Long-Term CPU Load Prediction System for Scheduling of Distributed Processes and Its Implementation

doi: 10.1109/AINA.2008.135

Long-Term CPU Load Prediction System for Scheduling of Distributed Processes and Its Implementation

Yoshihiro Sugaya[†] Hiroshi Tatsumi[†‡] Mitiharu Kobayashi[†] Hirotomo Aso[†]

[†] Graduate School of Engineering, Tohoku University, Japan

[‡] NTT DATA CORPORATION, Japan

Abstract

There exist distributed processing environments composed of many heterogeneous computers. It is required to schedule distributed parallel processes in an appropriate manner. For the scheduling, prediction of execution load of a process is effective to exploit resources of environments. We propose long-term load prediction methods with references of properties of processes and of runtime predictions. Since an appropriate prediction method is different according to the situation, we propose a prediction module selection to select an appropriate prediction method according to a state of changing CPU load using a neural network. We also discuss about the implementation of a long-term CPU load prediction system, which provides information including prediction of load for schedulers, system administrators and users.

1. Introduction

In heterogeneous computer network environment, a load balancing system, which allocates and schedules processes appropriately and automatically, is required to exploit computing resources efficiently[2]. However, in real world appropriate scheduling and task allocation are sometimes difficult, since runtime of a process often changes irregularly because of changing of CPU load particularly in multiple users environment. If changing of CPU load can be predicted, it will assist process scheduling and task allocation for utilizing computing resources efficiently.

Wolski et al. developed Network Weather Service (NWS) [5], which predicts CPU load by using simple and traditional methods. Yang et al. tried to improve the prediction method of NWS, and they proposed a method to exploit a tendency of load changing just before the prediction [7, 6]. Dinda et al. proposed a method of a runtime prediction of application programs by using short-term prediction of CPU load based on autoregressive prediction [1]. Whereas these are short-term prediction methods, Resource Information Server (RIS)[3] employed not only a simple short-term load prediction but also a long-term load prediction by a so-called similarity method. Smith et al.[4] proposed a method to predict runtime of application programs, and they confirmed the effectiveness to exploit the predictions for scheduling and load balancing.

Although many researchers have tried to predict CPU load, conventional CPU load prediction methods[5, 7, 1, 3] are difficult to predict unsteady changing of CPU load when it sharply changes. A sharp change is often occurred by launching or ending of a task. In this paper, we propose novel CPU load prediction methods, which are named *process search method* and *runtime-prediction-based method*, aiming at more accurate prediction of unsteady changing of CPU load. We also propose a prediction module selection for an appropriate prediction method according to a state of changing CPU load using a neural network. The selection is expected to improve prediction accuracy under circumstances both of steady and unsteady states because each prediction method can predict more accurately in each different condition. To build a real-time load prediction system by using the proposed prediction methods, we discuss about data structures and about influence of the gap between sampling timing and prediction timing.

The remaining parts of this paper are organized as follows. In section 2, we describe two long-term CPU load prediction methods. In section 3, we describe the prediction module selection using a neural network and then experimental results are shown in Section 4. We discuss problems about the implementation of the load prediction system in section 5. Finally we summarize our work in section 6.

2. CPU load prediction

In this paper, as an benchmark of CPU load, we use *load average*, which is offered by operating system and which

1550-445X/08 $25.00 © 2008 IEEE
DOI 10.1109/AINA.2008.135

are used by many researchers. The load average takes less cost to be acquired and it is a reasonable index of CPU load, although it is not completely proportional to runtime of a process.

Let $V(t)$ and $P(t)$ be an observed load average value and a predicted value at time t, respectively. The prediction start time, which is usually the present time, is denoted by T, and a span of prediction is denoted by p. In brief, we try to predict a sequence of future load values $P(T+1), \cdots, P(T+p)$ using a sequence of past load values $V(0), \cdots, V(T)$. We also define a *task* as a process whose CPU percentage exceeded 5% at least once during the process running. We confirmed that processes whose CPU percentage do not exceed 5% can scarcely affect load average.

2.1. Process search method

Similarity method[3], which is one of a conventional method, is based on a presumption that subsequent sequences of similar sequences are still similar.

