International Journal of Future Generation Communication and Networking
Vol. 9, No. 12 (2016), pp. 39-54
http://dx.doi.org/10.14257/ijfgcn.2016.9.12.04

A Power and Performance Management Simulation Platform for Web Application Server Cluster

Zhi Xiong, Zhongliang Xue, Weihong Cai, Lingru Cai and Juan Yang

Department of Computer Science and Technology
Shantou University
Shantou, Guangdong Province, China
{zxiong, 15zlxue, whcai, lrcai, 13jyang2}@stu.edu.cn

Abstract

Web application server cluster has been widely used to improve the performance of web application servers. Because web load is highly variable, we need to dynamically manage cluster's deployment so as to reduce power consumption and meanwhile satisfy load performance demand. To facilitate researchers to evaluate a management strategy or choose key parameters for it, we propose a CloudSim-based simulation platform in this paper. It can simulate different cluster deployment algorithm, request scheduling algorithm and load feature, where cluster's deployment includes the on/off state, CPU frequency and request scheduling parameter(s) of each server. By the aid of HookTimer component, the platform supports periodical and conditional deployment trigger modes, and can calculate some common performance indicators. The usage of interface, dynamic proxy technique and XML configuration file make the platform have good extensibility and configurability. In addition, a request-number-triggered management strategy is proposed and simulated by the platform. The simulation results demonstrate the feasibility of the platform.

Keywords*: Web application server cluster, power management, performance management, simulation, CloudSim*

1. Introduction

Web is one of the most popular applications in the Internet. Web requests can be divided into static requests and dynamic requests. We usually separate them in practice, use web server (*e.g.*, Nginx) to serve static requests, and use web application server (*e.g.*, JBoss) to serve dynamic requests. Compared with static request, the response to a dynamic request needs to be generated on the fly and consume more server resource. Web application server cluster (web cluster for short), for its traits of scalability, high performance, high cost-effectiveness, high availability, and transparency for clients, has been widely used to improve the performance and reliability of web application servers. Hence, various large-scale web applications, such as e-commerce, e-banking and SaaS (Software as a Service) applications, usually adopt web clusters to supply services.

Web cluster is usually deployed to handle peak load that may be significantly larger than in off-peak conditions. Researches show that servers are now gobbling up a huge amount of energy [1], but they, most of the time, are loaded between 10% and 50% of peak, with CPU utilization that rarely surpasses 40% [2]. This leads to an excessive waste of energy. Moreover, the load of web cluster is highly variable. Therefore, we need to dynamically manage cluster's deployment so as to reduce power consumption and meanwhile satisfy load performance demand. The deployment should include the on/off state, CPU frequency (CPU is the largest energy consumption component of a server, and mainstream CPUs all support dynamic frequency scaling technology [1] today) and request scheduling parameter(s) of each server.

When researchers present a new management strategy for web cluster and need to evaluate it, or choose key parameters (*e.g.*, some thresholds) for a strategy, actual tests will consume a lot of equipment, time and energy. Conversely, if using simulation tests, it will effectively save resources. In this paper, we propose a simulation platform based on CloudSim [3], which can simulate different management strategies for web cluster. It supports periodical and conditional deployment trigger modes; can simulate cluster deployment algorithm, request scheduling algorithm and load feature; and can calculate some common performance indicators such as energy consumption, mean response time, drop rate, *etc*. It also has good extensibility and configurability. Besides, we propose a request-number-triggered management strategy, and use the platform to simulate the strategy. The simulation results demonstrate the feasibility of the platform.

The rest of this paper is organized as follows. The related works are introduced in Section 2. Section 3 and 4 describe the outline and detailed design of the simulation platform respectively. The implementation of some important components are given in Section 5. We propose a request-number-triggered management strategy in Section 6, and simulate the strategy with our simulation platform in Section 7. Finally, we conclude in Section 8.

2. Related Works

In the study of cluster's power and performance management, some works [4-6] test their management strategies in real environment, but the cluster scale is small, so the effectiveness of their strategies is only verified in small-scale clusters. Some works [1, 7-12] use simulation tests to evaluate their strategies. However, [7-8] and [9] do not give simulation methods; [1] and [10] just simulate in computational level rather than cluster's actual operation; [11] and [12] use their own simulator to simulate, but do not give the design details of their simulators.

CloudSim is one of the most sophisticated and comprehensive simulation platforms in cloud computing. It is a frame that can be used to model and simulate cloud computing infrastructures and application services, as well as energy-aware computational resources. CloudSim package provides some examples about power-saving simulation of data center, but in these examples, tasks are batch-produced, while actual web request is a kind of streaming load, and energy consumption calculation is not very accurate as they do not consider all possible states of server (such as ignoring the process of server switching on/off). CloudSim has been widely used in various simulation of cloud computing platform, but they all deploy multiple VMs on a Host [13] and save energy by migration and integration of VMs (Virtual Machines), which do not accord with the deployment manner of web cluster [14].

As far as we know, there is no special power and performance management simulation platform for web cluster, and no reference bases on CloudSim to simulate power and performance management strategy for web cluster.

3. Outline Design

This section presents the outline design of our simulation platform.

