PAPER

A Prediction-Based Green Scheduler for Datacenters in Clouds

Truong Vinh Truong DUY[†a)], *Nonmember*, Yukinori SATO[††], *and* Yasushi INOGUCHI[††], *Members*

SUMMARY With energy shortages and global climate change leading our concerns these days, the energy consumption of datacenters has become a key issue. Obviously, a substantial reduction in energy consumption can be made by powering down servers when they are not in use. This paper aims at designing, implementing and evaluating a Green Scheduler for reducing energy consumption of datacenters in Cloud computing platforms. It is composed of four algorithms: prediction, ON/OFF, task scheduling, and evaluation algorithms. The prediction algorithm employs a neural predictor to predict future load demand based on historical demand. According to the prediction, the ON/OFF algorithm dynamically adjusts server allocations to minimize the number of servers running, thus minimizing the energy use at the points of consumption to benefit all other levels. The task scheduling algorithm is responsible for directing request traffic away from powered-down servers and toward active servers. The performance is monitored by the evaluation algorithm to balance the system's adaptability against stability. For evaluation, we perform simulations with two load traces. The results show that the prediction mode, with a combination of dynamic training and dynamic provisioning of 20% additional servers, can reduce energy consumption by 49.8% with a drop rate of 0.02% on one load trace, and a drop rate of 0.16% with an energy consumption reduction of 55.4% on the other. Our method is also proven to have a distinct advantage over its counterparts.

key words: *energy savings, green scheduling, neural predictor, cloud computing, datacenters*

1. Introduction

Cloud computing [1] has emerged as a new business model of computation and storage resources based on on-demand access to potentially significant amounts of remote datacenter capabilities. However, the deployment of datacenters in Clouds has put more and more computers in use each year, increasing energy consumption and pressure on the environment. Research shows that running a single 300-watt server during a year can cost about $338 and, more importantly, can emit as much as 1,300 kg CO_2 [2]. A recent report has estimated the datacenters in the US consumed approximately 1.5% of the total electricity consumption in 2006, and this number is projected to double in 2011 [3]. Japan has been actively involved in efforts to establish a post-Kyoto framework, and has proposed to reduce CO_2 emissions at least by half by 2050 [4].

Apart from well-established energy saving areas such as cooling and electrical management systems, the existing techniques for energy savings in the area of enterprise power management at a server farm can roughly be divided into two categories: dynamic voltage/frequency management inside a server, and shutting down servers when not in use. In the former, power savings are gained by adjusting the operating clock to scale down the voltage supply for the circuits. Although this approach can provide a significant reduction in power consumption, it depends on the hardware components' settings to perform scaling tasks. On the other hand, the latter promises most power savings, as it ensures near-zero electricity consumed by the off-power servers. However, previous work had difficulties in assuring service-level agreement due to the lack of a reliable tool for predicting future demand to assist in the ON/OFF decision-making process.

In this paper, we aim to design, implement and evaluate a Green Scheduler for reducing energy consumption of datacenters in Cloud computing environments by shutting down unused servers. The neural predictor which we developed earlier has been proven to have accurate prediction ability with low overhead, suitable for dynamic real-time settings [5]. The use of this predictor will help the scheduler make appropriate ON/OFF decisions, and will make the approach more practical. As virtual machines are spawned on demand to meet the users' needs in Clouds, the neural predictor will be employed to predict future load demand on servers based on historical demand.

Our scheduler is composed of four algorithms that work as follows. The predictor is used in the prediction algorithm to predict request load, based on which the ON/OFF algorithm computes the number of servers needed to process the load [6]. Then unnecessary servers are turned off in order to minimize the number of servers running, thus minimizing the energy use at the points of consumption to provide benefits to all other levels. The task scheduling algorithm schedules the incoming tasks, representing the current aggregate load level, to run on the active server set. Finally, the evaluation algorithm monitors the performance to dynamically adapt to load changes over time.

For evaluation, we perform simulations using the CloudSim and GridSim toolkits. We add an energy dimension to the toolkits, and implement the components of the Green Scheduler. In addition, we have developed five running modes to examine the impact of different methods on estimating the number of servers required. They are executed with two real workload traces collected on the Inter-

Manuscript received August 13, 2010.
Manuscript revised February 25, 2011.

[†]The author is with School of Information Science, JAIST, Nomi-shi, 923–1292 Japan.
[††]The authors are with Center for Information Science, JAIST, Nomi-shi, 923–1292 Japan.
a) E-mail: duytvt@jaist.ac.jp

DOI: 10.1587/transinf.E94.D.1731

net, and on four datacenter architectures. We also compare our method with two other methods.

Our main contribution in this paper is the design, implementation, and experimental evaluation of the Green Scheduler. The novelty of our solution lies in its integration of the neural predictor and the development of policies for dynamic training and dynamic provisioning of additional servers, as well as considering server's timing requirements. Another unique feature of our work is to compare the performance of our algorithms with optimal solutions and another algorithm empirically, in addition to performance comparison of running modes among themselves.

The remainder of this paper is organized as follows. Section 2 describes the system model, states the problem, and examines the power consumption of servers. Architecture of the Green Scheduler is presented in Sect. 3. Section 4 analyzes experimental results, and Sect. 5 draws comparisons between our method and two other methods. Finally, we introduce related work in Sect. 6 and conclude our study in Sect. 7.

2. System Model, Problem Statement, and Server Power Consumption

2.1 System Model

Figure 1 depicts the system model that we consider in this paper. Actually, it represents a simple architecture of Cloud computing, where a Cloud provider, consisting of a collection of datacenters and CISRegistry (Cloud Information Service Registry), provides utility computing service to Cloud users/DCBrokers. The Cloud users in turn use the utility computing service to become SaaS providers and provide web applications to their end users.

A request from a Cloud user is processed in several steps, as follows.

1. Datacenters register their information to the CISRegistry.
2. A Cloud user/DCBroker queries the CISRegistry for the datacenter information.
3. The CISRegistry responds by sending a list of available datacenters to the user.
4. The user requests processing elements through virtual machine creation.
5. The list of available virtual machines is sent back for processing requests from end users to the services hosted by the user.

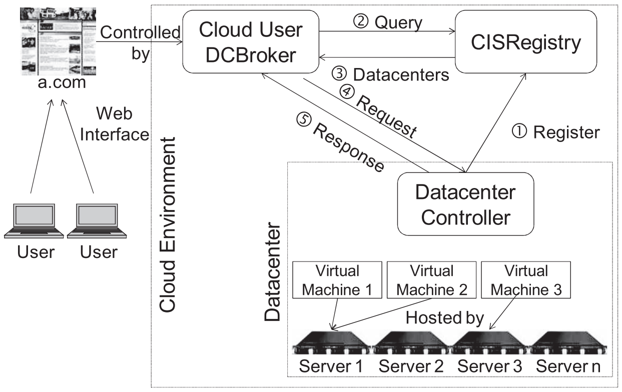

Fig. 1 The system model.

