International Journal of Applied Engineering Research ISSN 0973-4562 Volume 11, Number 24 (2016) pp. 11794-11802
© Research India Publications. http://www.ripublication.com

Performance Evaluation of Static VM Consolidation Algorithms for Cloud-Based Data Centers Considering Inter-VM Performance Interference

Young-Chul Shim

Young-Chul Shim is with Department of Computer Engineering, Hongik University, Seoul, Rep. of Korea.

Abstract

Energy efficiency in data centers is a very important issue and getting growing attention from researchers. One approach to reduce energy consumption is to allocate virtual machines (VMs) to physical machines (PMs) in such a way that the number of idle PMs is maximized. Approaches of this kind are called VM consolidation methods. Idle PMs can be put into an energy-saving sleep mode, in which PMs consume significantly lower energy than in the normal operation mode. But if too many VMs are packed into a single PM without considering their execution characteristics, the performance interference among VMs can cause significant slowdown to jobs. When a VM for a new job arrives at a cloud, this VM should be allocated to a PM and we consider this job allocation problem as a static VM consolidation algorithm. We introduce several static VM consolidation algorithms which not only increase number of idle servers and, therefore, reduce the total energy consumption on a cloud-based data center but also consider performance degradation on jobs due to inter-VM interference. We analyze introduced algorithms through simulation assuming more realistic data centers where each server consists of many cores and multiple types of resources including CPUs, memory, and IOs. We measure the idle server ratio and the total energy consumption in a cloud-based data center and the service level agreement violation ratio of executed jobs and compare these values with those of other existing algorithms.

Keywords: Cloud-based data centers, Energy-efficiency, Inter-VM performance interference, Multiple types of resources, SLA violation, Static VM consolidation algorithm

INTRODUCTION

SINCE the emergence of the concept of cloud computing, it is getting more widely adopted and deployed in the IT industry sector and receiving more attention from computer scientists and engineers. NIST defines cloud computing to be a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. From a hardware provisioning and pricing point of view, a cloud computing environment has advantages as follows [2]:

1) The appearance of infinite computing resources available on demand, quickly enough to follow load surges, thereby eliminating the need for cloud computing users to plan far ahead for provisioning.

2) The elimination of an up-front commitment by cloud users, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs.

3) The ability to pay for use of computing resources on a short-term basis as needed (for example, processors by the hour and storage by the day) and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

Examples of well-known cloud computing systems include Amazon EC2, Microsoft Azure, Google AppEngine, and Rackspace Cloudservers [3,4].

Virtualization is an essential mechanism in providing the computing-as-a-service vision of cloud-based data centers. By providing physical resource sharing, fault isolation, security isolation, and live migration, virtualization allows diverse applications to run in isolated environments through creating multiple virtual machines (VMs) on shared hardware platforms [5]. When a user rents VMs to run an application on them, he specifies the amount of resources allocated to each VM by stating required CPU cycles, memory, and I/O bandwidth. VM monitor (VMM) or hypervisor manages and multiplexes access to the physical resources, maintaining isolation between VMs at all times. As the physical resources are virtualized, several VMs, each of which is self-contained with its own operating system, can be executed on a physical machine (PM) [6].

Scheduling in cloud-based data centers is a problem of mapping VMs to PMs. Of course, the mapping should be done in such a way that the resources needed by all VMs running on a PM should not exceed the capacity of that PM so that the PM may not be overloaded and the performance of its VMs may not be degraded.

Another important issue in the scheduling problem is the minimization of the energy consumed in the cloud-based data center. The mapping should be performed in such as way that the total amount of energy consumed by all the PMs which run all the VMs should be minimized. It is reported that worldwide, the data centers use about 30 billion kilowatts of electricity, roughly equivalent to the output of 30 nuclear power plants [7] and this number is expected to grow 12% a year. This excessive use of energy in data centers not only raises the operation cost of data centers heavily but also creates environmental issues such as air pollution. Therefore, minimizing energy consumption in data centers is becoming a more and more important issue.

In many literatures, the power consumption of a server is modeled in its simplest form as follows:

$$P(u) = P_{min} + (P_{max} - P_{min}) \times u$$

where u is the CPU utilization, P_{min} is the power consumed when the server is idle, P_{max} is the power consumed when the server is fully utilized. In this paper we use two terminologies, PMs and servers, interchangeably. For most of all the servers available in the current market, it is well-known that P_{min} is almost 50% or more of P_{max} [8]. We can think of turning off a server when it becomes idle. But the procedure of turning off and then on a server takes too much time and, therefore, cannot be thought of as a practical solution. But some researchers developed a power-saving mechanism in which an idle server can be rapidly put into a sleep mode and consume just 10% of P_{min}. When a job arrives at the server, it can be put back into the normal operation mode rapidly [9]. With this kind of power-saving mechanism, a server consumes power with the above power model when it has one or more jobs to run but can consume very little power when it has no job.

