Dependable Virtual Machine Allocation

Hiroki Yanagisawa, Takayuki Osogami, Rudy Raymond
IBM Research - Tokyo
IBM Japan, Ltd.
Tokyo, Japan
{yanagis,osogami,raymond}@jp.ibm.com

Abstract—The difficulty in allocating virtual machines (VMs) on servers stems from the requirement that sufficient resources (such as CPU capacity and network bandwidth) must be available for each VM in the event of a failure or maintenance work as well as for temporal fluctuations of resource demands, which often exhibit periodic patterns. We propose a mixed integer programming approach that considers the fluctuations of the resource demands for optimal and dependable allocation of VMs. At the heart of the approach are techniques for optimally partitioning the time-horizon into intervals of variable lengths and for reliably estimating the resource demands in each interval. We show that our new approach allocates VMs successfully in a cloud computing environment in a financial company, where the dependability requirement is strict and there are various types of VMs exist.

Index Terms—server consolidation; capacity planning; fault tolerance; mixed integer programming; dynamic programming;

I. INTRODUCTION

In cloud computing environments, virtualization technologies are used to run many virtual machines (VMs) on smaller numbers of physical servers. Thus we can significantly reduce the IT costs and energy usage by turning off the unneeded servers. It is important to determine the allocation of VMs to servers such that each VM is allocated with sufficient resources (such as CPU resources, network bandwidth, and memory).

There are two types of cloud computing services, public clouds and private clouds. Whereas a public cloud service provider offers its services to external users, a private cloud service provider offers its services to internal users. Since users who do not rely on external cloud service providers tend to construct their own private clouds, the users of the private cloud are relatively conservative. For example, they are often interested in the locations of their VMs and the frequency of VM migrations during use, because the VM migration may slightly degrade the performance of the VMs.

Our study involves a bank that uses private clouds and that is quite concerned about their special mission-critical requirements for dependability. The IT administrators of this bank try to minimize the number of times when server resources are exhausted due to high loads of the VMs on any server, because they use this number as an indicator as to whether or not current VM locations are adequate. They are also interested in the VM allocations not only in normal use, but also in the presence of server failures and maintenance work. Our work provides a VM allocation algorithm for such highly conservative users.

One of the key requirements of such highly conservative users is that VM migrations between physical servers are allowed only at the times of maintenance or server failures. This is because they want to minimize the possibilities of VM failure and performance degradation due to the VM migrations. As a result, we need to carefully and statically provision and allocate the physical resources for the VMs. Such capacity planning is typically done monthly or when the physical resources or VMs are changed.

The most important requirement of dependability to be considered in capacity planning is that sufficient resources are available at every moment, even upon server failures. Two standard fault-tolerant mechanisms to achieve such dependability are Active-Standby (AS) pairs or Active-Active (AA) clusters [1]. In our mixed context, each service is provided by a pair of VMs (an active and a standby) or by a cluster of (active) VMs. With an Active-Standby pair, the active VM usually provides the service. When the physical server where the active VM is allocated experiences a failure, the corresponding standby VM becomes active and provides the service. With the Active-Active cluster, a job is usually dispatched to any of the VMs in that cluster. Upon the failure of a physical server, the job is dispatched to one of the VMs that are allocated on other physical servers.

The users also require guarantees that sufficient resources are available to allow for maintenance. The users handle maintenance differently from failures, because maintenance can be scheduled in advance. In particular, when one of the physical servers is undergoing maintenance, all of the VMs allocated to that physical server are migrated to other physical servers until the maintenance is finished. A requirement of the users is to determine at the time of capacity planning the physical server to which each VM will be migrated during maintenance. This VM relocation mechanism for server maintenance offers more fault-tolerance than that for server failure. However, the relocation mechanism for maintenance cannot be used for server failures, because we cannot know in advance when server failures will occur.