The method searches a past sequence of load values for the most similar pattern to the latest pattern, and then it outputs the sequence subsequent to the found pattern as a prediction result. This method exploits only a time series for prediction and does not employ other knowledge, for example task properties. Although it is a general prediction method and is applicable to predicting any time series, it would be difficult to improve prediction accuracy since it uses only a past sequence and does not use other property of processes. If we can exploit not only time series but also other properties, it would be a good solution to improve prediction accuracy.

We propose a new prediction method called *process search method* which refers properties of running tasks together with a past sequence of load averages for seeking similar sequences.

2.1.1 Making of a sequence of difference

While similarity method uses a sequence of observed load averages as it is, process search method uses a sequence of difference of load averages. The difference of load average is defined as $V'(t) = V(t) - V(t-1)$. Let us define a window of a difference sequence as $W_t = (V'(t-D+1), V'(t-D+2), \cdots, V'(t))$, and define the i-th element of W_t as $W_t(i)$, where D is a parameter which denotes the width of the window.

2.1.2 Search of process properties

Since similarity method searches all of the past sequence for the most similar window indifferent to the properties of processes, it may make the prediction accuracy worse. Although a different task might generate a different load sequence, the first half of which may accidentally resemble the current window and similarity method may choose such an accidentally similar sequence sometimes.

It is expected that the sequence of loads after the present time T is more similar to a past sequence of the same tasks as current running tasks. Unlike similarity method, process search method searches specific past sequences which are specified by task properties. The specific searching sequences are denoted by a set Q of time instants as follows.

1. Search the past sequence for time instants at which a set of running task is identical with the set of current running task. Let us define the set Q as a set of all the found time instants.

2. If the set Q is empty, search the past sequence for time instants at which at least one task in the set of current running task was running. Let us redefine Q as the set of the found time instants.

3. If the set Q is still empty, we set Q to the set of all the time instance in the past sequence. In this case, process search method is almost same as similarity method.

We do not aim to predict starting of a new task because a new task will be independent of sequences of past CPU loads and task properties. Therefore, time instants after which tasks except current running task were launched are removed from the set Q.

2.1.3 Calculation of prediction sequence

Let us define an error (un-similarity) between a window W_t and the current window W_T as:

$$err(t) = \sum_{i=1}^{D} |\mathbf{W}_t(i) - \mathbf{W}_T(i)|, \tag{1}$$

where t is an element of Q which was described in the previous subsection. (1) is calculated for each $t \in Q$. A sequence of load values which is predicted by the window W_t is determined as:

$$P_t(T+k) = V(T) + \sum_{j=1}^{k} V'(t+j), \tag{2}$$

for each k=1,..., p. This is the sequence that differences after t are accumulated on $V(T)$.

Since a window with less error is more similar to the current window, it seems that the predicted sequence by the window with less error is more reliable. However, there is often the case that predicted sequences by second or later

similar windows are more appropriate than the predicted sequence by the most similar window. This is occurs accidentally. Therefore, we calculate an output predicted sequence by averaging of predicted sequences by N most similar windows. It is defined as

$$P(T+k) = \frac{1}{N} \sum_{t \in S} P_t(T+k), \tag{3}$$

where S is a subset that consists of N of time instants t in the ascending order of $err(t)$, and parameter N is determined by preliminary experiments.

2.2. Runtime-prediction-based method

We propose another long-term CPU load prediction method, which is named runtime-prediction-based method. The method consists of two steps. First, it predicts remaining CPU time of current tasks by using task histories which are records of task properties. The method is based on an assumption that tasks with similar properties consume similar CPU time. As the property, we consider user name, task name and task arguments. Second, it predicts the future sequence of CPU loads by using the predicted remaining CPU time of current running tasks.

2.2.1 Remaining CPU time prediction of current running task

The proposed method picks out L (or less) tasks which have the identical property to a current running task and consume CPU time longer than the age of the current running task in CPU time. And then, the standard deviation of the set of tasks in consumed CPU time is calculated.

If the standard deviation is larger than the prediction span p, it is thought that remaining CPU time $Task\_X.runtime$ may exceed the prediction span, therefore, we set $Task\_X.runtime = p$. Otherwise, those tasks are arranged in chronological order of start time, and then it is divided in two classes so that the between-class variance becomes maximum. A predicted CPU runtime of the task is determined as an average CPU time of the later class, and remaining CPU time $Task\_X.runtime$ is calculated by subtracting the age of the task from the predicted CPU runtime. The reason why we use only the later class is that the character of tasks in the earlier class is sometimes different from current tasks in comparison with the tasks in the later class; users sometimes run a same name task with a subset data to debug the program before running the task with a normal data set, for example.

