Energy Efficient Allocation of Virtual Machines in Cloud Computing Environments Based on Demand Forecast

Jian Cao, Yihua Wu, and Minglu Li

Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{cao-jian,li-ml}@cs.sjtu.edu.cn,
darwink@sjtu.edu.cn

Abstract. In cloud computing environments, demands from different users are often handled on virtual machines (VMs) which are deployed over plenty of hosts. Huge amount of electrical power is consumed by these hosts and auxiliary infrastructures that support them. However, demands are usually time-variant and of some seasonal pattern. It is possible to reduce power consumption by forecasting varying demands periodically and allocating VMs accordingly. In this paper, we propose a power-saving approach based on demand forecast for allocation of VMs. First of all, we forecast demands of next period with Holt-Winters' exponential smoothing method. Second, a modified knapsack algorithm is used to find the appropriate allocation between VMs and hosts. Third, a self-optimizing module updates the values of parameters in Holt-Winters' model and determines the reasonable forecast frequency. We carried out a set of experiments whose results indicate that our approach can reduce the frequency of switching on/off hosts. In comparison with other approaches, this method leads to considerable power saving for cloud computing environments.

Keywords: cloud computing, power consumption, demand forecast, allocation of virtual machines, modified knapsack algorithm, self-optimization.

1 Introduction

Cloud computing emerges as an efficient approach to meet the growing computational requirements of commercial projects and scientific research ones in a cost-effective way. IaaS (Infrastructure as a Service), the delivery of the computing infrastructure as a fully outsourced service, accelerates the development of cloud computing. In recent years, increasing demands for computational resources have led to significant growth in the number of cloud computing servers, along with an estimated doubling in the energy used by these servers and the power and cooling infrastructures that support them [1]. High power consumption gives rise to a large amount of operational cost which can accumulate more than the construction cost of servers and infrastructures in a short period of time. As a result, power management is important to cost control for cloud computing environments.

The adoption of virtualization brings about the problem of how to allocate VMs among hosts reasonably. On the one hand, more hosts are added to current environment when computing resources cannot meet with the requirements of all VMs. Service providers of cloud computing have to guarantee Quality of Service (QoS) according to Service Level Agreement (SLA), which results in switching on more hosts to deal with resource demands when needed. On the other hand, shutting down some hosts on which all deployed VMs are idle is a way to conserve power consumption. But energy is consumed while a host is being powered up or down and not performing any useful work. The frequent switching on/off hosts which usually needs two to three minutes also leads to the delay of task execution. One way to avoid this situation is to keep idle hosts in standby mode, which consumes much less power than switching on/off frequently. Therefore if we know that the amount of demand will increase in a very short period, we can keep some idle hosts waiting for the future demand rather than switching them off now and switching them on when needed.

In this paper, we propose an approach to make this decision with the help of demand forecast. Our approach can allocate VMs to hosts in accordance with time-variant demand of seasonal pattern. We define demand as the number of requests for VMs from each user. The result of our work is verified by experiments performed on the basis of CloudSim [2], a simulation framework for cloud computing environments.

The remainder of this paper is organized as follows. Section 2 presents the related works. Section 3 describes the system model behind our implementation and experiments. Section 4 introduces the design of algorithm in details. Experimental results are presented in section 5. Section 6 concludes the paper and discusses about the possible directions for future work.

2 Related Work

In recent years, extensive efforts have been put into the research of the VM allocation in cloud computing environments. Walsh et al. [3-4] proposed an approach based on utility function to dynamically manage data centers with different hosts. The utility function is designed for two levels. The service level is used to calculate resource demands of application and the resource level is used to allocate resources among applications. Walsh laid emphasis on the increase of resource utilization, while our focus is on the reduction of power consumption.

Beloglazov and Buyya [5] proposed an approach using Best Fit Decreasing algorithm to allocate VMs among hosts. VMs are sorted in descending order by current utilization and allocated to a host that provides the least increase of power consumption. The advantage of this approach is the stable utilization of resources while the disadvantage is that threshold policy may results in unnecessary migration of VMs and switching on/off hosts.

Bobroff et al. [6] put forward a dynamic management algorithm to reduce required active physical resources without violating the SLA agreement. The resource demand of VMs is predicted and sorted in descending order. Furthermore, the first-fit bin-packing heuristic is used to minimize the active hosts at each interval. However, this algorithm only takes CPU into account, which is not enough for real application. This approach is similar to our proposed one, but it doesn't count the wasted energy due to the unnecessary switching on and off. More often than not, it is more energy efficient to keep hosts standby than switch them off and on in a very short period.

Khanna et al. [7] proposed a method to determine a VM configuration while minimizing the number of hosts needed. The placement of VMs is triggered to revise by events such as change of CPU utilization and memory availability. The strength of this method is that the authors take into account VM migration costs in terms of the additional CPU cycles and memory needed to stop a VM. But the placement algorithm in [7] does not consider power saving by switching off unneeded hosts.