With the above observation, if we run all the VMs on the smallest number of PMs and, therefore, maximize the number of idle PMs, we can greatly reduce the amount of energy consumed in data centers. This feature of scheduling, packing VMs into a small number of PMs, is called VM consolidation. It is known that VM consolidation is classified into (i) static consolidation, (ii) semi-static consolidation, and (iii) dynamic consolidation [10]. In static consolidation the sizing and placement of VMs on PMs is determined statically when a job arrives and does not change over a period of time. Semi-static consolidation attempts to take advantage of medium to long term workload variations by periodically re-sizing workloads and relocating them on target PMs. Semi-static consolidation typically takes advantage of intra-week variations or intra-month variations. Semi-static consolidation is performed by re-sizing and relocating workloads once a week or once a month. Dynamic VM consolidation extends the idea of semi-static consolidation event further by consolidating workloads daily or multiple times on the same day. Consolidations on such a frequent basis cannot be performed using VM relocation due to downtime required for VM relocation and, therefore, needs live VM re-sizing and live VM migration. While semi-static consolidation is suitable for very long jobs running for several weeks or months and dynamic consolidation is suitable for from long to very long jobs running for several days, static consolidation will be suitable for short jobs running for a couple of hours.

In this paper, we consider an energy-efficient job scheduling problem in data centers as a static consolidation problem. The static consolidation problem has been studied extensively by many researchers as will be described in the next section on related works. But most of their works have the following problems.

1) They assume that the performance of a VM is not affected by other VMs running on the same PM. But recent research results, which will be summarized in the next section, show that current virtualization technology does not provide perfect performance isolation. This imperfect performance isolation will have the effect of making job completion time longer than calculated assuming perfect performance isolation and in some cases can cause service level agreement (SLA) violations for some jobs.

2) Most researchers consider only CPU utilization. But there can be various kinds of jobs. They can be CPU-bound, memory-bound, or IO-bound. For CPU-bound jobs, CPU utilization is an important factor to consider but for IO-bound jobs, IO bandwidth utilization is an important factor. To be able to handle mix of various kinds of jobs, utilization of resources including CPU, memory, and IO should be considered simultaneously.

3) Most researchers assume that a server has one core and a server can run as many a number of VMs as possible unless the total resource requirements of all VMs on that server exceed the capacity of the server. But in reality a server has many cores and the number of VMs that can be allocated on a core is limited, for example 2 in Amazon EC2. If a server has two cores, each of which has 1 GIPs computation power, and has 2 GIPs computation power in total, it seems that we can assign 3 jobs, each of which requires 0.65 GIPs. But in reality only two jobs can be assigned to this server, if we consider the fact that this server consists of 2 cores.

4) Many energy-efficient static consolidation algorithms assume that we have the knowledge of both the completion time of a job and its resource requirements. But the job completion time varies significantly depending upon the amount of inputs and is hard to predict while approximate resource requirement of job can be predicted relatively easily.

In this paper we present several static VM consolidation algorithms considering (i) imperfect performance isolation of virtualization technology, (ii) mix of various kinds of jobs, and (iii) servers consisting of multiple cores and the limitation on the number of VMs for a core. Although we assume that we have approximate knowledge of resource requirements of jobs, we do not require that we know the completion time of jobs a priori as many researchers do. The static VM consolidation problem is modeled as a bin packing problem as will be explained in the next section. We will present random and first-fit algorithms as base cases for performance comparison purpose assuming the more realistic characteristics of servers as explained in the above. In the design of a best-fit algorithm, it is not quite obvious how to define which servers is the best when considering multiple resource requirements simultaneously. We will present several approaches to decide the best server while considering multiple resource requirements. Among this presented approaches, we will select the best one through simulation. But it will be shown that just simply using a best-fit algorithm can lead to the high ratio of SLA violation of jobs. We will augment the best-fit algorithm to drastically reduce the SLA violation ratio. In the design of data center simulator, we assume that each server has multiple cores, the number of VMs that can be allocated to a core is limited, and the performance isolation among VMs running on a PM is not perfect. Through simulation we measure values of three metrics for each algorithm, the idle PM ratio, the SLA violation ratio, and the total amount of energy consumed in the data center, while varying the workload level on the data center.

The rest of the paper is organized as follows. Section 2 summarizes related works. Section 3 describes how much performance degradation a VM may suffer due to imperfect performance isolation when it is run with other VMs on the same PM. Section 4 presents the proposed static VM consolidation algorithms. Section 5 presents the simulation results and is followed by the conclusion in Section 6.