2.2.2 Load prediction using predicted CPU runtime

Depending on CPU percentage, which is the utilization ratio of the CPU, an estimated load $Task\_X.eload$ generated by a task X is defined as follows:

$$Task\_X.eload = \begin{cases} 1 & \cdots \text{if } CPU\% \geq M\%, \\ 0 & \cdots \text{otherwise}, \end{cases} \tag{4}$$

where M is a threshold to judge whether the task X will exploit CPU resources significantly or not, in other words, whether the task is a CPU-bounded task or not. The threshold M is determined by preliminary experiments.

Since each task X will generate a load $Task\_X.eload$ during $Task\_X.runtime$, the load generated by the task X at time $t = T+1, \ldots, T+p$ is described as follows:

$$Task\_X.load(t) = \begin{cases} Task\_X.eload & (t \leq T + Task\_X.runtime), \\ 0 & (t > T + Task\_X.runtime). \end{cases} \tag{5}$$

A predicted total load of the CPU at the time $t = T+1, \ldots, T+p$ is calculated by sum of $Task\_X.load(t)$ of all tasks X as follows:

$$P(t) = \sum_{X}^{\text{all\_task}} Task\_X.load(t). \tag{6}$$

2.2.3 Regulation of predicted load

Indeed, wall-clock runtime of a task depends on the CPU load at that time. When the number of CPU-bounded tasks exceeds the number of CPU, wall-clock runtime of each task becomes longer than $Task\_X.runtime$, because it is not a predicted remaining runtime in wall-clock time but in CPU time. In case of CPU-bounded tasks, wall-clock runtime increases approximately linear as the load increases. The fact is confirmed by preliminary experimental observations.

Therefore, we have to regulate predicted sequence of loads. Let I_a denote the span at which a predicted load before the regulation is $P(t) = a$. I_a is extended to I'_a as follows:

$$I'_a = \begin{cases} I_a * \frac{a}{C} & \cdots \text{if } a \geq C, \\ I_a & \cdots \text{otherwise}, \end{cases} \tag{7}$$

where C is the number of CPU. The regulated load $P'(t)$ is straightforwardly obtained by the regulation, this is a final result of this method. Figure 1 demonstrates an example of the regulation.

3. Prediction module selection

We evaluated several conventional methods, similarity method[3], process search method and runtime-prediction-based method by preliminary experiments. When sequence

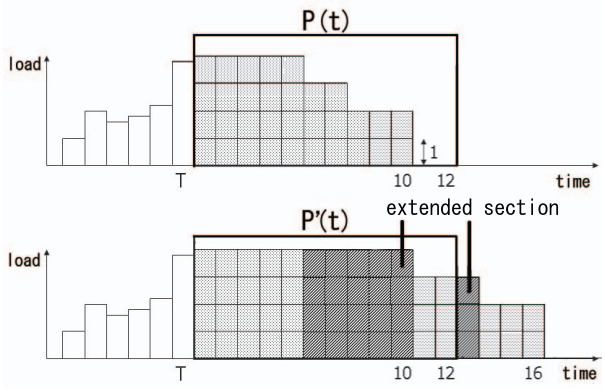

Figure 1. An example of P(t) and P'(t) of the runtime-prediction-based method in case of a dual CPU computer.

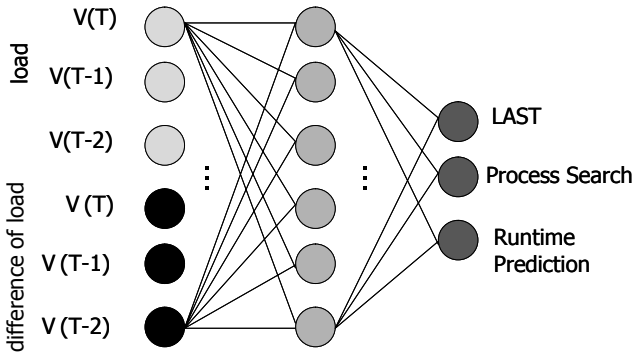

Figure 2. Neural network for prediction module selection.

of loads is steady state, LAST which simply assumes the present load value to be predicted value can predict fastest and more properly in many case. On the other hand, process search method or runtime-prediction-based method can predict more properly in unsteady load state. From the observations, we can conclude that an appropriate prediction method is different according to the state of load changing. It can be expected that prediction accuracy will be improved if we can dynamically select a proper prediction module depending on the condition of changing of load.