Steinder et al. [8] presents a system that manages heterogeneous workloads to their performance goals by dynamic reallocation of VMs. The VMs are migrated as needed between host machines as new tasks arrive. While the placement approach in [8] consolidates workloads, it does not save power by switching off unneeded machines, and does not consider the cost of the control incurred during migration of the VMs.

The authors of [9] solved the problem of placing application instances on a given set of server machines to adjust the amount of resources available to applications in response to varying resource demands. Though this problem is similar to the allocation of VMs in cloud computing environments, they do not consider the physical machines in standby mode to reduce power consumption.

3 A System Model for VMs Allocation

In cloud computing environments, it is allowed to launch VMs with a variety of operating systems, load them with users' own custom application, manage network's access permissions and finish computing tasks as required. These diverse applications result in unpredictable demands. In general, service providers start, terminate and monitor hosts to present on-demand service, but the following issues cannot be avoided: firstly, some hosts have to be switched on or off frequently; secondly, unbalanced distribution of load exists among different hosts.

Without loss of generality, suppose there are m users $\{U_1, U_2, \cdots, U_m\}$, corresponding to m types of VMs $\{V_1, V_2, \cdots, V_m\}$. The demand from U_i can only be handled by V_i and V_i can be launched on various hosts depending on its resource requirements. An allocation algorithm should be applied to map VMs to hosts on condition that information of needed VMs to satisfy demands from users is gathered.

Figure 1 is the system architecture to implement our solution. As shown in Figure 1, the cloud computing environment is made up of many heterogeneous hosts. It is assumed that each user's demand corresponds to one VM which can be allocated to any host as long as this host can provide enough computing resources to run this VM.

The key components in this architecture are demand analyzer, scheduler and self-optimizing module. Demand analyzer is responsible for collecting, analyzing and predicting users' demands. It relies on a prediction model to forecast future load change. In order to make the prediction model work, its parameters' values should be estimated based on analysis of historical data. Demand analyzer provides scheduler with the forecasted resource demands as input for allocation process of VMs. In addition to allocate VMs to some hosts, scheduler also determines which hosts to be shut down and which ones to be kept standby. The tasks of self-optimizing module are to estimate and update the values for the parameters in prediction and to select the reasonable forecast frequency which has a big influence on the results of our method.

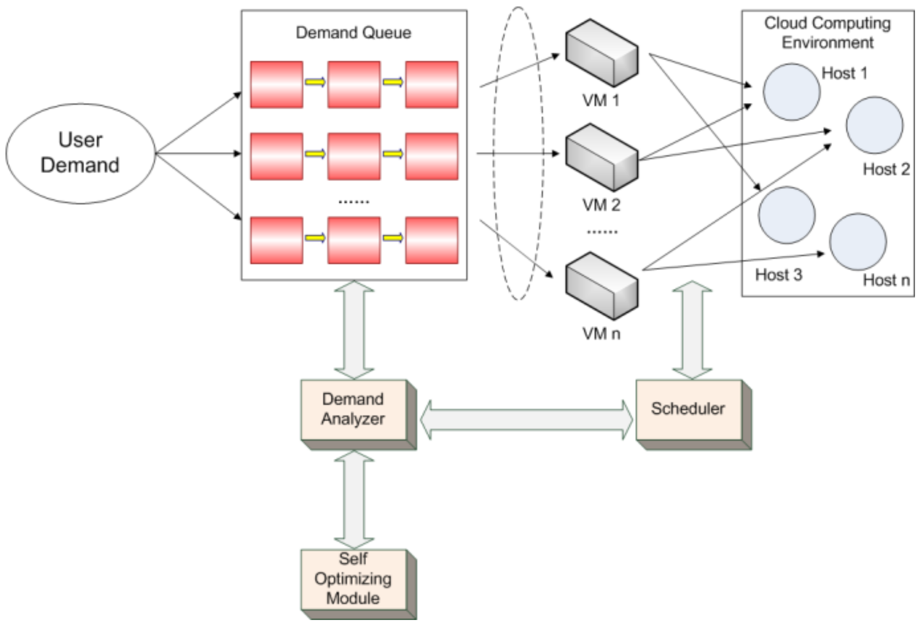

Fig. 1. System architecture for VM allocation

In this system, Holt-Winters' additive model is selected for demand prediction. A modified knapsack algorithm is embedded in the scheduler. And self-optimizing module makes use of hill climbing method to determine optimized values for parameters and forecast frequency automatically.

4 Algorithms

4.1 Problem Statement

In this paper, we take two kinds of resources into consideration: computing cores and memory. The formal representation of this problem is as below:

For VMs:
$\{V_1, V_2, \cdots, V_m\}$, m is total number of types.
V_i: $\{v_{i1}, v_{i2}, \cdots, v_{ip_i}\}$, p_i is total number of VMs for type i.
For hosts:
$\{h_1, h_2, \cdots, h_n\}$, n is total number of hosts.