3.1. CloudSim Introduction

CloudSim is an event-based simulator, and the communication between entities is based on message event. It is written by Java. The core entity class in CloudSim is *SimEntity*, which is responsible for sending messages to other entities and processing the received messages. Each entity that needs to communicate with other entities must extend *SimEntity* and override its some methods. The *send* and *sendNow* methods are used to send events to other entities. The important basic classes in CloudSim include *Cloudlet*, *Datacenter*, *Host*, *VM*, as well as *PowerDatacenter*, *PowerHost* and *PowerVM* which can be used for energy-aware simulation.

3.2. Cluster Model

The simulation object of this paper is dispatcher-based web cluster, which consists of a dispatcher and a number of (back-end) servers. We suppose each server node is a real server or the sole VM of a real server [14]. Nowadays, mainstream CPU usually has four or more cores, and each core has many available discrete frequencies, *e.g.*, Intel Xeon E5-1650 has 6 cores and each core has 15 frequencies. For a multi-core server, if we allow different cores to work at different frequencies (including on/off state), the deployment algorithm will be very complicated and we have to determine a lot of power parameters; therefore, for simplification, we simultaneously adjust each core's frequency and keep them consistent.

3.3. Platform's Overall Structure

The platform includes seven components: request generator, dispatcher, server pool, load feature, scheduling algorithm, deployment algorithm and HookTimer, where the first three are entity components and the last four are extensible components. The entity components are responsible for handling request, and the extensible components are used to realize some user-defined algorithms or functions. These components all have their respective classes.

RequestGenerator, *Dispatcher* and *ServerPool* are the corresponding classes of the three entity components, where the first two are subclasses of *SimEntity* and the last one is a subclass of *PowerDatacenter*. *RequestGenerator* generates requests flow according to request rate, size distribution and time interval distribution. The request size distribution and time interval distribution are supplied by load feature component. *Dispatcher* calls request scheduling algorithm to dispatch requests. Requests are served in *ServerPool*. *ServerPool* also maintains each server's state during the simulation process and modify each server's deployment according to deployment algorithm. The main event relationship among them is shown in Figure 1. These events will be discussed in latter appropriate sections.

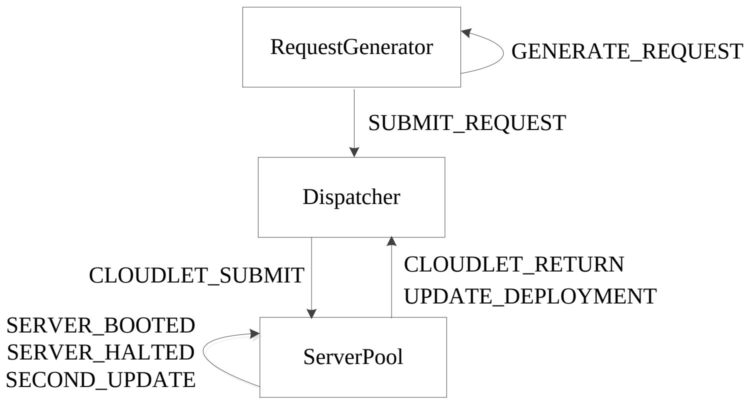


Figure 1. Main Event Relationship among Entity Components

Each kind of extensible component corresponds to an interface, and each extensible component must implement its corresponding interface. The relationship among entity components, extensible components and interfaces is as Figure 2 shows.

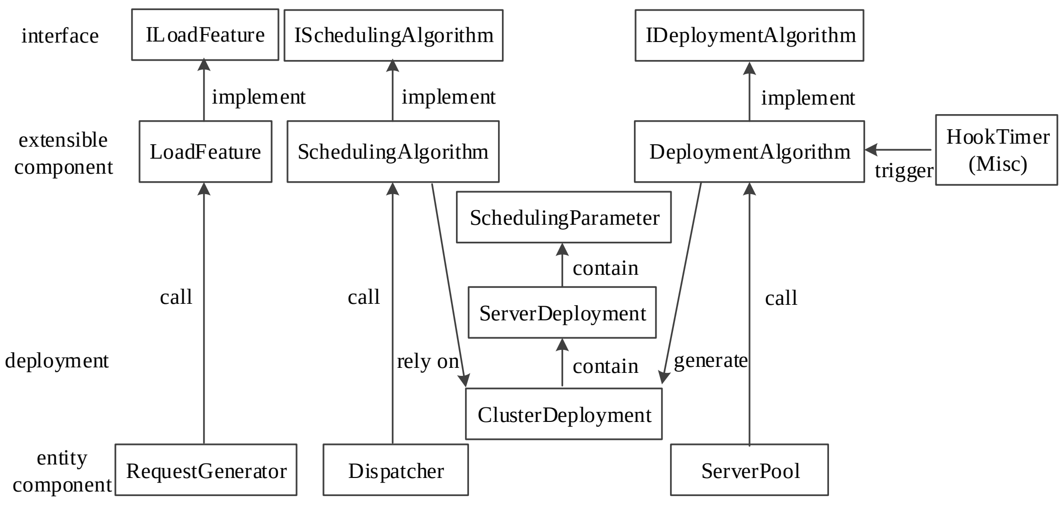


Figure 2. The Relationship between Components and Interfaces

4. Detailed Design

This section discusses the detailed design of our simulation platform.

