J. Parallel Distrib. Comput. 74 (2014) 1820–1832

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Google hostload prediction based on Bayesian model with optimized feature combination

Sheng Di [a,*], Derrick Kondo [a], Walfredo Cirne [b]

[a] INRIA, 51, avenue Jean Kuntzmann, Grenoble, France

[b] Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA

HIGHLIGHTS

- We devise an exponentially segmented pattern model for the hostload prediction.
- We devise a Bayes method and exploit 10 features to find the best-fit combination.
- We evaluate the Bayes method and 8 other well-known load prediction methods.
- The experiment is based on Google trace with over 10 k hosts and millions of jobs.
- The pattern prediction with Bayes method has much higher precision than others.

ARTICLE INFO

Article history:
Received 4 May 2013
Received in revised form
18 August 2013
Accepted 4 October 2013
Available online 18 October 2013

Keywords:
Hostload prediction
Bayesian model
Google data center

ABSTRACT

We design a novel prediction method with Bayes model to predict a load fluctuation pattern over a long-term interval, in the context of Google data centers. We exploit a set of features that capture the expectation, trend, stability and patterns of recent host loads. We also investigate the correlations among these features and explore the most effective combinations of features with various training periods. All of the prediction methods are evaluated using Google trace with 10,000+ heterogeneous hosts. Experiments show that our Bayes method improves the long-term load prediction accuracy by 5.6%–50%, compared to other state-of-the-art methods based on moving average, auto-regression, and/or noise filters. Mean squared error of pattern prediction with Bayes method can be approximately limited in $[10^{-8}, 10^{-5}]$. Through a load balancing scenario, we confirm the precision of pattern prediction in finding a set of idlest/busiest hosts from among 10,000+ hosts can be improved by about 7% on average.

1. Introduction

Accurate prediction of the host load in a Cloud computing data center is essential for achieving service-level agreements (SLA's). In particular, effective prediction of host load can facilitate the proactive job scheduling or host load balancing decisions. This, in turn, can improve resource utilization, lower data center costs (if idle machines are shutdown), and improve the job performance.

Compared with traditional Grids [16] and HPC systems, the host load prediction in Cloud data centers is arguably more challenging as the Cloud host load has much higher variance. This stems from differences in the workloads on top of such platforms. Unlike the scientific applications commonly used in Grid or HPC platforms, Cloud tasks tend to be shorter and more interactive, including (instant) keyword, image, or email search. In fact, by comparing the load traces of a Google data center [18,33,24] and the AuverGrid cluster [3,30], we observe that Cloud task lengths are only $[\frac{1}{20}, \frac{1}{2}]$ of Grid task lengths. We find that this difference leads to more drastic and short-term load fluctuation in Clouds compared to Grids [9,10].

Most prior work in Cloud Computing has focused primarily on the application workload characterization versus long-term host load prediction. For instance, there are several works on the characterization of task placement constraints [28], task usage shape [37], and their impacts to host load [24].

Most prior prediction work in Grid Computing or HPC systems [6,7,1,11,38,34,31,35] has focused mainly on using moving averages, auto-regression, and noise filters. These prediction methods have been evaluated with traces of load in Grids or HPC systems. When applied to bursty Cloud workloads, they have limited accuracy. Moreover, these works do not attempt to predict the long-term future load fluctuation pattern (e.g., the load changes over consecutive time intervals).

In this paper, we design an effective Cloud load prediction method that can accurately predict the host load fluctuation pattern over a long-term period up to 16 h in length. We focus

* Corresponding author.
E-mail addresses: sheng.di@inria.fr, disheng222@gmail.com (S. Di), derrick.kondo@inria.fr (D. Kondo), walfredo@google.com (W. Cirne).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.10.001

on two critical metrics, CPU and memory, highlighted in [24]. Our approach is to use a Bayesian model for prediction as it effectively retains the important information about the load fluctuation and noise. We evaluate our prediction method, not only using a detailed 1-month load trace of a Google data center with 10,000+ machines, but also based on an emulated 1-year host load for each machine.

In particular, our contributions are listed below:

- *What we predict*: we accurately predict both mean load over a future time interval (up to 16 h), and the mean load over consecutive future time intervals (which we refer to as a *pattern*).
- *How we predict*. We craft novel features used for the Bayesian prediction that capture the important and predictive statistical properties of host load. These properties include the expectation, stability, trends, and patterns of host load. We determine which of these features are complementary to one another and improve the predictive power of the Bayesian model.
- *How we evaluate and compare*: our evaluation is done using a 1-month load trace of a Google data center with over 10,000 machines. We compare comprehensively our Bayesian prediction methods with 8 other baseline and state-of-the-art methods that use a variety of techniques, including moving averages, noise filters, and auto-regression. For the long-term load prediction, our Bayesian method outperforms others by 5.6%–50% on average. The mean-squared error (MSE) of the Bayesian method for a single interval is 0.0014, and for a pattern is about 10^{-5} or lower. Through a load balancing scenario, we confirm the precision of pattern prediction in finding a set of idlest/busiest hosts from among totally 10 k hosts can be improved by about 7% on average.

The rest of the paper is organized as follows. In Section 2, we formulate the Cloud load prediction problem as a pattern prediction model. In Section 3, we present the overall design of the Bayes classifier and propose 10 candidate features used as the evidence for prediction with an in-depth analysis of their mutual correlation. In addition to our Bayes method, we rigorously implement many other solutions (including models based on moving averages, auto-regression, and noise filters) for comparison. We present the experimental results based on Google's load trace data in Section 4 and discuss related work in Section 5. Finally, we conclude the paper with our future work in Section 6.

2. Prediction formulation

Our predictive study is based on the load measurements of a Google data center. Google [18] traced over 670,000 jobs and over 40 million task events at minute resolution across over 10,000 machines in a production system in 2011 over a one-month period. Users submit jobs to a batch scheduler, where each job consists of a set of tasks and a set of resource constraints (on CPU and memory, for example). The batch scheduler in turn allocates those tasks to hosts. Load on the hosts is a function of the incoming workload at the batch scheduler and its scheduling strategy.

Our objective is to predict the fluctuation of host load over a long-term period. First, at a current time point t_0, we would like to predict the mean load over a single interval, starting from t_0. Second, we would like to predict, the mean load over consecutive time intervals. We propose a new metric, namely *exponentially segmented pattern (ESP)*, to characterize the host load fluctuation over some time period. For any specified prediction interval, we split it into a set of consecutive segments, whose lengths increase exponentially. We predict the mean load over each time segment.

We show in Fig. 1an example of ESP. We denote the total prediction interval length as s. The first segment (denoted by s_1) is

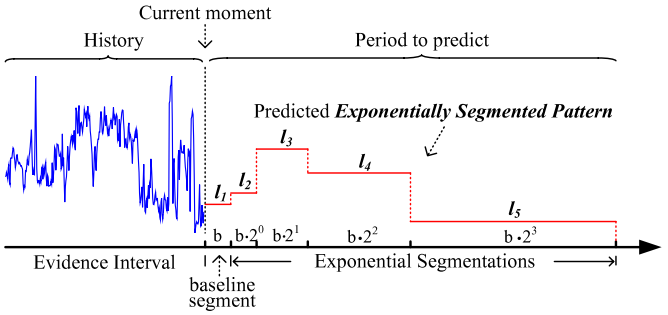


Fig. 1. Illustration of exponentially segmented pattern. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

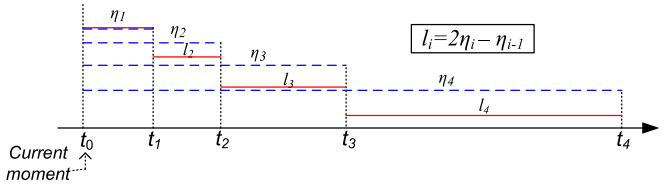


Fig. 2. Induction of segmented host load. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

called *baseline segment* with length b, starts from the current time point t_0 and ends at $t_0 + b$. The length of each following segment (denoted by s_i) is $b \cdot 2^{i-2}$, where $i = 2, 3, 4, \ldots$. For example, if b is set to 1 h, the entire prediction interval length s could be equal to 16 (=1 + 1 + 2 + 4 + 8) h. For each segment, we predict the mean host load. The mean values are denoted by l_i, where $i = 1, 2, 3, \ldots$. From this example, it is clear that the prediction granularity is finer in the short-term than in the long-term. This is useful for two reasons. In general, the short-term load is easier to predict precisely than the long-term load. This is because of the higher correlation of host load found among short consecutive time segments. Also, tasks in Cloud systems are typically short (less than 1 h) in length. So, users or schedulers would value the prediction of short-term load fluctuations more than the long-term ones.

As illustrated above, our aim is to predict the vector of load values (denoted by $\boldsymbol{l} = (l_1, l_2, \ldots, l_n)^T$), where each value represents the mean load value over a particular segment.

To predict load, a predictor often uses recent load samples. The interval that encloses the recent samples used in the prediction is called *evidence interval* or *evidence window*.

Transformation of pattern prediction

According to our prediction model formulated previously, the prediction of each segmented mean load is the key step of the whole process. Since the host load always appears with high correlation between the adjacent short-term intervals but not for the non-adjacent ones, it is straight-forward to predict the load in the successive intervals based on the evidence window. Hence, we convert the segment representation formulated above into another one, where each interval to predict is adjacent to the evidence window.

In the new representation, we only need to predict a set of mean host loads for different lengths of future intervals, *each starting from the current time* t_0. We denote the mean load levels of the prediction intervals as $\eta_1, \eta_2, \ldots, \eta_n$, where $\eta_{i+1} = 2 \cdot \eta_i$. The target is to predict such a vector, $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_n)^T$, rather than the vector $\boldsymbol{l}$. In fact, the vector $\boldsymbol{l}$ can be converted from $\boldsymbol{\eta}$ through the following induction. We use an example to show the idea, as shown in Fig. 2.