$capacity_{core}(k)\ \forall k \in \{1,2,\cdots,n\}$ represents available cores of host k.
$capacity_{memory}(k)\ \forall k \in \{1,2,\cdots,n\}$ represents available memory of host k.
For each VM:

$$core_{ij}(k) = \begin{cases} c_{ij} & if\ v_{ij}\ is\ deployed\ on\ h_k \\ 0 & if\ v_{ij}\ is\ not\ deployed\ on\ h_k \end{cases}$$

$$memory_{ij}(k) = \begin{cases} m_{ij} & if\ v_{ij}\ is\ deployed\ on\ h_k \\ 0 & if\ v_{ij}\ is\ not\ deployed\ on\ h_k \end{cases}$$

Where c_{ij} represents the required number of cores for some VM, m_{ij} represents the required amount of memory for some VM.

Then the problem becomes:

For $\forall i \in \{1,2,\cdots,m\}$ $p_i = Forecast_i(t+\Delta t)$, we should keep: $\forall k \in \{1,2,\cdots,n\}$,

$$\sum_{i=1}^{m}\sum_{j=1}^{n} core_{ij}(k) \le capacity_{core}(k)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} memory_{ij}(k) \le capacity_{memory}(k)$$

Where for each type of VM, the actual number of VMs to be deployed on hosts should be equal to the forecasted value $Forecast_i(t+\Delta t)$. On the other hand, $\sum_{i=1}^{m}\sum_{j=1}^{n} core_{ij}(k)$or $\sum_{i=1}^{m}\sum_{j=1}^{n} memory_{ij}(k)$, total resources required for all the VMs deployed on a certain host should be less than or equal to the available resources on that host.

For example, $\{V_1, V_2, V_3\}$ (i.e. $m = 3$) stands for three different types of VMs. According to the prediction, the number of VMs for each type is 120, 42 and 12 respectively. $\{h_1, h_2, \cdots, h_{18}\}$ (i.e. $n = 18$) stands for the set of hosts. Each host has limited resources, for instance, $capacity_{core}(10) = 24$ represents the number of available cores of the 10th host is 24. $capacity_{memory}(10) = 65536$ represents the amount of available memory of the 10th host is 65536MB. Then the problem becomes how to allocate 120 VMs of type 1, 42 VMs of type 2 and 12 VMs of type 3 to 18 hosts according to the resource limitations of each host and the resource requirements of each VM.

The details of demand forecast, self-optimizing module, etc. will be discussed in the following part.

4.2 Demand Forecast

In our proposed approach, demand actually refers to the number of requests for VMs. In order to let allocation of VMs to hosts keep pace with dynamic demands from different users, demand forecast is an effective way to deal with such issue. It can be observed from real applications that the following two assumptions are reasonable for the analysis of demands in cloud computing environments: First, demands from the same user are similar. Second, a certain type of demand follows some seasonal pattern. For example, daily load curve, 10:00-11:00 and 14:00-15:00 are two peaks, while 02:00-03:00 is a valley.

On the basis of these assumptions, we make use of Holt-Winters' exponential smoothing method [10-12] which has ability to adapt to changes in trends and seasonal patterns. The equations of additive model for Holt-Winters are defined as [11]:

$$level: L_h = \alpha(y_h - S_{h-c}) + (1-\alpha)(L_{h-1} + T_{h-1}) \tag{1}$$

$$trend: T_h = \beta(L_h - L_{h-1}) + (1-\beta)T_{h-1} \tag{2}$$

$$seasonal: S_h = \gamma(y_h - L_h) + (1-\gamma)S_{h-c} \tag{3}$$

$$\text{forecast: } \mathrm{F_{h+k}} = \mathrm{L_h} + \mathrm{kT_h} + \mathrm{S_{h+k-c}} \tag{4}$$

Where c is the length of the seasonal cycle, for $0 \le \alpha \le 1,\ 0 \le \beta \le 1,\ 0 \le \gamma \le 1$. We can choose the best values of α, β and γ through trial-and-error process with mean absolute percentage error (MAPE) [10-13] or mean absolute scaled error (MASE) [14] as evaluation criteria. On the other hand, initial values of the level, trend and seasonality are usually determined by two complete seasonal cycles. Equations are listed as below [11]:

$$L_c = \frac{1}{c}\sum_{i=1}^{c} y_i \tag{5}$$

$$T_c = \frac{1}{c}\left[\frac{y_{c+1} - y_1}{c} + \frac{y_{c+2} - y_2}{c} + \cdots + \frac{y_{2c} - y_c}{c}\right] \tag{6}$$

$$S_i = y_i - L_c, i = 1,2,\cdots,c \tag{7}$$

Holt-Winters' exponential smoothing method assumes the time series is composed by a linear trend and a seasonal cycle. It constructs three statistically correlated series (smoothed, seasonal and trend) and projects forward the identified trend and seasonality. Obviously, the seasonal component is prerequisite for this method to model different demands. Furthermore, how to stop this method from being unduly influenced by demands that are unusually high or low (i.e. outliers) is another issue to deal with. In a word, Holt-Winters' exponential smoothing method can only be used to model time-varying demands or user behaviors of some seasonal pattern.