Figure 1 shows examples of the VM relocation mechanisms. In these examples, one AA-cluster which consists of three VMs (AA1-1, AA1-2, and AA1-3) and two AS-clusters (a pair of VMs (AS2-A, AS2-S) and a pair of VMs (AS3-A, AS3-S)) are running on four physical servers before the server failure or maintenance. On the failure of Server 2, the workload on AA1-2 will be taken over by the other VMs in the same
AA-cluster (AA1-1 and AA1-3) and the workload on AS2-A will be handled by the corresponding standby VM (AS2-S). Notice that these corresponding VMs use more resources after the server failure than before. In contrast, when Server 2 is undergoing maintenance, the VMs on Server 2 will be relocated to other physical servers. In this example, AA1-2 migrates to Server 4 and AS2-A migrates to Server 3.

The objective of our capacity planning is to minimize the required server resources while ensuring fault-tolerance. If a private cloud service provider owns the servers, as in a typical scenario, then the required server resources mean the number of servers. Otherwise, the private cloud service provider rents the servers from vendors, and often pays fees depending on the number of assigned CPUs to the servers. (The rented servers typically have large CPU capacities and the vendors assign a requested number of CPUs for the servers.) For this case, the required server resources is the total number of CPUs allocated for the servers. (The rented servers typically have large CPU capacities and the vendors assign a requested number of CPUs for the servers.) For this case, the required server resources is the total number of CPUs allocated for the servers. Although many existing studies seek to minimize the number of servers [2], [3], some of them seek to minimize the number of allocated CPUs [4]. We consider the CPU-minimization objective in this study unless otherwise stated, but our approach can be easily adapted to the server-minimization objective.

In this study, we allocate VMs mainly focusing on CPU resources. An important property of CPU demand that needs to be considered in capacity planning is that the CPU demands of the VMs change dynamically over time, and they change differently depending on the services that they provide. Some VMs have peak CPU demands during banking hours. Other VMs use their CPUs at relatively constant rates throughout the day. A key to achieving high CPU utilization is to allocate each physical machine to those VMs that use their CPUs at different times, whenever possible. That property also holds for other resources such as network bandwidth and disk IOs, and so our approach can be easily adapted to such resources.

The practice at the bank before our work was to first estimate the peak CPU usage for each VM, allocate the VMs to physical servers, and calculate the number of CPUs needed for each physical server based on the estimated peak CPU usage. Because the estimates of the CPU usage were unreliable, they added extra CPUs for insurance on each physical machine.

**Our contributions.** The first contribution of our work is to formulate our optimization problem as a Mixed Integer Program (MIP) to determine the optimal allocations of VMs to physical servers taking into account the requirements for dependability. The dependability requirements include the fault-tolerance mechanisms for server failure and the migration plans during maintenance work. The consideration of the fault-tolerance mechanisms in the formulation of an MIP is a new concept. The MIP also takes into account the properties of the varying CPU demands depending on the particular services that are provided by the VMs. The variations of the CPU demand over time have been observed and incorporated in the MIP formulation of prior work [2], [3], [4], where the time horizon is partitioned into multiple slots. In [2], [3], [4], however, each slot has a constant and given length. The constant-length partitioning with a small number of slots does not fully take into account the variations of the CPU demand over time, whereas the time partitioning with many slots results in prohibitive computational time for solving the MIP.

Our second contribution is an algorithm for determining an optimal partition of the time horizon into a given number of slots of variable lengths. We formulate the problem of finding the optimal partition so that the total CPU resources required for the VMs is minimized. We then develop a dynamic programming algorithm to find the optimal partitioning with these slots of variable lengths.

Our third contribution is a framework for determining the
CPU demands reliably and without being conservative. By studying the logged data of the CPU use of the bank’s VMs for each minute during one month, we found statistical regularities in the CPU use patterns. In particular, we found that the percentiles of CPU use show periodic cycles each day. Using bootstrap confidence intervals, we show that the estimated values of the 95th percentile of the CPU use are relatively reliable. Although the accuracy of the estimated percentiles depends on the amount of available log data, we found that higher percentiles such as the 98th percentile cannot be estimated with sufficient accuracy from one month of log data. However, we see that the maximum CPU demands can still be estimated reliably. At the bank of our research, the maximum amount of CPU power that each VM can use is specified in advance, and that specified value can be used as a conservative estimate of the maximum CPU demand if no more reliable estimates are available.