Suppose that the current moment is t_0, and we have already predicted two mean load values (η_{i-1} and η_i, the blue dotted-line segment) over two different intervals, $[t_0, t_{i-1}]$ and $[t_0, t_i]$, respectively. Then, by making the areas of the two shaded squares (S_1 and S_2) equal to each other, we can easily derive the mean load value in $[t_{i-1}, t_i]$. The transformation is shown in Formula (1), where l_i is the predicted mean load in the new segment $[t_{i-1}, t_i]$, corresponding to the red solid-line segment in Fig. 2.

$$l_i = \eta_i + \frac{t_{i-1} - t_0}{t_i - t_{i-1}}(\eta_i - \eta_{i-1}). \tag{1}$$

Taking into account $t_i = 2t_{i-1}$ and $t_0 = 0$, we can further simplify the Formula (1) as Eq. (2).

$$l_i = 2\eta_i - \eta_{i-1}. \tag{2}$$

This new representation is useful for two reasons. First, it simplifies and generalizes predictor implementation; any predictor that can predict load over a single load interval can be converted to predict a load pattern. Second, it gives the resource or job management system the option of predicting different load intervals starting at the current time point, or consecutive load intervals, without any additional overheads. Transforming from one representation to another is trivial in terms of complexity.

We show the pseudo-code of our Cloud load pattern prediction method in Algorithm 1.

Algorithm 1 PATTERN PREDICTION ALGORITHM

Input: baseline interval (b); length of prediction interval ($s = b \cdot 2^{n-1}$, where n is the number of segments to be split in the pattern prediction);
Output: mean load vector l of Exponentially Segmented Pattern (ESP)

1: **for** ($i = 0 \rightarrow n-1$) **do**
2: $z_i = b \cdot 2^i$;
3: $\varpi_i = \frac{z_i}{2}$; /*ϖ_i is the length of the evidence window*/
4: Predict the mean load η_i, whose prediction length is equal to z_i, based on a predictor - PREDICTOR(ϖ_i,z_i);
5: **end for**
6: Segment transformation based on Equation (2): $\boldsymbol{\eta} \rightarrow \boldsymbol{l}$;

Basically, there are two key steps in the Pattern Prediction algorithm, namely, mean load prediction (lines 1–5) and segment transformation (line 6). Note that each prediction interval always starts from the current moment, unlike the segments defined in the first representation $\boldsymbol{l}$ (Fig. 1). In next section, we focus on the mean load prediction based on the Bayes Classifier [5,13,32] with our elaborately designed features.

3. Mean load prediction based on Bayes model

The fundamental idea is to generate the posterior probability from the prior probability distribution and the run-time evidence of the recent load fluctuations, according to a Bayes Classifier. We first describe how we construct the Bayes model and then intensely discuss 10 key features designed.

3.1. Bayes classifier

The Bayes Classifier [5,13,32] is a classic supervised learning classifier used in data mining [14]. Bayesian classification consists of five main steps: (1) determine the set of target states (denoted as the vector $\boldsymbol{W} = (\omega_1, \omega_2, \ldots, \omega_m)^T$, where m is the number of states), and the evidence vector with h mutually-independent features (denoted as $\chi = (x_1, x_2, \ldots, x_h)^T$); (2) compute the prior probability distribution for the target states, $P(\omega_i)$, based on the samples; (3) compute the joint probability distribution $p(\chi|\omega_i)$ for each state ω_i; (4) compute the posterior probability based on some evidence, according to Formula (3); (5) make the decision based on a risk function $\lambda(\widehat{\omega}_i, \dot{\omega}_i)$, where $\widehat{\omega}_i$ and $\dot{\omega}_i$ indicate the true value and predicted value of the state, respectively.

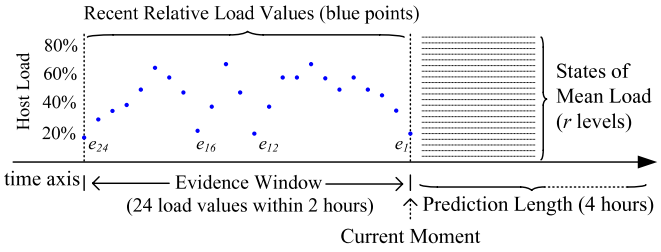


Fig. 3. Illustration of evidence window and target load states.

$$P(\omega_i|x_j) = \frac{p(x_j|\omega_i)P(\omega_i)}{\sum_{k=1}^{m} p(x_j|\omega_k)P(\omega_k)}. \tag{3}$$

Based on different risk functions, there are two main ways for making decisions, namely Naïve Bayes Classifier (abbreviated as *N-BC*) [32,13] and Minimized MSE (MMSE) based Bayes Classifier (abbreviated as *MMSE-BC*) [5]. Their corresponding risk functions are shown in Formula (4) and Formula (5) respectively.

$$\lambda(\dot{\omega}_i, \widehat{\omega}_i) = \begin{cases} 0 & |\dot{\omega}_i - \widehat{\omega}_i| < \delta \\ 1 & |\dot{\omega}_i - \widehat{\omega}_i| \geq \delta \end{cases} \tag{4}$$

$$\lambda(\dot{\omega}_i, \widehat{\omega}_i) = (\dot{\omega}_i - \widehat{\omega}_i)^2. \tag{5}$$

According to the different risk functions, the predicted value of the state ($\widehat{\omega}_i$) is determined by Formula (6) and Formula (7) respectively. It is easy to prove that the former leads to the minimal error rate and the latter results in the minimal MSE [5], where the error rate is defined as the number of wrong decisions over the total number of tries.

$$\widehat{\omega}_i = \arg\max p(\omega_i|x_j) \tag{6}$$

$$\widehat{\omega}_i = E(\omega_i|x_j) = \sum_{i=1}^{m} \omega_i p(\omega_i|x_j). \tag{7}$$

Based on the above analysis, the target state vector and the evidence feature vector are the most critical for accurate prediction. In our design, we split the range of host load values into small intervals, and each interval corresponds to a load level (or load state). The number of intervals in the load range [0, 1] is denoted by r, which is set to 50 in our experiment. So there are 50 load states in total, [0, 0.02), [0.02, 0.04), ..., [0.98, 1]. As shown in Algorithm 1, the length of the evidence window is set equal to half of the prediction interval length, which maximizes accuracy, based on our experimental results. The whole evidence window will also be split into a set of equally-size segments. If the prediction interval length is 8 h, the evidence window length will be set to the recent past 4 h. 48 ($=\frac{4\times 60}{5}$) successive load values (if the sample interval is 5 min) in this period will serve as the fundamental evidence, based on which we can extract many interesting features for the Bayes prediction. We use Fig. 3 to illustrate the discretized evidence window and target load states. In this example, the prediction interval length is assumed to be 4 h, so the evidence window length is 2 h and there are 24 load values in the evidence window. In next section, we will present how to extract the features from the load values in the evidence window.

3.2. Features of load fluctuation

Through the analysis of Google's one-month trace, we extract 9 candidate features to be used as the evidence in the Bayes model, each of which can partially reflect recent load fluctuation. In this section, we first present these features, and then discuss their mutual correlation.

We denote the load vector in the evidence window as $\boldsymbol{e} = (e_1, e_2, \ldots, e_d)^T$, where d is the number of the samples in the evidence window, also known as *window size*. The elements in the vector are organized from the most recent one to the oldest one. For example, e_1 indicates the newest sample that is closest to the current moment. We summarize the 10 features as follows.

- *mean load* ($F_{ml}(\boldsymbol{e})$): the mean load is the mean value of the load vector $\boldsymbol{e}$, as shown in Eq. (8). Its value range is [0, 1] in principle, and we split such a range into r even fractions, each corresponding to a load level (or type). For instance, r is set to 50 in our experiment, so there are 50 levels (or types) to characterize the recent mean load level, [0, 0.02), [0.02, 0.04), ..., [0.98, 1]. For this feature, its value must be one of the 50 levels, constructing partial evidence for the Bayes Classifier.

$$F_{ml}(\boldsymbol{e}) = \frac{1}{d}\sum_{i=1}^{d} e_i. \tag{8}$$

- *weighted mean load* ($F_{wml}(\boldsymbol{e})$): weighted mean load refers to the linear weighted mean value of the load vector $\boldsymbol{e}$, as shown in Eq. (9).

$$F_{wml}(\boldsymbol{e}) = \frac{\sum_{i=1}^{d}(d-i+1)e_i}{\sum_{i=1}^{d} i} = \frac{2}{d(d+1)}\sum_{i=1}^{d}(d-i+1)e_i. \tag{9}$$

  Rather than the mean load feature, the weighted mean load weights the recent load values more heavily than older ones. Similar to the mean load feature, the value range of this feature is also within [0, 1], which will also be split into 50 levels to choose, serving as the partial evidence for the succeeding Bayes prediction.
- *fairness index* ($F_{fi}(\boldsymbol{e})$): the fairness index [20] (a.k.a., Jain's fairness index) is used to characterize the degree of the load fluctuation in the evidence window. The fairness index is defined in Formula (10).

$$F_{fi}(\boldsymbol{e}) = \frac{\left(\sum_{i=1}^{d} e_i\right)^2}{d\sum_{i=1}^{d} e_i^2}. \tag{10}$$