A datacenter consists of a set of hundreds to thousands of processing servers. It has several controllers which have three main functions: (1) registering the datacenter information to the CISRegistry, (2) accepting requests from Cloud users, and (3) running the Green Scheduler to distribute load among virtual machines, making decisions on creation and suppression of virtual machines in servers, and turning off/on servers for energy savings. A server is responsible for managing virtual machines it is hosting. A server can host multiple virtual machines at the same time, but one virtual machine can be hosted in only one server. Virtual machines appear as processing elements from the viewpoint of users.

2.2 Problem Statement

Given the above system model, we cast the problem as: dynamically scale the datacenter up and down to the minimum possible number of servers running based on future load prediction from historical load, and distribute clients' requests to the active servers in such a way that none of them are overloaded, for the purpose of minimizing the energy consumption and performance loss.

Cloud computing environments host a wide variety of applications that have different characteristics and requirements. They can be roughly classified into two major types: Web-based request-response applications and HPC applications. The Web-based request-response applications have short-lived requests that are processed in short processing cycles, usually immediately after their arrival within the current cycle, with almost equal processing time. Examples include online shopping sites, social networking sites, and news sites. On the other hand, the HPC applications contain long-running tasks, such as image processing and flight simulation. In this paper, the Green Scheduler is designed with a view to accommodating both types of applications, although the experimental evaluations focus on the more popular Web-based applications. Evaluations of the Green Scheduler with HPC applications are left for future work.

In this context, we define the term "task" as both long-running applications in the HPC area, and a chunk of requests which is commonly used to mention short-lived client-server connections in Web-based applications. Although the term "task" generally refers to long-running applications, it is such a general term that we can treat it as a chunk of requests as well. In addition, the term "load" refers to the amount of work that a computer system performs. For Web-based applications, load is expressed in terms of the number of requests or connections from clients per unit time.

2.3 Server Power Consumption

Understanding the relationship between power consumption and CPU utilization of servers is essential to design efficient strategies for energy savings. We examined this relationship by measuring power consumption of typical machines in different states. The machines we used are a Linux machine with AMD Phenom 9500 Quad-Core Processor 2.2 GHz, and a Windows machine with AMD Athlon 64 X2 Dual-Core Processor 5000 + 2.6 GHz. They were connected to a System Artware SHW3A watt-hour meter at the power plug, to record power consumption of the whole machine.

Figure 2 shows power consumption of the two machines in the idle state and at different CPU utilization levels, ranging from 10% to 100%. In the Linux machine, the CPU load is generated using the lookbusy load generator, to attempt to keep the CPUs at a chosen utilization level, while in the Windows machine, load is generated by a simple loop written in C# by changing the parameter LOAD [20]. To obtain more accurate data, the CPU utilization is maintained at a stable state for 5 minutes, and the average recorded power consumption over the period is reported. The power consumption appears to be almost linear to CPU utilization. An increase of 10% in CPU utilization leads to increases of approximately 6.5% and 3% in power consumption in the quad-core and dual-core machines, respectively. Also, we observe that the idle state consumes a substantial amount of energy, as much as 62%, in the case of the quad-core machine, and 78% in the case of the dual-core machine, of peak power. This observation implies that there is room for power conservation and hence, a large energy reduction can be achieved by sending idling servers to a lower power state.

Based on these empirical measurements, we can model the relationship between energy consumption and CPU utilization using the following formula:

$$E_n = P_I + (P_M - P_I) \times \frac{n}{100} \tag{1}$$

where E_n is the energy consumption at $n\%$ CPU utilization, and P_M and P_I are the power consumption at maximum utilization and idle, respectively.

We then calculate the energy consumption of a server operating at $n(t)\%$ CPU utilization over a period T_{run} as:

$$E_{run} = \sum_{t=1}^{T_{run}} \left(P_I + (P_M - P_I) \times \frac{n(t)}{100} \right) \tag{2}$$

3. The Green Scheduler

In this section we present the architecture of the Green Scheduler, focusing primarily on its components and their main duties. As plotted in Fig. 3, the scheduler is composed of four algorithms in order of execution: the prediction, ON/OFF, task scheduling, and evaluation algorithms. The scheduler starts execution at regular time intervals by running the prediction algorithm to collect historical load, and predict loads in the future based on the historical load. Depending on the predicted future load, the ON/OFF algorithm dynamically adjusts server allocations to minimize energy consumption. The task scheduling algorithm then schedules the incoming tasks, representing the current aggregate load level, to run on the active server set. Finally, the performance is evaluated and monitored by the evaluation algorithm to ensure the ability of the scheduler to adapt to load changes over time.

3.1 The Prediction Algorithm

The prediction algorithm actually makes use of a neural predictor. A three-layer neural network predictor in operation with a time series input is displayed in Fig. 4. The network has 4 network inputs where external information is received, and 1 output layer C with one unit where the solution is ob-

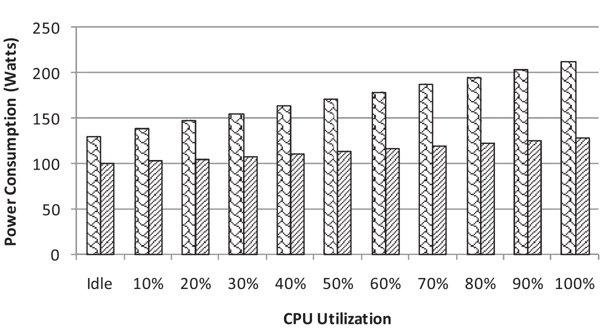

Fig. 2 CPU utilization and power consumption.

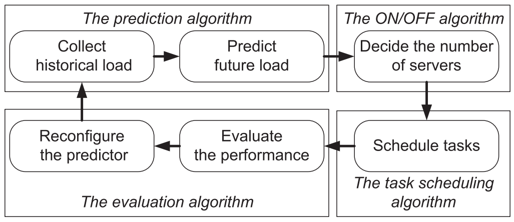

Fig. 3 Architecture of the Green Scheduler.

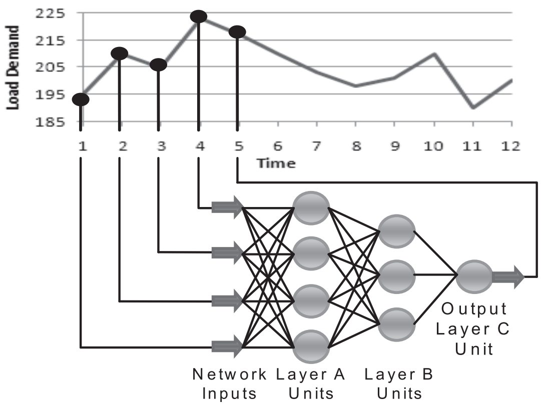

Fig. 4 A three-layer neural network predictor.

tained. The network input and output layers are separated by 2 hidden layers: layer A with 4 units and layer B with 3 units. The connections between the units indicate the flow of information from one unit to the next, i.e., from left to right.

In order to make meaningful predictions, the neural network needs to be trained on an appropriate data set. Basically, training is a process of determining the connection weights in the network. Examples of the training data set are in the form of <input vector, output vector> where input vector and output vector are equal in size to the number of network inputs and outputs, respectively. The final goal is to find the weights that minimize an overall error measure, such as the sum of squared errors or mean squared errors.