We propose a prediction module selection using a neural network(NN) as shown in Fig 2. The NN is a 3-layer perception and each layer is fully connected. The number of cells in each layer is 6-6-3 (input-hidden-output), respectively. The last three observed load values $V(T), V(T-1), V(T-2)$ and three of those differences $V'(T), V'(T-1), V'(T-2)$ are fed to the input layer. Each cell in output layer corresponds to each prediction module, which is LAST or process search method or runtime-prediction-based method, and the cell that most strongly fires indicates the prediction module to be selected at that time.

By backpropagation learning, the NN is trained which module should be selected when some kind of load sequence was observed. The training is performed in advance using results of preliminary experiments. The teacher data are made so as to choose the best method whose error is the smallest among three methods, but LAST should be chosen as a teacher when errors of them are almost same.

4. Experiments

We have conducted experiments to compare with LAST, MEAN, MEDIAN, AR[5], similarity method and our proposed methods. LAST is a method to assume the present value to be a predicted value as is. MEAN and MEDIAN assume the mean or the median of last D' values to be a predicted value, respectively, where D' is dynamically set so that the error of prediction $P(T)$ of the present load value using $V(T-1), V(T-2), \cdots$ becomes smallest. AR is a prediction method by autoregressive model.

Sequences of CPU loads are obtained by system load averages for five minutes sampled every five minutes on four Linux 2.4.x machines equipped with dual Xeon 2.4GHz processors, which are utilized for scientific computations or computer simulations. Task properties are obtained by *ps* command at the same timing with sampling of CPU load values and are also obtained from the process accounting file logged by Linux operating system. We attempt to predict a sequence of total loads of the four machines on the assumption predicting load of a computer cluster. The span to be predicted is $p = 12$ (60 minutes), and data from prediction start time T to three months before are employed to search for similar windows and similar tasks. In the experiments, we use difference sequence of CPU loads not only in proposed methods but also conventional methods.

Parameters are determined by preliminary experiments. The width of the search window is $D = 3$ (15 minutes), the number to average predicted sequences in process search method is $N = 5$, and the maximum number of tasks to acquire in runtime-prediction-based method is $L = 10$. We set the threshold of CPU percentage to $M = 80$ when the number of running tasks is two or less, or set it to

$$M = \frac{200}{\text{the number of tasks}} \times 0.8,$$

when the number of running tasks exceeds two, because each computer has two CPU and a maximum of a total CPU percentage is about 200% in our environment. Training data used for NN in prediction module selection are gathered

Table 1. The results of evaluation experiments of prediction methods in steady state.

method	AVG	STD	≤ 0.2	≥ 1.0
LAST	0.102	0.353	89.8%	2.4%
MEAN	0.105	0.365	89.0%	2.4%
MEDIAN	0.252	0.466	66.2%	6.6%
AR	0.119	0.322	86.2%	2.4%
similarity	0.176	0.497	82.4%	4.0%
process	0.135	0.363	81.8%	2.4%
runtime	0.213	0.620	84.6%	7.8%
NN	0.101	0.349	90.4%	2.2%

The ratio of methods selected by NN
LAST:93%process:2%runtime:5%

Table 2. The results of evaluation experiments of prediction methods in unsteady state.

method	AVG	STD	≤ 0.2	≥ 1.0
LAST	0.797	0.834	21.8%	26.0%
MEAN	0.851	0.852	14.4%	28.0%
MEDIAN	0.917	0.868	15.0%	32.4%
AR	1.347	1.500	12.8%	43.6%
similarity	0.951	1.097	28.2%	35.6%
process	0.779	0.812	21.6%	26.2%
runtime	0.861	1.090	38.6%	30.0%
NN	0.747	0.954	32.6%	22.0%

The ratio of methods selected by NN
LAST:22%process:37%runtime:41%

from the identical sequence with testing data, but each prediction start time is distinct. We employed 300 start time for learning, and used the rest for testing.

We evaluate those methods in terms of average error between the observed value and a predicted value, which is defined as

$$PredictionErr(T) = \frac{1}{p'} \sum_{k=1}^{p'} |V(T+k) - P(T+k)|, \quad (8)$$

where p' satisfies $0 \leq p' \leq p$ and it is the time when a new task which was not running at the prediction start time T is launched. When there is not such a new task, p' is same as p. That is because we decided not to consider launch of new tasks in this paper.