  Its value is normalized in [0, 1], and a higher value indicates more stable load fluctuation. Its value is equal to 1 if and only if all the load values are equal to each other. Since the target state in our model is the mean load value of the future prediction interval, the mean load feature seems more important than the fairness index, which will also be confirmed in our experiment. However, in some situations, e.g., when the load in prediction interval changes similarly to the training period to a certain extent, fairness index could effectively improve the prediction effect, to be shown later.
- *noise-decreased fairness index* ($F_{ndfi}(\boldsymbol{e})$): the noise-decreased fairness index is also computed using the fairness index formula. However, one or two load values that may significantly degrade the whole fairness index, would be excluded in the computation of fairness index. Via our experiment over Google's trace, we found that the degree of load fluctuation can be split into 5 levels, whose fairness index values range in [0, 0.4), [0.4, 0.65), [0.65, 0.9), [0.9, 0.97) and [0.97, 1] respectively. In our implementation, if the original fairness index changes across different ranges when removing one or two host load values, the excluded load values would be deemed *load outliers*, which will be considered noises.
- *type state* ($F_{ts}(\boldsymbol{e})$): the type state feature is used to characterize the load range in the evidence window and the degree of jitter. Specifically, as aforementioned, there are $r = 50$ types split in the load range. The type state feature is defined as a two-tuple, denoted by $\{\alpha, \beta\}$, where α and β refer to the number of *types* involved and the number of *state changes* respectively.
- *last load* ($F_{ll}(\boldsymbol{e})$): the last load is referred to as the most recent load value in the evidence window $\boldsymbol{e}$. Like the *mean load* feature, the value range of the last load is also in [0, 1], thus there are r different levels for this feature, and r is set to 50 in our experiment.
- *first–last load* ($F_{fll}(\boldsymbol{e})$): the first–last load feature is used to roughly characterize the changing trend of the host load in the recent past. It is also a two-tuple, denoted as $\{\tau, \iota\}$, indicating the first load value and the last one recorded in the evidence window. Obviously, this is just a rough feature which needs to be combined with other features in practice.
- *N-segment pattern* ($F_{N\text{-}sp}(\boldsymbol{e})$): we also characterize the segment patterns based on the evidence window. The evidence window is evenly split into several segments, each of which is reflected by the mean load value. For example, if the window size is 48, the 4-segment pattern is a four-tuple, whose elements are the means of the following load values respectively, $[e_1, e_{12}]$, $[e_{13}, e_{24}]$, $[e_{25}, e_{36}]$, and $[e_{37}, e_{48}]$.

So far, we have presented 10 features to be used in the Bayes model. (Note N-segment pattern can be deemed three features with $N = 2, 3$, and 4 respectively.) Some of them, however, are mutually correlated, which violates the condition of Bayes' theorem that the features used in Formula (3) should be mutually independent. For example, the fairness index feature and the noise-decreased fairness index feature could be closely correlated, implying that they cannot be used meanwhile. We list the linear correlation coefficients in Table 1. We observe that some correlation coefficients (such as F_{fi} & F_{ndfi}) can be as high as 0.99, while those of the intuitively non-correlated features (such as F_{ts} & F_{fll}) are below 0.85 and even down to 0.15. We also observe N-segment pattern features are always extremely correlated mutually.

Much research on the independence constraint of Bayes Classifier [12,13,32] shows that the optimal situation might still happen when a few features are correlated to a certain extent. Hence, we set the compatibility of the features in Table 2, based on the Formula (11), where $\mathrm{Comp}(F_x, F_y)$ and $\mathrm{Corr}(F_x, F_y)$ refer to the compatibility and correlation coefficient of two features respectively.

$$\mathrm{Comp}(F_x(\boldsymbol{e}), F_y(\boldsymbol{e})) = \begin{cases} Y & \mathrm{Corr}(F_x(\boldsymbol{e}), F_y(\boldsymbol{e})) \le 0.84 \\ N & \mathrm{Corr}(F_x(\boldsymbol{e}), F_y(\boldsymbol{e})) \ge 0.96. \end{cases} \tag{11}$$

Two features are considered incompatible iff their correlation coefficients are greater than 0.96, and compatible iff their coefficients are less than 0.84.

There are only 143 viable combinations of the features, based on the following analysis in terms of the compatibility table. Since there are 10 features (F_{ml}, F_{wml}, F_{fi}, F_{ndfi}, F_{ts}, F_{ll}, F_{fll}, $F_{2\text{-}sp}$, $F_{3\text{-}sp}$, $F_{4\text{-}sp}$) in total, the number of their combinations is at most 2^{10}. Yet, many of the combinations are not viable according to the Table 2. For instance, F_{ml} and F_{wml} should not be used together. By observing this table, all of 10 features can be classified into 5 groups, $\{F_{ml}, F_{wml}, F_{2\text{-}sp}, F_{3\text{-}sp}, F_{4\text{-}sp}\}$, $\{F_{fi}, F_{ndfi}\}$, $\{F_{ts}\}$, $\{F_{ll}\}$, and $\{F_{fll}\}$. The elements in the same group cannot be used meanwhile in one combination. So, the numbers of compatible combinations (denoted by *NCC*) for the five groups are 6, 3, 2, 2, and 2 respectively. Hence, the total number of compatible combinations can be computed as follows.

Table 1
Linear correlation of the features.

	F_{ml}	F_{wml}	F_{fi}	F_{ndfi}	F_{ts}	F_{ll}	F_{fll}	$F_{2\text{-}sp}$	$F_{3\text{-}sp}$	$F_{4\text{-}sp}$
F_{ml}	1	0.98	0.46	0.46	0.15	0.69	0.82	0.990	0.990	0.990
F_{wml}	0.98	1	0.45	0.45	0.15	0.75	0.81	0.967	0.968	0.968
F_{fi}	0.46	0.45	1	0.99	0.3	0.30	0.36	0.467	0.467	0.467
F_{ndfi}	0.46	0.45	0.99	1	0.3	0.30	0.36	0.464	0.464	0.464
F_{ts}	0.15	0.15	0.3	0.3	1	0.15	0.17	0.167	0.167	0.167
F_{ll}	0.69	0.75	0.30	0.30	0.15	1	0.84	0.706	0.707	0.707
F_{fll}	0.82	0.81	0.36	0.36	0.17	0.84	1	0.830	0.831	0.831
$F_{2\text{-}sp}$	0.99	0.97	0.46	0.46	0.17	0.71	0.83	1	0.999	0.999
$F_{3\text{-}sp}$	0.99	0.97	0.46	0.46	0.17	0.71	0.83	0.999	1	0.999
$F_{4\text{-}sp}$	0.99	0.97	0.46	0.46	0.17	0.71	0.83	0.999	0.999	1

Table 2
Compatibility of the features.

	F_{ml}	F_{wml}	F_{fi}	F_{ndfi}	F_{ts}	F_{ll}	F_{fll}	$F_{N\text{-}sp}$
F_{ml}	N	N	Y	Y	Y	Y	Y	N
F_{wml}	N	N	Y	Y	Y	Y	Y	N
F_{fi}	Y	Y	N	N	Y	Y	Y	Y
F_{ndfi}	Y	Y	N	N	Y	Y	Y	Y
F_{ts}	Y	Y	Y	Y	N	Y	Y	Y
F_{ll}	Y	Y	Y	Y	Y	N	Y	Y
F_{fll}	Y	Y	Y	Y	Y	Y	N	Y
$F_{N\text{-}sp}$	N	N	Y	Y	Y	Y	Y	N

$NCC(10\text{ features}) = NCC(\text{Group 1}) \cdot NCC(\text{Group 2}) \cdot NCC(\text{Group 3}) \cdot NCC(\text{Group 4}) \cdot NCC(\text{Group 5}) = 6 \times 3 \times 2 \times 2 \times 2 = 144$.

By excluding the case where no feature is selected, there are 143 viable combinations of the features, all of which will be evaluated in our experiment under the Bayes model.

4. Performance evaluation

4.1. Algorithms for comparison

In addition to our Bayes estimator, we also implemented eight other load prediction methods. These baseline solutions are extensively studied in the load prediction domain.

- *Last-state based method* (*last-state*): the last recorded load value in the evidence window will be used as the predicted mean load for the future period.
- *Simple moving average method* (*SMA*): the mean load value of the evidence window will serve as the prediction for the future mean load.
- *Linear weighted moving average method* (*Linear_WMA*): the linear weighted mean load (based on Formula (9)) will be considered as the mean load prediction for the future.
- *Second-order moving average method* (*Sec-Order_MA*) [2]: the second-order moving average (denoted as $M_t^{(2)}$, where t refers to the current time point) is defined in Formula (12), where $N = \frac{d}{2}$ and d means evidence window length.

$$M_t^{(2)} = \sum_{t=0}^{N} M_t^{(1)}, \quad \text{where } M_t^{(1)} = \frac{\sum_{i=1}^{N} e_{t+i}}{N}. \tag{12}$$

If the future host load follows a linear trend with the evidence window, the load state at future time point $t + T$ (denoted as $X(t + T)$) could be estimated by Formula (13).

$$X(t+T) = a_t + b_t \cdot T, \quad \text{where } \begin{cases} a_t = 2M_t^{(1)} - M_t^{(2)} \\ b_t = \dfrac{2}{N-1}(M_t^{(1)} - M_t^{(2)}). \end{cases} \tag{13}$$

In the experiment, we use $X(t + 1)$ and $X(t + \frac{s}{2})$ to be the predicted state value of the future s-step length. In the following text, the corresponding two prediction results are denoted as "secOrder-MA-1" and "secOrder-MA-s/2" respectively.

- *Exponential Moving Average method* (*EMA*): this predicted value (denoted $S(t)$ at time t) is calculated based on the Formula (14), where e_1 is the last load value and α is tuned empirically to optimize accuracy.

$$S(t) = \alpha \cdot e_1 + (1 - \alpha) \cdot S(t - 1). \tag{14}$$

- *Prior probability based method* (*PriorPr*): this method uses the load value with highest prior probability as the prediction for the future mean load, regardless of the evidence window.
- *Auto-regression method* (*AR*): the classic AR method is performed according to the Formula (15), where $X(t)$, p and ε_t refer to the predicted value, the order and the white noise at time point t respectively.

$$X(t) = \sum_{i=1}^{p} \varphi_i e_i + \varepsilon_t. \tag{15}$$

In general, the AR method can only predict the load value for the next moment, while previous works [36,35] extended it to the long-term point prediction by applying the AR method recursively on the predicted values. Based on our prediction model, the mean value of the AR-based predicted values at different time points in the prediction interval will serve as the prediction value.