4.1. Interfaces

The *ILoadFeature* interface contains two methods: "*double getRequestSize()*" and "*double getRequestInterval(double rate)*". They are used to generate the required request size distribution and time interval distribution. Request time interval is obviously related to request rate, so *getRequestInterval* has an input argument.

The *ISchedulingAlgorithm* interface contains two methods. The "*int scheduleRequest(Cloudlet request)*" method is an important method. It selects an appropriate server for the request passed in, and returns the serial number of the selected server. Some scheduling algorithms depend on some parameters to select server, *e.g.* the weights in WLC (Weighted Least Connection) algorithm and the probabilities in probability-based algorithm. So the interface supplies an *initialize* function to initialize scheduling parameters.

The *IDeploymentAlgorithm* interface contains seven methods, and user can realize periodical and/or conditional deployment trigger mode through implementing these methods.

(1) The "*ClusterDeployment requestNumberReport(int serverID, int requestNumber)*" method is used in the case that the cluster's redeployment is triggered by request number, and it will be automatically called by the platform when a server's request number is updated. The typical application pattern is that, if a server's request number is above a given threshold or below another given threshold, redeployment will be triggered. If redeployed, it returns the deployment result; otherwise, *null* is returned.

(2) The "*ClusterDeployment dropRateReport(double[] dropRate)*" method is used in the case that the cluster's redeployment is triggered by request drop rate, and it will be periodically called by the platform when all servers' request drop rates are calculated. Its returned value is similar to that of *requestNumberReport*. The *responseTimeReport and cpuUtilizationReport* are the other two similar methods.

(3) The "*void setInterval(int interval)*" and "*int getInterval()*" method are used to set and get the deployment period respectively in periodical trigger mode. If the returned value of "*getInterval()*" is zero, periodical trigger mode is disabled.

(4) The "*ClusterDeployment deploy()*" method is used in periodical trigger mode. If the returned value of "*getInterval()*" is not zero, this method will be called by the platform every "*getInterval()*" seconds.

4.2. HookTimer Component

Performance measurement is based on information sampling, and the calculation of different performance indicators may need different information sampling modes. Generally, there are two information sampling modes: periodical mode and event-triggered mode. Moreover, the platform supports periodical and conditional deployment trigger modes. So we design an important component — HookTimer, which includes two functions: Hook and Timer.

Hook allows us to hang our customized codes at some pivotal places; Timer allows us to do certain work at regular intervals. HookTimer component must implement *IHookTimer* interface, and the methods defined in *IHookTimer* will be called by the platform at some particular time. Figure 3 gives the nine methods defined in *IHookTimer*, and also illustrates the time when these methods will be called from the angles of simulation time and request handling processes.

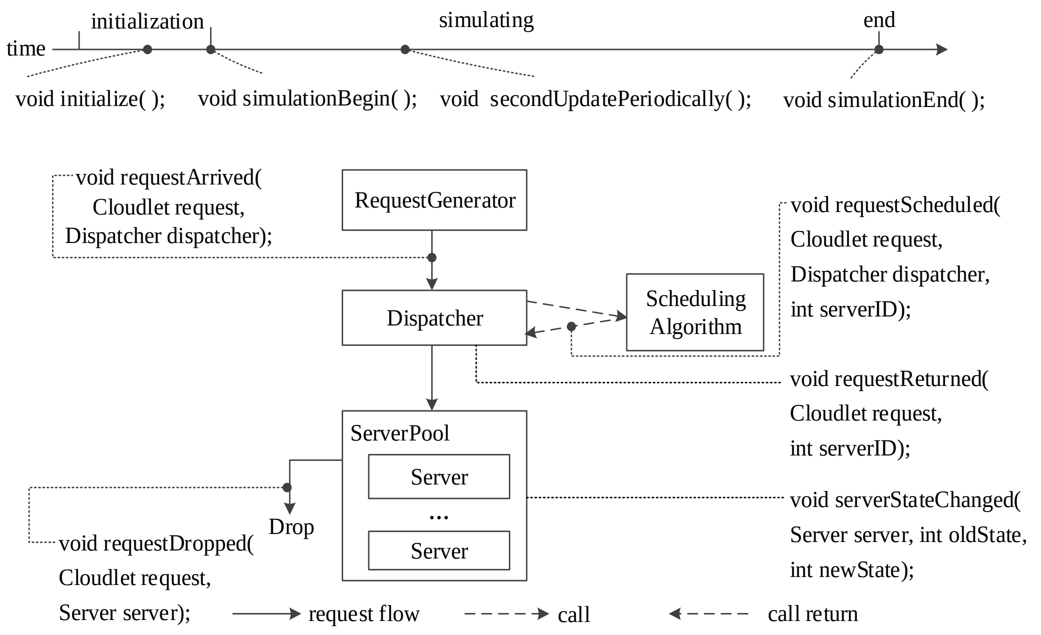

time
initialization
simulating
end
void initialize( );
void simulationBegin( );
void secondUpdatePeriodically( );
void simulationEnd( );
void requestArrived(Cloudlet request, Dispatcher dispatcher);
RequestGenerator
void requestScheduled(Cloudlet request, Dispatcher dispatcher, int serverID);
Dispatcher
Scheduling Algorithm
void requestReturned(Cloudlet request, int serverID);
ServerPool
Server
...
Server
Drop
void requestDropped(Cloudlet request, Server server);
void serverStateChanged(Server server, int oldState, int newState);
request flow
call
call return

Figure 3. The Methods Defined in IHookTimer

4.3. Misc Component

Misc is a class that implements *IHookTimer* interface. Its functions include three aspects: the calculation of some common performance indicators, the realization of two deployment trigger modes, and energy consumption calculation. In addition, users can add their customized HookTimer components to the platform to realize some special functions, *e.g.*, calculate some less-used performance indicators.