We have developed a neural predictor and performed experiments to prove its accurate prediction ability with low overhead, suitable for dynamic real-time settings similar to this system model [5]. For example, the 20:10:1 network with a learning rate of 0.3 has reduced the mean and standard deviation of the prediction errors by approximately 60% and 70%, respectively. The network needs only a few seconds to be trained with more than 100,000 samples, and then makes tens of thousands of accurate predictions within a second, without the need to be trained again.

3.2 The ON/OFF Algorithm

The ON/OFF algorithm, detailed in Fig. 5, is a key component in determining which servers should be turned ON/OFF. Due to the wear-and-tear problem, it recruits and retires servers using a simple round-robin policy to evenly distribute the on/off cycles among them. This algorithm will turn on servers when the load increases and conversely, turn off servers when the load decreases. However, as it takes some time for a server to come to full operation, a server must be turned on before it is actually needed. Hence, the number of servers running at any time t must be sufficient to tolerate the peak load until more servers are ready to share the load. Also, to assure service-level agreement, no server can be loaded to more than its capacity C, and one processing core should be allocated to only one virtual machine.

In a previous work [6], we measured power consumption of predefined low power states, and found that suspend-to-RAM and standby are the best states for Linux and Windows machines, respectively, in terms of power consumption and transition delays. To emphasize these low power states and transition delays, we assume that a server can be in one of the following four states: OFF, RESTARTING, ON, and SHUTTING. Initially all servers are in the OFF state, which is actually a selected low-power state to which a server is sent for energy savings. For instance, the OFF state may refer to suspend-to-RAM for a Linux machine and standby for a Windows machine. Upon receiving a restart signal, the server moves from OFF to RESTARTING. It will stay in this state for $T_{RESTARTING}$ seconds before coming to ON. The ON state implies that the server is idling, waiting for a user's request or processing it. Likewise, when a server is signaled to turn off, it will change state and stay in the SHUTTING state for $T_{SHUTTING}$ seconds before completely changing its state to OFF. The energy consumption in ON state is estimated from equation (2), where the CPU utilization level can be approximated as:

$$n(t) = \frac{r(t)}{C} \times 100\% \tag{3}$$

where $r(t)$ is the number of requests being processed by the server at the time.

3.3 The Task Scheduling Algorithm

The task scheduling algorithm finds scheduling mappings between tasks and VMs, and spreads request loads from all services evenly across the entire VM pool available. As can be seen in Fig. 6, it applies the Earliest Deadline First strategy to the task queue, and the Largest Capacity First strategy to the VM pool. Tasks from the task queue are scheduled to

```
Inputs: list of servers in the datacenter and their current states; $T_{RESTARTING}$: delay necessary for a server to come to ON from OFF; C: server capacity.
Output: ON/OFF decisions, and updated list of servers.
Do
    Ask the predictor to predict loads from time t to time t + $T_{RESTARTING}$ based on the collected historical loads during the period of [0, t - 1]
    Find the peak load $L_p$ from time t to time t + $T_{RESTARTING}$
    Find the number of servers necessary at time t: $N_t = \lceil L_p \text{ div } C \rceil$
    Assume $N_c$ = number of servers in ON state
    If $N_t = N_c$: no action
    Else if $N_t > N_c$: choose $(N_t - N_c)$ servers in OFF state and signal them to restart
    Else if $N_t < N_c$: choose $(N_c - N_t)$ servers in ON state with free processing cores, and signal them to shutdown.
```

Fig. 5 The ON/OFF algorithm.

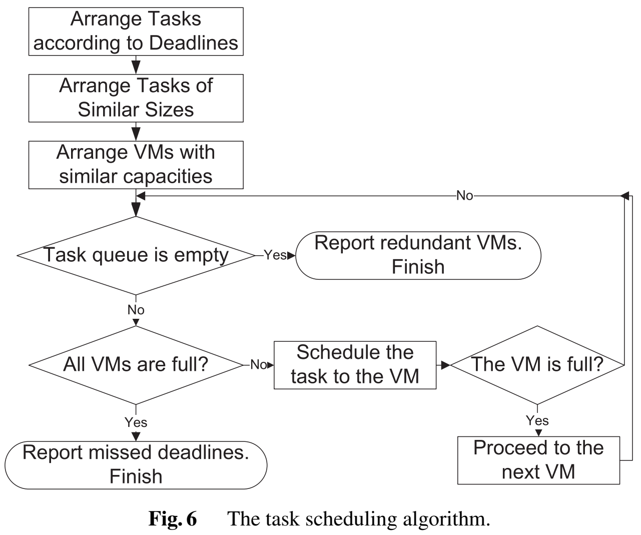
Arrange Tasks according to Deadlines
Arrange Tasks of Similar Sizes
Arrange VMs with similar capacities
No
Task queue is empty
Yes
Report redundant VMs. Finish
No
All VMs are full?
No
Schedule the task to the VM
The VM is full?
Yes
Yes
Report missed deadlines. Finish
Proceed to the next VM

Fig. 6 The task scheduling algorithm.

run on VMs in the VM pool, until there are no tasks available or all VMs have become full. Once it has finished processing, it will report the number of missed deadlines, or drop rate, and redundant virtual machines, if any.

3.4 The Evaluation Algorithm

This algorithm involves developing a performance monitoring mechanism to sufficiently adapt to meaningful workload changes over time. However, it must also have the ability to avoid overreacting to noise in workload fluctuations. That said, the performance monitoring mechanism must balance adaptability with stability. One unique feature of the neural predictor is that it can be trained with the most up-to-date training data to reflect workload changes when possible. A major duty of this algorithm is to find the suitable training epochs during execution, because the system may oscillate if the epochs are too short for the servers to stabilize after each round of training, or may not be able to dynamically adapt if the epochs are too long. In addition, it establishes a policy for dynamic provisioning of additional servers, with the aim of improving performance. The algorithm works on the workload to increase the prediction accuracy, assumed to relate directly to applications' performance.

We propose two training policies, which we call static training and dynamic training. In static training, the training phase takes place at regular time intervals, making it easy to identify training epochs. As persistent load increase and decrease caused by shifts in users' requests tend to occur on the scale of hours rather than seconds, the training phase can be performed on a daily or hourly basis. The downside of this policy is that training occurs regardless of performance.

In dynamic training, our solution holds a sliding window moving average of size $S1$, and maintains an error term $MSPE$ (Mean Squared Prediction Error). If $MSPE$ of the moving average of recent observations exceeds a predefined error threshold Err, the training process will be triggered. Otherwise, the system is said to be stable, and no training is needed. $MSPE$ is identified based on the following equation.

$$MSPE = \frac{1}{S_1} \sum_{i=T-S_1}^{T-1} (P_i - A_i)^2 \tag{4}$$

where T is the current time, S_1 is the window size, and P_i and A_i are, respectively, the predicted workload and the actual workload at time i.