- *Hybrid model* proposed in [36] (*HModel*): this method integrates the Kalman filter [21] and Savitzky–Golay smoothing filter [25] with auto-regression. There are four steps in the load prediction: (1) use the Kalman filter to eliminate noise in the evidence window; (2) smoothen the curve by using a Savitzky–Golay filter; (3) compute the AR coefficients and predict the usage values for future time points, by recursively calling the AR method; (4) smoothen the AR-predicted values by a Savitzky–Golay filter and estimate the confidence window. In our experiment, we also calculate the mean load of the predicted values as the prediction result.

We optimize the coefficients for the baseline algorithms in Table 3.

4.2. Method of training and evaluation

Each evaluation involves two duration, a *training period* and a *test period*. The training period is used to fit the models, for instance, for computing the prior probability ($P(\omega_i)$ in Formula (3)) and the conditional probability ($p(x_j|\omega_k)$ in Formula (3)) and estimating auto-regressive coefficients and noise covariance for the auto-regression method and HModel. The test period is used to validate the prediction effect of different methods.

For the baseline algorithms, the length of the evidence interval (or the evidence window length) is always set to the half of the prediction interval length. This is because we found empirically via the Google trace that this maximizes the prediction accuracy for the baseline algorithms.

Table 3
Optimized parameters for the baseline methods.

	Key parameters	Values
EMA	α	0.95
AR	Order of AR	7
Hybrid model	Order of AR	4
	Degree of SGFilter	4
	Covariance of Kalman Filter's process-noise (Q)	0.00001
	Covariance of Kalman Filter's measurement noise	0.028^2

The experiments can be categorized into three situations. They differ based on whether there are sufficient number of load samples in the training period. In the first situation (referred to as *evaluation type* A), we set the training period to be [day 1, day 25] and test period to be [day 26, day 29]. We found that the distribution of features of host load in the training period are likely different from those in the test period, which is due to the lack of enough training data.

In the second situation (referred to as *evaluation type* B), we emulate the scenario where we have enough training data to observe the repeated load fluctuations. In this case, the distribution of features between the training and test periods are more similar. The training period is changed to be [day 1, day 29] accordingly. Note that the load forecasting is still performed based on the posterior probability calculated by the Bayes predictor.

In the third situation (referred to as *evaluation type* C), we emulate the case with a large number of samples in the training period. In particular, we characterize the probability distribution of task arrival rate, task execution length and task usage on different resources. Then, we emulate the task schedule/execution events by randomly reproducing the tasks for each host based on the corresponding probability distributions. We finally compute the hostload series for each host over one year. We find the emulated one-year hostload exhibits with fairly similar features to the hostload computed based on the original one-month Google-trace. The training period is set to [day 1, day 330], while the test period is [day 331, day 360].

4.3. Metrics for accuracy

In terms of evaluating prediction accuracy, we use two metrics. We desire to minimize the mean squared error (MSE) between the predicted load values and the true values in the prediction interval. We denote the true mean values in the segments by $L_1, L_2, \ldots, L_n$. Then, the value of MSE can be calculated with Formula (16), where $s_1 = b$, $s_i = b \cdot 2^{i-2}\ \forall\, i \geq 2$, $s = \sum_{i=1}^{n} s_i$, and n is the total number of the segments in the prediction interval.

$$mse(s) = \frac{1}{s}\sum_{i=1}^{n} s_i(l_i - L_i)^2. \tag{16}$$

Second, we measure the *success rate*, which is defined as the ratio of the number of accurate predictions to the total number of predictions. A prediction is deemed accurate if it falls within some delta of the real value. We use a delta of 10%. In general, the higher the success rate, the better, and the lower the MSE.

4.4. Experimental results

We evaluate the mean load prediction before evaluating the pattern prediction, because the latter can be derived from the former.

4.4.1. Evaluation of mean host load prediction

A. *Experiments with evaluation type* A.

For evaluation type A, we first compare the prediction effects when using different risk functions (either Formula (4) or Formula (5)) and traversing all compatible combinations of the evidence features, under our designed Bayes Classifier model. Based on our previous analysis, there are 143 compatible combinations of our designed features. We denote them via the binary numerical system according to the following order, $\{F_{ml}, F_{wml}, F_{fi}, F_{ndfi}, F_{ts}, F_{ll}, F_{fll}, F_{2\text{-}sp}, F_{3\text{-}sp}, F_{4\text{-}sp}\}$. For example, 1000000000 denotes the single feature F_{ml}, and 1010101000 indicates the combination of the four features F_{ml}, F_{fi}, F_{ts}, and F_{fll}. Due to the heavy computation in traversing all 143 combinations, we sampled 2000 hosts in Google's trace. In this test, we discretize the host loads using two-minute sample intervals.

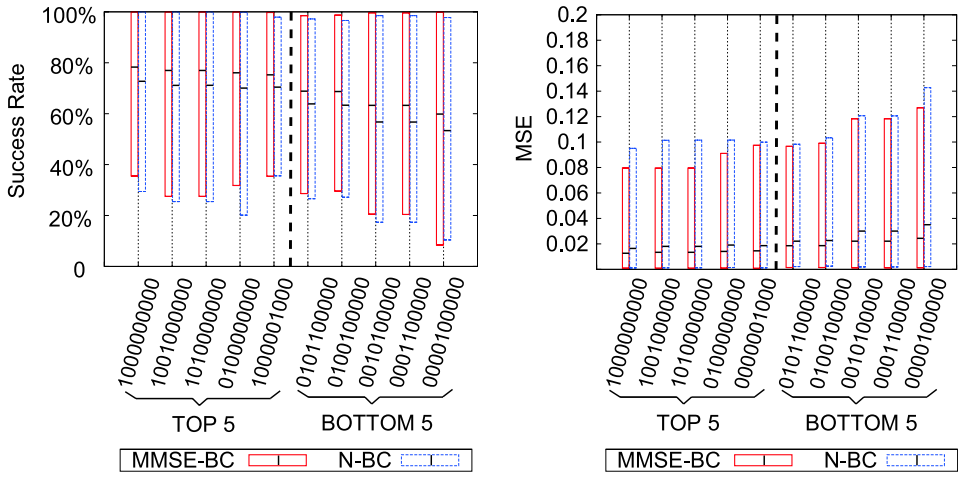
We select top 5 and bottom 5 combinations based on the success rate and MSE. In Fig. 4, we present the range of the evaluation results via rectangles, where the bottom-edge, middle black line, and upper-edge refer to the minimum, mean, and maximum values respectively. The top 5 (bottom 5) combinations are selected, with either the 5 highest (5 lowest) success rates or 5 lowest (5 highest) MSEs respectively.

In Fig. 4, we can see that the best feature combination is always 1000000000, while the worst one is always 0000100000, regardless of the prediction interval length. We can also observe that the N-segment pattern feature ($N = 2$, 3, or 4) is neither in the top five nor bottom five. In addition, since most of the top five and bottom five feature combinations almost always contain the feature F_{ml} and f_{ts} respectively, we can conclude F_{ml} plays the greatest positive effect while F_{ts} plays the most significant negative effect.

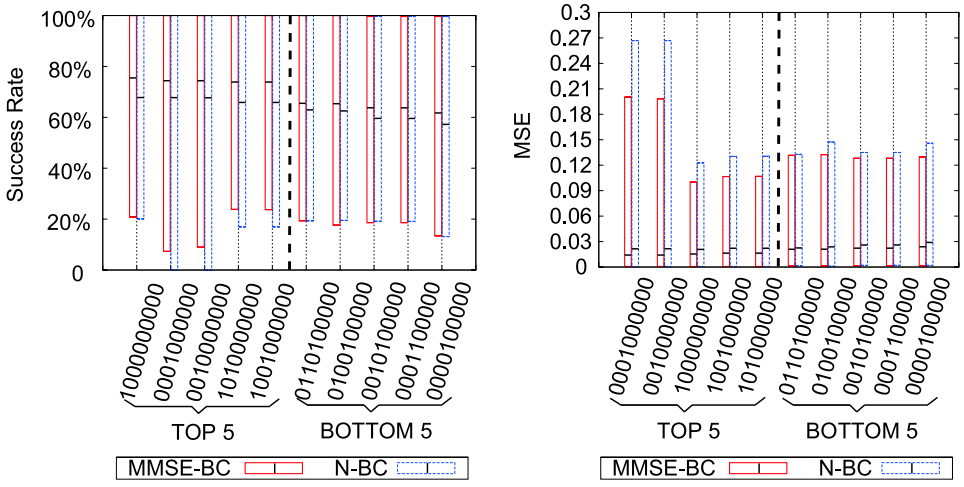
Moreover, it is also observed that the prediction of MMSE-BC is always more accurate (with higher success rate or lower MSE) than that of N-BC. This can be explained as follows: MMSE-BC adopts the mathematically expected value of the predicted load, which has the highest probability of being located in the real load level. In contrast, N-BC selects the level with the highest posterior probability as the prediction result, which may be significantly skewed from the expected value. From the above analysis, we can conclude the best strategy under the Bayes Classifier is using MMSE-BC with the single feature F_{ml}.

We comprehensively compare MMSE-BC to other prediction methods, with respect to the success rate and MSE respectively, by using 11 K hosts in the Google trace. Fig. 5 shows the cumulative distribution function (CDF) of the success rate and MSE for different prediction methods. It is clear that MMSE-BC's prediction effect on the CPU-load is better than all the other approaches. Specifically, its advantage becomes more prominent with the increase of prediction length (see Fig. 5(a), (c) or (b), (d)). Our statistics show that the MMSE-BC's success rate is higher than that of the second best solution (Linear-WMA) by 5.6% and 7.8% in the mean load prediction with 6.4 h-ahead and 12.8 h-ahead length respectively, and also higher than other approaches by up to 50%.