We performed predictions by 500 times using each method in both of steady state and unsteady state. Table 1 and Table 2 show the average of $PredictionErr$(AVG), the standard deviation(STD), the ratio of predictions that are $PredictionErr \leq 0.2$ and the ratio of predictions that are $PredictionErr \geq 1.0$, respectively. We defined that steady state is the case that the variance of the last three load values just before T is less than 0.1 and unsteady state is the case that the variance is 1.0 or more.

In steady state (Table1), LAST is excellent among individual methods. Process search and runtime-prediction-based method, which are our proposed methods, are not superior in comparison with conventional methods in terms of prediction accuracy. Accordingly, we confirmed that some conventional methods can predict well in steady state. Prediction selection by NN is the best among all methods. It selects LAST at the ratio of 93%, which is an appropriate selection in steady state, and furthermore, it is thought that an appropriate method for unsteady state can be also selected when a task is launched just before prediction.

In unsteady state (Table2), process search method is the best among individual methods in terms of accuracy, the second is LAST, and the third is runtime-prediction-based method. The prediction error of process search is 18% smaller than the error of similarity method. This result indicates that task properties are beneficial for long-term CPU load prediction. The average accuracy of runtime-prediction-based method is not so good, but there are a lot of cases that the prediction errors are 0.2 or less. It means that runtime-prediction-based method can often predict a long-term sequence of CPU loads considerably precisely compared with other methods. Also in unsteady state, prediction selection by NN is the best among all methods. It selects process search or runtime-prediction at the ratio of 78%. The fact indicates that it can select an appropriate method in also unsteady state. It seems that it can select an appropriate method accordingly whether in steady or unsteady or intermediate. By backpropagation learning, NN can select an appropriate prediction module according to the state of changing load, it improves prediction accuracy.

Figure 3 is an example of a prediction results. Runtime-prediction-based method can predict a sequence of loads considerably precisely in this case, and process search method also can predict a rough tendency.

5. Implementation of load prediction system

We are building a real-time load prediction system to provide predicted loads for task scheduler, system administrators and users. The process search method already has been implemented in the system. To exploit the system for scheduling or task allocation, it is required that it can quickly return a result with low system load. Therefore, we have to consider following two issues for implementation.

5.1 Data structure of load average history

For low system load and fast search of a sequence of difference of load values, we decided to store all the history of the sequence on memory by PackBits encoding, which

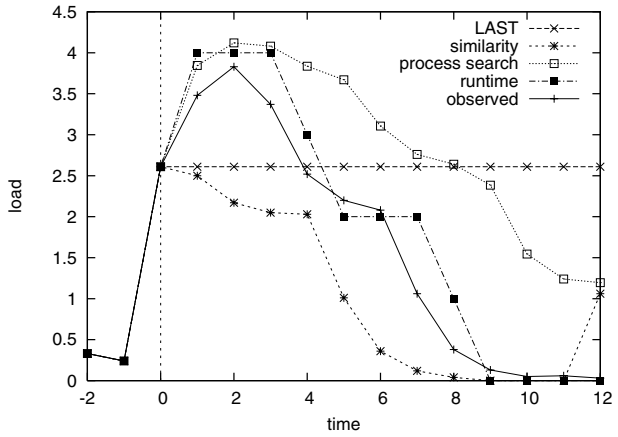

Figure 3. An example of prediction results. The span to predict at time 0 is from 1 to 12 (from 5 min to 60 min).

Table 3. Compress rates of PackBits and run-length encoding.

	PackBits	run-length
compress rate	3.55%	21.35%

switches run-length encoding and raw data sequence by flag bits. Although it requires some overhead for decoding, PackBits encoding will sometime reduce the number of calculation of error between windows; an error between a window and the current window can be calculated without decoding in run-length encoding part except the neighbor of border. When a value of load average is constant in a certain range, the sequence of all the windows in the range are identical, and the calculation of error can be skipped.

Table 3 shows compress rate of PackBits and run-length encoding, where compress rate is defined as

$$\text{compress rate} = \frac{\text{memory usage with encoding}}{\text{memory usage without encoding}}. \quad (9)$$

We confirmed that the compress rate of PackBits is very high, because there are many constant data corresponding to idle time on our environment. Note that PackBits does not increase memory usage different from run-length even if there is less constant range.