The fourth contribution of this work is to formulate the MIP so that the CPU demand is expressed using two percentiles (e.g., 95th and 100th percentiles) that can be estimated reliably. In existing studies, the CPU demand of a VM is represented by a single value as opposed to our dual percentiles approach. The prior work such as [2] uses the peak usage (corresponding to the 100th percentile) for the CPU demand value. However, Bichler et al. [2] report that a CPU allocation based on peak CPU usage is conservative, because the typical CPU usage of a VM is far smaller than its peak CPU usage. We can use the 95th percentile results instead of the peak CPU usage, but this might result in consolidating too many VMs into a server, because we ignore the peak CPU usage data for the VM allocation. When we want to achieve high dependability that cannot be guaranteed with the 95th percentile results but want to utilize the CPUs at levels that cannot be attained by allocations based on the peak CPU demand, we might be able to use higher percentiles (such as 98th percentiles), but the 98th percentiles are less reliable as mentioned above. The use of dual percentiles allows us to achieve a level of dependability and CPU utilization that can only be achieved with intermediate percentiles.

With the proposed approach, we reduced the number of allocated CPUs by 30%, compared to the allocation by the bank’s expert, and hosted 20 VMs on four physical servers. The expert evaluated our allocation and has agreed that our allocation indeed provides the necessary dependability for the virtualized system under consideration.

In this paper, we primarily used IBM logical partitions (LPAR) as the means of virtualization. LPAR is a hardware partitioning approach, where a physical server can be subdivided into multiple servers and operating systems that run on them. The techniques introduced in this paper can be applied to other hardware-based partitioning approaches such as HP nPAR and also to software virtualization, where hypervisor software such as VMware, Xen, or KVM Linux creates virtual servers for various operating systems.

The rest of the paper is organized as follows. In Section II, we start by studying the characteristics of the workload of the VMs at the bank. In Section III, we formulate a simple MIP without consideration of the fault-tolerant mechanisms. We focus on how to use the dual percentiles in the MIP formulation. In Section IV, we describe the dynamic programming for optimally partitioning the time horizon into slots of variable lengths. In Section V, we extend the MIP to include the fault-tolerant mechanisms. In Section VI, we present the results of our experiments with the proposed approach. In Section VII, we discuss related work, followed by our conclusions.

II. CPU USAGE CHARACTERISTICS

In this section, we describe the CPU usage characteristics of the workloads in this bank’s data center. We obtained CPU usage data for 20 workloads for one month at one minute intervals, including both weekdays and weekends. Figure 2 shows the statistics of each workload, where the CPU usage of each workload was normalized as a percentage of the peak CPU usage and the workloads are sorted by their average CPU usage. For example, the average CPU usage of the rightmost workload (Workload ID 20) is slightly over 20% of its maximum CPU usage, and the 98th percentile of its CPU usage is approximately 80% of its maximum CPU usage. From Figure 2, we see that all of the workloads have low average CPU usages compared to their peak CPU usages (less than 25% for all of the workloads) and even the 98th or 99th percentile point of each CPU usage is less than 60% of the peak CPU usage for most of the workloads. These results show that CPU usage is typically low and rarely becomes high. In other words, the CPU usage history has spikes and the spikes can be regarded as outliers. Similar workload characteristics were also observed in [5], where their data was obtained from a data center for enterprise applications including customer relationship management.