The reason why Bayesian prediction outperforms other methods is its features, which capture more complex dynamics (such as trends, predictability, and expectation). Last-State performs poorly because of irregular fluctuations in load. Prior-Probability performs poorly because the distribution of load values is roughly uniform, and there is no load value that is superior to others. While



(a) Success rate (6.4 h). (b) MSE (6.4 h).



(c) Success rate (12.8 h). (d) MSE (12.8 h).

Fig. 4. Success rate & MSE of Bayes classifier with different prediction length *s*.

moving averages perform well in general, they cannot capture features to adapt to load dynamics. For example, second-order moving average method assumes future load changes with a linear trend, while the true load may not change like that. The prediction effects of AR and HModel are far worse than other moving average strategies (e.g., SMA and Linear-WMA), because they both use recursive AR steps that may cause cumulative prediction errors. Experiments show a worse effect under HModel that uses filtering, since it filters useful information about load dynamics.

We compare the CDFs of Prediction Methods w.r.t. memory host load in Fig. 6. Since the memory load does not fluctuate as drastically as CPU, the MMSE-BC and all moving average solutions work well, with the average success rate up to 93% and the average MSE down to 0.004 respectively. However, AR and HModel still suffer with a low success rate and high MSE in predicting the mean memory load of long-term intervals.

So far, we have evaluated our Bayes Classifier based on the evaluation type A. We conclude that the MMSE-BC with the single feature F_{ml} performs the best among all of strategies, for the evaluation type A.

B. *Experiments with evaluation type* B.

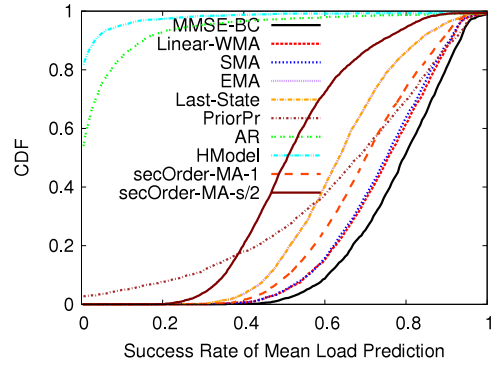
We evaluate the solutions based on evaluation type B, where the load fluctuation in the test period is similar to that of training period to a certain extent. The discretized interval of the host loads is set to 5 min and the prediction length is set to 8 h and 16 h respectively.

We traverse all 143 combinations and observe that the best feature combination is $\{F_{ml}, F_{fi}, F_{ts}, F_{fll}\}$ instead of the single feature F_{ml}. The results about the probability density function (PDF) of success rate and MSE can be found in our previous work [10]. Experiments show that the MMSE-BC under the four features (abbreviated as MMSE-BC(4F)) is surprisingly good. Its mean success rate is up to 0.975 and the average MSE is about 0.0014, significantly outperforming Linear-WMA or the MMSE-BC based on the single feature F_{ml}, whose values are about 0.68 and 0.017 respectively.

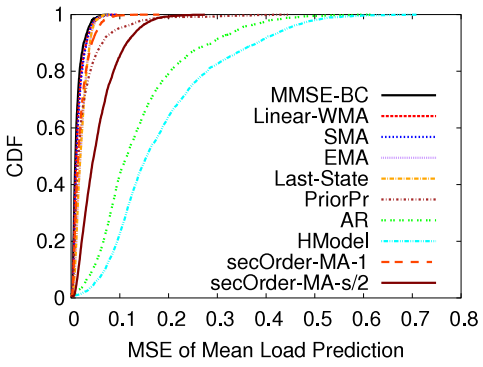
The different prediction results between MMSE-BC(4F) and MMSE-BC(F_{ml}) across the two evaluation types is mainly due to the different sets of samples. The host load in one is dissimilar to that in the test period, while the other one has the similar distribution. With more historical trace data, the test period should be more consistent with the training period, which can be confirmed in the evaluate type C.

C. *Experiments with evaluation type* C.

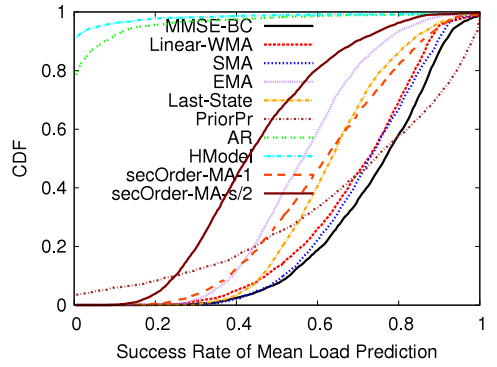
In the above text, we compare the prediction effects under different mean hostload prediction methods, based on the original trace with insufficient load samples or the ideal case with hypothetically sufficient load samples in the training period. In



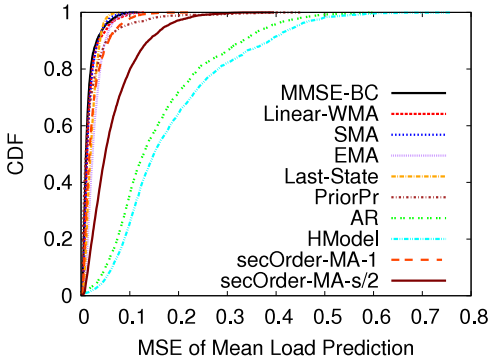
(a) Success rate ($s = 6.4$ h).



(b) MSE ($s = 6.4$ h).

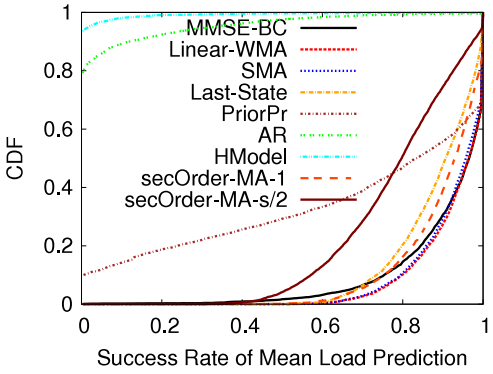


(c) Success rate ($s = 12.8$ h).

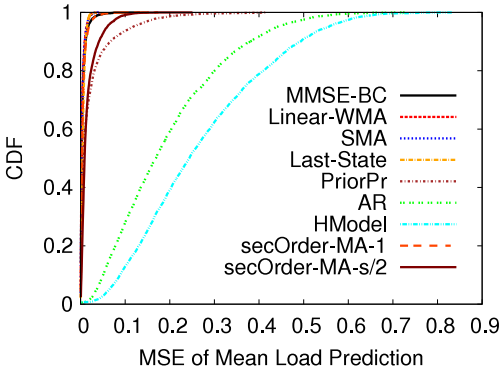


(d) MSE ($s = 12.8$ h).

Fig. 5. CDF of prediction (CPU load with evaluation type A).



(a) Success rate ($s = 6.4$ h).



(b) MSE ($s = 6.4$ h).

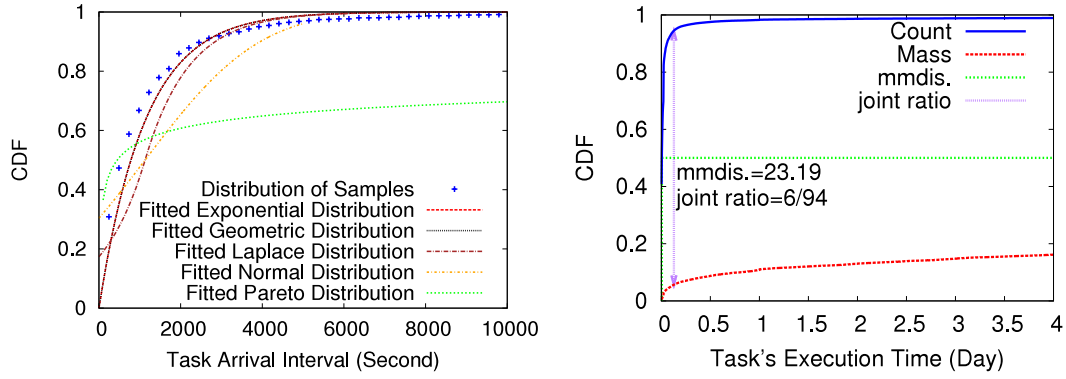
Fig. 6. CDF of prediction (memory load with evaluation type A).

this part, we evaluate the mean hostload prediction effect, in the situation with a relatively large amount of load samples given in a 11-month training period.

We emulate one-year hostload series for 11 k hosts, based on various probability distributions about task events and usage reported in the original one-month Google trace. On each host, thousands of tasks are reproduced based on the probability distributions characterized, with respect to task arrival rate, task execution length and task resource utilization.