Although we adopt a relatively aggressive performance monitoring mechanism in training policies, prediction errors are unavoidable. To tackle the shortage of servers in case the requested load is more than the capacity of the provided servers, a given number of servers, called additional servers, are added to assure service-level agreement. For example, if the predictor predicts 5 servers and we use 2 additional servers, we will actually use 5 + 2 = 7 servers instead of only 5. In order to avoid over-provisioning, we establish a policy to dynamically adjust the number of additional servers based on two factors: missed deadlines and redundant virtual machines. In this policy, we examine two series, MDL (missed deadlines) and RVM (redundant virtual machines) in a sliding window moving average of size $S2$. Suppose we are at current time T. If $MDL = \{X_i = 0 : \forall i \in [T - S2, T - 1]\}$, where X_i is the number of missed deadlines at time i, and $RVM = \{Y_i > 0 : \forall i \in [T - S2, T - 1]\}$, where Y_i is the number of redundant virtual machines at time i, we will decrease the number of additional servers due to possible over-provisioning. If $MDL = \{X_i > 0 : \forall i \in [T - S2, T - 1]\}$ and $RVM = \{Y_i = 0 : \forall i \in [T - S2, T - 1]\}$, this number will be increased to avoid under-provisioning of servers. The dynamic training policy and dynamic provisioning policy of additional servers are illustrated in Fig. 7.

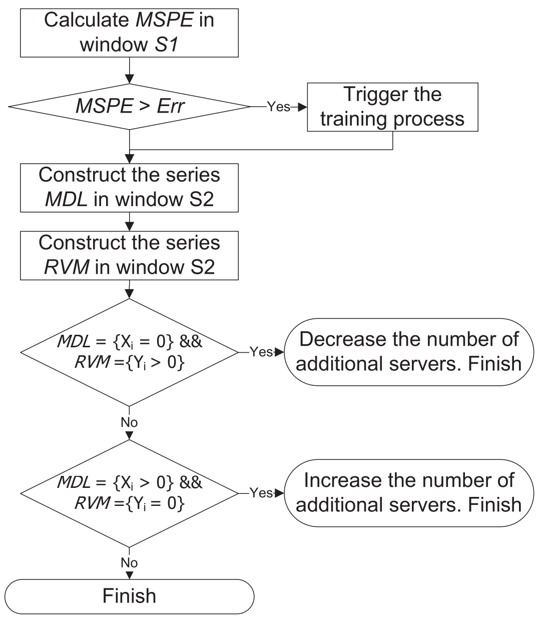
Calculate *MSPE* in window *S1*
MSPE > *Err*
Yes
Trigger the training process
Construct the series *MDL* in window S2
Construct the series *RVM* in window S2
MDL = {Xᵢ = 0} && *RVM* ={Yᵢ > 0}
Yes
Decrease the number of additional servers. Finish
No
MDL = {Xᵢ > 0} && *RVM* ={Yᵢ = 0}
Yes
Increase the number of additional servers. Finish
No
Finish

Fig. 7 The evaluation algorithm.

4. Experimental Evaluations

4.1 Simulator Description

The simulations were conducted on our SGI Altix XE nodes having configuration: Intel Quad-Core Xeon, 8GB RAM, Linux OS, and JDK 1.6. We performed simulations using the CloudSim and GridSim toolkits [7], [8]. Certain considerable modifications were made to meet our needs, notably:

- We added a new dimension to the toolkits, the energy dimension, to calculate energy consumption, to enable the servers' states, to shutdown and restart servers, etc.
- We added the components of the Green Scheduler.

In addition, we modified the original CloudSim communication flow, to the flow shown in Fig. 8. First, each datacenter registers itself with the CISRegistry. The datacenter broker queries the CISRegistry for a list of datacenters which offer services matching the user's application requirements, on behalf of users. The broker then deploys the application (with the matching datacenter) for processing. The simulation ends after this process has been completed in the original flow. Therefore, we added a new entity, called User Workload Generator, to periodically impose load on the system for N time intervals. Virtual machines are created and destroyed at each step, without virtual machine migration, because client's requests are supposed to be completely processed within the step.

The workload is defined as the number of requests from end users. The loads are generated in the same shapes as the traces containing all requests to NASA and ClarkNet web servers [9]. In the generated traces, timestamp is compressed to 5 second resolution, and the load curve is scaled to the total capacity of all processing cores in the datacenters in the simulations. The characteristics of these workloads are displayed in Table 1. They exhibit typical workload characteristics of web servers: heavily loaded during daytime and lightly loaded during the night.

In the simulations, we assume that each server has a capacity C of 1000 requests in an interval for one processing core, and calculate the number of servers required by the requests. Two types of servers are considered: single-core servers and quad-core servers. Server's total capacity is assumed to be linear with the number of processing cores. The number of requests that exceed capacity is considered to be drops. The drop rate is defined as the ratio of the number of requests that exceed servers' capacity to the total number of requests. Based on the measurement results in Sect. 2.2, we assume that states OFF, RESTARTING, SHUTTING, and idle consume 7 W, 150 W, 150 W, and 100 W, respectively. Also, transition delays $T_{RESTARTING}$ and $T_{SHUTTING}$ are set to 20 seconds and 10 seconds, respectively.

In order to compare different methods for estimating the number of servers required, we examine the following five different running modes.

- Normal mode (NM) - The traditional mode where all the servers are kept running all the time regardless of load. It is actually a baseline mode for calculating the energy consumption reduction rate in other modes.
- Optimal green mode (OP) - Future load is exactly known in advance and the number of servers at each step can be correctly identified.
- Prediction green mode (PR) - Future load is predicted by the predictor, and the number of servers required at each step is identified based on the predicted load. The predictor is employed as the network of 20:10:1 with a learning rate of 0.3 as mentioned earlier.
- Prediction mode with a tendency-based strategy (PT) - Future load is predicted by a five-step-ahead prediction strategy, as proposed by Zhang et al. in [10].
- Prediction mode plus additional servers (PP): similar to PR, except for the provision of some additional servers. In the simulations, this mode is run with approximately 10% and 20% of the total number of available servers as additional servers.

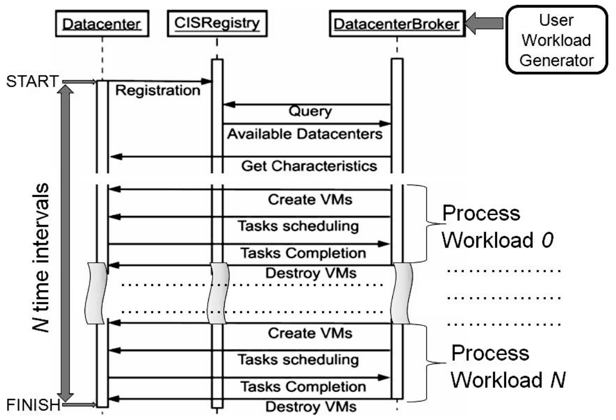

Fig. 8 The modified communication flow.

Table 1 Workload characteristics.