The execution time of a prediction is shown in Table 4. It is confirmed that there is overhead by encoding but it is not so significant. The variances by using PackBits or run-length are much greater than one by no encoding because it depends on whether the window of which error should be calculated is constant.

Table 4. Execution time of a prediction by using PackBits, run-length and no encoding.

encoding	average (ms)	variance (ms^2)
PackBits	9.970	15.826
run-length	9.801	14.855
no encoding	6.795	2.627

Table 5. Error of predictions when it is not a sampling instant.

	average	variance
ignored	0.361	0.184
shortened	0.368	0.226
exhaustive	0.329	0.182

5.2 Prediction between sampling timings

In preceding section, it is implicitly assumed that prediction start time T, which is the present time in load prediction system, is also a sampling instant. If a prediction will be performed when it is not a sampling instant, the current window can not be made correctly. It is necessary to discuss about this influence for implementing the process search method into the system. To solve the problem, we consider the following two manners.

The first manner is to ignore the gap between the latest sampling timing and the prediction start time. Even if the present time is not a sampling instant, eventually if time lag with the last sampling instant $T-1$ is not five minutes before, sample the present load average and exploit it to make the current window. It may make an error of under five minutes when sampling timing is five minutes.

The second manner is to shorten sampling timing only in the range of window width. If the normal sampling timing is five minutes and the shortened sampling time is one minute, errors of sampling timing can be decreased to under one minute. The error of sampling timing under one minute has to be ignored same as the preceding manner, but the error is decreased. The extra samples are required only in the present window, and they will be discarded when they become older than the present window.

We performed experiments to examine these two manners. The comparison results are shown in Table 5. We performed 2000 times of predictions in unsteady state by each manner. "ignored" and "shortened" indicate first manner and second manner, respectively. "exhaustive" is the way that sampling timing is one minutes and keep all samples different from the second manner, in other words, it can be regarded as the result of the case that all predictions are performed when it is just a sampling instant. Note that we may not adopt this timing for actual system because of the

necessity of low system load. It is confirmed that prediction errors increased a little by using both "ignored" and "shortened", but it does not seem critical. The error of "ignored" is slightly larger than the error of "shortened", but we think that "ignored" is acceptable because it is simple and does not require extra memory.

6. Conclusions

In this paper, we proposed novel long-term CPU load prediction methods, namely process search method and runtime-prediction-based method, which refer properties of processes. We showed that these methods can predict long-term CPU load more accurately compared with conventional methods in unsteady state when process properties can be acquired and can be exploited.

Furthermore, we proposed that prediction module selection using a neural network. It is confirmed that it can select an appropriate module according to a state of load changing by backpropagation learning and that it can improve prediction accuracy.

To build a real-time load prediction system, we also discussed about some issues for the implementation. PackBits encoding is reasonable to store load average history, and it is an acceptable manner that the present load value is exploited to make the current window even if the present time is not a sampling instant.

In future works, we will refine parameters to improve prediction accuracy and will develop scheduling algorithms to exploit long-term CPU load predictions.

References

[1] P. A.Dinda and D. R.O'Hallaron. Host load prediction using linear models. *J. Cluster Computing*, 3:265–280, 2000.

[2] S. Akioka and Y. Muraoka. An extended forecast of cpu and network load on the computational grid. *IEICE Trans. Information and Systems*, J87-D-I(9):845–854, 2004 (in Japanese).

[3] H. Koide, N. Yamagishi, H. Takemiya, and H. Kasahara. Evaluation of the resource information prediction in the resource information server. *IPSJ Transactions on Programming*, 42(SIG3(PRO10)):65–73, 2001 (in Japanese).

[4] W. Smith, I. Foster, and V. Taylor. Predicting application run times with historical information. *J. Parallel Distib. Comput.*, 64:1007–1016, 2004.

[5] R. Wolski. Dynamically forecasting network performance using the network weather service. *J. Cluster Computing*, 1:119–132, 1998.

[6] L. Yang, I. Foster, and J. M. Schopf. Homeostatic and tendency-based cpu load predictions. In *Proc. of the 17th International Symposium on Parallel and Distributed Processing*, page 42b, 2003.

[7] L. Yang, J. M. Schopf, and I. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proc. Supercomputing 2003*, volume 11, pages 15–11, 2003.