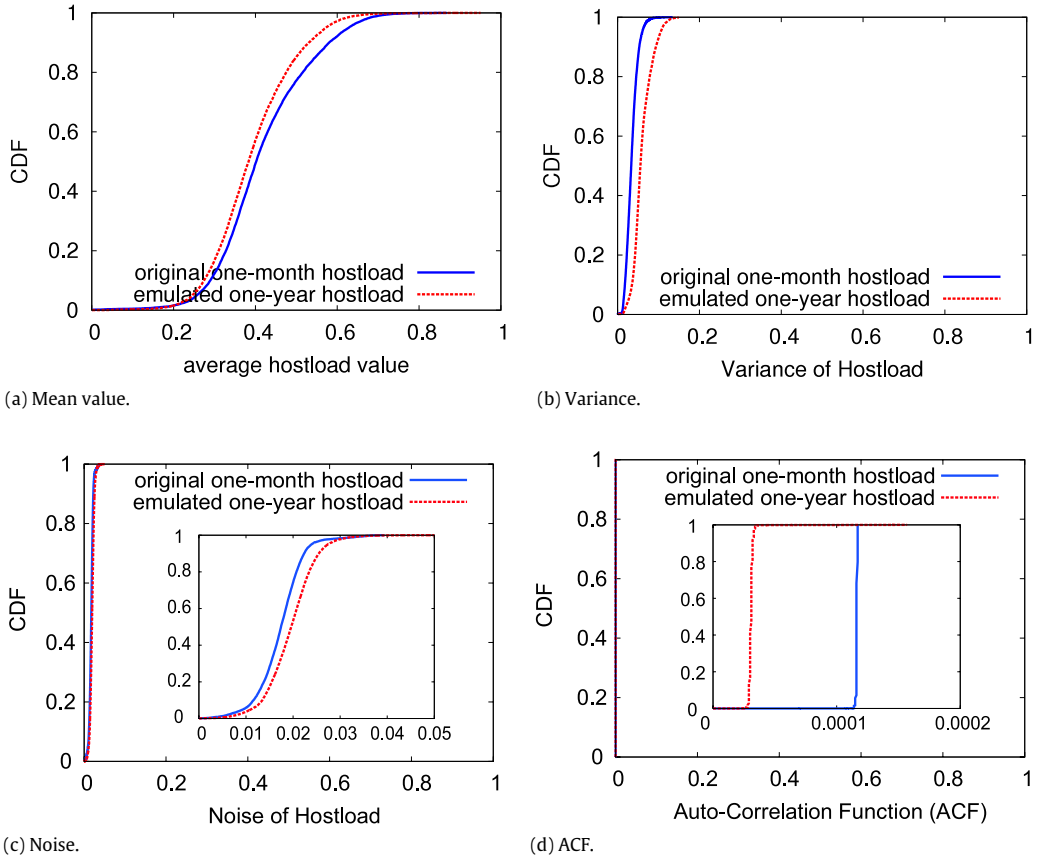
Specifically, we find that the cumulative distribution function (CDF) of task arrival interval on each host matches the exponential distribution mostly among well-known distributions, which means the task arrival rate follows a Poisson-similar process. Fig. 7(a) and (b) shows the distribution of task arrival interval and task execution length respectively. In Fig. 7(a), we also present the maximum likelihood based distribution curves based on different distribution functions. Fig. 7(b) shows the mass and count disparity of task length. It is observed that the task length follows a Pareto-similar distribution, where majority of tasks are of short execution length. Specifically, 94% of the tasks only account for 6% of the summed execution length and 6% of the tasks account for 94% of the execution length.

In our one-year hostload emulation, there are two steps to reproduce a Google task. (1) task arrival emulation: we list all of task arrival intervals for each host based on the one-month trace and randomly make selections over time. (2) task usage emulation: we extract the usage trace for each task in the one-month Google trace, and randomly select one task upon a task arrival. Such a one-year hostload emulation for the 10 k+ machines is fairly close to the real situation, with respect to the mean hostload value, variance of



(a) Task arrival interval. (b) Task length.

Fig. 7. Distribution of task properties.



(a) Mean value. (b) Variance.

(c) Noise. (d) ACF.

Fig. 8. CDF of hostload measurements (original trace vs. emulation).

hostload, noise and auto-correlation function (ACF) of the hostload series. We present the evaluation of the emulated hostload vs. the real hostload in Fig. 8.

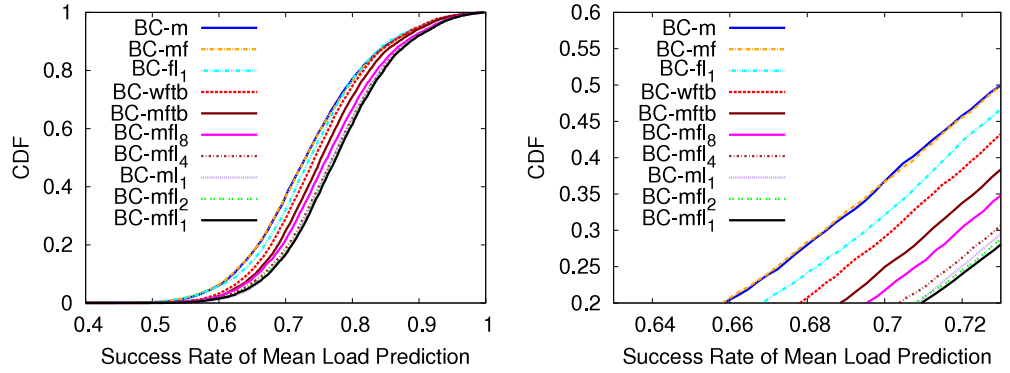
We evaluate all the prediction methods with the one-year hostload series. We set the training period and test period to the first 11 months and the last month respectively. We still set the evidence window length to the half of prediction length, since this results in the best prediction effect as observed. We not only investigate the best-fit feature combination under Bayes Model, but also compare our Bayes method to other well-known prediction methods.

C.1. *Investigation of best-fit feature combination.*

We evaluated the compatible combination of 10 features, F_{ml}, F_{wml}, F_{fi}, F_{ndfi}, F_{ts}, F_{fll}, F_{ll}, F_{ll2}, F_{ll4}, and F_{ll8}. The first 7 features refer to mean load, weighted mean load, fairness index, noise-decreased fairness index, type state, first-last load (a.k.a., both-ends), and last-load respectively. F_{ll2} (F_{ll4}, F_{ll8}) computes the mean of the last two (four, eight) hostload values in the evidence window as one feature. For the simplicity of representation, we denote the 10 features as $m, w, f, n, t, b, l_1, l_2, l_4$, and l_8 respectively. Fig. 9 presents the prediction accuracies based on different feature combinations under the Bayes Model, using the emulated one-year hostload series. Through Fig. 9(b), it is observed that the best-fit feature combination is BC-mfl$_1$ = $\{F_{ml}, F_{fi}, F_{ll}\}$. It is far better than the two feature combinations ($\{F_{ml}\}$ and $\{F_{ml}, F_{fi}, F_{ts}, F_{fll}\}$)—the best choices in evaluation type A and B.

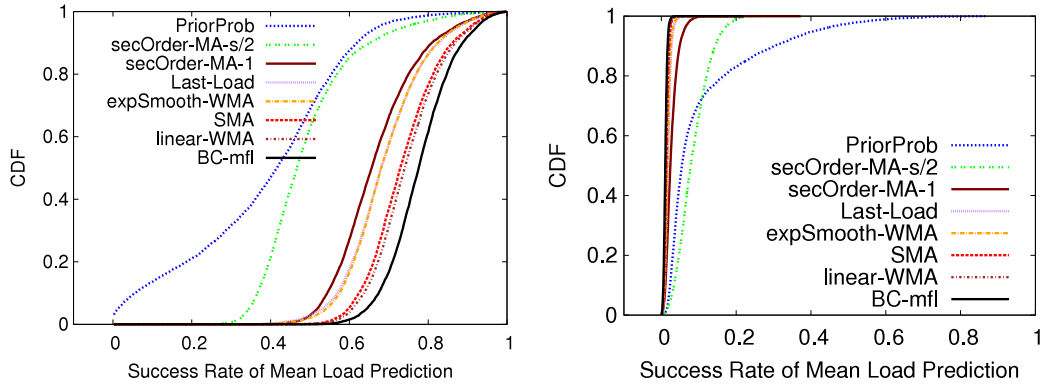
C.2. *Comparison of Bayes method to other methods.*

We compare the prediction effect of the Bayes method to those of other well-known prediction methods, as shown in Fig. 10 (ev-



(a) Success rate = [0, 1]. (b) Success rate = [0.63, 0.73].

Fig. 9. Mean load prediction effect under Bayes model.



(a) Success rate. (b) MSE.

Fig. 10. Mean load prediction effect (Bayes model vs. others).

idence window length = 1.6 h, prediction length $s = 3.2$ h). We evaluate all of the methods with the emulated one-year host load over 11 k hosts and characterize the distribution of success rate and MSE. It is observed that for the 11 k hosts, the success rate of Bayes method (with the combination of three features $\{F_{ml}, F_{fi}, F_{ll}\}$) is better than that of the second one (Linear-WMA) by 4.13% on average. By comparing to Fig. 5(a) (average success rate = 1.5%), we confirm that the prediction effect can be improved a lot by leveraging multiple features on longer training period with more samples. In addition, the lowest success rates under Bayes method, Linear-WMA, SMA, lastLoad, expSmooth-WMA, and secOrder_MA-1 are 50.91%, 45.97%, 44.91%, 33.6%, 35.14%, and 37.35% respectively. This means Bayes method gets at least $\frac{50.91\%}{45.97\%} - 1 = 10.8\%$ improvement compared to other approaches, at the worst case.

4.4.2. Evaluation of pattern prediction effect

Finally, we evaluate the prediction effect of our pattern prediction model (Algorithm 1), by performing the pattern prediction every 2 min over Google's trace. The total number of pattern prediction events in the test period (day 26–day 29) is about $11\,000 \times \frac{4\times 86400}{120} \approx 31.7$ million. We first evaluate the overall prediction errors and present the snapshot of prediction effect with Bayes method. And then, we further evaluate the precision in predicting idle/busy hosts for the purpose of load balancing.