4.3 Modified Knapsack Algorithm for VMs Allocation

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a cost and a value, determine the number of each item to include in a collection so that the total cost is less than or equal to a given limit and the total value is as large as possible [15-16]. The VM allocation is a problem for the admission of new requests for virtual machines provisioning and the placement of virtual machines on hosts. Its optimization goal is to produce more benefits with reduction of cost, for example, charge for running instances. Hence, these two problems are similar with each other and we can use knowledge of knapsack problem to deal with VM allocation.

Our algorithm is made up of bounded and multi dimensional knapsack problems. In our algorithm, we regard each host as a knapsack and each VM as an item. The capacity of knapsack consists of available cores and memory for each host. As mentioned in the problem statement, each item has two different kinds of cost (i.e. cores and memory) and a value. We describe cost by using resource requirements of each VM. On the other hand, value is expressed in terms of Amazon's Elastic Cloud pricing [17]. It is assumed that there are three kinds of items corresponding to three types of VMs: small, large and extra large. Values for them are \$0.12, \$0.48 and \$0.96 per hour respectively. Our aim is to find how to allocate VMs among hosts with the largest value under the conditions of limited capacity.

Table 1. Modified Knapsack Algorithm Pseudo Code

Steps of Pseudo Code

```
 1. Forecast demand;
 2. Get predicted amount of VMs for each type;
 3. for each host in hosts
 4.     Get capacity of host;
 5.     Initialize array of VM selection;
 6.     Initialize array of state transition;
 7.     iterate over each type of VM
 8.         if both limit are observed
 9.             Calculate new value by using state transition;
10.             if new value > current max value
11.                 if bounded conditions are met
12.                     Update current max value;
13.                     Record results into VM selection;
14.                 End if;
15.             End if;
16.         End if;
17.     End iterate;
18.     Update bounded conditions for next host;
19. End for;
```

The equation of state transition is needed to solve knapsack problem with dynamic programming. Here we take 2-dimension knapsack problem as our basic model. The knapsack (i.e. host) will have to bear two kinds of cost when choosing any item (i.e. VM). The 1st kind of cost is the number of cores. The 2nd kind of cost is the amount of memory.

Then the equation of state transition is:

$$r[c(i)][u][w] = max\begin{Bmatrix} r[c(i-1)][u][w] \\ r[c(i-1)][u-k*x[i]][w-k*y[i]]+v[i] \end{Bmatrix} \quad (8)$$

Where $0 \leq k*x[i] \leq U$ and $0 \leq k*y[i] \leq W$

For this equation:

i: The i^{th} category for item
u: Sum for 1st kind of cost
w: Sum for 2nd kind of cost
$x[i]$: The 1st kind of cost for item of i^{th} category
$y[i]$: The 2nd kind of cost for item of i^{th} category
$v[i]$: The value for item of i^{th} category
$c(i)$: The first i categories of items
$r[c(i)][u][w]$: The largest value by choosing the first i categories of items when the sum for 1st kind of cost is u and the sum for 2nd kind of cost is w.
U: The maximum for 1st kind of cost (i.e. capacity of knapsack)
W: The maximum for 2nd kind of cost (i.e. capacity of knapsack)

From the equation of state transition, we know that there are multiple choices (i.e. 0 pieces, 1 piece, 2 pieces...) for each item. No matter which choice is made, conditions $0 \leq k*x[i] \leq U$ and $0 \leq k*y[i] \leq W$ should be met to ensure the destination host has enough resources to deal with demands. For item i, we assign $s[i]$ to represent the range of k and it is:

$$0 \leq s[i] \leq \left\lfloor min\left(\frac{U}{x[i]}, \frac{W}{y[i]}\right)\right\rfloor \quad (9)$$

As a result, the time complexity of our algorithm is $O(max(U,W)\sum s[i])$. The better solution in dynamic programming for knapsack problem is to use monotone optimal policy, which is beyond the scope of this paper.

4.4 Self-optimizing Module

The self-optimizing module has two tasks: (1) updating values of parameters in forecasting model; (2) determining reasonable forecast period.