Characteristic	NASA	ClarkNet
Mean request	4.4	13.51
Standard deviation	3.64	8.89
Maximum request	33	80
Minimum request	0	0

4.2 Static Training vs. Dynamic Training

This section compares static and dynamic training policies to find suitable training epochs during execution. Figure 9 shows the experimental results of static and dynamic training policies with NASA and ClarkNet on a 32-single-core datacenter. In Fig. 9 (a), the neural predictor is trained ev-

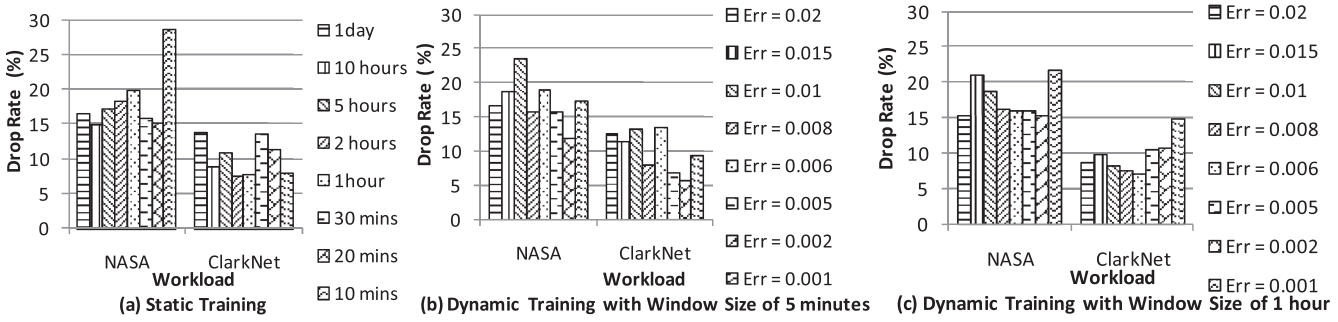

Fig. 9 Static training vs. Dynamic training.

ery 1 day, 10 hours, 5 hours, 2 hours, 1 hour, 30 minutes, 20 minutes, and 10 minutes. In Fig. 9 (b) and (c), 8 different values of the error threshold *Err* taken in the range of [0.001, 0.02] is enforced in the system, along with the window sizes $S1$ of 5-minute-period and 1-hour-period.

In static training policy, it appears that no common training frequency can lead to the best performance in both workloads, as the drop rate fluctuates according to the frequency of training. On NASA, the lowest drop rate (14.9%) is gained with a period of 10 hours, while a period of 2 hours offers the lowest drop rate of 7.3% on ClarkNet. The results imply that choosing an appropriate training period will improve performance, even though the period can mainly be identified empirically.

However, choosing the right training period dynamically is perhaps a clever option. In that regard, the dynamic training policy is likely more promising in terms of both adaptability and performance. An error threshold of 0.002 with a window size of 5-minute-period leads to the best performance in both workloads, with drop rates of 11.8% (NASA) and 5.6% (ClarkNet). This policy requires an average training period of 12 minutes to achieve this improved performance. The window size of 1-hour-period cannot outperform the static training, however, possibly due to low training frequency caused by the large window size. A one-hour window results in high drop rates, starting from 15.2% on NASA and 7.0% on ClarkNet. The outcome confirms the importance of window size in dynamic training policy, and a window size of 5-minute-period with an error threshold of 0.002 proves to be the best. Consequently, we used the configuration of 5-minute-period window and threshold of 0.002 in our subsequent experiments.

4.3 Combination of Dynamic Training and Dynamic Provisioning of Additional Servers

Despite the fact that dynamic training has an edge over static training, it still needs provisioning of additional servers to reduce the drop rate. Figure 10 presents the results of combination of dynamic training and dynamic provisioning with NASA and ClarkNet on a 32-single-core datacenter. We use the best configuration of a 5-minute-period window and an error threshold of 0.002 in dynamic training, combined with 10% (PP10) and 20% (PP20) of the total servers as additional servers. These numbers, however, are dynamically adjusted in different window sizes of 1 minute (WS1), 2 minutes (WS2), 3 minutes (WS3), and 4 minutes (WS4). A window size of zero (WS0) indicates no adjustment in the number of additional servers.

In PP10 and PP20 modes with both workloads, WS3 and WS4 show no effect, as their drop rates and energy consumption amounts remain unchanged in comparison with WS0. This means dynamic provisioning policy does not work with large windows. With smaller windows WS1 and WS2, it tends to reduce the energy consumption, with the trade-off of causing slightly higher drop rates. In the case of either WS1 or WS2, the energy consumption is decreased,

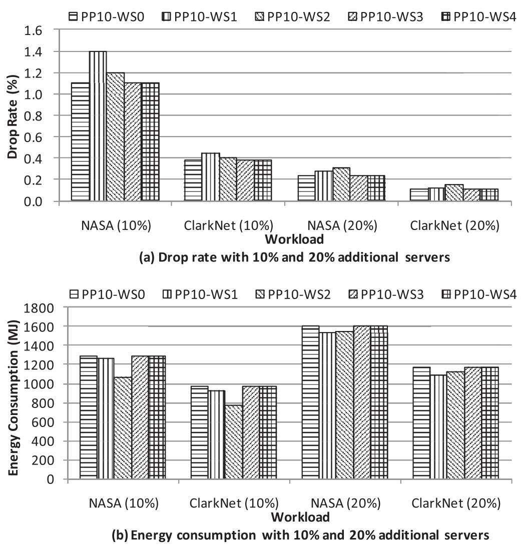

Fig. 10 Dynamic training and dynamic provisioning.

and up to 5.7% (on NASA) and 7.4% (on ClarkNet) lower than the baseline WS0. Overall, the provisioning of additional servers drastically cuts the drop rates, with the lowest drop rates of 0.24% on NASA and 0.11% on ClarkNet. This is a significant improvement compared with the pure dynamic training policy.

4.4 Power and Performance

In this section, we rigorously perform experiments with the two workloads in five different running modes and four datacenters to:

- Find out the relationship between the drop rate (performance) and the energy reduction rate (power);
- Identify the best prediction mode that can offer high energy reduction rates while maintaining low drop rates;
- Calculate the training and ON/OFF periods required in each mode;
- Examine the impact of datacenter architecture on energy consumption; and
- Make comparisons between the two workloads.

Tables 2 and 3 show the results on NASA and ClarkNet load traces, respectively, with the best of each case displayed in boldface. The number of servers in the datacenters was varied from 64, a representation of small-size datacenters, to 512, a representation of medium-size datacenters, each with two types of single-core and quad-core. PR employs a dynamic training policy with 5-minute-period window size and an error threshold of 0.002. Meanwhile, PP20 operates with dynamic provisioning of 20% additional servers and a monitor window size of 2 minutes. The results presented here include the energy consumption reduction rate, which is the ratio of the energy consumption of each mode to that of the baseline normal mode, the drop rate, the required training period of the neural predictor, and the average ON/OFF period for one server. The energy consumption has a direct

Table 2 Performance on NASA with the best of each case displayed in boldface.