In the methods like *requestDropped*，*requestArrived*，*requestScheduled*, *etc* of *Misc*, we record and update corresponding variables so as to calculate some common performance indicators, such as mean response time, drop rate and CPU utilization.

Misc cooperates with *IDeploymentAlgorithm* interface to realize periodical and conditional deployment trigger modes. Once the request number of a server is changed, *Misc* will call the *requestNumberReport* method of *IDeploymentAlgorithm*. *Misc* periodically calculate the performance indicators including mean response time, drop rate and CPU utilization, once these indicators are calculated, it will call the *dropRateReport*, *responseTimeReport* and *cpuUtilizationReport* method of *IDeploymentAlgorithm*, respectively. User can determine whether to trigger redeployment in these functions. As for periodical trigger mode, if the returned value of *getInterval()* of

IDeploymentAlgorithm is not zero, then *Misc*'s *secondUpdatePeriodically()* method will call the *deploy()* method of *IDeploymentAlgorithm* every "*getInterval()*" seconds.

Energy consumption calculation is based on the state transition of each server, so we calculate energy consumption in the *serverStateChanged* method of *Misc*. We define six states:

(1) OFF: The server is shutdown. We usually use suspending-to-RAM to replace it in practice so as to shorten the time of server's switching on/off [6].

(2) BOOTING: The server is starting. Once entering this state, SERVER_BOOTED event will be triggered after a certain period of time.

(3) IDLE: The server provides normal service, but there is no request in the server at this moment.

(4) BUSY: The server provides normal service, and it is serving request at this moment.

(5) CLOSED: The server is ready to shut down and stop receiving new requests, but it is serving the received requests.

(6) HALTING: The server is shutting down. Once entering this state, SERVER_HALTED event will be triggered after a certain period of time. Figure 4 gives the transition conditions among various states.

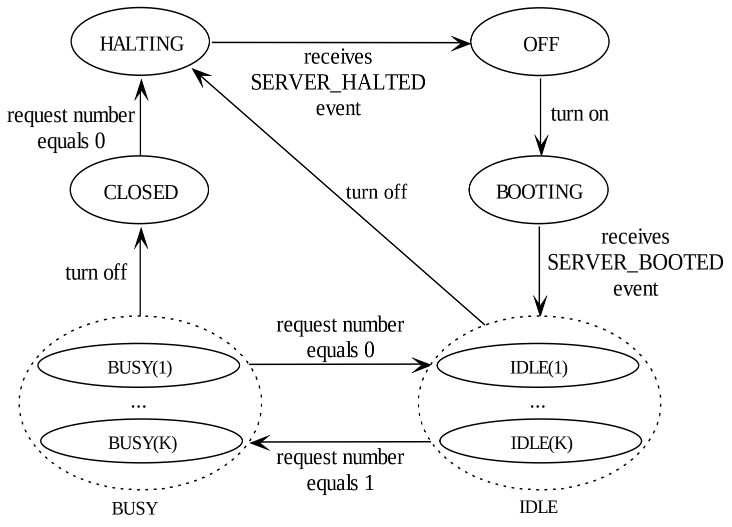


Figure 4. Transition Conditions among Various States

When a server is in IDLE or BUSY state, its frequency may be changed by deployment algorithm, so the two states contain some substates. We take a server that has six available frequencies as an example. Substate BUSY(1) denotes it is busy at the lowest frequency, and IDLE(6) denotes it is idle at the highest frequency. Strictly speaking, CLOSED, HALTING and BOOTING also should have analogous substates. However, considering that these three states are instantaneous states, whether distinguishing substates or not has little effect on the calculation of energy consumption; therefore, we ignore their substates.

For each server, once its state/substate changes (say from *A* to *B*) or a statistic period terminates (say the current state/substate is *A*), we will compute the residence time and power of the state/substate (*i.e.*, *A*), then calculate energy consumption and accumulate it. When a statistic period terminates, the accumulated energy consumption will be outputted and cleared to zero.

4.4. Extensibility and Configurability

Users are allowed to add new extensible components to the platform. Of course, they can name the new components as they like, that is to say, the class names of these new components are unknown in advance. How to use these user-defined new component classes only through modifying configuration file (without modifying platform's codes) is a key problem, which determines the convenience of component extension. In other words, we must guarantee the PnP (Plug and Play) characteristic of extensible components.

Java dynamic proxy technique [15] means using Java reflection to create dynamic implementations of interfaces at runtime. Since each extensible component corresponds an interface, we use Java dynamic proxy technique to realize component dynamic proxy, and then use component dynamic proxy to realize the PnP characteristic of extensible components. Specifically, for every extensible component, the platform reads its configuration information from configuration file, then creates a dynamic proxy of the corresponding component interface and a component object in the proxy at runtime, and then uses the component object through its host proxy.