Next we investigated the relationships between the CPU usage and the time of day. We picked two workloads from the 20 workloads for detailed investigation. We divided each day into 24 hour slices and computed the 95th and 98th percentiles.
of the CPU usage for each one-hour slice. Figures 3 and 4 show the results, in which the error bar of each percentile point shows its confidence interval (the bootstrap percentile interval (Section 8.3 from [6])). These results show that lower percentile points (e.g., 95th percentile points) are more reliable than higher percentile points (e.g., 98th percentile points), due to the need for more data to compute higher percentile points with comparable accuracy. We also see that the CPU usage depends on the time. From the 95th percentile points of Figure 3, this workload seems to have high loads between 8 am and 10 am and lower loads at other times. Based on these characteristics, we might be able to infer that the application of this workload involves some kind of authentication service for the users’ logon requests. Next, from the 95th percentile points of Figure 4, this workload is consistently heavy from 8 am to 7 pm and has another peak time around midnight. This VM might provide some services to users during the day and run heavy batch jobs at midnight. Similarly, we confirmed that the other 18 workloads also have their own time trends depending on their applications. For example, some VMs have steady CPU usages, and some workloads have high loads at night and low usages during the day.

What we find from the results in this section is that the CPU demand at the bank can be best characterized by percentiles rather than the peak CPU demands, which are too conservative for the server consolidation. In addition, since higher percentile points are less reliable, we need to be careful when we determine which percentile points to use for server consolidation. Throughout this paper, we use the 95 percentile points as the best estimates of the CPU demands for the bank’s situation.

III. DUAL PERCENTILE TRANSFORMATION

The purpose of this section is to introduce our idea about the use of dual percentiles in a mixed integer programming (MIP) formulation for finding the optimal allocation of VMs. To clarify the technique of using dual percentiles, this section ignores the fault-tolerant mechanisms and the plans for migration during maintenance work, which will be incorporated in Section V.

We start by presenting a standard MIP formulation as in [2], [4]. The time is divided into a set of time segments \( \tau_t \) for \( t \in T \). Let \( r_{i,t} \) be the required CPU resources for VM \( i \in V \) in time segment \( \tau_t \), where \( V \) is a set of VMs. Let \( C_k \) be the capacity of the CPU resource of server \( k \in K \), where \( K \) is the set of physical servers. Since the objective of the bank...
is to minimize the total number of allocated CPUs, the VM packing problem can be formulated as

\[
\min \sum_{k \in K} z_k \\
\text{s.t. } \sum_{i \in V} r_{i,t} x_{i,k} \leq z_k, \forall k \in K, t \in T \\
\sum_{k \in K} x_{i,k} = 1, \forall i \in V \\
z_{k} \in \{0, 1, 2, \ldots \}, \forall k \in K,
\]

where the variable \( x_{i,k} \) indicates whether or not VM \( i \) is allocated to server \( k \), and the variable \( z_k \) denotes the number of CPUs allocated to server \( k \). The objective function minimizes the total number of the CPUs allocated to each server. Constraint (1) ensures that the number of allocated CPUs does not exceed the server capacity. Constraint (2) ensures that the number of allocated CPU resources for server \( k \) must be, at every time segment, at least the total amount of required CPU resources for the VMs allocated to that server. Constraint (3) ensures that each VM must be assigned to a server.

This problem is often called the vector bin packing problem, and it is known to be NP-hard. Therefore, researchers often use an exponential time algorithm (such as mixed integer programming solvers) to obtain an optimal solution [2], [4] or an approximation algorithm with heuristics specialized for this problem (such as a genetic algorithm [4]).

It is important to use an appropriate value for each \( r_{i,t} \). If we wanted to be very conservative, we could use the maximum CPU usage of VM \( i \) in time segment \( \tau \) as in [2] or perhaps even add some margin to the maximum CPU usage. However, as we have already seen in the previous section, the maximum CPU usage can be seen as an outlier of the CPU usage history and so it is too pessimistic to use the maximum CPU usage for \( r_{i,t} \). Our approach is to use a percentile point of the CPU history instead, which mitigates the influence of the outliers in the CPU usage history. Unless otherwise stated, we use the 95th percentile point for each \( r_{i,t} \).

We regard the prediction of future workload and the allocation of VMs as separate problems, and focus on the allocation in this paper. This is because many workloads in the real world, such as those of the bank servers we considered, have statistical regularities, so that percentiles of CPU usage in the future can be estimated reliably. We believe that the performances of the allocation algorithms can be compared by isolating them from the effects of incorrect estimates of future workloads. We acknowledge that the prediction of future workloads with high accuracy is often difficult and that making accurate future workload predictions is an important topic for future research.