Data-center	Mode	Energy (MJ)	Reduction Rate (%)	Drop Rate (%)	Training Period (minute)	ON/OFF Period (minute)
64 Single-Core Servers	OP	2005.9	74.1	0.0	n/a	5.8
	PR	**1362.6**	**82.4**	24.84	12.1	24.2
	PT	1551.2	80.0	36.59	n/a	8.1
	PP20	3040.6	60.7	**0.41**	**12.1**	**24.7**
512 Single-Core Servers	OP	15619.0	74.8	0.0	n/a	5.8
	PR	**11528.6**	**81.4**	20.6	12.0	28.2
	PT	11965.0	80.7	40.06	n/a	8.1
	PP20	22449.6	63.8	**0.79**	**12.0**	**28.5**
16 Quad-Core Servers	OP	553.0	71.4	0.0	n/a	5.7
	PR	**368.6**	**81.0**	19.9	11.9	25.2
	PT	436.7	77.4	26.92	n/a	8.0
	PP20	862.6	55.4	**0.16**	**12.1**	**26.6**
128 Quad-Core Servers	OP	3948.1	74.5	0.0	n/a	5.8
	PR	**2804.0**	**81.9**	22.59	**12.2**	**25.8**
	PT	3051.4	80.3	39.76	n/a	8.1
	PP20	5990.0	61.3	**0.49**	12.1	25.6

Table 3 Performance on ClarkNet with the best of each case displayed in boldface.

Data-center	Mode	Energy (MJ)	Reduction Rate (%)	Drop Rate (%)	Training Period (minute)	ON/OFF Period (minute)
64 Single-Core Servers	OP	1481.0	72.7	0.0	n/a	6.1
	PR	1264.3	76.7	8.37	13.1	15.4
	PT	**1251.4**	**76.9**	26.08	n/a	7.9
	PP20	2284.6	57.8	**0.14**	**13.4**	**15.6**
512 Single-Core Servers	OP	11550.2	73.4	0.0	n/a	6.1
	PR	**9119.2**	**79.0**	15.3	13.5	15.8
	PT	9717.1	77.6	28.8	n/a	7.8
	PP20	17269.9	60.2	**0.21**	**13.5**	**16.0**
16 Quad-Core Servers	OP	400.0	70.5	0.0	n/a	6.1
	PR	**315.7**	**76.7**	9.55	**13.5**	**15.8**
	PT	343.1	74.7	18.96	n/a	7.9
	PP20	680.0	49.8	**0.02**	13.1	15.0
128 Quad-Core Servers	OP	2918.5	73.1	0.0	n/a	6.1
	PR	**2509.6**	**76.8**	8.7	13.3	13.2
	PT	2517.1	76.8	25.74	n/a	7.6
	PP20	4519.1	58.3	**0.12**	**13.6**	**14.0**

relationship with CO_2 emissions. They do not reflect the energy savings for air conditioning systems, resulting from the reduced thermal load from active servers.

First of all, we note an obvious relationship between the energy reduction rate and the drop rate in the simulations. The reduction rate is always directly proportional to the drop rate, except for OP, where the drop rate is maintained at 0%. OP is optimal, but infeasible, since there is no way to exactly know in advance the future load. In OP, a significant energy consumption reduction rate can be achieved, up to 74.8% on NASA and 73.4% on ClarkNet, without affecting performance, as the drop rate is 0% in either case. One of the major drawbacks of this OP is the short ON/OFF period; one server is required to change its status approximately every 5–6 minutes on average.

In contrast, PR and PT are feasible because they apply a prediction mechanism to historical loads for predicting future loads, and then make decisions based on them. PR apparently offers better performance than PT in terms of reduction rate, drop rate, and ON/OFF period. While the difference in reduction rates between them is quite small, the drop rates of PR are much lower than those of PT, approximately 1/3 in the cases of 64 single-core servers and 128 quad-core servers with ClarkNet, and 1/2 in most of other cases. This verifies the advantage of the neural predictor over the Zhang prediction method. The downside is that it must be trained every 12–13 minutes. However, servers in PR need to change status every 25 minutes with NASA and 15 minutes with ClarkNet, as opposed to 8 minutes in PT. In general, PR saves most energy: up to 82.4% on NASA with 64 single-core servers, and 79% on ClarkNet with 512 single-core servers. However, the cost is quite high, as the best drop rates it can offer are as much as 19.9% and 8.37% on NASA and ClarkNet, respectively.

Among the prediction modes, PP20 eclipses the others by proving to be able to provide low drop rates with high energy reduction rates. Obviously, the provisioning of additional servers helps to greatly reduce the drop rate. In the case of 16 quad-core servers with approximately 20% = 4 additional servers, it provides a drop rate of 0.16%, and an energy reduction rate of 55.4% on NASA, and a drop rate of 0.02% with a reduction rate of 49.8% on ClarkNet. It is expected that the drop rate can be reduced further, to a near-zero level, provided that more additional servers are added. Similar to PR, this mode requires its neural predictor to undergo training every 12–13 minutes. The ON/OFF periods in PP20 are the longest: 26 minutes on NASA and 15 minutes on ClarkNet. As a result, PP20 is the most practical mode in real-world systems.

In addition, larger datacenters are likely to yield larger potential relative energy savings because dynamic resizing of the datacenter may take place on a finer granularity to more correctly approximate the load curve. For example, the energy reduction rates of OP on NASA are 74.1% and 74.8% on 64 and 512 single-core servers, respectively. Similar results on ClarkNet: 70.5% and 73.1% (energy reduction rate) on 16 and 128 quad-core servers.

Finer granularity in dynamic datacenter resizing is also gained by a fewer number of cores in a server. The number of cores tends to be inversely proportional to the reduction rate: the fewer cores the server has, the higher the energy reduction rate is. Nevertheless, the difference becomes trivial with a high number of servers. On NASA with OP, for instance, the reduction rate is 74.1% in the case of 64 single-core servers, as against 71.4% in the case of 16 quad-core servers, but it stands at 74.8% and 74.5%, not much difference, in the cases of the 512 single-core and 128 quad-core servers. This tendency also appears on ClarkNet, where a difference of 2.2% in the reduction rates in cases of 64 single-core servers and 16 quad-core servers decreases to only 0.3% in the cases of 512 single-core servers and 128 quad-core servers.

Finally, the overall results suggest that the drop rate on ClarkNet is lower than that on NASA in all cases. This outcome is perhaps due to a higher level of self-similarity of the ClarkNet load trace, which leads to more accurate predictions. This again asserts the strong influence of workload characteristics on the prediction accuracy, and eventually on the overall performance of the system.

5. Energy and Performance Comparison with Other Methods

This section draws energy and performance comparisons between our method and two other methods proposed by Rajamani et al. in [17] and Heath et al. in [18]. As there exist a number of discrepancies in the experimental conditions, our environment and conditions are brought into line with those described in these works to try to make the comparisons as fair as possible.

5.1 Comparison with the Method by Heath et al. [18]

Heath et al. [18] employs modeling and optimization to conserve energy in a heterogeneous server system. The problem is to find the optimal configuration (for identifying the number of servers required) and request distribution (for distributing the requests from clients to the active servers) periodically, so that no resource is overloaded more than its bandwidth and the total energy consumption is minimized. The best configuration is identified based on prediction of future load on the system. The prediction uses a first order derivative of the current average load, in the form of an exponentially weighted moving average, and the last average load of the previous interval, i.e. $predicted\_load = avg\_load + (avg\_load - last\_avg\_load)$ [18]. This prediction scheme is quite simple compared to our neural predictor, and does not take the transition delays of servers into account as we do. Furthermore, the optimization procedure is so time-consuming that it must be solved off-line.