RELATED WORKS

The VM consolidation problem is often formulated as a bin packing problem which can be described as follows. Given *n* items and *m* bins with

$$w_j = weight\ of\ item\ j,$$

$$c = capacity\ of\ each\ bin,$$

assign each item to one bin so that the total weight of the items in each bin does not exceed *c* and the number of bins used is minimum [11]. This problem is known to be NP-hard and several approximate algorithms are introduced: First-Fit (FF), Best-Fit (BF), First-Fit Decreasing (FFD), and Best-Fit Decreasing (BFD) algorithms. We assume that items and bins are indexed. The FF algorithm considers the items according to increasing indices and assigns each item to the lowest indexed bin which it fits. The BF algorithm is obtained from FF by assigning the current item to the feasible bin having the smallest residual capacity. If items are sorted in the non-increasing order of their weight and then FF and BF algorithms are applied, the resulting algorithms are called FFD and BFD algorithms, respectively. Making an item, w_j, a bin, and *c* correspond to a VM, the resource requirements of VM_j, a PM, and the resource capacity of a PM, respectively, the static VM consolidation problem exactly becomes a bin packing problem. When jobs, each of which requires one VM, arrive one at a time, either FF or BF algorithms can be used. When many jobs, each of which requires one or more VMs, are allocated to PMs simultaneously, FF, BF, FFD, or BFD algorithms can be applied. In this paper we address the situation where jobs, each of which is run on a VM, arrive one at a time and, therefore, we do not need to consider using FFD or BFD. So we use terms, VM and a job, interchangeably in this paper.

Eucalyptus, which is open source software for building AWS-compatible clouds, provides the following VM allocation algorithms: a greedy algorithm, a round-robin algorithm, and a power save algorithm [12]. The greedy algorithm is an FF algorithm explained above. The round robin algorithm mainly focuses distributing the load equally to all the PMs and is outside out interests. The power save algorithm optimizes the power consumption by turning off PMs which are not currently used, which is considered impractical.

Lin et al proposed a hybrid approach which combines FF and dynamic round robin algorithms [13]. The dynamic round-robin algorithm uses two rules to help consolidate VMs. The first rule is that if a VM has finished and there are still other VMs hosted on the same PM, this PM will accept no more new VMs. Such PMs are referred to as being in the retiring state. The second rule is that if a PM is in the retiring sate for a sufficiently long period of time, instead of waiting for the residing VMs to finish, the PM will be forced to migrate the rest of the VMs to other PMs and be shutdown after the migration is finished. The hybrid approach uses the FF algorithm during rush hours and dynamic round-robin during non-rush hours. This approach is a combination of static and dynamic VM consolidation methods

[14] describes two energy saving static VM consolidation algorithms: ECTC and MaxUtil. When a new job, which is to be run on a new VM arrives, a cost function is computed for each PM and the PM which has the lowest cost is selected to run the VM. [15] introduces a modified BFD algorithm which allocates each VM to a PM that provides the least increase of power consumption due to this allocation. The algorithm is used to allocate multiple VMs for multiple jobs simultaneously. The algorithms in [14] and [15] assume that the exact completion time of a job is known a priori and is not affected by other VMs collocated on the same PM. Moreover, they consider only CPU requirements for jobs.

While all of the above works focus on the utilization of only one resource type, mostly CPU, there are VM allocation and migration algorithms which deal with multiple resource types simultaneously. The Dynamic Integrated Resource Scheduling algorithm in [16], the VectorDot algorithm in [17], and ESWCT and ELMWCT algorithms in [18] take multiple resource types, including CPU, memory, and IO, into consideration simultaneously. The goal of these algorithms is to balance resource utilization among PMs and they are not concerned about minimization of energy consumption in a cloud-based data centers. But their idea of combining requirements for multiple resource types can be somewhat relevant to our research.

None of the works that we described above consider the performance interference among VMs running on the same PM. Many researchers have studied this performance interference issue [19-21]. They take various types of applications, CPU-intensive, memory-intensive, IO-intensive, and mixed, run them on a PM together, and study how much performance degradation applications of one type cause to applications of another types. Pu et al in [5] states that network I/O applications are becoming dominating workloads in most cloud-based data centers and studies the performance interference among multiple VMs running on the same hardware platform with the focus on network I/O processing. Kang et al in [22] especially focus on the performance impact of contention in a last-level shared cache (LLC). Through measurement they find that the degree of impact on performance degradation heavily depends on the LLC reference ratio of applications and suggests a throughput-maximizing VM consolidation policy based on this observation. They suggest that if two jobs have heavy LLC reference ratio, it is better to allocate them on different PMs. We use results of these works in designing and analyzing static VM allocation algorithms.

PERFORMANCE INTERFERENCE AMONG JOBS

In this section we describe how much performance degradation one job is causing to another job running in the same PM. Each job is executed in a separate VM. We consider three types of jobs as follows.

1) CPU-bound jobs: Consist of arithmetic operations mostly and require little IO operations. Because they do not need large size data structures, their requirements for cache and memory are modest and much lower than those for memory-bound jobs. We assume that their CPU requirements are on average 90% of a core's CPU capacity.