For the first task, initial values of parameters α, β and γ in forecasting model are chosen by MAPE or MASE as evaluation criteria. We take 0.1 as the minimum step to iterate from 0 to 1 for each parameter. The group of α, β and γ with the smallest forecast error is the initial values of parameters for future forecast. Along with the varying demands, MAPE or MASE for current parameters becomes larger and larger, which means the accuracy of forecast declines. Therefore, our self-optimizing approach is to set a threshold for forecast error. Given that forecast error exceeds the threshold, parameters are needed to update with the recent several complete cycles of data through trial-and-error process.

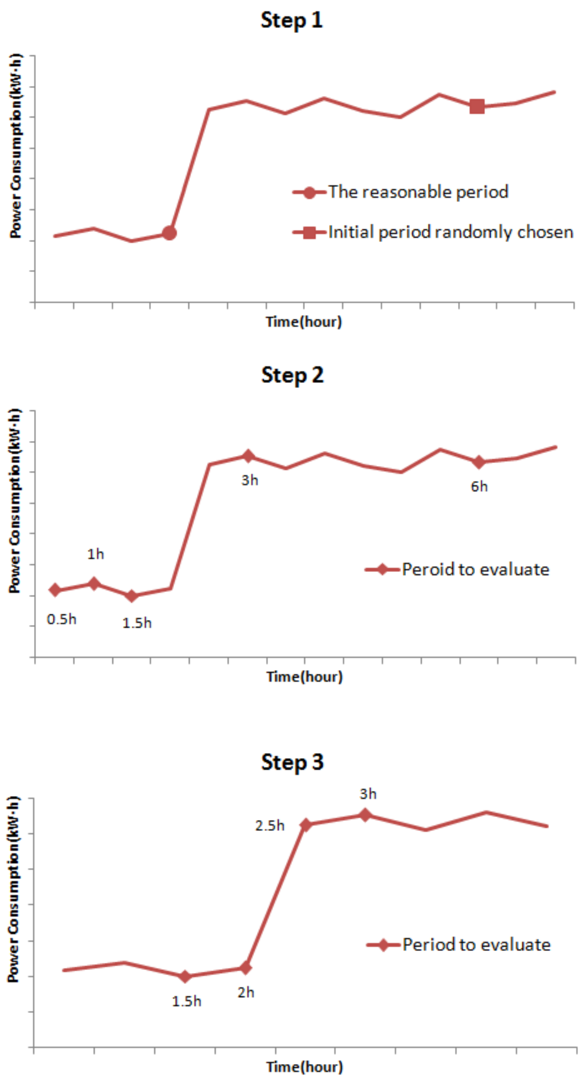

Fig. 2. Process of self-optimization for forecast period

For the second task, self-optimization of forecast period makes use of hill climbing method. Based on our experimental results, there is a sharp increase in the curve of power consumption for different periods and the most reasonable forecast period is close to the start point of this sharp increase. The first step is to select initial period randomly. The second step is to use neighborhood function to get a set of new periods in certain range and evaluate their effect on power consumption. These two steps will be repeated until a platform of low power consumption appears which means the start point of the sharp increase is found. The third step is to evaluate periods on this platform with number of iterations for knapsack as criterion. Small forecast period leads to more iteration that is needed to work out allocation of VMs to hosts. For example, we make two assumptions: (1) the minimum step of period is 0.5 hour; (2) the most reasonable period is 2 hours. The initial random period in the first step is 6 hours. The set of new periods generated by the neighborhood function in second step is 6, 3, 1.5, 1 and 0.5 (the latter is half of the former). Based on the evaluation result, period of 1.5 hours becomes the start point of the sharp increase, while period of 3 hours is the end point. The set of periods to evaluate in the third step is 3, 2.5, 2 and 1.5. In the end, the forecast period with low power consumption and small number of iterations is found. The self-optimizing process of this example is illustrated in Figure 2.

5 Experiments

5.1 Experimental Setup

Our experiments are performed on CloudSim which is a new, generalized, and extensible simulation framework that allows seamless modeling, simulation, and experimentation of emerging cloud computing infrastructures and application services [2]. By using CloudSim, researchers and industry-based developers can test the performance of a newly developed application service in a controlled and easy to set-up environment [2].

To simulate our algorithm and collect the results, we need to set up the configuration of hosts and VMs.

Host Configuration. To simulate computing and storage abilities of hosts, we take IBM System X3850 X5 as our sample host. Configuration of X3850 is collected from IBM website: 4 CPUs (each with 6 cores), 32GB DDR3 1066Mhz memory [18]. Power parameters are listed in Table 2.

Table 2. Host Power Parameters [18]

Host	Power Rating	Peak Power	Standby Power	Start Duration	Idle Duration
IBM System X3850	1975W*2	2765W*2	98.75W*2	2min	20min

VM Configuration. Experiments are conducted with three types of VMs. Configurations of our VMs are determined with reference to standard instance types of Amazon EC2 which are well suited for most applications. The detailed configurations are listed in Table 3.

