Host Load Prediction in a Google Compute Cloud with a Bayesian Model

Sheng Di[1], Derrick Kondo[1], Walfredo Cirne[2]
[1]INRIA, France, [2]Google Inc., USA
{sheng.di,derrick.kondo}@inria.fr, walfredo@google.com

***Abstract*—Prediction of host load in Cloud systems is critical for achieving service-level agreements. However, accurate prediction of host load in Clouds is extremely challenging because it fluctuates drastically at small timescales. We design a prediction method based on Bayes model to predict the mean load over a long-term time interval, as well as the mean load in consecutive future time intervals. We identify novel predictive features of host load that capture the expectation, predictability, trends and patterns of host load. We also determine the most effective combinations of these features for prediction. We evaluate our method using a detailed one-month trace of a Google data center with thousands of machines. Experiments show that the Bayes method achieves high accuracy with a mean squared error of 0.0014. Moreover, the Bayes method improves the load prediction accuracy by 5.6-50% compared to other state-of-the-art methods based on moving averages, auto-regression, and/or noise filters.**

I. Introduction

Accurate prediction of 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 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 job performance.

Compared with traditional Grids and HPC systems, host load prediction in Cloud data centers is arguably more challenging as it has higher variance. This stems from differences in the workloads run 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 [9], [10], [11] and the AuverGrid cluster [12], 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 fluctuations in Clouds compared to Grids.

Most prior work in Cloud Computing has focused primarily on application workload characterization versus long-term host load prediction. For instance, there are several work on characterizing task placement constraints [13], task usage shape [14], and its impact on host load [11].

Most prior prediction work in Grid Computing or HPC systems [1], [2], [3], [4], [5], [6], [7], [8] 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 long-term future load over consecutive time intervals.

In this paper, we design an effective Cloud load prediction method that can accurately predict host load over a long-term period up to 16 hours in length. We focus on two critical metrics, CPU and memory, highlighted in [11]. Our approach is to use a Bayesian model for prediction as it effectively retains the important information about load fluctuation and noise. We evaluate our prediction method using a detailed 1-month load trace of a Google data center with thousands of machines.

In particular, our contributions are as follows:

- *What we predict.* We accurately predict both mean load over a future time interval (up to 16 hours), and also mean load over consecutive future time intervals (which we refer to as a **pattern**).
- *How we predict.* We craft novel features used for Bayesian prediction that capture important and predictive statistical properties of host load. These properties include the expectation, predictability, 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. To the best of our knowledge, this is one of the first works to show the effectiveness of a Bayesian model for host load prediction in the context of Cloud Computing.
- *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 7 other baseline and state-of-the-art methods that use a variety of techniques, including moving averages, noise filters, and auto-regression. Our Bayesian method outperforms others by 5.6-50%. In absolute terms, the mean-squared error (MSE) of the Bayesian method for a single interval is 0.0014, and for a pattern is less than or equal to 10^{-5}.

For the remainder of the paper, we use the terms **host load**, **cloud load** and **load** interchangeably. In Section III, we formulate the Cloud load prediction problem. In Section II, we carefully analyze the prediction problem from two perspectives, characterizing Cloud host load for

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00

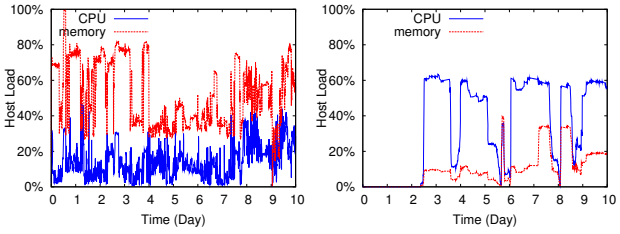
the succeeding prediction work and simplifying the pattern prediction model to a set of more straight-forward mean load prediction steps. In Section IV, we present the overall design of the Bayes classifier and propose 9 candidate features to be used as the evidence for prediction, and we analyze their 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 V and discuss related work in Section VI. Finally, we conclude the paper with the future work in Section VII.

II. Google Load Measurements and Characterization

Our predictive study is based on load measurements of a Google data center. Google [9] traced over 670,000 jobs and over 40 million task events at minute resolution across about 12,000 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.

Host load at a given time point is the total load of all running tasks on that particular machine. By leveraging Google's machine event trace, which contains each host's (re-scaled) capacity, we calculate the relative load values by dividing the absolute load values by the corresponding capacities. Thus, the load values range between 0 and 1 for each resource (such as memory and CPU). Then, we discretize all the load values by recomputing each host's relative load over consecutive fixed-length periods, each having length on the order of a few minutes. This discretized load trace is the basis of this work.

To motivate host load prediction in Clouds, we compare the characteristics of the load fluctuation in Cloud data centers with that in Grids. For comparison with the Google trace, we use a one-year load trace from the Grid called AuverGrid [12]. For this platform, we also compute the relative host load at each period, using the same method applied to the Google trace.