2) Memory-bound jobs: Consist of arithmetic operations mostly and require little IO operations like CPU-bound jobs. Basically they read data from a large size array, perform arithmetic operations on them, and write the results to another large size array. We set the size of arrays to be a couple of times larger than the LLC size and, therefore, their requirement for cache is very heavy. Their LLC reference ratio is very high and cache fault ratio at LLC is also very high. If the main memory size of a PM is *M* giga-bytes and there are *N* cores in a PM, we assume that their memory requirements are on average 90% of *M/N* giga-bytes. Their CPU requirements are on average 90% of a core's CPU capacity while their IO requirement is very low like CPU-bound jobs.

3) IO-bound jobs: Consists of disk IO operations mostly, make little use of arithmetic or logical operation, and incur little burden on CPU. Basically they read data from a large size file, perform very simple operations on them, and write the results back to another file, repeatedly. If the IO bandwidth of a PM is *B* GBps and there are *N* cores in a PM, we let the bandwidth requirement of an IO-bound jobs be on average 90% of *B/N* GBps. Their CPU and memory requirements are very low.

To explain the performance interference between jobs running on the same PM, we use the metric called the *slowdown*. First the completion time of *job1* is measured without any other jobs in the PM. Then we measure the completion time of *job1* while *job2* is running in the same PM. *Job2* can be run on the same core or a different core. *Slowdown(job1@job2)* is defined to be the ratio of the second value to the first value and is greater than 1. The larger this value gets, the larger performance degradation job1 suffers from job2.

We analyzed the data from [19-22] and performed experiments ourselves to measure the slowdown values for various combinations of jobs. The results are shown in Table 1 and Table 2. The element at the row-i and the column-j represents the value *of slowdown((row-I type job)@(column-j type job))*. Table 1 gives slowdown values when two jobs are run on the same core while Table 2 presents slowdown values when two jobs are run on different cores on one PM. From Table 1 we see that slowdown values between CPU-bound jobs and memory-bound jobs are greater than 2. This means that the job completion time increases more than two times longer. This is because CPU and memory-bound jobs compete heavily for the CPU resource and, moreover, memory-bound jobs contend excessively for cache against other jobs. We see around 10% slowdown increase between memory and IO-bound jobs. This is because (1) memory-bound jobs' heavy demand on cache hurts the performance of IO-bound jobs and (2) IO-bound jobs' frequent IO requests incur frequent service requests on the Dom0, which deteriorates the performance of other jobs. From Table 2 we see that the slowdown values become much lower when two jobs are run on different cores. One case that requires special mention is when a memory-bound job is run against another memory-bound jobs. In this case two memory-bound jobs contend heavily for LLC and the slowdown value becomes 1.25, which is much higher than other cases in Table 2.

Table 1. Interference between Jobs Executed on the Same Core

Against	@ CPU	@ Memory	@ IO
CPU	> 2	> 2	1.05
Memory	> 2	> 2	1.11
IO	1.02	1.09	1.11

Table 2. Interference between Jobs Executed on Different Core

Against	@ CPU	@ Memory	@ IO
CPU	1.02	1.04	1.02
Memory	1.05	1.25	1.05
IO	1.01	1.03	1.05

INTERFERENCE AWARE STATIC VM CONSOLIDATION ALGORITHMS

In this section we explain the proposed inter-VM performance interference aware static VM consolidation algorithm. Before we describe proposed algorithms, we first explain two existing algorithms: random (RD) and First Fit (FF) algorithms. Then we describe the Best Fit (BF) algorithm with several different metrics for selecting the best PM for a submitted VM. Then we explain the fourth algorithm called Interference Aware Best Fit (IABF) which considers the inter-VM interference to reduce the SLA violation ratio. In all the algorithms to be explained, we assume that at most one CPU or memory-bound jobs can be allocated at a core. According to Table 1, if two CPU or memory-bound jobs are allocated to one core, their contention for CPU is excessive and their job completion times become longer more than two times, which may cause the requested SLA to be violated.

Before we explain 4 classes of consolidation algorithms, we describe the function which checks whether a selected PM can accommodate a new job in Figure 1. The data structure *Job* includes three fields which describes CPU, memory, and IO requirement of a job, the data structure *PM_i* has two fields which present available memory and IO capacity, and for each core in *PM_i*, there is a field which presents the available CPU capacity of that core. In all the functions presented below, we assume that the newly assigned *CoreNo* parameter value is returned to the calling function.

```
IsPMAvailableForJob (Job, $PM_i$, CoreNo) {
  if (Job's memory requirement is larger than available
        memory in $PM_i$ or Job's IO requirement is larger
        than available IO in $PM_i$))
    return (False);
  else
    if (idle core exists) {
      set CoreNO to the number of that idle core;
      return (True);
    }
    else if (we can find a core which can satisfy the
          CPU requirement of the job by rearranging
          existing jobs among cores in $PM_i$) {
      rearrange existing jobs among cores;
      set CoreNo to the number of that selected core;
      return (True);
      }
    else
      return (False);
}
```

