, Server workload analysis for power minimization using consolidation, in: Proceedings of the 2009 Conference on USENIX Annual Technical Conference, USENIX Association, 2009, pp. 28–28.
[38] M. Wang, A. Ailamaki, C. Faloutsos, Capturing the spatio-temporal behavior of real traffic data, Perform. Eval. 49 (2002) 147–163.
[39] Y. Wu, G. Min, K. Li, B. Javadi, Modeling and analysis of communication networks in multicluster systems under spatio-temporal bursty traffic, IEEE Trans. Parall. Distrib. Syst. 23 (2012) 902–912.
[40] W. Xiong, H. Hu, N. Xiong, L.T. Yang, W.C. Peng, X. Wang, Y. Qu, Anomaly secure detection methods by analyzing dynamic characteristics of the network traffic in cloud communications, Inf. Sci. (2013).
[41] A. Youssef, D. Krishnamurthy, Cloud service level planning under burstiness, in: International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2013, IEEE, pp. 107–114.

Jianwei Yin is a full professor at the College of Computer Science and Technology, Zhejiang University, China. He received his Ph.D. in Zhejiang University, China in 2001. His current research interests include service computing, middleware, semantic web, cloud computing, workload analysis, performance prediction, etc. He has published more than 120 research papers in major peer-reviewed international journals and conference proceedings in these areas. Furthermore he has received more than 30 patents in the past five years. E-mail: zjuyjw@zju.edu.cn.



Xingjian Lu received his B.S. degree in computer science from Xidian University (P.R. China) in 2009. He is a Ph.D. candidate in the College of Computer Science, Zhejiang University. His research interest is in the area of service computing, workload analysis and performance prediction, especially in the context of cloud computing. E-mail: zjulxj@zju.edu.cn.



Hanwei Chen received his B.S. degree in computer science from Zhejiang University (P.R. China) in 2007. He is a Ph.D. candidate in the College of Computer Science, Zhejiang University. His research interest is in the area of service computing, cloud computing, workload analysis and performance predicting. He has published some papers in the famous international conference and jours, such as Internet Computing and Cluster. E-mail: chw@zju.edu.cn.

ARTICLE IN PRESS

Xinkui Zhao received his B.S. degree in computer science from Northwestern Polytechnical University (P.R. China) in 2011. He is a Ph.D. candidate in the College of Computer Science, Zhejiang University. His research interest is in the area of cloud computing and performance engineering, including modeling, analyzing, and prediction. E-mail: zhaoxinkui@zju.edu.cn.

Neal N. Xiong Neal N. Xiong is current a Professor at School of Computer Science, Colorado Technical University, Colorado Spring, CO, USA. His research interests include Cloud Computing, Security and Dependability, Networks, and Optimization Theory.

Xiong published over 150 international journal papers and over 100 international conference papers. Some of his works were published in IEEE JSAC, IEEE or ACM transactions, ACM Sigcomm workshop, IEEE INFOCOM, ICDCS, and IPDPS. He has been a General Chair, Program Chair, Publicity Chair, PC member and OC member of over 100 international conferences, and he is serving as an Editor-in-Chief, Associate editor-in-chief, or Editor member for over 10 international journals (including **Associate Editor for "IEEE Tran. on Systems, Man** & **Cybernetics: Systems" and "INFORMATION SCIENCE", and Editor-in-Chief for Journal of Parallel** & **Cloud Computing (PCC)**), and a guest editor for over 20 international journals, including Sensor Journal, WINET and MONET. He has received the Best Paper Award in HPCC-08 and the Best student Paper Award in NAFIPS2009.

He is the Chair of "Trusted Cloud Computing" Task Force, IEEE Computational Intelligence Society (CIS), http://www.cs.gsu.edu/cscnxx/index-TF.html, and the Industry System Applications Technical Committee, http://ieee-cis.org/technical/isatc/; He is a **Senior member of IEEE Computer Society**, E-mail: nxiong@coloradotech.edu.