In addition, the configuration of extensible components and the parameter settings of all components are realized through a configuration file of XML (eXtensible Markup Language) format. Figure 5 gives an example of configuration file.

```
<Configuration>
        <RequestGenerator>
                <LoadFeasure name="Exponential"></LoadFeasure>
        </RequestGenerator>
        <Dispatcher>
                <SchedulingAlgorithm name="Hybrid"
parameters=""></SchedulingAlgorithm>
        </Dispatcher>
        <ServerPool>
                <DeploymentAlgorithm name="RequestNumberThreshold" period="0">
                </DeploymentAlgorithm>
                <Server name="type1" memSize="2048" suspendPower="4.85"
coreNumber = "2"
                count="2">
                        <Frequencies>
                                <Frequency mips="1000" idlePower="65.8"
busyPower="82.5"></Frequency>
                                ......
                                <Frequency mips="2600" idlePower="76.9"
busyPower="140.1"></Frequency>
                        </Frequencies>
                </Server>
                ......
                <Server ......>
                ......
                </Server>
        </ServerPool>
        <HookTimers>
                <HookTimer name="Misc" reportPeriod="60"></HookTimer>
                <HookTimer name="DelayOffAssist" delay="120"></HookTimer>
        </HookTimers>
 </Configuration>
```

Figure 5. An Example of XML Configuration File

5. The Implementation of Entity Components

In this section, we give the implementation of the three entity components, namely request generator, dispatcher and server pool.

5.1. Request Generator

When *RequestGenerator* receives GENERATE_REQUEST event, it proceeds as follows:

```
double size = loadFeature.getRequestSize();
Cloudlet request = generateRequest(size);
sendNow(dispatcherID, Tags.SUBMIT_REQUEST, request);
double interval = loadFeature.getRequestInterval(rate);
send(getId(), interval, Tags.GENERATE_REQUEST, null);
```

User can implement the *getRequestSize* and *getRequestInterval* functions of load feature component to generate the required request size and time interval distribution.

5.2. Dispatcher

Every time *Dispatcher* receives SUBMIT_REQUEST event, it proceeds as follows:

```
Cloudlet request = (Cloudlet)event.getData();
for (i=0; i<hooktimer.length; i++)
    hooktimer[i].requstArrived(request, dispatcher);
int sel = schedulingAlgorithm.scheduleRequest(request);
request.setVmId(sel);
sendNow(serverPoolID, Tags.CLOUDLET_SUBMIT, request);
for (i=0; i<hooktimer.length; i++)
    hooktimers[i].requestScheduled(request, dispatcher, sel);
```

Besides, when it receives CLOUDLET_RETURN event, it will call the *requestRetruned* method of each HookTimer component.

5.3. Server Pool

ServerPool sends itself a SECOND_UPDATE event every one second. When it receives SECOND_UPDATE event, it will call the *secondUpdatePeriodically* method of each HookTimer component. When a new deployment is generated, it will compare new deployment with old deployment, and then adjust each server's on/off state and CPU frequency. It is relatively easy to realize the switching on/off of a server. Below we will discuss how to realize CPU frequency adjustment.

Since VM is the component which directly supplies task processing service in CloudSim, we use a Host and a sole VM deployed in the Host (occupying all resources) to simulate a server node. The running configuration information of Host and VM are set when corresponding entities are created, so CPU frequency adjustment is a little complicated and includes two processes: Host's frequency adjustment and VM's frequency adjustment. Host's frequency can be modified by rewriting the CPU cores' MIPS (Million Instructions Per Second) values of the Host. The host can be obtained through the path "ServerPool->characteristics->hostList->host".

As the modification of VM frequency involves several properties (*e.g.*, "cloudletScheduler", "vmStateHistoryEntry" and "currentAllocatedMips" are all related to frequency information), we use the following simple method to avoid the complexity of code modification. Firstly, we obtain the task processing list "cloudletExeList" in original VM (it records all running tasks' progress information); then, create a new VM with the specified frequency to replace the original one and put the "cloudletExeList" into the new VM; finally, call *updateCloudletprocessing()* to update all tasks.

6. Management Strategy

To verify our platform, we propose a request-number-triggered management strategy suitable for homogeneous web cluster, which bases on request number thresholds and greedy idea to schedule requests and adjust cluster's deployment.

6.1. Request Number Thresholds

Many indicators can be used to measure server load, such as CPU utilization, memory utilization, (current) request number, *etc*. Here, we choose request number, because the dispatcher can easily get the current request number of each server without any additional measures. We set a high threshold (HIGH_THRESHOLD) and a low threshold (LOW_THRESHOLD). When a server's request number is equal or bigger than the high threshold, it means the server is overloaded and we should improve its performance by turning up its frequency. Conversely, when a server's request number is equal or less than the low threshold, it means the server is underloaded and we should turn down its frequency.

In order to guarantee QoS (Quality of Service) and avoid too much request response time, we also set a drop threshold (DROP_THRESHOLD). When the request number of a server goes over the threshold, the server will discard the new requests which are scheduled to it.