Figure 1. PM checking algorithm

In the RD algorithm, a PM is selected randomly and checked for availability with the algorithm in Figure 1. If available, the available core number is also returned. Otherwise, another PM is selected randomly and this process is repeated until an available PM is found. If no available PM is found, even after all the PMs are examined, this unavailability information is returned.

```
FF (Job) {
  for (i = 0; i < NoOfPM; i++)
    if(IsPMAvailableForJob(Job, $PM_i$, CoreNo))
      return (True, $PM_i$, CoreNo);
  return(False, Don't care, Don't care);
}
```

Figure 2. FF algorithm

Figure 2 shows the FF algorithm. FF checks PMs in the order of PM indices until an available PM is found. Basically it finds the first PM which can accommodate the new job. If an available PM is not found, this unavailability information is notified.

```
BF(Job) {
  sort non-idle PMs in either increasing or decreasing order
      of used metrics;
  append idle PMs at the end of the sorted non-idle PMs;
  while (the list of sorted PMs is not empty) {
    retrieve $PM_i$ from the head of the PM list;
    if (IsPMAvailableForJob(Job, $PM_i$, CoreNo))
      return (True);
  }
  return (False);
}
```

Figure 3. BF algorithm

Fig 3. shows the BF algorithm, which selects among non-idle PMs the best fitting PM for a new job. If such a non-idle PM is not found, one of idle PMs is assigned. In the BF algorithm we use 4 different metrics as follows. We use notations *CPU_U_i*, *MEM_U_i*, *IO_U_i* as the utilization of CPU, memory, and IO respectively of *PM_i*.

The first metric called Integrated Load Imbalance (ILI) is defined in (1) [16]. In the equation *CPU_U* is defined to be the arithmetic average of CPU utilization of all PMs and *MEM_U* and *IO_U* are defined likewise.

$$ILI = \frac{\sqrt{(CPU\_U_i - AVG)^2 + (MEM\_U_i - AVG)^2 + (IO\_U_i - AVG)^2}}{AVG} \quad (1)$$

where

$$AVG = \frac{\sum_i (CPU\_U_i + MEM\_U_i + IO\_U_i)/3}{No. of\ PMs}$$

If the ILI value of a certain PM is low, it means that that PM's load is well-balanced. With this metric, non-idle PMs are ordered in the non-decreasing order of the ILI value. The purpose of using this metric is to make loads be distributed among PMs as evenly as possible. Actually this is irrelevant to our goal of minimizing energy usage. But we include this metric just for comparison purpose. There are other metrics which are proposed to achieve load balancing among PMs and consider multiple resource types but we ignore them in our experimentation except ILI in (1).

The second metric called Integrated Inverse Utilization (IIU) is defined in (2) [23]. If this value is high, it means that the utilization is high. With this metric, non-idle PMs are ordered in the non-increasing order of the IIU value.

$$IIU = \frac{1}{(1 - CPU\_U_i)(1 - MEM\_U_i)(1 - IO\_U_i)} \quad (2)$$

The third metric called Vector Distance Utilization (VDU) is defined as in (3). If this value is low, it means that the utilization is high. With this metric, non-idle PMs are ordered in the non-decreasing order of the VDU value.

$$VDU = \sqrt{(1 - CPU\_U_i)^2 + (1 - MEM\_U_i)^2 + (1 - IO\_U_i)^2} \quad (3)$$

The fourth metric called Arithmetic Average Utilization (AAU) is defined in (4). If this value is high, it means that the utilization is high. With this metric, non-idle PMs are ordered in the non-increasing order of the AAU value.

$$AAU = \frac{CPU\_U_i + MEM\_U_i + IO\_U_i}{3} \quad (4)$$

When using metrics (2), (3), or (4), we try to assign a new VM to a more highly loaded PM.

From Table 2 we observe that memory-bound jobs slow down other jobs running on other cores non-trivially and especially other memory-bound jobs much more heavily. We made further experiments to measure the performance interference among memory-bound jobs. Experiments were made on a server with 4 cores. We increased the number of memory-bound jobs from 1 to 4 and measured the slowdown of jobs. Results are shown in Table 3.

Table 3. Performance Interference among Memory-Bound Jobs

No. of jobs	1	2	3	4
Slowdown	1	1.04	1.25	1.61

From this observation and previous results in Table 2, we restrict the number of memory-bound jobs that can be simultaneously run on a PM and we choose to limit to *N/2* memory-bound jobs on a PM, where *N* is the number of cores in a PM. This additional policy can be implemented by modifying the PM checking algorithm in Fig. 1 by inserting several statements which check how many memory-bound jobs are already running on a PM if a new memory-bound job is submitted. The PM selection algorithm with this modified PM checking algorithm is called an IABF algorithm.

EXPERIMENTAL RESULTS

In this section we first explain simulation environments for analyzing the performance of proposed algorithms. Then we present the experimental results.