We propose a new MIP formulation, since the 95th percentile might not provide sufficient dependability, but the higher percentiles cannot be estimated reliably. If we use the 95th percentile point for every \( r_{i,t} \), then the obtained results are based only on the 95th percentile points and the information of the maximum CPU usage of each VM is completely ignored. This can cause a problem if the maximum CPU usage and the 95th percentile point are too far apart, which is indeed the case in Figure 2. Therefore, we propose characterizing the CPU usage of each VM with two values: one is the 95th percentile point of the CPU usage, and the other is the maximum CPU usage. Recall that a limit on the CPU usage is manually specified at the bank (by setting a virtual processor (VP) value for each LPAR), and this limit can be used as the maximum CPU usage if the maximum CPU usage cannot be estimated reliably with other methods. Then we reformulate the VM packing problem in such a way that a sufficient amount of CPU resources are available even when one of the VMs uses its maximum CPU usage as long as the CPU usages of the other VMs are at most 95th percentile of the CPU usage. This can be done by replacing Inequality (2) of the standard MIP formulation with the following two inequalities.

\[
\sum_{i \in V} r_{i,t} x_{i,k} + u_k \leq z_k, \forall k \in K, t \in T, \\
(r_{i,t}^{\text{100}} - r_{i,t}) x_{i,k} \leq u_k, \forall i \in V, \forall k \in K, t \in T,
\]

where the constant \( r_{i,t}^{\text{100}} \) represents the maximum CPU usage of VM \( i \) in time segment \( \tau \) and variable \( u_k \) for server \( k \in K \) represents the surplus resources that can be consumed by at most one VM. We call this transformation of the MIP formulation the dual percentile transformation and use this technique in Section V.

Note that, for more conservative users, we can extend this dual percentile formulation so that at most two (or more) VMs can be at full utilization instead of one VM. We omit the details of the extension because of the length.

**IV. OPTIMAL TIME PARTITIONING**

The time partitioning in the MIP formulation is important to capture the periodic CPU-usage trends. Given the number of partitions \( N \) as an input, prior work [2], [4] partitions the time horizon into fixed-length intervals of equal size. Variable-length intervals are, however, superior to fixed-length ones. For example, the load is often stable during the night except when scheduled jobs are running, while the load can fluctuate significantly in the morning. Figure 3(a) shows an example of such behavior. Therefore, it makes sense to partition the night into long intervals and the morning into short intervals.

Here we propose an optimal partitioning algorithm for variable-length time segments. For ease of understanding, we consider the problem of partitioning 24 hours into \( N \) time segments \( \tau_1, \ldots, \tau_N \). We assume that the minimum partitioning unit is one hour, that \( \tau_1 \) starts at midnight, and that \( \tau_N \) ends at the following midnight. Let \( T = \{1, 2, \ldots, 23\} \) be the set of candidate hours where the time is partitioned. Our problem is to compute \( t_1, \ldots, t_{N-1} \in T \) such that \( t_n = [t_{n-1}, t_n] \) for \( n = 1, \ldots, N \), where we define \( t_0 = 0 \) and \( t_N = 24 \).
The objective of our partitioning is to minimize
\[
\sum_{n=1,...,N} \left( (t_n - t_{n-1}) \sum_{i \in V} c_i(t_{n-1}, t_n) \right),
\]
where \( V \) denotes the set of VMs, and \( c_i(t_{n-1}, t_n) \) denotes the required CPU resources for VM \( i \) in time segment \([t_{n-1}, t_n]\). We assume that \( c_i(t_{n-1}, t_n) \) is precalculated based on the CPU history of VM \( i \). The term \((t_n - t_{n-1}) c_i(t_{n-1}, t_n)\) can be interpreted as the consumed resources on a server by VM \( i \) in the time segment from \( t_{n-1} \) to \( t_n \) (see Figure 5 for an intuitive representation). Thus the optimal partitioning with this objective function minimizes the total resources consumed by the VMs.