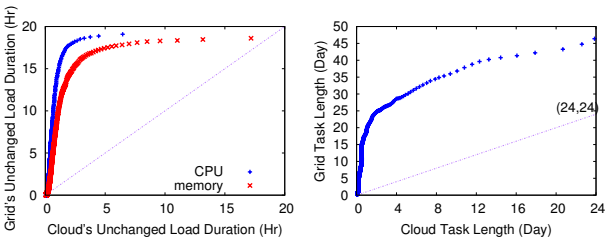


(a) Google's Host Load (b) AuverGrid's Host Load

Figure 1. Load Comparison between Google & AuverGrid

In Figure 1, we show CPU and memory load for one Google host (left), and one AuverGrid host (right). Clearly, Google load exhibits higher noise compared with AuverGrid. In particular, the minimum/mean/maximum noise of AuverGrid's CPU load over all hosts computed using a mean filter [18] are 0.00008, 0.0011, 0.0026 respectively. For Google, the minimum/mean/maximum noise are 0.00024, 0.028, 0.081 respectively.

We also compare the distributions of host load using a quantile-quantile plot [19] (see Figure 2). We split the range of load values into five sub-ranges ([0,20%],[20%,40%],$\cdots$,[80%,1]). A load duration is defined to be the length of time where load on a host constantly stays within a single sub-range. We compare the distribution of these load durations between Google and AuverGrid in Figure 2. The figure shows the points at which the load durations from AuverGrid and Google have the same probability. It is clear that Google's host load changes much more frequently than that of AuverGrid, which is consistent with Figure 1.



(a) QQPlot of Unchanged Duration (b) QQplot of Task Length

Figure 2. Quantile-Quantile Plot of Continuous Duration and Task Length

Such a drastic load fluctuation in Google's trace is mainly due to the fact that Google's tasks are much shorter than AuverGrid's. Figure 2 (b) shows the QQ-Plot of task length between the two platforms. It is observed that with the same probability, the ratio of the Google's task size to AuverGrid's task size is usually $[\frac{1}{20}, \frac{1}{2}]$, which leads to much finer resource allocation on Google's data centers.

The differences in load fluctuation between Cloud and Grid systems introduces new challenges for accurate load prediction, especially for long-term intervals. Curve-fitting solutions [5], [20] or auto-regression methods [4], [8] may not be as effective in Clouds (versus Grids) due to the drastic fluctuations of host load. Also, filtering the noise of host load in Clouds may remove important and real fluctuations, required for accurate prediction. So conventional load prediction solutions that use noise filtering [8], which have been effective for Grid systems, cannot be directly used in the Cloud data centers. For the reasons, we have to take a new (Bayesian) approach for prediction.

III. Prediction Formulation

Our objective is to predict the fluctuation of host load over a long-term period, and our aim is two-fold. 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 a 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 Figure 3 an example of ESP. We denote the total prediction interval length as s. The first segment (denoted by s_1) is 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, $\cdots$.

For example, if b is set to 1 hour, the entire prediction interval length s could be equal to 16 (=1+1+2+4+8) hours. For each segment, we predict the mean host load. The mean values are denoted by l_i, where i = 1, 2, 3, $\cdots$.

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, 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 (the majority being less than 1 hour) in length. So, users or schedulers would value prediction of short-term load fluctuations more than long-term ones.

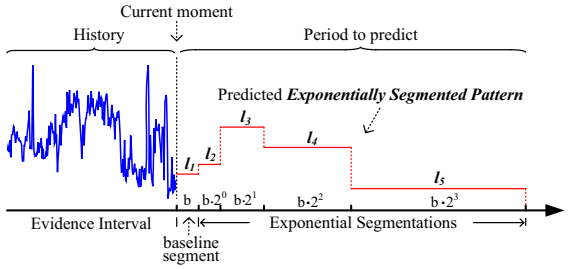


Figure 3. Illustration of Exponentially Segmented Pattern

As illustrated above, our aim is to predict the vector of load values (denoted by $\boldsymbol{l}$=$(l_1,l_2,\cdots,l_n)^T$), 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 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. Without loss of generality, we convert the segment representation formulated in Section III into another one, in which each interval to be predicted 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 η_1, η_2, $\cdots$, η_n, where $\eta_{i+1} = 2 \cdot \eta_i$. The target is to predict such a vector, $\boldsymbol{\eta}$=$(\eta_1, \eta_2, \cdots, \eta_n)^T$, 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 Figure 4.

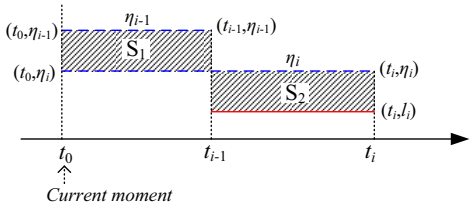


Figure 4. Induction of Segmented Host Load

Suppose 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 Figure 4.

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

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

$$l_i = 2\eta_i - \eta_{i-1} \quad (2)$$

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.