Table 3. Virtual Machine Types[19]

Virtual Machine Type	Core(s)	Memory	Storage
Small	1	1.7GB	160GB
Large	4	7.5GB	850GB
Extra Large	8	15GB	1690GB

5.2 Demand Collection

According to our assumption, three types of time-varying resource demands are collected and predicted. The first one is collected from statistics of visit requests published by Shanghai Education Website. The second one is provided by R2Group as example of their tool for website statistics. The third one is from sample.org. All of them include data of three continuous days (72 hours). A complete cycle is set to 24 hours, from 0:00AM-24:00PM, considering seasonal patterns of collected data. The first two complete cycles will help us calculate initial values of level, trend, seasonality and parameters α, β, γ in the equations. The third cycle is used to evaluate the accuracy of our forecasting model.

5.3 Performance of Forecast

Due to similar process of forecast for different types of demands, we just take data from Shanghai Education Website as an example for performance evaluation. Figure 3 illustrates the comparison of actual and forecasted demands. The parameters α, β and γ in forecasting model are set to real value between 0.1 and 1. The exact values of parameters are listed below in Table 4.

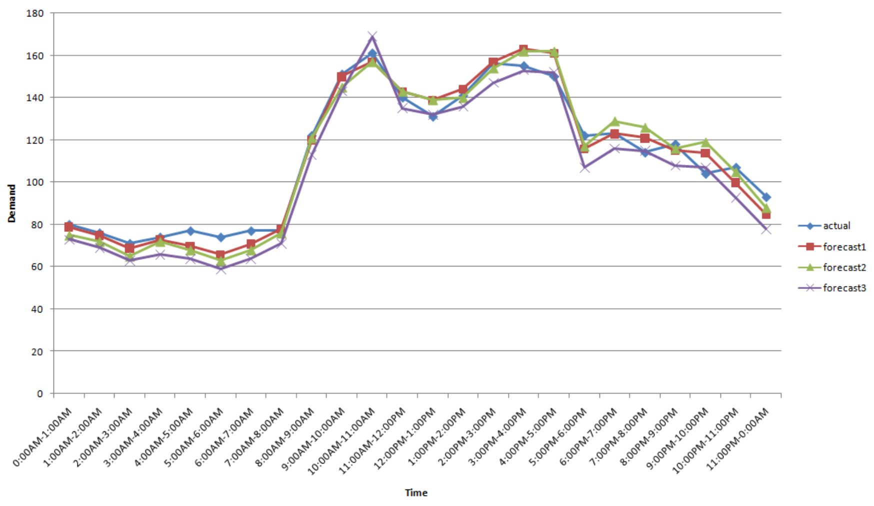

Fig. 3. Comparison of actual and forecasted demands

Table 4. Values of Model Parameters

Forecast	α	β	γ	MAPE
forecast1	0.1	0.2	0.3	0.0434
forecast2	0.9	0.2	0.1	0.0562
forecast3	0.5	0.1	0.4	0.0824

As illustrated in Figure 3, Holt-Winters' exponential smoothing method with appropriate parameters has good accuracy of forecast. Here parameters' values are determined by comparing the values of MAPE. The lower the value of MAPE, the better accuracy the forecast has. Based on current collected data, we set parameters' values as $\alpha = 0.1$, $\beta = 0.2$ and $\gamma = 0.3$.

In our experiment, we make predictions for three types of demands which correspond to three types of VMs. These VMs are treated as items to go through the allocation process. The accuracy of forecast for different demands influences the performance of our algorithm (i.e. sum of power consumption) by changing allocation of VMs to hosts.

5.4 Performance of Algorithm

In this part, we evaluate the performance of our algorithm by comparing it with the others: one is without forecast, WF for short and the other is with constant number of hosts which are never shut down, CN for short. It is assumed that the arrival time of demand fits normal distribution of (0, 25) and the execution time of demand fits normal distribution of (30, 144).

The evaluation criteria of our experiment are the total power consumption during a certain period. Here we take 10 hosts as example for CN. As shown in Figure 4, our algorithm with forecast remains lower power consumption than WF for the complete cycle of 24 hours. The major features of WF are: (1) on-demand service; (2) shut