A. *Overall prediction errors and snapshots.*

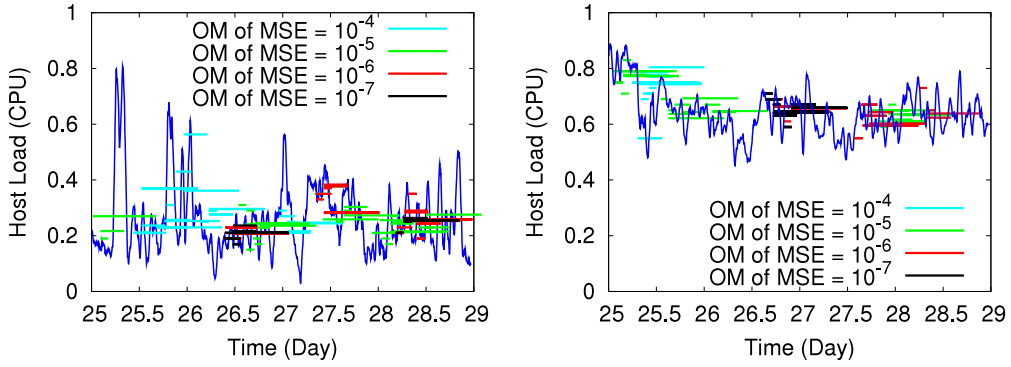
The mean prediction error is computed by Formula (17), where the notations are defined as the same as in Formula (16). Based on our experiments, the mean errors on majority of machines can be limited under 2×10^{-5}. The MSE of MMSE-BC(F_{ml}) is near to 10^{-5}, and MMSE-BC(4F)'s is about 10^{-6} with high probability. That is, Bayes Classifier outperforms other methods on pattern prediction, in that other methods suffer from remarkable errors. More details can be found in our previous work [10].

$$\bar{e}(s) = \frac{1}{s}\sum_{i=1}^{n} s_i \left| l_i - L_i \right|. \tag{17}$$

We also illustrate the pattern prediction of Algorithm 1, based on the Bayes method with the single feature F_{ml}. The experiment is performed using evaluation type A, with two hosts that have different levels of load fluctuation. In Fig. 11, we show that the predicted segment load matches on average the true load. In particular, the dark blue, jittery line represents the true host load. The y-value of each horizontal line represents the predicted load value over some segment. The duration of this segment is given by the horizontal segment length. The segment color gives the order of magnitude of the MSE. For clarity, we randomly chose a subset of segments to illustrate in the figure, based on different orders of magnitude (OM) of the MSE. Note that the horizontal line segments shown do not indicate the mean load predictions, but the ones derived from Formula (1) (i.e., l_i shown in Fig. 1). We observe that the line segments (black or red lines) with MSE of about 10^{-5} or lower are quite close to the real fluctuation of the host load (the dark blue curve).

B. *Prediction precision in context of load balancing.*

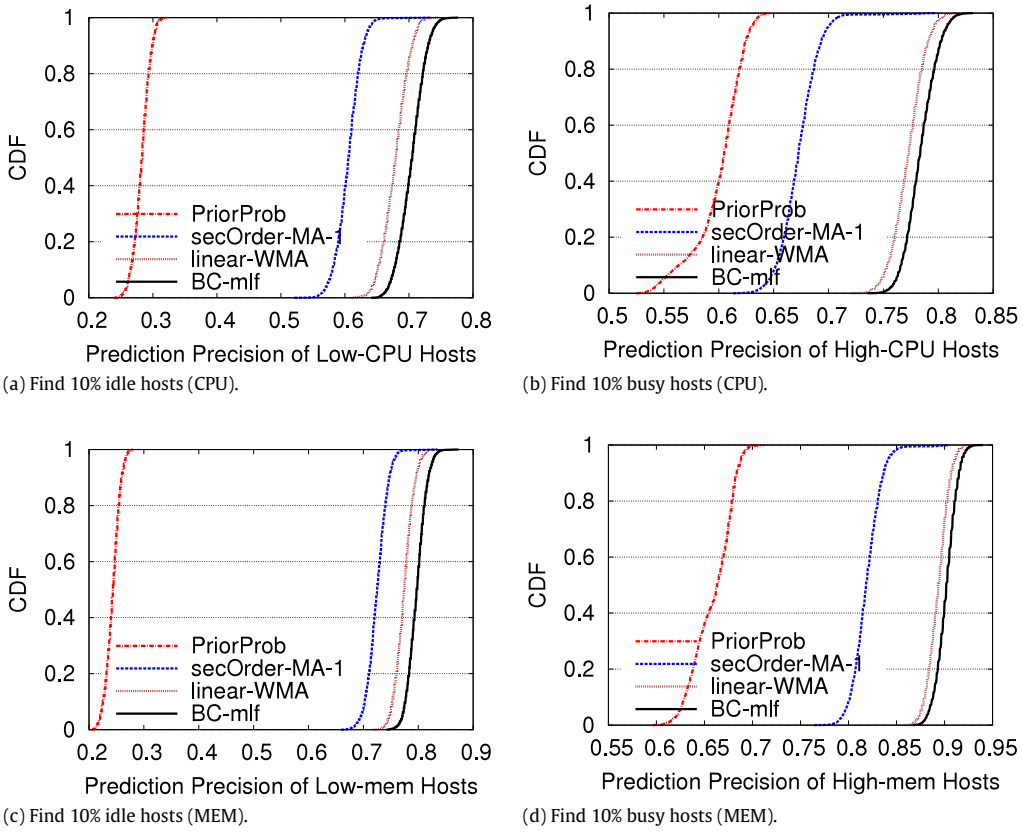
We evaluate different prediction methods in the context of load balancing scenario. Load balancing is widely used for improving the resource usage in distributed systems, and precisely finding a set of idle or busy hosts based on a threshold is a preliminary condition for efficient load balancing. We perform the node searching every two minutes in the one-month experiment, aiming to find 1 k idlest hosts and 1 k busiest hosts from among the totally



(a) Load with wide amplitude.

(b) Load with narrow amplitude.

Fig. 11. Snapshot of pattern prediction. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) Find 10% idle hosts (CPU).

(b) Find 10% busy hosts (CPU).

(c) Find 10% idle hosts (MEM).

(d) Find 10% busy hosts (MEM).

Fig. 12. CDF of prediction in the context of load balancing.

10 k hosts, with respect to CPU and memory usage respectively. The prediction period is always set as the future segment interval [96 min, 192 min] compared to the current moment. We present the precision in finding the 10% of idlest/busiest hosts in Fig. 12. It is clearly observed that the best solution is our Bayes method with the three features $\{F_{ml}, F_{fi}, F_{ll}\}$. In particular, the average precisions of Bayes method in predicting 10% idlest hosts with respect to CPU and memory usage, are higher than those of the linear-WMA method by 7.8% and 6.4% respectively.

5. Related work

At present, most of work on workload characterization and prediction for Cloud systems mainly focus on tasks' placement constraints [28], tasks' usage shapes [37] or task workload prediction [17]. Sharma et al. [28] carefully studied the performance impact of task placement constraints based on the resource utilization from the view of tasks, while Zhang et al. [37] designed a model that can accurately characterize the task usage shapes in Google's compute clusters. Barnes et al. [4] introduced a regression-based approach for predicting the parallel application's workload, and the prediction errors are between 6.2% and 17.3%. Ganapathi et al. [17] adopted a statistical model, namely *Kernel Canonical Correlation Analysis (KCCA)*, to model and predict the workload for map-reduce jobs [8] based on Hadoop Distributed File System (HDFS) [29]. Li et al. [23] proposed a *CloudProphet* framework to predict the application performance, in terms of the trace generated by a prototype. Jackson et al. [19] provided a performance analysis of HPC applications on the Amazon Web Service platform.

Although there exist some host load prediction methods proposed [6,7,1,11,38,34,31,36,35], most are designed for traditional distributed systems such as Grid platforms. Carrington et al. [6], for example, predict the load values based on convolution that maps a

scientific application's signature (a set of fundamental operations) onto a machine profile. Dabrowski et al. [7] perform the host load prediction by leveraging the Markov model via a simulated environment. Akioka, et al. [1] combine the Markov model and seasonal variance analysis to predict the host load values only for the next moment (next discretized time point) on a computational Grid. There also exist some regression based methods (such as polynomial fitting [38] or auto-regression [11]) in the Grid host load prediction work.

Cloud host load observed via Google's trace has more drastic fluctuation and higher noise than that in Grid systems [9]. Based on our experiments, the improved AR method that is recursively performed on predicted values for long-term mean load prediction suffers quite low prediction accuracy, due to the accumulated prediction errors. The HModel [36] also suffers significant prediction errors, since it not only relies heavily on the load values recursively predicted by AR method but filters noises by filtering methods. Hence, it is necessary to revisit Cloud host load prediction such as Google's.

Some load prediction methods are proposed for adapting Cloud systems. Khan et al. [22] proposed a model to capture the CPU workload auto-correlation within and across data centers, also by leveraging Hidden Markov Model (HMM). The effectiveness of their method, however, strongly relies on the assumption that the host loads are predictable such that the time series must follow some repeatable patterns. Moreover, their method is fairly complex and time-consuming, because it needs to iteratively analyze the (auto)correlation within and across multiple time series. Saripalli et al. [27] adopted a cubic spline Interpolation to predict the load trend and used a hotspot detection algorithm for sudden spikes. As confirmed in their experiment, such a method can only predict short-term future host load with a reasonable accuracy. This is due to the fact that the Cloud host load usually fluctuates with large noises and variable patterns, such that no fixed linear models like curve-fitting [15] or auto-regression could always fit it well. Prevost et al. [26] aimed to predict the number of requests for NASA data centers, mainly by auto-regressive filter, which cannot fit long-term Google host load prediction due to significantly larger noises based on our experiments.

In this paper, we propose a more flexible and effective host load prediction in Cloud data centers. In order to predict the fluctuation patterns for the long-term future period, we comprehensively studied 10 features extracted from a moving evidence window under Bayes model, and explored the best-fit choices for different cases. We address 3 advantages about the Bayes method: (1) through probability theory, Bayes method can retain important information like noise, rather than filtering them, such that the predicted states can be consistent with the true noisy load values; (2) Bayes method is pretty flexible, because it is not limited to any linear model [38,11,15] or load fluctuation pattern [22,7,1] and the features can be combined based on different cases and user demands; (3) Bayes method suffers quite low computation complexity (and small disk space size) as it just needs to compute (and keep) a set of probability values related to load fluctuation features. Through Google trace, the Bayes method significantly outperforms other methods (including auto-regression, moving average, noise-filtering) from the perspective of success rate and mean squared error (MSE).