6.2. Deployment Algorithm

Suppose a server's current frequency is j, *i.e.*, all the cores are working at the jth frequency. When it is overloaded, we turn up its frequency to j+1. Conversely, when it is underloaded, we turn down its frequency to j-1.

Anti-thrashing is a practical issue to be resolved, *i.e.*, we must avoid servers oscillating between the on/off states caused by small fluctuations of the load. Therefore, when a server's request number becomes zero, we do not turn it off immediately, but adopt the following turning-off scheme. We check each server every D seconds, only if a server's request number equal zero all the time in the past D seconds, we turn it off. Bigger D is more helpful to reduce thrashing, but it will waste a little more energy when the load is in falling stage. We use a HookTimer component *DealyOffAssist* (see Figure 5) to help realize the delay turning-off function.

Obviously, the deployment trigger mode is conditional and the deployment is triggered by request number. So the deployment trigger codes are written in the *requestNumberReport* method *of IDeploymentAlgorithm.* Algorithm 1 gives the pseudo-code of deployment algorithm. In the algorithm, S_{top} denotes the set of servers whose frequency is adjusted to the maximum and meanwhile whose request number is bigger or equal to the high threshold.

6.3. Request Scheduling Algorithm

We combine greedy idea, thresholds and request number to schedule requests. When all the turned-on servers are over-loaded, we use least request number algorithm. Otherwise, we schedule requests on a greedy fashion, specifically, we concentrate load on some servers and let them work at the highest possible frequency, thus turning off other servers as many as possible. Algorithm 2 gives the pseudo-code of request scheduling algorithm. It should be noted that if the request number of the selected server is bigger than DROP_THRESHOLD, the request will be dropped by the server.

7. Simulation Tests

In this section, we simulate the proposed request-number-triggered management strategy with our simulation platform.

7.1. Simulation Scenario

Suppose a web cluster consists of 50 homogeneous servers. Each server has an AMD Athlon 64 X2 Dual-Core 5000+ CPU and 2G memory, Table 1 gives its power parameters which are supplied in [6]. We use a server of this hardware configuration to test, and the

Algorithm 1. Deployment Algorithm

```
ClusterDeployment requestNumberReport(int serverID, int requestNumber) {
    i = ServerID;
    S[i].req_num = requestNumber;
    if (S[i].req_num==0) {
        S[i].delay_off = CloudSim.clock();
        return null;
    }
    if (S[i].delay_off>0 && S[i].req_num==1) {
        S[i].delay_off  = 0;
        return null;
    }
    if (S[i].req_num<=LOW_THRESHOLD) {
        if(S[i].frequency != 1) {
            turn_down (S[i], deploymnet);
            return deploymnet;
        }
        return null;
    }
    if (S[i].req_num==HIGH_THRESHOLD-1 && $i \in S_{top}$) {
        $S_{top} = S_{top}$ - {i};
        return null;
    }
    if (S[i].req_num==HIGH_THRESHOLD && $i \notin S_{top}$) {
        if(S[i]. frequency !=  S[i].maxFrequency) {
            turn_up(S[i], deployment);
            if (S[i]. frequency==S[i].maxFrequency) {
                $S_{top} = S_{top}$ + {i};
                if ($S_{on}$==$S_{top}$ ) {
                    fectch j from $S_{off}$;
                    $S_{off} = S_{off}$ - { j };
                    turn_on(S[j], deployment);
                }
            }
            return deployment;
        }
    }
    return null;
}
```

Algorithm 2. Request Scheduling Algorithm

```
int scheduleRequest(Cloudlet request) {
    select the first S[i] whose req_num < HIGH_THRESHOLD from $S_{on}$;
    if (success)
4             **return** i;
5         select S[i] whose req_num is minimal from $S_{on}$;
6         **return** i;
7     }
```

results show that: the off (suspending-to-RAM) power is about 4.85W; booting and halting process lasts about 2s; the booting power is approximately equal to the mean of busy power and idle power; the halting power approximates to idle power.

Table 1. Power Parameters

Frequency (GHz)	Idle power (W)	Busy power (W)
1.0	65.8	82.5
1.8	68.5	99.2
2.0	70.6	107.3
2.2	72.3	116.6
2.4	74.3	127.2
2.6	76.9	140.1

Suppose request size follows a negative exponential distribution with a mean of 10. Every two minutes is deemed as a time unit. Suppose the request time interval in a time unit also follows negative exponential distribution, *i.e.*, the request arrival process is a Poisson process. We define request mean rate as load. The loads of different time units may be different. In addition, the delay parameter *D* is set to 120 seconds.

7.2. Simulation Results and Analysis

We use a group of thresholds (namely, DROP_THRESHOLD, HIGH_THRESHOLD and LOW_THRESHOLD) (35, 30, 10) to simulate. Our simulation lasts 51 time units. Figure 6 gives load curve, Figure 7, 8 and 9 give the curve of energy consumption, mean response time and drop rate respectively.