This is how we solve the problem of minimizing Equation (6) with dynamic programming. Let \( \text{cost}(n, t) \) be the objective value when we optimally partition time segment \([0, t]\) into \( n \) time segments. If \( n = 1 \), then it is easy to see that \( \text{cost}(1, t) = \sum_{i \in V} c_i(0, t) \). Next, if \( n > 1 \), then this recurrence formula holds:
\[
\text{cost}(n, t) = \min_{t' \in T} \left\{ \text{cost}(n - 1, t') + (t - t') \sum_{i \in V} c_i(t', t) \right\}.
\]

Since the first term on the right-hand side corresponds to the \( n - 1 \) partition of time segment \([0, t']\) and the second term on the right-hand side corresponds to a single time segment \([t', t]\), the left-hand side is defined over the minimum of possible \( t' \in T \). Then the optimal partitioning can be solved by recursively calculating \( \text{cost}(N, 24) \).

It is easy to extend the time partitioning algorithm for an arbitrary set \( T \), so we can use arbitrary time windows for the partitioning. (For example, we can use \( T = \{1, 2, \ldots, 47\} \) when the minimum partitioning unit is 30 minutes.) It would be better to use a larger set \( T \) to capture the time trends of the usage of each CPU more accurately, but the shorter the time window (i.e., the larger the set \( T \)), the more difficult it is to estimate the percentiles accurately. For our bank’s case, we found that a one-hour time window is the best choice.

Let us analyze the time complexity of the time partitioning. The dynamic programming takes \( O(N |T| |V|) \) time. Since we assumed that \( t_0 = 0 \), this result is restricted to the case where the first partition starts at midnight. Removing the restriction of \( t_0 = 0 \) requires that we solve the dynamic programming while changing \( t_0 \), which yields an \( O(N |T|^2 |V|) \) time algorithm overall. Note also that precalculating \( c_i(t_{n-1}, t_n) \) requires a total of \(|T|^2\) combinations of \((t_{n-1}, t_n)\) for each VM.

V. MIP FOR DEPENDABLE VM ALLOCATIONS

In this section, we incorporate the fault-tolerant mechanisms into the basic MIP formulation described in Section III. In financial companies, it is standard to use multiple clusters of VMs for a single business application, and each of the clusters are operated in redundancy mode in case of server failure or maintenance. There are two types of clusters for redundancy: the Active-Standby (AS) cluster and the Active-Active (AA) cluster. Let us refer to Figure 6 for an example of applications \( b1 \) and \( b2 \), where \( b1 \) consists of 7 clusters of VMs (four AS clusters and three AA clusters), and \( b2 \) with 2 clusters (one AS cluster and one AA cluster). Our task is to allocate each of VMs in the clusters to serve all of the applications with a minimum number of CPUs in a dependable manner.

Each AS cluster contains exactly two VMs: an active VM that fully handles the assigned workload, and a standby VM that becomes active if the server that hosts the active VM fails. This means that the active and standby VMs of the same cluster cannot be placed on the same server. If properly configured, the standby VM usually consumes only a small percentage of the resources (perhaps 20%) compared to the active VM.

In contrast, the AA cluster contains at least two active VMs that handle its workload in the same way. For example, an AA cluster with four VMs constitutes a single application, and each of the four VMs consumes one fourth of the resources required by the application. Again, the VMs in the same AA cluster cannot be placed on the same physical server. When a VM in an AA cluster experiences a server failure, the other VMs in the AA cluster must take over the VM that was running on the failed server. As an example, if one of the VMs in an AA cluster of size four experiences a server failure, then the required resources for each of the other three VMs should increase by approximately 33%.