Our experimental conditions are modified as below for a fair comparison:

- The datacenter consists of 8 single-core servers, with the states OFF, RESTARTING, SHUTTING, and idle consuming 7 W, 94 W, 94 W, and 76 W, respectively. Transition delays $T_{RESTARTING}$ and $T_{SHUTTING}$ are set to 80 seconds and 45 seconds, respectively.
- The same trace WorldCup98, consisting of all the requests made to the 1998 World Cup Web site, during the period of June 23 to June 24 is also used in our experiments [9]. The trace is compressed to 20 second resolution, and the last requests of the trace are removed to shorten the experiments to 7500 seconds. Also, it is scaled down by a factor of 60.9 so that the total number of requests becomes 1,268,444, just a little higher than 1,267,475 experienced in their work. The scaled trace has mean, maximum, and minimum requests of 170, 923, and 54. Finally, since the maximum request is 923 and there are 8 servers in the datacenter, the processing capacity of a server is set to 120.

Table 4 Performance comparison.

Method	Mode	Energy (MJ)	Reduction Rate (%)	Requests Serviced	Requests Lost	Drop Rate (%)	Training Times	ON/OFF Period (minute)
Heath [18]	E-O-LC	4.54	0.0	1267475	0	0.0	n/a	n/a
	A-LC	3.24	28.6	1256091	11436	0.9	n/a	n/a
	Model	2.65	41.6	1262736	4417	0.35	n/a	n/a
Rajamani [17]	S_{max}	2.61	42.8	1261242	7202	0.57	n/a	7.6
	S_{max} (1)	2.99	34.4	1268444	0	0.0	n/a	8.4
	S_N	2.56	43.9	1258316	10128	0.8	n/a	8.3
	S_N (1)	2.88	36.8	1268444	0	0.0	n/a	9.3
	S_t	1.39	69.5	1268444	0	0.0	n/a	28.5
Our Method	NM	4.56	0.0	1268444	0	0.0	n/a	n/a
	OP	1.39	69.5	1268444	0	0.0	n/a	28.5
	PR	1.36	70.2	1257360	11084	0.87	1	33.3
	PP (1)	1.87	59.0	1265742	2702	0.21	1	35.7
	PP (2)	2.39	47.6	1268444	0	0.0	1	37.0

Table 4 details the results of our method with the above experimental conditions. Some important results of their proposed method are also reproduced in the table. In theory, our NM and its counterpart E-O-LC (Energy-Oblivious-LC) should consume the same amount of energy for the same number of requests, as they keep the servers continuously running regardless of load. The results show that our NM consumes 4.56MJ to process 1,268,444 requests, almost equivalent to their E-O-LC, which uses 4.54MJ for 1,267,475 requests. The outcome suggests that the conditions for a fair comparison are guaranteed to some extent.

In the ideal case, OP can reduce the energy consumption by 69.5% while ensuring no performance loss. Recall that PR adopts the neural predictor as the network of 20:10:1 and a learning rate of 0.3, combined with dynamic training policy with 5-minute-period window size and an error threshold of 0.002. PP is the prediction mode plus 1 additional server, PP (1), and 2 additional servers, PP (2). Compared with A-LC (Adaptive-LC), PR offers a much higher energy reduction rate, 70.2% as against 28.6%, while maintaining a lower drop rate of 0.87%. With 1 additional server, PP (1) is better than their best Model in both energy reduction rate (59.0% compared to 41.6%) and drop rate (0.21% compared to 0.35%). When the number of additional servers is raised to 2, PP (2) has a distinct advantage over Model, with the energy consumption reduced by 47.6% and no performance loss.

5.2 Comparison with the Method by Rajamani et al. [17]

In [17], Rajamani et al. has proposed several power aware request distribution schemes for server systems, where energy reduction is obtained by turning off some servers under light load conditions. Among them, history-based schemes have been proven to be the most effective. Basically, they base the ON/OFF decisions on future load prediction, similar to our method. Three such schemes, S_{max}, S_N, and S_t, have been proposed with given knowledge of the load. S_{max} assumes the maximum number of servers encountered with the load. S_N is aware of the maximum number of servers required up to the current load point. S_t is the ideal case, like our OP, which knows the load in advance. Accordingly, each scheme identifies the number of active servers at any time t as follows:

$$N_t = (L_t + S_{max} \times D)/C \tag{5}$$

$$N_t = (L_t + S_N \times D)/C \tag{6}$$

$$N_t = (L_t + S_t \times D)/C \tag{7}$$

where L_t is the load at time t, D is the transition delay $T_{RESTARTING}$, and C is the processing capacity of a server.

For performance comparison, we implemented these three schemes in our platform with the same conditions described in the previous subsection. We were not able to obtain the load traces, collected from sites of a financial organization and the 1998 Winter Olympics, used in their experiments. To the best of our knowledge, the data they used are business data and are not open to the public.

The results of their method are also shown in Table 4. In schemes S_{max} and S_N, we added 1 additional server to the schemes, denoted by S_{max} (1) and S_N (1), similar to our PP. First, we see that S_t is identical to OP, since the assumptions are the same. Their highest performance is achieved by S_N (1), when energy consumption is reduced by 36.8% and the drop rate is 0%. It is still, however, lower than PP (2), which has a higher energy reduction rate of 47.6%. In addition, the ON/OFF period of our method is about 3–4 times longer than that of their method, on average more than half an hour as against 8–9 minutes. The overhead is negligible, as the number of trainings is only 1 during the 2-hour period.

6. Related Work

Many papers have studied the dynamic voltage/frequency scaling technique for managing energy and server resources in clusters and data/hosting centers [11]–[13]. The work in [11] has mainly focused on a single server setting, and its energy consumption is reduced by adaptive algorithms for frequency scaling. In [12], a cluster-level power controller has been proposed, although the actual energy reduction is gained at processor level, also by adjusting the frequency. An interesting work was introduced in [13] to find the specific relationship between power and frequency for optimal power allocation at the level of server farms. Even though frequency scaling technique offers substantial power savings, it relies on the settings of hardware components to perform scaling tasks.

A recent trend is to define special states of servers, which can provide energy savings while being able to perform some pre-defined tasks. In [14], PowerNap was proposed as an approach to energy conservation, where the server moves rapidly between an active state and a near-zero-power idle state, called “nap” state, in response to load. Another special state of server, called “Somniloquy”, was presented in [15] to augment network interfaces and enable a server to respond to network traffic such as remote desktop and VoIP in the S3 state for saving energy. [16] introduced a similar barely-alive state that allows remote access to a server’s main memory even when many of its other components have been turned off. This approach has a downside, however, as it requires additional specially-designed hardware to implement the special state.