For simulation we consider a data center consisting of 32 homogeneous PMs. Each PM has 4 cores and each core can run up to 2 VMs. Therefore at maximum 256 VMs can be executed simultaneously. As explained in Section 3, we consider three types of jobs: CPU-bound, memory-bound, and IO-bound. Each job is executed on one VM. Jobs of these three types are created with equal probability. The size of a job is defined to be the optimal execution time of the job under no inter-VM interference from other jobs. We assume that the inter-arrival time and the job size follow the exponential distribution.

From the simulation we measure three metrics: (1) the ratio of idle PMs, (2) the ratio of SLA violations, and (3) the total amount of energy consumed in the data center during the simulation. The ratio of idle PMs is obtained as follows. First the total idle time of each PM during the simulation is calculated and this per-PM idle time is summed up for all the PMs. Then this sum is divided by the number, *(simulation time) × (total number of PMs)*. The ratio of idle PMs is a rough indicator of the energy efficiency of the simulated algorithm and takes the value from 0 to 1. The larger this ratio gets, the more idle PMs there are in the data center during the simulation and, therefore less energy consumption.

The actual completion time of a job during the simulation is defined to be the time between the arrival time of the job and the finish time of the job. If a job is allocated to a core as soon as it arrives, the actual completion time of the job consists of only its actual execution time. But if a job has to wait in the job queue for a core until it is allocated, the actual completion time consists of waiting time and actual execution time. Moreover the execution time can become longer than its ideal execution time due to the performance interference from other jobs running in the same PM as described in Tables 1, 2, and 3. The slowdown of a job is defined to be the ratio of the actual completion time to the ideal execution time. The slowdown has a value greater than or equal to 1. If the slowdown value gets higher than some threshold, we say that the SLA of the job is violated. In this paper, we choose 1.25 for this threshold. To calculate the ratio of SLA violation, we count the number of completed jobs whose SLA has been violated and then divide this value by the total number of completed jobs during the simulation. The SLA violation ratio gets value from 0 to 1. The smaller this ratio gets, the more reliable the consolidation algorithm becomes.

To calculate the energy consumed in a PM, we use the equation for the energy consumption model stated in Section 1. We let $P_{min} = 0.5 \times P_{max}$ and the energy consumption model becomes as follows.

$$P(u) = 0.5 \times (u+1) \times P_{max}$$

When at least one job is running at a PM, the energy consumption in the PM is defined as above. When a PM is idle, we assume that it is put into a sleep mode quickly and consumes as low as 10% of P_{min} or 5% of P_{max}. When a job arrives, the PM can quickly return to the normal operation mode as suggested in [9]. With the above model, the energy consumption is calculated for each PM and it is summed up for all the PMs in the data center to calculate the total energy consumption of the data center. The unit of the calculated energy is P_{max}.

For the simulation we vary the load level of a data center. The load level value reflects how many and how large jobs are submitted to the data center. The load level is calculated as follows assuming that each core has the computation power of 1 GIPs.

$$((\text{average job size}) \times (\text{average number of jobs per sec})) / ((\text{number of PMs}) \times (\text{number of cores per PM}) \times 2)$$

In this paper we consider four load levels: 0.25, 0.375, 0.5, 0.625. The load level 0.25 means that the data center is lightly loaded while 0.625 means that the data center is heavily loaded. When we increased the load level to 0.75, the SLA violation ratio became too high, which should be avoided in the real situation, and we set the maximum load level to be 0.625. We adjusted the load level by changing the average job length as in Table 4 while fixing the average job number per second to be 16.

Table 4. Average Job Length Values for Load Levels

Load level	0.25	0.375	0.5	0.625
Average job length	1	1.5	2	2.5

Each run of the simulation is terminated when 10,000 jobs are completed. Tables 5, 6 and 7 show the simulation results. In tables, columns represent load levels and rows represent the consolidation algorithms as follows: RD (Random), FF (First-Fit), ILI (Best-Fit with Integrated Load Imbalance metric), IIU (Best-Fit with Integrated Inverse Utilization metric), VDU (Best-Fit with Vector Distance Utilization metric), AAU (Best-Fit with Arithmetic Average Utilization metric), IABF (Interference Aware Best Fit) algorithms.

Table 5 shows the idle PM ratio (in percentage) of various VM consolidation algorithms under varying load levels. Among RD, FF, and BF algorithms (except the BF algorithm with ILI metric), BF algorithms has the highest idle PM ratio, the FF algorithm has slightly lower number, and the RD algorithm has the worst result. This result is quite obvious because BF and FF algorithms try to allocate a new job to an already packed PM and try to maximize the number of idle PMs. On the other hand, the BF algorithm using ILI metric tries to distribute job loads as evenly as possible among non-idle PMs and, therefore, its idle PM ratio is even lower than the FF algorithm. From the simulation we see that the BF algorithm using AAU metric exhibits the highest idle PM ratio. So for the IABF algorithm we decided to use AAU metric. We see that the idle PM ratio of the IABF algorithm is better than the BF with ILI metric but slightly worse than the FF algorithm. This is because the IABF algorithm tries to avoid the situation where too many memory-bound jobs are allocated to the same PM and, therefore, the performance of not only themselves but also other jobs running on the same PM are seriously degraded.