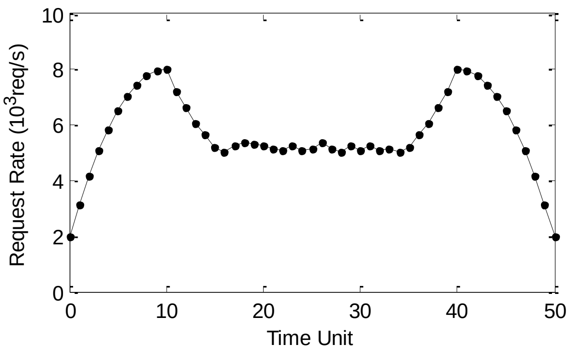


Figure 6. Load

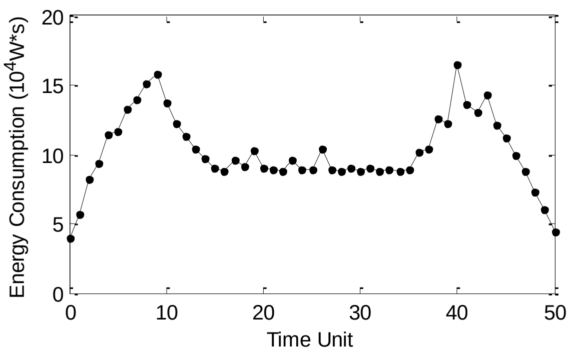


Figure 7. Energy Consumption

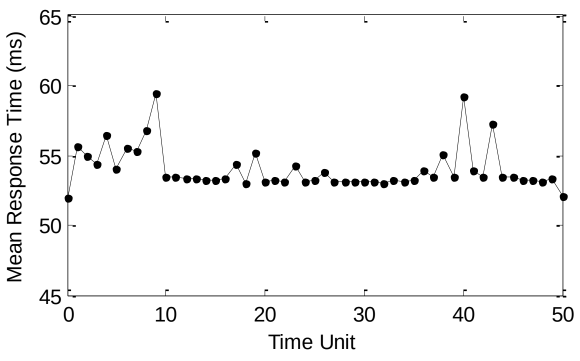


Figure 8. Mean Response Time

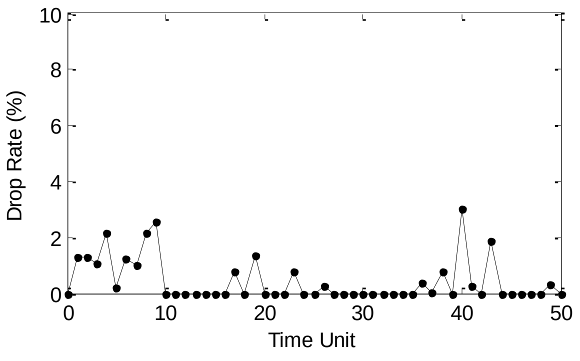


Figure 9. Drop Rate

From the results we can see that: (1) cluster's energy consumption changes along with the load and their trends are basically identical; and (2) when the load increases rapidly, new server will be turned on, because server's booting needs some time, the mean response time and drop rate will increase significantly. The results accord with our expectations for the management strategy on both power and performance control.

We fix the load equal to 5000 req/s, and use three groups of thresholds (35, 30, 10), (38, 33, 13) and (41, 36, 16) to simulate. Each simulation lasts 10 time units. Figure 10 gives the simulation results of mean response time. In the proposed management strategy, bigger threshold means more concurrent request number. When concurrent request number increases, each request will wait more time, so request mean response time will increase. The results coincide with this conclusion.

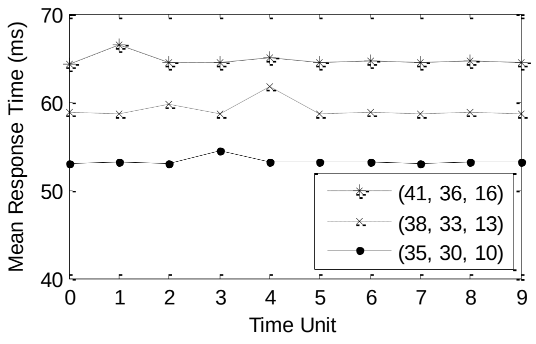


Figure 10. Comparison of Mean Response Time

8. Conclusion

Along with the wide usage of web cluster, its dynamic power and performance management is an urgent problem to be resolved. Evaluating a proposed management strategy by actual tests will consume a lot of equipment, time and energy. This paper proposes a CloudSim-based simulation platform. The main traits and merits of the platform are as follows:

(1) It can simulate web cluster's actual operation and cluster's dynamic deployment, where the deployment includes server dynamic switching on/off, CPU dynamic frequency scaling and scheduling-parameter dynamic adjustment.

(2) It can calculate cluster's energy consumption, request mean response time, request drop rate, and other common performance indicators.

(3) It supports periodical and conditional deployment trigger modes for the introduction of HookTimer component.

(4) Users can conveniently define their own load feature, deployment algorithm and request scheduling algorithm.

(5) It has good extensibility and configurability for the usage of interface, dynamic proxy technique and XML configuration file.

Finally, we put forward a request-number-triggered management strategy, and simulate it with our platform. The simulation results accord with our expectations for the management strategy on both power and performance control, which demonstrates the feasibility of the platform.