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 l (Figure 3).

Given the prediction problem, one approach for prediction is to use feedback control. One could dynamically validate the prediction accuracy at runtime, adjusting the predicted

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): $\boldsymbol{\eta} \rightarrow \boldsymbol{l}$;

values in the next interval by the error in the previous one. Then, prediction error could converge to a low level. This idea is based on the feed-back control model, which is often used in the one-step look-ahead prediction scenario. For example, a Kalman filter [21] can produce relatively precise estimates of unknown variables, using the recursive one-step process on the current estimates and the validated errors of the previous estimates. However, such an approach is infeasible in our situation in that the validation of the previous estimates always suffers significant lag compared to the current time point. For example, if the prediction interval is set to 16 hours, the current prediction for the mean load value of the future period cannot be validated until 16 hours later. Such a high lag makes the feed-back control hard to apply in a timely manner.

Another approach is to use error of short-interval prediction to tune the long-term prediction. For instance, using the prediction error in a 4-hour interval may forecast the prediction error in the 8-hour interval, such that the predicted values could be tuned accordingly. However, this idea is also inapplicable to Cloud load prediction in that short-term prediction error always lags behind long-term error (see Figure 5). In this figure, we present the error (true mean load − predicted mean load) of the simple moving average prediction method (to be described in Section V) with different prediction lengths, over one-day of one host of a Google trace. We observe that the short-term prediction error almost always lags behind the long-length prediction error, and the lag time is close to the interval length. So, most of the prediction errors cannot be detected in time because the drastic fluctuation of the host load.

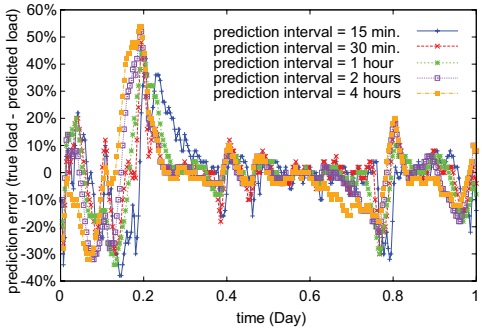


Figure 5. Prediction Errors in Different Prediction Lengths

As we believe feedback control is not effective for long-term prediction, we investigate other methods based on Bayes Model in the following section.

IV. 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 9 key features designed.

A. Bayes Classifier

The Bayes Classifier [15], [22], [16] is a classic supervised learning classifier used in data mining [23]. Bayesian classification consists of five main steps: (1) determine the set of target states (denoted as the vector $\boldsymbol{W}$=(ω_1,ω_2,$\cdots$,ω_m)^T, where m is the number of states), and the evidence vector with h mutually-independent features (denoted as χ=(x_1,x_2,$\cdots$,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.

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

Based on different risk functions, there are two main ways for making decisions, namely Naïve Bayes Classifier (abbreviated as **N-BC**) [16], [22] and Minimized MSE (MMSE) based Bayes Classifier (abbreviated as **MMSE-BC**) [15]. 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 [15], 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 usage level. 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), $\cdots$, [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 hours, the evidence window length will be set to the recent past 4 hours. 48 ($=\frac{4\times60}{5}$) successive load values (if the sample interval is 5 minutes) in this period will serve as the fundamental evidence, based on which we can extract many interesting features for the Bayes prediction. We use Figure 6 to illustrate the discretized evidence window and target load states. In this example, the prediction interval length is assumed to be 4 hours, so the evidence window length is 2 hours 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.

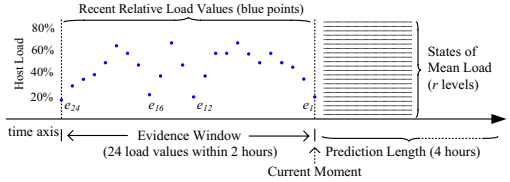


Figure 6. Illustration of Evidence Window and Target Load States

B. 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 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 $\boldsymbol{e}$=$(e_1,e_2,\cdots,e_d)^T$, 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 9 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 Equation (8).

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

  The value range of the mean load is [0,1] in principle, hence, 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),$\cdots$,[0.98,1]. For this feature, its value must be one of the 50 levels, constructing partial evidence for the Bayes Classifier.
- *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 Equation (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 [24] (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).

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

  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 the prediction interval changes with the similar fluctuation rule to the statistics, fairness index could effectively improve the prediction effect, to be confirmed later.
- *noise-decreased fairness index* ($F_{ndfi}(\boldsymbol{e})$): The noise-decreased fairness index is also computed using the fairness index formula. But, if there exist one or two load values (a.k.a. *load outliers*) that may significantly degrade the whole fairness index, they would not be counted in. That is, such load outliers are likely supposed to be considered *noise* or irregular jitters, which is independent of the load levels.
- *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. For example, if the window vector is (0.023,0.042,0.032,0.045,0.056,0.036), then there are only two types (or levels) involved, [0.02,0.04) and [0.04,0.06), yet there are four state changes: 0.023→0.042, 0.042→0.032, 0.032→0.045, 0.056→0.036. Note that 0.056 is not a state change since its preceding state is at the same level.
- *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 length is 4 hours (i.e., the window size is 48), then 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 $[e_{37},e_{48}]$. In our experiment, N is set to 2, 3, and 4 respectively. Hence, there are actually three features w.r.t the N-segment pattern in our experiment.