6. Conclusion and future work

We design a Cloud load prediction method, and evaluate it using Google's one-month trace and one-year host loads emulated based on consistent probability. Our objective is to predict the load fluctuation patterns for long-term prediction intervals. We first reduce the pattern prediction to a set of mean load predictions, each starting from the current time and being with different prediction lengths. Then, we design a mean load prediction approach by leveraging Bayes model with 10 interesting features. We explore the best-fit combinations of the features with different sufficient levels of samples. To the best of our knowledge, this is the first attempt to make use of the Bayes model to predict the host load, especially for the long prediction length in Cloud data centers. With Google's large-scale trace, our designed Bayes method outperforms other solutions by 5.6%–50% in the long-term prediction. With insufficient samples, the best-fit feature combination is the one with the single mean load features. With more samples, more features selected appropriately (e.g., $\{F_{ml}, F_{fi},$ and $F_{ll}\}$ and $\{F_{ml}, F_{fi}, F_{ts},$ and $F_{fll}\}$) can prominently improve the prediction accuracy. Such an improvement is regardless of how drastic the host load fluctuates. The MSE of predicting the load fluctuation patterns for the long-term interval is usually low within the range $[10^{-8}, 10^{-5}]$. Through a load balancing scenario, we confirm the pattern prediction's precision in finding a set of idlest/busiest hosts from among totally 10 k hosts can be improved by about 7% on average.

Acknowledgments

We thank Google Inc, in particular Charles Reiss and John Wilkes, for making their invaluable trace data available. This work was made possible by a Google Research Award and by the ANR project Clouds@home (ANR-09-JCJC-0056-01).

References

[1] S. Akioka, Y. Muraoka, Extended forecast of CPU and network load on computational grid, in: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid, CCGRID'04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 765–772.
[2] E. Alessio, A. Carbone, G. Castelli, V. Frappietro, Second-order moving average and scaling of stochastic time series, Eur. Phys. J. B 27 (2) (2002) 197–200.
[3] Auvergrid: Online at: http://www.auvergrid.fr/.
[4] B.J. Barnes, B. Rountree, D.K. Lowenthal, J. Reeves, B. de Supinski, M. Schulz, A regression-based approach to scalability prediction, in: Proceedings of the 22nd Annual International Conference on Supercomputing, ICS'08, ACM, New York, NY, USA, 2008, pp. 368–377.
[5] J.O. Berger, Statistical Decision Theory and Bayesian Analysis, Springer-Verlag, New York, 1985.
[6] L. Carrington, A. Snavely, N. Wolter, A performance prediction framework for scientific applications, Future Gener. Comput. Syst. 22 (2006) 336–346.
[7] C. Dabrowski, F. Hunt, Using Markov chain analysis to study dynamic behaviour in large-scale grid systems, in: Seventh Australasian Symposium on Grid Computing and e-Research, AusGrid 2009, in: Ser. CRPIT, vol. 99, ACS, Wellington, New Zealand, 2009, pp. 29–40.
[8] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: 5th USENIX Symposium on Operating Systems Design and Implementation, OSDI'04, 2004, pp. 137–150.
[9] S. Di, D. Kondo, W. Cirne, Characterization and comparison of cloud versus grid workloads, in: Proceedings of the IEEE International Conference on Cluster Computing, Cluster'12, 2012, pp. 230–238.
[10] S. Di, D. Kondo, W. Cirne, Host load prediction in a Google compute cloud with a Bayesian model, in: Proceedings of the IEEE/ACM Conference on High Performance Computing Networking, Storage and Analysis, SC'12, 2012, pp. 21:1–21:11.
[11] P.A. Dinda, D.R. O'Hallaron, Host load prediction using linear models, Clust. Comput. 3 (2000) 265–280.
[12] P. Domingos, M.J. Pazzani, Beyond independence: conditions for the optimality of the simple Bayesian classifier, in: Proceedings of the 13th International Conference on Machine Learning, ICML'96, 1996, pp. 105–112.
[13] P. Domingos, M. Pazzani, On the optimality of the simple Bayesian classifier under zero–one loss, Mach. Learn. 29 (2–3) (1997) 103–130.
[14] M.H. Dunham, Data Mining: Introductory and Advanced Topics, first ed., Prentice Hall, 2002.
[15] M.A. Farahat, M. Talaat, A new approach for short-term load forecasting using curve fitting prediction optimized by genetic algorithms, in: Proceedings of the 14th International Middle East Power Systems Conference, MEPCON'10, 2010, pp. 106–110.
[16] I. Foster, C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, in: The Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann, 2003.
[17] A. Ganapathi, Y. Chen, A. Fox, R.H. Katz, D.A. Patterson, Statistics-driven workload modeling for the cloud, in: ICDE Workshops'10, 2010, pp. 87–92.
[18] Google Cluster-Usage Traces: Online at: http://code.google.com/p/googleclusterdata.

[19] K.R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H.J. Wasserman, N.J. Wright, Performance analysis of high performance computing applications on the amazon web services cloud, in: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, CloudCom'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 159–168.
[20] R.K. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling, John Wiley & Sons, 1991.
[21] R.E. Kalman, A new approach to linear filtering and prediction problems, Trans. ASME D 82 (1960) 35–45.
[22] A. Khan, X. Yan, S. Tao, N. Anerousis, Workload characterization and prediction in the cloud: a multiple time series approach, in: 3rd IEEE/IFIP International Workshop on Cloud Management, Cloudman'12, 2012.
[23] A. Li, X. Zong, S. Kandula, X. Yang, M. Zhang, Cloudprophet: towards application performance prediction in cloud, in: ACM SIGCOMM Student Poster, 2011.
[24] A.K. Mishra, J.L. Hellerstein, W. Cirne, C.R. Das, Towards characterizing cloud backend workloads: insights from Google compute clusters, SIGMETRICS Perform. Eval. Rev. 37 (4) (2010) 34–41.
[25] S.J. Orfanidis, Introduction to Signal Processing, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.
[26] J.J. Prevost, K. Nagothu, B. Kelley, M. Jamshidi, Prediction of cloud data center networks loads using stochastic and nearal models, in: 6th International Conference on System of Systems Engineering, SoSE'11, 2011, pp. 276–281.
[27] P. Saripalli, G.V.R. Kiran, R.R. Shankar, H. Narware, N. Bindal, Load prediction and hot spot detection models for autonomic cloud computing, in: 4th IEEE International Conference on Utility and Cloud Computing, UCC'11, 2011, pp. 394–402.
[28] B. Sharma, V. Chudnovsky, J.L. Hellerstein, R. Rifaat, C.R. Das, Modeling and synthesizing task placement constraints in Google compute clusters, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, SOCC'11, ACM, New York, USA, 2011, pp. 3:1–3:14.
[29] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST'10, May 2010, pp. 1–10.
[30] The Grid Workloads archive: Online at: http://gwa.ewi.tudelft.nl/pmwiki/.
[31] J.M. Tirado, D. Higuero, F. Isaila, J. Carretero, Multi-model prediction for enhancing content locality in elastic server infrastructures, in: Proceedings of 18th High Performance Computing, HiPC'2011, 2011.
[32] G.I. Webb, J.R. Boughton, Z. Wang, Not so naive Bayes: aggregating one-dependence estimators, Mach. Learn. 58 (1) (2005) 5–24.
[33] J. Wilkes, More Google Cluster Data, Google Resarch Blog, November 2011. Online at: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.
[34] R. Wolski, Dynamically forecasting network performance using the network weather service, Clust. Comput. 1 (1998) 119–132.
[35] Y. Wu, K. Hwang, Y. Yuan, W. Zheng, Adaptive workload prediction of grid performance in confidence windows, IEEE Trans. Parallel Distrib. Syst. 21 (7) (2010) 925–938.
[36] Y. Wu, Y. Yuan, G. Yang, W. Zheng, Load prediction using hybrid model for computational grid, in: The 8th IEEE/ACM International Conference on Grid Computing, Grid'07, 2007, pp. 235–242.
[37] Q. Zhang, J.L. Hellerstein, R. Boutaba, Characterizing task usage shapes in Google's compute clusters, in: Large Scale Distributed Systems and Middleware Workshop, LADIS'11, 2011.
[38] Y. Zhang, W. Sun, Y. Inoguchi, CPU load predictions on the computational grid, in: IEEE International Symposium on Cluster Computing and the Grid, CCGrid'2006, Los Alamitos, CA, USA, 2006, pp. 321–326.

Sheng Di received his Master (M.Phil.) degree from Huazhong University of Science and Technology in 2007 and Ph.D. degree from The University of Hong Kong in 2011, both in Computer Science. He is currently a post-doctoral researcher at INRIA. Dr. Di's research interest involves the optimization of distributed resource allocation in large-scale cloud platforms, characterization and prediction of workload at Cloud data centers, and so on. His background is mainly on the fundamental theoretical analysis and practical system implementation. Contact him at the MESCAL Group, INRIA, Grenoble, sheng.di@inria.fr.

Derrick Kondo is a tenured research scientist at INRIA, France. He received his Bachelor's at Stanford University in 1999, and his Master's and Ph.D. at the University of California at San Diego in 2005, all in computer science. His general research interests are in the areas of reliability, fault-tolerance, statistical analysis, job and resource management. His research projects are supported by the national, European, and industrial grants. In 2009, he received a Young Researcher Award (similar to NSF's CAREER Award). He received an Amazon Research Award in 2010, and Google Research Award in 2011. He is the co-founder of the Failure Trace Archive, which serves as a public repository of failure traces and algorithms for distributed systems. He has won numerous best paper awards at IEEE/ACM conferences for work in these projects. Contact him at the MESCAL Group, INRIA, Grenoble, derrick.kondo@inria.fr.

Walfredo Cirne is with Google infrastructure group in California, USA, where he works on cluster management. He is on leave of his faculty position at the Computer Science Department of the Universidade Federal de Campina Grande, in Brazil. Dr. Cirne holds a Ph.D. degree from the University of California San Diego, in the USA. Since 1997, his research focuses on the Distributed Systems and Resource Management, having led the OurGrid project from 2001 to 2006. Contact him at walfredo@google.com.