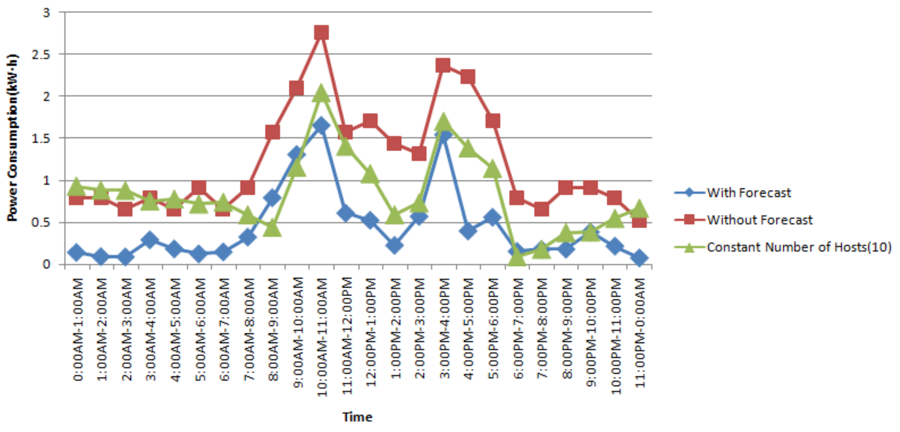

Fig. 4. Performance of different methods with interval of 1 hour

down hosts immediately. Due to no forecasted demand for reference, WF cannot keep hosts standby in advance and wait for future demands in a reasonable period of time. A lot of power is consumed by switching on/off hosts frequently. On the other hand, for every period in CN, a constant set of hosts are kept in active mode (full power state) or standby mode (low power state), which means they will never be shut down. Other hosts work in the same way as the ones in WF. According to our experimental results in Figure 4, WF consumes 29.625 kW·h in total, while our algorithm only consumes 10.734 kW·h, which saves energy by up to 60%.

Figure 5 shows the results with interval of 0.5, 1, 1.5 and 2 hours for comparison among our algorithm, WF and CN with 5, 10 and 15 hosts respectively. Based on actual demand from same users, our algorithm keeps its advantage over WF and CN in power consumption for different intervals. In addition, CN with different number of hosts has lower power consumption than WF for any interval, which indicates that it can save energy to keep reasonable set of hosts in standby mode for each period. This lays the experimental basis for our algorithm. On the other hand, the performance of all algorithms becomes worse with change of period from smaller ones to larger ones, for example, from 1 hour to 1.5 hours. This is because of the extension of period, which results in ignoring many changes during longer interval. Therefore, we design the self-optimizing module to find the reasonable forecast period which is important to the performance of our algorithm.

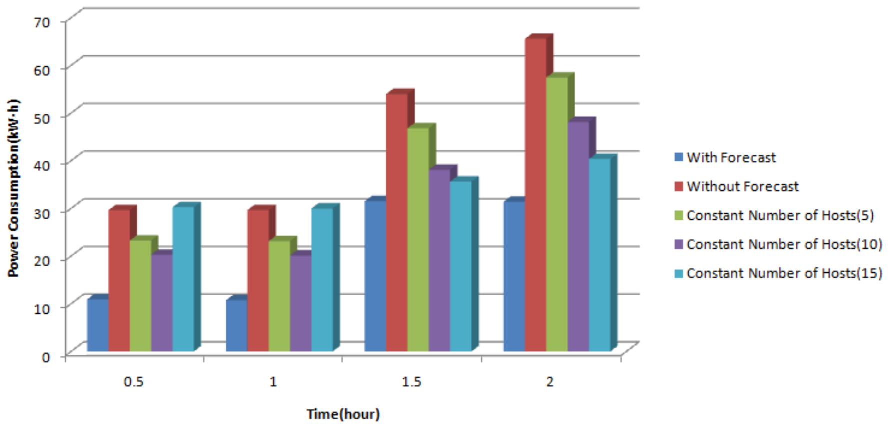

Fig. 5. Sum of power consumption with different intervals

We can conclude from Figure 5 that there is a sharp increase in the curve of power consumption for our algorithm. Self-optimizing module can help us find the most reasonable forecast period. The initial random period is 2 hours. The set of new periods generated by the neighborhood function is 2, 1 and 0.5. Based on our evaluation results of power consumption, the start point and end point of this sharp increase is 1 and 2 respectively. Then the set of periods to evaluate in the third step is 2, 1.5 and 1. Based on Figure 5 and Figure 6, we can see that the most reasonable forecast period with lower power consumption and less iteration is 1 hour.