So far, we have presented 9 features to be used in the Bayes model. Some of them, however, are mutually correlated, which violates the assumption of feature independence in Bayes' theorem. 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 and Spearman's rank correlation coefficients [25] in Table I. 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. In addition, among $F_{2\text{-}sp}$, $F_{3\text{-}sp}$, and $F_{4\text{-}sp}$, the correlation coefficients are always close to 0.999, implying that they are extremely correlated to each other.

Table I
CORRELATION OF THE FEATURES (LINEAR_CORR/RANK_CORR)

	F_{ml}	F_{wml}	F_{fi}	F_{ndfi}	F_{ts}	F_{fll}	$F_{N\text{-}sp}$
F_{ml}	1/1	0.98/0.97	0.46/0.51	0.46/0.51	0.15/0.21	0.82/0.78	0.99/0.99
F_{wml}	0.98/0.97	1/1	0.45/0.5	0.45/0.5	0.15/0.2	0.81/0.76	0.97/0.96
F_{fi}	0.46/0.51	0.45/0.5	1/1	0.99/0.99	0.3/0.43	0.36/0.4	0.47/0.51
F_{ndfi}	0.46/0.51	0.45/0.5	0.99/0.99	1/1	0.3/0.43	0.36/0.4	0.46/0.51
F_{ts}	0.15/0.21	0.15/0.2	0.3/0.43	0.3/0.43	1/1	0.17/0.19	0.17/0.21
F_{fll}	0.82/0.78	0.81/0.76	0.36/0.4	0.36/0.4	0.17/0.19	1/1	0.83/0.79
$F_{N\text{-}sp}$	0.99/0.99	0.97/0.96	0.46/0.51	0.46/0.51	0.17/0.21	0.83/0.79	1/1

Much research on the independence constraint of Bayes Classifier [26], [22], [16] 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 II, based on the Formula (11), where *Comp*(F_x,F_y) and *Corr*(F_x,F_y) refer to the compatibility and correlation coefficient of two features respectively.

$$Comp(F_x(\boldsymbol{e}),F_y(\boldsymbol{e}))=\begin{cases}Y & Corr(F_x(\boldsymbol{e}),F_y(\boldsymbol{e}))\leq 0.83\\ N & Corr(F_x(\boldsymbol{e}),F_y(\boldsymbol{e}))\geq 0.96\end{cases} \quad (11)$$

Two features are considered incompatible iff their correlation coefficients are greater than 0.96, and compatible iff their coefficients are less than 0.83.

Table II
COMPATIBILITY OF THE FEATURES

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

There are only 71 viable combinations of the features, based on the following analysis in terms of the compatibility table. Since there are 9 features (F_{ml}, F_{wml}, F_{fi}, F_{ndfi}, F_{ts}, F_{fll}, $F_{2\text{-}sp}$, $F_{3\text{-}sp}$, $F_{4\text{-}sp}$) in total, the number of their combinations is at most 2^9. Yet, many of the combinations are not viable according to the Table II. For instance, F_{ml} and F_{wml} should not be used together. By observing this table, all 9 features can be classified into 4 groups, $\{F_{ml}$, F_{wml}, $F_{2\text{-}sp}$, $F_{3\text{-}sp}$, $F_{4\text{-}sp}\}$, $\{F_{fi}$, $F_{ndfi}\}$, $\{F_{ts}\}$, 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 four groups are 6, 3, 2 and 2 respectively. Hence, the total number of compatible combinations can be computed as follows.

NCC(9 features) = *NCC*(Group 1) · *NCC*(Group 2) · *NCC*(Group 3) · *NCC*(Group 4) $= 6 \times 3 \times 2 \times 2 = 72$

By excluding the case where no feature is selected, there are 71 viable combinations of the features, all of which will be evaluated in our experiment under the Bayes model.

V. PERFORMANCE EVALUATION

A. Algorithms for Comparison

In addition to our Bayes estimator, we also rigorously and comprehensively implemented seven other load prediction methods. These baseline solutions (listed below) are extensively studied in the load prediction domain, and some of them have been shown to be effective in Grid environments. Under our formulated prediction model, we make them uniformly aim to predict the mean load of the future interval, based on the evidence window.