Table 5. The Ratio of Idle PMs (in %)

	0.25	0.375	0.5	0.675
RD	43.8	26.1	13.3	5.4
FF	62.44	46.3	29.8	15.4
BF(ILI)	61.4	35.5	25.8	11.2
BF(IIU)	63.2	47.0	31.0	16.6
BF(VDU)	63.0	46.4	30.2	15.7
BF(AAU)	63.5	47.5	31.1	17.1
IABF	61.5	44.3	38.5	14.6

Table 6 presents the SLA violation ratio (in percentage) of 7 consolidation algorithms under varying load levels. We see the SLA violation ratio increases as the load level increases in all the algorithms. The table shows that the IABF algorithm result in zero or almost zero SLA violation because it tries to reduce performance interference among VMs at a PM. When the load level becomes 0.625, the SLA violation is not zero even for the IABF algorithm because as the load level rises, new jobs find it more difficult to find PMs which can accept them and, therefore, their waiting time in the queue increases. We see that the BF algorithms using IIU, VDU, and AAU metrics have the highest violation ratio, the FF algorithm has the lower violation ratio, and the RD algorithm has even lower violation ratio. We notice that the violation ratio of the BF algorithm using ILI metric rises relatively slowly compared with that of the RD algorithm and is lower than that of the FF algorithm although its idle PM ratio is lower than the FF algorithm. This is because the BF algorithm using ILI tends to distribute loads evenly among non-idle PMs, including the memory requirements of jobs.

Table 6. The Ratio of SLA Violation (in %)

	0.25	0.375	0.5	0.675
RD	0.73	1.67	3.09	5.44
FF	6.29	6.80	7.13	7.90
BF(ILI)	2.35	2.30	2.56	3.26
BF(IIU)	8.47	9.62	10.52	10.75
BF(VDU)	8.31	9.46	10.29	10.64
BF(AAU)	8.94	9.64	10.78	11.04
IABF	0.00	0.00	0.00	0.08

Table 7 shows the total energy consumption of all the algorithms under varying load levels. The unit of the consumed energy is P_{max} as explained before. From Tables 5 and 7, we see that as the idle PM ratio rises, the consumed energy falls. The IABF algorithm consumes slightly more energy than the BF algorithms using IIU, VDU, and AAU metrics and the FF algorithm but less energy than the RD algorithm and the BF algorithm using ILI metrics.

Table 7. Total Energy Consumption

	0.25	0.375	0.5	0.675
RD	5753	7663	9283	10660
FF	4520	6515	8444	10163
BF(ILI)	4683	6804	8577	10356
BF(IIU)	4457	6468	8390	10118
BF(VDU)	4469	6496	8433	10122
BF(AAU)	4450	6410	8368	10046
IABF	4596	6609	8447	1017

We want to minimize the energy consumption in a data center but at the same time we do not want the job completion time to become excessively longer to save energy. From simulation results in Tables 5, 6, and 7, we conclude that the IABF algorithm succeeds in achieving in almost zero SLA violation ratio under any load level conditions at the cost of consuming slightly higher energy consumption than any other more energy efficient algorithms

CONCLUSION

In this paper we approached the scheduling problem in a data center as solving a static VM consolidation problem, which was then modeled as a bin packing problem. The consolidation algorithm should try to minimize not only the total energy consumption in the data center but also the SLA violation ratio of submitted jobs. Moreover, we must consider more realistic situations when we deal with the consolidation algorithm as follow: (i) the performance of a VM is affected by other VMs running in the same PM, (ii) not only CPU resources but also other resources such as memory and IO should be simultaneously considered, (iii) a single PM consists of many cores in general, and (iv) the job completion time is difficult to expect because it heavily depends upon the amount of input.

We modified existing random, first-fit, and best-fit algorithms for the static VM consolidation algorithm considering the above 4 realistic situations. We studied the performance interference among VMs running in the same PM through analyzing existing research results and making further experiments. We considered four metrics which integrate CPU, memory, and IO resource requirements/utilization, especially for the best-fit algorithms. We assumed a PM consists of several cores. Finally we assumed that we did not know the job completion time but could know the resource requirement characteristics.

We analyzed the random, first-fit, and best-fit algorithms through simulation. Their performance was compared using three metrics: the ratio of idle PMs, the ratio of SLA violations, and the total amount of consumed energy in a data center. We found that the best-fit algorithm using the arithmetic average utilization achieved the highest idle PM ratio and the lowest energy consumption but its SLA violation ratio was the highest. We modified this algorithm so that memory-bound jobs are more evenly distributed among PMs and this modified algorithm was called as IABF (Interference Aware Best Fit) algorithm. The IABF algorithm was shown to have slightly lower idle PM ratio and slightly higher energy consumption than the first fit algorithms and other best fit algorithms but achieves almost zero SLA violation ratio from very low to high load level conditions.