Acknowledgment

The authors would like to thank the referees and editors for providing very helpful comments and suggestions. This work is partially supported by the National Natural Science Foundation of China (No. 61202366), the Natural Science Foundation of Guangdong Province (No. S2012010010023), Engineering and Technology Research Center of Digital Content Management of Guangdong Province (No. 2016B090920095), Engineering and Technology Research Center of Guangdong Higher Education Institutes (No. GCZX-A1306), and the Science and Technology Program of Shantou (No. 2014-98).

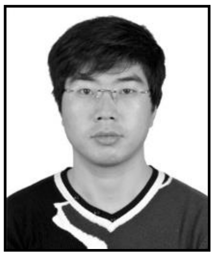
References

[1] K. Bilal, A. Fayyaz, S. U. Khan and S. Usman, "Power-Aware Resource Allocation in Computer Clusters Using Dynamic Threshold Voltage Scaling and Dynamic Voltage Scaling: Comparison and Analysis", Cluster Computing, vol. 18, no. 2, **(2015)**, pp. 865-888.
[2] L. A. Barroso and U. Holzle, "The Case for Energy-Proportional Computing", Computer, vol. 40, no. 12, **(2007)**, pp. 33-37.
[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms", Software: Practice and Experience, vol. 41, no. 1, **(2011)**, pp. 23-50.

[4] S. Kiertscher, J. Zinke and B. Schnor, "CHERUB: Power Consumption Aware Cluster Resource Management", Cluster Computing, vol. 16, no. 1, **(2013)**, pp. 55-63.
[5] C. Lei, L. Luo and W. Wu, "Cloud Computing Based Cluster Energy Monitoring and Energy Saving Method Study", Computer Applications and Software, vol. 28, no. 11, **(2011)**, pp. 242-244, 251.
[6] L. Bertini, J. C. B. Leite and D. Mosse, "Power Optimization for Dynamic Configuration in Heterogeneous Web Server Clusters", Journal of Systems and Software, vol. 83, no. 4, **(2010)**, pp. 585-598.
[7] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler and R. Katz, "NapSAC: Design and Implementation of a Power-Proportional Web Cluster", Computer Communication Review, vol. 41, no. 1, **(2011)**, pp. 102-108.
[8] D. Duolikun, T. Enokido, A. Aikebaier and M. Takizawa, "Energy-Efficient Dynamic Clusters of Servers", Journal of Supercomputing, vol. 71, no. 5, **(2015)**, pp. 1642-1656.
[9] T. Enokido, M. Takizawa and S. M. Deen, "The Delay Time-Based (DTB) Algorithm for Energy-Efficient Server Cluster Systems", Proceedings of 2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems, Birmingham, UK, **(2014)**, pp. 294-301.
[10] P. J. Kuehn and M. E. Mashaly, "Automatic Energy Efficiency Management of Data Center Resources by Load-Dependent Server Activation and Sleep Modes", Ad Hoc Networks, vol. 25, **(2015)**, pp. 497-504.
[11] X. Zheng and Y. Cai, "CMDP Based Adaptive Power Management in Server Clusters", Sustainable Computing: Informatics and Systems, vol. 3, no. 2, **(2013)**, pp. 70-79.
[12] G. Terzopoulos and H. Karatza, "Energy-Efficient Real-Time Heterogeneous Cluster Scheduling with Node Replacement Due to Failures", Journal of Supercomputing, vol. 68, no. 2, **(2014)**, pp. 867-889.
[13] H. He and D. Liu, "Optimizing Data-Accessing Energy Consumption for Workflow Applications in Clouds", International Journal of Future Generation Communication and Networking, vol. 7, no. 3, **(2014)**, pp. 37-48.
[14] Z. Xiong, S. Zeng and H. Lu, "Online Automatic Energy-Saving Deployment under QoS Guarantee for Web Server Cluster", Proceedings of IEEE International Conference on Information and Automation, Yinchuan, China, **(2013)**, pp. 25-30.
[15] W. V. Heiningen, T. Brecht and S. MacDonald, "Exploiting Dynamic Proxies in Middleware for Distributed, Parallel, and Mobile Java Applications", Proceedings of the 20th International Parallel and Distributed Processing Symposium, Rhodes Island, Greece, **(2006)**.

Authors

Zhi Xiong, he received the Ph. D. degree in Communication and Information System from Wuhan University, China. He is currently an associate professor in the Department of Computer Science and Technology at Shantou University, China. His research interests include server cluster, green computing and cloud computing.



Zhongliang Xue, he received the B.E. degree in Computer Science and Technology from Jiangsu University, China. He is currently pursuing the M.E. degree in Computer Application Technology at Shantou University, China. His research interests include big data and server cluster.

Weihong Cai, he received the Ph.D. degree in Communication and Information System from South China University of Technology, China. He is currently a professor in the Department of Computer Science and Technology at Shantou University, China. His research interests include information security, network and communications.

Lingru Cai, she received the Ph.D. degree in System Engineering from Huazhong University of Science and Technology, China. She is currently an associate professor in the Department of Computer Science and Technology at Shantou University, China. Her research interests include modelling and simulation, system dynamics and game theory.

Juan Yang, she received the B.E. degree in Computer Science and Technology from Tianjin Polytechnic University, China. She is currently pursuing the M.E. degree in Computer Application Technology at Shantou University, China. Her research interests include cloud computing and server cluster.