- *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 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.
- *Exponential Moving Average method* (**EMA**): This predicted value (denoted $S(t)$ at time t) is calculated based on the Formula (12), where e_1 is the last load value and α is tuned empirically to optimize accuracy.

$$S(t) = \alpha \cdot e_1 + (1-\alpha) \cdot S(t-1) \quad (12)$$

- *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 Formula (13), where $X(t)$, p and ε_t refer to the predicted value, the order and the white noise at time point t respectively.

$$X(t) = \sum_{i=1}^{p} \varphi_i e_i + \varepsilon_t \quad (13)$$

In general, the AR method can only predict the load value for the next moment, while previous works [27], [8] extended it to 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 [27] (**HModel**): This method integrates the Kalman filter [21] and Savitzky-Golay smoothing filter [28] 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 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.

B. Method of Training and Evaluation

We split Google's one-month trace data into two durations, a **training period** (from the beginning to the 25th day unless stated otherwise) and the **test period** (always from the 26th day to the end). 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.

During the test period, the length of the evidence interval (or the evidence window length) is always set to the half of the prediction interval length. We found empirically via the Google trace that this maximizes prediction accuracy.

In addition, we optimize the coefficients for the baseline algorithms in Table III. The window length is always set to the half of the prediction length, because often leads to the highest accuracy based on our experiments. This setting is in stark contrast to the conclusion in the study based on Grid traces [8], where using a large window size may lead to higher accuracy. This is mainly due to the fact that Google host load fluctuates much more drastically with higher noise, such that successive load values have a weaker relationship.

Table III
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 split into two situations. They differ depending on whether the load values in the test period are exhibited 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 probably different from those in the test period. We believe this is due to the lack of enough training data. In reality, we would have more than 25 days of training data, and this in turn would improve accuracy.

So, in the second situation (referred to as **evaluation type B**), we emulate the scenario where we have enough training data to observe 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 under our Bayes predictor.

C. 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, $\cdots$, L_n. Then, the value of MSE can be calculated with Formula (14), 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.

$$mse(s) = \frac{1}{s} \sum_{i=1}^{n} s_i (l_i - L_i)^2 \quad (14)$$

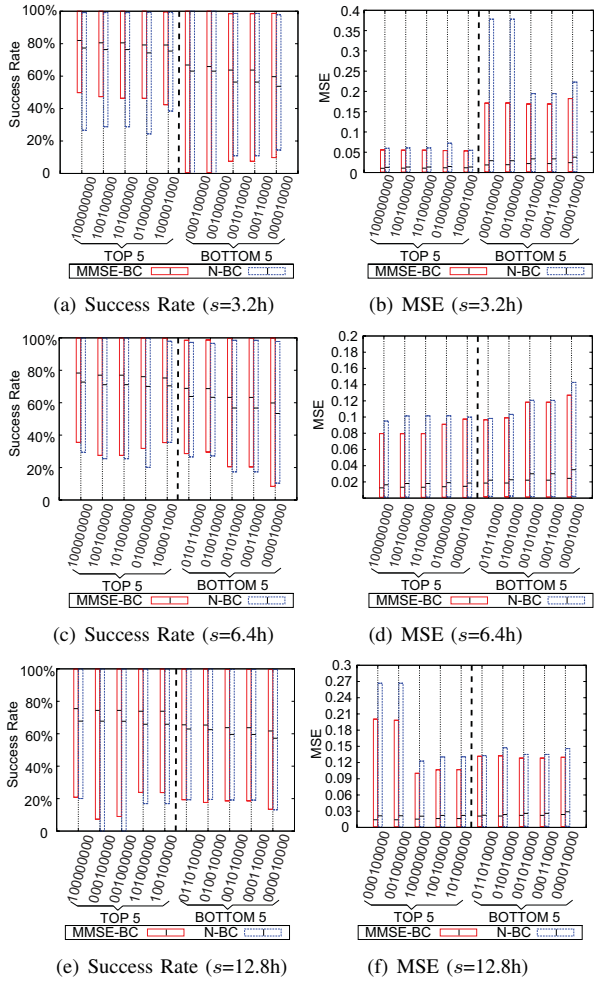
Second, we measure **success rate**, which is defined as the ratio of the number of accurate predictions to total number of predictions. A prediction is deemed accurate if it falls within some delta of the real value. We use a delta of 10%. In general, the higher the success rate, the better, and the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

D. Experimental Results

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

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 71 compatible combinations of our designed features. We denote them via the binary numerical system according to the following order, $\{F_{ml},\ F_{wml},\ F_{fi},\ F_{ndfi},\ F_{ts},\ F_{fll},\ F_{2\text{-}sp},\ F_{3\text{-}sp},\ F_{4\text{-}sp}\}$. For example, 100000000 denotes the single feature F_{ml}, and 101011000 indicates the combination of the four features F_{ml}, F_{fi}, F_{ts}, and F_{fll}. Due to the heavy computation in traversing all 71 combinations, we sampled 2000 machines from among the 12000 machines 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 Figure 7, we present the range of the evaluation results via rectangles, where the bottom-edge, the middle black line, and the upper-edge refer to the minimum, mean, and maximum values respectively. The top 5 (bottom 5) combinations are selected based on MMSE-BC, with either the 5 highest (5 lowest) success rates or 5 lowest (5 highest) MSEs respectively.