In the future we plan to relax some assumptions made in this paper and continue our research to consider (i) long-lived jobs whose execution may require multiple cooperating VMs, (ii) a heterogeneous data center which consists of PMs of different hardware capacities, and (iii) mixed jobs.

ACKNOWLEDGMENT
This work was supported by 2015 Hongik University Research Fund

REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *NIST Special Publication* 800-15.
[2] M. Armbrust et al., "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010.
[3] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloud-Cmp: Comparing Public Cloud Providers," In *Proc. ACM Internet Measurement Conference*, 2010.

[4] G. Lu and W. Zeng, "Cloud Computing Survey," *Applied Mechanics and Materials*, vol. 530-531, pp. 650-661, 2014.

[5] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Who is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds" *IEEE Transactions on Services Computing*, vol. 6, no. 3, pp. 314-329, 2013.

[6] M. Mishra, A. Das, P. Kukarni, and A. Sahoo, "Dynamic Resource Management Using Virtual Machine Migrations," *IEEE Communication Magazine*, pp. 34-40, Sep. 2012.

[7] J. Glanz, "Power, Pollution and the Internet," *The New York Times*, Sep. 22, 2012, Available: http://www.nytimes.com.

[8] L. A. Barroso and U. Hoezle, "The Case for Energy-proportional Comuting," *IEEE Computer*, pp. 33-37, Dec. 2007.

[9] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating Server Idle Power," In *Proc. ACM ASPLOS*, 2009, pp. 205-216.

[10] A. Verma, J. Bagrodia, and V. Jaiswal, "Virtual Machine Consolidation in the Wild," In *Proc. ACM Middleware*, 2014, pp. 313-324.

[11] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley and Sons Ltd., 1990

[12] S. Subramanian et al., "An Adaptive Algorithm for Dynamic Priority based Virtual Machine Scheduling in Cloud," *Int. Journal of Computer Science Issues*, vol. 9, no. 2, pp. 397-402, 2012.

[13] C. –C. Lin, P. Liu, and J. –J. Wu, "Energy-Efficient Virtual Machine Provision Algorithms for Cloud Systems," In *Proc. IEEE Int. Conf. on Utility and Cloud Computing*, 2011, pp. 81-88.

[14] Y. C. Lee and A. Y. Zomaya, "Energy Efficient Utilization of Resources in Cloud Computing Systems," *Springer Journal of Supercomputing*, vol. 60, pp. 268-280, 2012.

[15] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing," *Future Generation Computer Systems*, vol. 28, pp. 755-768, 2012.

[16] W. Tian et al., "Dynamic and Integrated Load-Balancing Scheduling Algorithm for Cloud Data Centers," *China Communications*, vol. 8, no. 6, pp. 117-126, 2011.

[17] A. Singh, M. Korupolu, and D. Mohaptra, "Server-Storage Virtualization: Integration and Load Balancing in Data Centers," In *Proc. ACM/IEEE Conf. on Supercomputing*, 2008.

[18] H. Li, J. Wang, J. Peng, J. Wang, and T. Liu, "Energy-Aware Scheduling Scheme Using Workload-Aware Consolidation Technique in Cloud Data Center," *China Communications*, pp. 114-124, 2013.

[19] Y. Koh. R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An Analysis of Performance Interference Effects in Virtual Environments," In *Proc. IEEE Int. Symp. on Performance Analysis of Systems & Software*, 2007, pp. 200-209.

[20] F. Oh, H. S. Kim, H. Eom, and H.Y. Yeom, "Enabling Consolidation and Scaling Down to Provide Power Management for Cloud Computing," In *Proc. 3rd USENIX Conf. on Hot Topics in Cloud Computing*, 2011.

[21] S. Govindan, J. Liu, A. Kansai, and A. Sivasubramaniam, "Cuanta: Quantifying Effects of Shared On-Chip Resource Interference for Consolidated Virtual Machines," In *Proc. SOCC*, 2011.

[22] S. Kang, S. –G. Kim, H. Eom, and H. Y. Yeom, "Toward Workload-Aware Virtual Machine Consolidation on Cloud Platforms," In *Proc. ICUIMC*, 2012

[23] W. Tian, Y. Zhao, Y. Zhong, M. Xu, and C. Jing, "A Dynamic and Integrated Load-Balancing Scheduling Algorithm for Cloud Datacenters," In *Proc. IEEE CCIS*, 2011.

ABOUT AUTHOR

Young-Chul Shim received the B.S. degree in Electronic Engineering from the Seoul National University, Korea, the M.S. degree in Electrical Engineering from KAIST, Korea, and the Ph.D. degree in Computer Science from the University of California, Berkeley, U.S.A. Currently he is a Professor of Computer Engineering at Hongik University, Seoul, Korea. His teaching and research areas include network protocols, network security, and cloud computing.