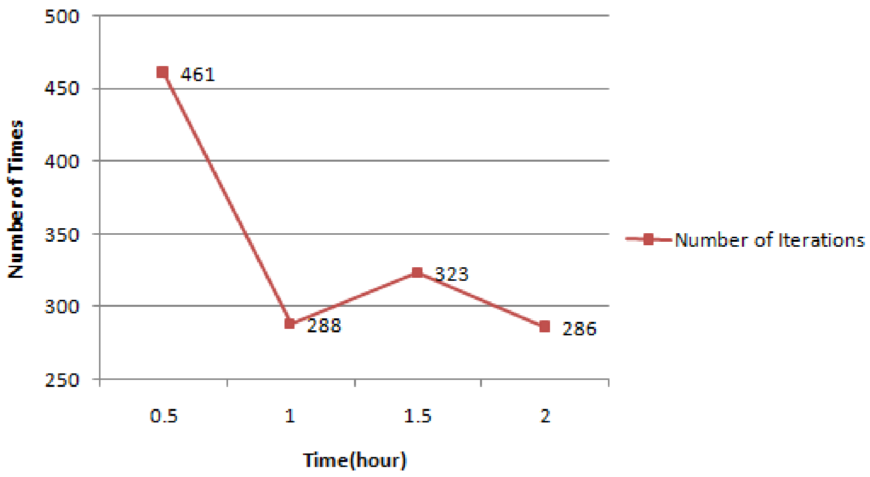

Fig. 6. Number of iterations with different intervals

6 Conclusions and Future Work

Along with the fast development of cloud computing, the number of servers grows rapidly. Apart from the construction cost of the equipment, how to control the operational cost becomes a major concern for every company. One possible way for operational cost control is to reduce power consumption of each host. This paper proposes an approach based on demand forecast to conserve energy. Our approach can predict users' demands of different seasonal patterns effectively and use modified knapsack algorithm to adjust physical locations of VMs. On the basis of forecasted resource demands and reasonable allocation of VMs, the frequency of switching on/off hosts can be reduced, which leads to a decrease in the total amount of energy and has practical value for cloud computing environments.

It is verified that our approach has good accuracy for demand forecast and an advantage in power consumption over other approaches. For the future work, we propose to take into account other kinds of resources in the allocation of VMs, such as disk storage and network bandwidth. The other research interest is to study the influence of VMs' live migration on power consumption, which may be another way to save energy.

Acknowledgements. This work is partially supported by China NSFC (Granted Number 61073021), Science and Technology Commission of Shanghai Municipality (Granted Number 10511501503, 10DZ1200200, 11511500102). The work described in this paper was also partially supported by Morgan Stanley. Morgan Stanley and Shanghai Jiao Tong University Innovation Center of Computing in Financial Services have entered into Collaboration Agreement No.CIP-A20110324-2.

References

1. U.S. Environmental Protection Agency:Report to Congress on Server and Data Center Energy Efficiency. Public Law, 109–431 (2007)
2. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software: Practice and Experience 41(1), 23–50 (2011)
3. Walsh, W.E., Tesauro, G., Kephart, J.O., Das, R.: Utility Functions in Autonomic Systems. In: 1st IEEE International Conference on Autonomic Computing, pp. 70–77. IEEE Press, New York (2004)
4. Tesauro, G., Das, R., Walsh, W.E., Kephart, J.O.: Utility-Function-Driven Resource Allocation in Autonomic Systems. In: 2nd IEEE International Conference on Autonomic Computing, pp. 342–343. IEEE Press, New York (2005)
5. Beloglazov, A., Buyya, R.: Energy Efficient Allocation of Virtual Machines in Cloud Data Centers. In: 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing, pp. 577–578. IEEE Press, New York (2010)
6. Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, pp. 119–128. IEEE Press, New York (2007)
7. Khanna, G., Beaty, K., Kar, G., Kochut, A.: Application performance management invirtualized server environments. In: Network Operations and Management Symposium, pp. 373–381. IEEE Press, New York (2006)
8. Steinder, M., Whalley, I., Carrera, D., Gaweda, I., Chess, D.: Server virtualization in autonomic management of heterogeneous workloads. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, pp. 139–148. IEEE Press, New York (2007)
9. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: 15th International Conference on World Wide Web, pp. 593–604. ACM (2006)
10. Time series Forecasting using Holt-Winters Exponential Smoothing, `http://www.it.iitb.ac.in/~praj/acads/.../04329008_ExponentialSmoothing.pdf`
11. Holt-Winters' Exponential Smoothing with Seasonality, `http://www.cec.uchile.cl/~fbadilla/Helios/referencias/08HoltWintersSeason.pdf`
12. Goodwin, P.: The Holt-Winters Approach to Exponential Smoothing: 50 Years Old and Going Strong. Foresight, 30–33 (2010)
13. Mean Absolute Percentage Error, `http://en.wikipedia.org/wiki/Mean_absolute_percentage_error`
14. Mean Absolute Scaled Error, `http://en.wikipedia.org/wiki/Mean_absolute_scaled_error`
15. Knapsack Problem, `http://en.wikipedia.org/wiki/Knapsack_problem`
16. Chu, P.C., Beasley, J.E.: A Genetic Algorithm for the Multidimensional Knapsack Problem. Journal of Heuristics, 63–86 (1998)
17. Amazon EC2 Pricing, `http://aws.amazon.com/ec2/pricing/`
18. IBM System x3850 X5 Specifications, `http://www-03.ibm.com/systems/x/hardware/enterprise/x3850x5/specs.html`
19. Amazon EC2 Instance Types, `http://aws.amazon.com/ec2/instance-types/`