(a) Success Rate (s=3.2h) (b) MSE (s=3.2h)

(c) Success Rate (s=6.4h) (d) MSE (s=6.4h)

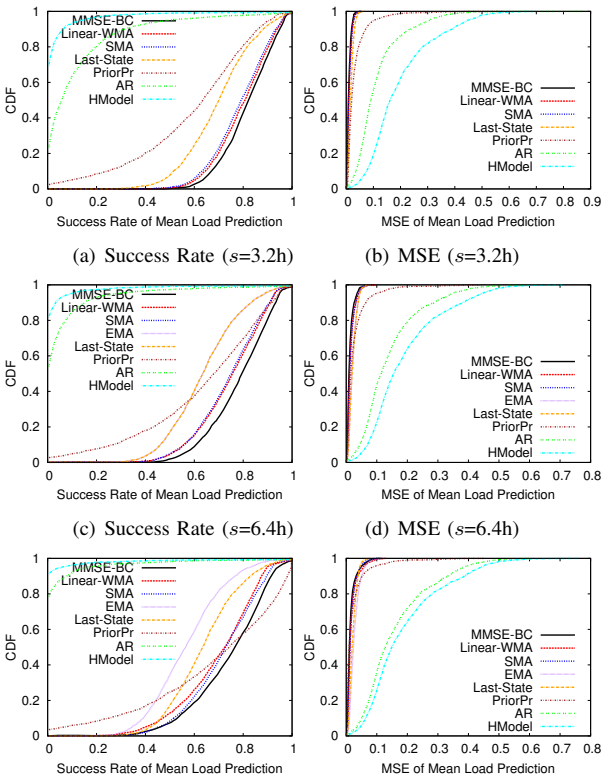
(e) Success Rate (s=12.8h) (f) MSE (s=12.8h)

Figure 7. Success Rate & MSE of Bayes Classifier

In Figure 7, we can see that the best feature combination is always 100000000, while the worst one is always 000010000, 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 7 prediction methods, with respect to the success rate and MSE respectively, by using 11K hosts in the Google trace. Figure 8 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 7 methods. Specifically, its advantage becomes more prominent with the increase of prediction length (See Figure 8 (a), (c), (e) or (b), (d), (f)). Our statistics show that the MMSE-BC's success rate is higher than the second best solution (Linear-WMA) by 5.6% in the 6.4h ahead mean load prediction and by about 7.8% in the 12.8h ahead prediction, and also higher than others by up to 50%.



(a) Success Rate (s=3.2h) (b) MSE (s=3.2h)

(c) Success Rate (s=6.4h) (d) MSE (s=6.4h)

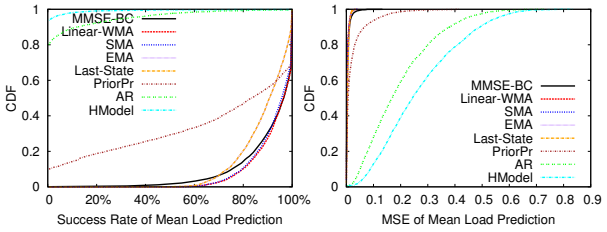
(e) Success Rate (s=12.8h) (f) MSE (s=12.8h)

Figure 8. CDF of Prediction (CPU Load with Evaluation Type A)

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 moving averages perform well in general, they cannot capture features such as predictability . 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, because it filters useful information about load dynamics.

We compare the CDFs of Prediction Methods w.r.t. memory host load in Figure 9. Since the memory load does not fluctuate as drastically as CPU, the MMSE-BC and all moving average solutions work very well, with the average success rate up to 93% and the average MSE down to 0.004 respectively. Our experiment also confirms AR and HModel still suffer with a low success rate and high MSE in predicting the mean memory load of long-term intervals.



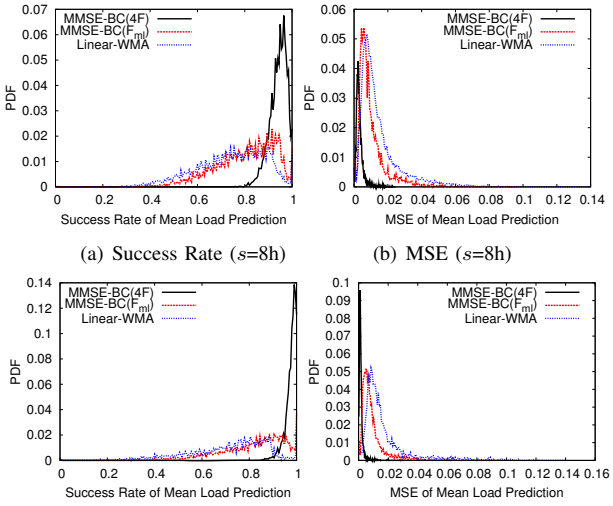
(a) Success Rate (s=6.4h) (b) MSE (s=6.4h)

Figure 9. CDF of Prediction (Memory Load with Evaluation Type A)

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. Next, we emulate the evaluation type B, where the load fluctuation in the test period is similar to that of training period. The discretization interval of the host loads is set to 5 minutes and the prediction length will be set to 8 hours and 16 hours respectively.