We believe that a software-based approach that takes advantage of currently available servers’ states would be more cost-efficient and easier for datacenter deployment. In that regard, workload concentration and temporary server turnoff promise the most power savings. The aforementioned power aware request distribution schemes proposed in [17], [18] reduce energy by turning off some servers when the current load can be served by fewer servers. Recently, the energy-aware consolidation problem for Clouds was investigated in [19] to show the performance-energy trade-offs and the existence of an optimal point. In this paper, we design a Green Scheduler that employs a neural predictor to predict user demand in turning servers on and off, considering the predicted demand and servers’ restart delay, as well as the dynamic adaptation to workload changes over time.

7. Conclusion

This paper has presented a Green Scheduler for energy savings in Cloud computing. It is composed of four algorithms: prediction, turning ON/OFF, task scheduling, and evaluation algorithms. We use a neural predictor in the prediction algorithm to predict aggregate request load, based on which the ON/OFF algorithm computes the number of servers needed to process the load. The task scheduling algorithm then directs request traffic away from powered-down servers and toward active servers. The performance is evaluated and monitored by the evaluation algorithm to ensure the ability of the scheduler to adapt to load changes over time. In order to demonstrate its efficacy, we have performed simulations with different parameters and running modes. From the results, we have concluded the best configuration is the prediction mode with a combination of dynamic training and dynamic provisioning of 20% additional servers to ensure service level agreement. It can offer 49.8% energy consumption reduction, while maintaining the drop rate at 0.02% on ClarkNet, and an energy reduction of 55.4% with a drop rate of 0.16% on NASA. The comparison results also show that our method has an advantage over its competitors.

In future work, we plan to extend the system model to deal with a greater diversity of workloads and application services, as well as architectures of datacenters, for a better simulation of cloud environments. Another plan is to compare our scheduler with other power management schemes which employ different load prediction mechanisms. A deployment of the scheduler to show its efficiency in real-world datacenters is also worth considering.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, and M. Zaharia, “Above the Clouds: A Berkeley View of Cloud computing,” Technical Report no.UCB/EECS-2009-28, University of California at Berkley, 2009.

[2] R. Bianchini and R. Rajamony, “Power and energy management for server systems,” Computer, vol.37, no.11, pp.68–74, 2004.

[3] EPA Datacenter Report Congress, http://www.energystar.gov/, accessed June 26, 2010.

[4] Green IT Initiative in Japan, http://www.meti.go.jp/english/policy/GreenITInitiativeInJapan.pdf, accessed June 26, 2010.

[5] T.V.T. Duy, Y. Sato, and Y. Inoguchi, “Improving accuracy of host load predictions on computational grids by artificial neural networks,” International Journal of Parallel, Emergent and Distributed Systems, vol.26, issue 4, pp.275–290, 2011.

[6] T.V.T. Duy, Y. Sato, and Y. Inoguchi, “Performance evaluation of a Green Scheduling algorithm for energy savings in Cloud computing,” Proc. 24th IEEE International Parallel and Distributed Processing Symposium, pp.1–8, April 2010.

[7] R. Buyya, R. Ranjan, and R.N. Calheiros, “Modeling and simulation of scalable Cloud computing environments and the CloudSim Toolkit: Challenges and opportunities,” Proc. 7th High Performance Computing and Simulation Conference, pp.1–11, June 2009.

[8] R. Buyya and M. Murshed, “GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing,” J. Concurrency and Computation, vol.14, no.13-15, pp.1175–1220, 2002.

[9] Traces in the Internet Traffic Archive, http://ita.ee.lbl.gov/html/traces.html, accessed June 26, 2010.

[10] Y. Zhang, W. Sun, and Y. Inoguchi, “Predict task running time in grid environments based on CPU load predictions,” Future Gener. Comput. Syst., vol.24, no.6, pp.489–497, 2008.

[11] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron, “Power-aware QoS management in Web servers,” Proc. 24th IEEE International Real-Time Systems Symposium, p.63, Dec. 2003.

[12] X. Wang and M. Chen, “Cluster-level feedback power control for performance optimization,” Proc. IEEE 14th International Symposium on High Performance Computer Architecture, pp.101–110, 2008.

[13] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, “Optimal power allocation in server farms,” Proc. 2009 Conference on Measurement and Modeling of Computer Systems, pp.157–168, 2009.

[14] D. Meisner, B.T. Gold, and T.F. Wenisch, “PowerNap: Eliminating server idle power,” Proc. 14th International conference on Architectural support for programming languages and operating systems, pp.205–216, 2009.

[15] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, V. Bahl, and R. Gupta, “Somniloquy: Augmenting network interfaces to reduce PC energy usage,” Proc. 6th USENIX symposium on Networked systems design and implementation, pp.365–380, 2009.

[16] V. Anagnostopoulou, S. Biswas, A. Savage, R. Bianchiniy, T. Yang, and F.T. Chong, “Energy conservation in datacenters through cluster memory management and barely-alive memory servers,” Proc. 2009 Workshop on Energy-Efficient Design, 2009.

[17] K. Rajamani and C. Lefurgy, “On evaluating request-distribution schemes for saving energy in server clusters,” Proc. IEEE International Symposium on Performance Analysis of Systems and Software, pp.111–122, 2003.

[18] T. Heath, W. Meira, Jr., B. Diniz, R. Bianchini, and E.V. Carrera, “Energy conservation in heterogeneous server clusters,” Proc. 10th Symposium on Principles and Practice of Parallel Programming, pp.186–195, 2005.

[19] S. Srikantaiah, A. Kansal, and F. Zhao, “Energy aware consolidation for Cloud computing,” Proc. USENIX Workshop on Power-Aware Computing and Systems, pp.1–5, 2008.

[20] http://www.pcreview.co.uk/forums/cpu-load-generator-t3723002.html

Truong Vinh Truong Duy received the B.E. degree in Computer Science from Hochiminh City University of Technology in 2002, and the M.S. degree in Information Science and Systems Engineering from Ritsumeikan University in 2007. He is currently a doctoral student at School of Information Science, JAIST. His research interests lie in the fields of high performance computing, parallel programming, and Grid, Cloud, and Green computing technologies.

Yukinori Sato received the BS, MS, and Ph.D. degree in Information Science from Tohoku University in 2001, 2003, 2006, respectively. From 2006, he engaged in embedded processor system design in Sendai Software Development center of FineArch Inc. and also became a joint research member at Tohoku University. From 2007, he has been working at JAIST as an assistant professor. His research interests include high-speed and low-power computer architectures and reconfigurable computing. Dr. Sato is a member of the IEEE, ACM, and IPS of Japan.

Yasushi Inoguchi received his B.E. degree from Department of Mechanical Engineering, Tohoku University in 1991, and received MS and Ph.D. degree from JAIST in 1994 and 1997, respectively. He is currently an Associate Professor of Center for Information Science at JAIST. He was a research fellow of the JSPS from 1994 to 1997. He was a researcher of PRESTO program of Japan Science and Technology Agency from 2002 to 2006. He was also Courtesy Senior Research Scholar at the University of South Florida from 2008 to 2009. His research interest has been mainly concerned with parallel computer architecture, interconnection networks, GRID architecture, and high performance computing on parallel machines. Dr. Inoguchi is a member of IEEE and IPS of Japan.