We traverse all 71 combinations and observe that the best feature combination is $\{F_{ml}$,F_{fi},F_{ts},$F_{fll}\}$ instead of just the single feature F_{ml}. Figure 10 presents the probability density function (PDF) of three methods. (The other methods perform significantly worse.) We observe that the MMSE-BC under the four features (abbreviated as MMSE-BC(4F)) shows surprisingly high accuracy. 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 corresponding 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 training period should be more consistent with the test period. Hence, we believe MMSE-BC(4F) in reality has the best accuracy when using load samples accumulated for a longer period of time.



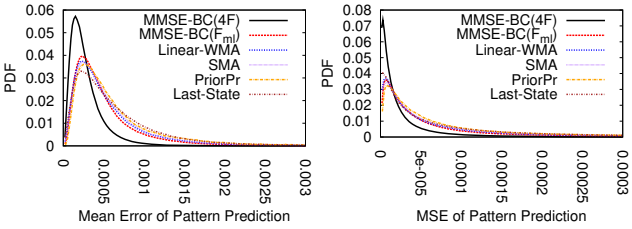
(a) Success Rate (s=8h) (b) MSE (s=8h)

(c) Success Rate (s=16h) (d) MSE (s=16h)

Figure 10. PDF of Prediction (CPU Load with Evaluation Type B)

Finally, we evaluate the prediction effect of our pattern prediction model (Algorithm 1), by performing the pattern prediction every 2 minutes over Google's trace. The total number of pattern prediction events in the test period (day 26 - day 29) is about $11000\times\frac{4\times 86400}{120}\approx$31.7 million. We present the PDF of the prediction error in Figure 11. Figure 11 (a) shows the mean error, which is computed by Formula (15), where the notations are defined as the same as in Formula (14). We observe that the mean errors on majority of machines can be limited under 2×10^{-5}. Figure 11 (b) indicates that the MSE of MMSE-BC(F_{ml}) is near to 10^{-5}, and MMSE-BC(4F)'s is about 10^{-6} with high probability. All in all, Bayes Classifier outperforms other methods on pattern prediction, in that the other methods suffer from the remarkable errors in the mean load prediction and their predicted mean loads are regardless of prediction length.

$$\overline{e}(s) = \frac{1}{s}\sum_{i=1}^{n} s_i\,|l_i - L_i| \qquad (15)$$

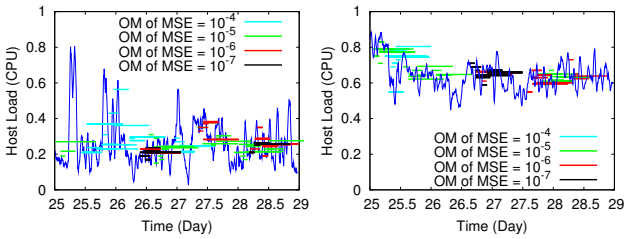


(a) Mean Error (b) Mean MSE

Figure 11. PDF of Pattern Prediction (about the CPU load)

We illustrate the pattern prediction of Algorithm 1, based

on 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 Figure 12, 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. We observe that the line segments (black or red lines) with MSE of about 10^{-5} or lower are quite close the the real fluctuation of the host load (the dark blue curve). As illustrated in Figure 11 (b), only a minor portion of pattern predictions have relatively higher MSEs at 10^{-4}. This means that our pattern prediction method under the Bayes Model leads to quite satisfactory accuracy with high probability.



(a) Load with Wide Amplitude (b) Load with Narrow Amplitude

Figure 12. Snapshot of Pattern Prediction

VI. Related Work

There exist many host load prediction methods [1], [2], [3], [4], [5], [6], [7], [27], [8] designed for traditional distributed systems such as Grid platforms. L. Carrington et al. [1], 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. C. Dabrowski et al. [2] perform the host load prediction by leveraging the Markov model via a simulated environment. S. Akioka, et al. [3] 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 [5] or auto-regression [4]) in the Grid host load prediction work. Moreover, such approaches are proposed by different researchers using hybrid or mixed models [6], [7], [27], [8], to suit different situations. However, as discussed in Section II, Cloud host load observed via Google's trace has more drastic fluctuation and higher noise than that in Grid systems, which will impact the prediction accuracy of these traditional methods significantly. Based on our experiments, for example, the HModel [27] suffers significant prediction errors in predicting the mean load over Google's trace, because its long-term prediction relies heavily on the load values recursively predicted by the AR method; prediction error can easily accumulate. Hence, it is necessary to revisit host load prediction for Cloud systems such as Google's.

While other works [13], [14], [29] already characterized workload features for Cloud systems, they mainly focus on tasks' placement constraints [13] or tasks' usage shapes [14]. Specifically, B. Sharma et al. [13] carefully studied the performance impact of task placement constraints based on the resource utilization from the view of tasks, while Q. Zhang et al. [14] designed a model that can accurately characterize task usage shapes in Google's compute clusters. A. Khan et al. [30] designed a model to capture the CPU workload auto-correlation in Cloud data centers by leveraging the Hidden Markov Model (HMM). B.J. Barnes et al. [29] introduced a regression-based approach for predicting parallel application's workload, and the prediction errors are between 6.2% and 17.3%. In comparison, we focus on the effective host load prediction in Cloud environments. Specifically, we designed a novel method based on the Bayes model and 9 extracted features from the evidence window, in order to predict the fluctuation patterns for the long-term future period. Through Google's trace, our Bayes method significantly outperforms the related works (including auto-regression, moving average, filter-based methods) from the perspective of success rate and mean squared error (MSE).

VII. Conclusion and Future Work

We design a Cloud load prediction method, and evaluate it using Google's one-month trace. Our objective is to predict the load fluctuation patterns for a long-term prediction interval. We first reduce pattern prediction to a set of mean load predictions each starting from the current time. Then, we design a mean load prediction approach by leveraging the Bayes model with our novel 9 features. To the best of our knowledge, this is the first attempt to make use of 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. In addition, the posterior probability analysis shows that four selective features (F_{ml}, F_{fi}, F_{ts}, and F_{fll}) can surprisingly characterize the mean load for the future 16 hours with high accuracy, regardless of how drastic it fluctuates. The MSE of predicting the load fluctuation patterns for the long-term period is usually low within the range [10^{-8},10^{-5}].

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] L. Carrington, A. Snavely, and N. Wolter, "A performance prediction framework for scientific applications," *Future Generature Computing Systems*, vol. 22, pp. 336–346, February 2006.

[2] C. Dabrowski and 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)*, ser. CRPIT, vol. 99. Wellington, New Zealand: ACS, 2009, pp. 29–40.

[3] S. Akioka and 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)*, Washington, DC, USA: IEEE Computer Society, 2004, pp. 765–772.

[4] P. A. Dinda and D. R. O'Hallaron, "Host load prediction using linear models," *Cluster Computing*, vol. 3, pp. 265–280, October 2000.

[5] Y. Zhang, W. Sun, and 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.

[6] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Cluster Computing*, vol. 1, pp. 119–132, January 1998.

[7] J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero, "Multi-model prediction for enhancing content locality in elastic server infrastructures," in *Proceedings of 18th High Performance Computing (HiPC'2011)*, 2011.

[8] Y. Wu, K. Hwang, Y. Yuan, and W. Zheng, "Adaptive workload prediction of grid performance in confidence windows," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 7, pp. 925 –938, july 2010.

[9] Google cluster-usage traces: online at http://code.google.com/p/googleclusterdata.

[10] J. Wilkes, "More Google cluster data," Google resarch blog, Nov 2011, online at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[11] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, pp. 34–41, Mar. 2010.

[12] The grid workloads archive: online at http://gwa.ewi.tudelft.nl/pmwiki/.

[13] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and 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)*, New York, USA: ACM, 2011, pp. 3:1–3:14.

[14] Q. Zhang, , J. L. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in google's compute clusters," in *Large Scale Distributed Systems and Middleware Workshop (LADIS'11)*, 2011.

[15] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis*. New York: Springer-Verlag, 1985.

[16] Webb, Boughton, and W. Zhihai, "Not so naive bayes: Aggregating one-dependence estimators," *Machine Learning*, vol. 58, no. 1, pp. 5–24, Jan. 2005.

[17] Auvergrid: online at http://www.auvergrid.fr/.

[18] P. S. R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*, softcover reprint of hardcover 3rd ed. 2008 ed. Springer, Oct. 2010.

[19] M. B. Wilk and R. Gnanadesikan, "Probability plotting methods for the analysis of data." *Biometrika*, vol. 55, no. 1, pp. 1–17, Mar. 1968.

[20] M. A. Farahat and 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.

[21] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of Basic Engineering*, no. 82 (Series D), pp. 35–45, 1960.

[22] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine Learning*, vol. 29, no. 2-3, pp. 103–130, Nov. 1997.

[23] M. H. Dunham, *Data Mining: Introductory and Advanced Topics*, 1st ed. Prentice Hall, Sep. 2002.

[24] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*. John Wiley & Sons, April 1991.

[25] C. Spearman, "The proof and measurement of association between two things. by c. spearman, 1904." *The American journal of psychology*, vol. 100, no. 3-4, pp. 441–471, 1987.

[26] P. Domingos and 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.

[27] Y. Wu, Y. Yuan, G. Yang, and 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.

[28] S. J. Orfanidis, *Introduction to signal processing*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[29] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22nd annual international conference on Supercomputing (ICS'08)*. New York, NY, USA: ACM, 2008, pp. 368–377.

[30] A. Khan, X. Yan, , S. Tao, and 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.