In our work, we assume that only one server at a time experiences a failure. In addition to the server failure, we must consider the server maintenance. When a physical server undergoes maintenance, all of the VMs on that server must be migrated to other servers.
Here is how we formulate the MIP, taking into account server maintenance and failures. Table I lists some of the notation used in our formulation. Let

\[ x_{b,c,i,k,l} \in \{0,1\}, \]

\[ \forall b \in B, \forall c \in C_{AS}(b), \forall i \in \{A,S\}, \forall k,l \in K : k \neq l (7) \]

be the decision variable for a VM in an AS cluster \( c \) in a business application \( b \), with a value of 1 if and only if the server \( k \) is used to serve the VM under normal operation, and the server \( l \) during maintenance. The index \( c \in C_{AS}(b) \) corresponds to an AS cluster in application \( b \), and the value of \( i \) is either active (A) or standby (S). Thus, the first three indices \( b, c, i \) uniquely determine a VM in an AS cluster of a business application.

Similarly for VMs in an AA cluster, let

\[ y_{b,c,k,l} \in \{0,1\}, \forall b \in B, \forall c \in C_{AA}(b), \forall k,l \in K : k \neq l \]

be the decision variable with a value of 1 if and only if the server \( k \) and \( l \) are used to serve \( b \) under normal operation and maintenance, respectively. The index \( c \in C_{AA}(b) \) corresponds to an AA cluster for application \( b \). Notice that the first two indices \( b \) and \( c \) are used to locate an AA cluster of \( b \), while the individual VMs in the cluster are indistinguishable, unlike the VMs in an AS cluster.

To cope with a server failure, we must add constraints that require the VMs of the same cluster to be allocated to different servers. For the AS cluster, this is achieved by adding the constraints

\[ \sum_{l \in K} x_{b,c,A,k,l} + \sum_{l \in K} x_{b,c,S,k,l} \leq 1, \]

\[ \forall b \in B, \forall c \in C_{AS}(b), \forall k \in K, \]

\[ \sum_{k,l : k \neq l} x_{b,c,i,k,l} = 1, \]

\[ \forall b \in B, \forall c \in C_{AS}(b), \forall i \in \{A,S\}. \] (10)

Constraint (9) ensures that the active and standby VMs in the same cluster cannot be placed on the same server, while Constraint (10) ensures that the server used in normal operation is different from the server used during maintenance and that exactly one server is used for each active and standby VM.

Meanwhile for the AA cluster, the corresponding constraints are

\[ \sum_{k \in K} y_{b,c,k,l} \leq 1, \forall b \in B, \forall c \in C_{AA}(b), \forall k \in K, \] (11)

\[ \sum_{k \in K, k \neq l} y_{b,c,k,l} = q_{b,c}, \forall b \in B, \forall c \in C_{AA}(b). \] (12)

Constraint (11) ensures that at most one server is reserved for maintenance of a VM of AA cluster at server \( k \), while Constraint (12) ensures that the server used in normal operation is different from the server used during maintenance and that exactly \( q_{b,c} \) servers are used for the AA clusters of application \( b \).

Here are the constraints for balancing the active VMs:

\[ |M_b| \leq \sum_{c \in C_{AS}(b)} \sum_{l \in K} x_{b,c,A,k,l} + \sum_{c \in C_{AA}(b)} \sum_{l \in K} y_{b,c,k,l} \leq |M_b|, \]

\[ \forall b \in B, \forall k \in K, \]

where

\[ M_b = \left| C_{AS}(b) \right| + \sum_{c \in C_{AA}(b)} q_{b,c} |K| \]

is introduced to simplify the notation. These constraints are used so that each of the servers hosts roughly the same number, \( M_b \), of active VMs for each \( b \). This mitigates the influence of the server failure on the active VMs.

We also introduce slack decision variables for increasing the efficiency in finding an optimal allocation. In theory, one can find an optimal allocation by just using the variables \( x_{b,c,i,k,l} \) and \( y_{b,c,k,l} \). However, in practice, to help a MIP solver limit the search space, it is better to add redundant constraints by introducing 0-1 slack variables. In addition, these slack variables will be useful in describing additional constraints for server failure and maintenance. The slack variables we added for the AS clusters are

\[ v_{b,c,k,k'} \in \{0,1\}, \]

\[ \forall b \in B, \forall c \in C_{AS}(b), \forall k,k' \in K : k \neq k'. \] (16)

The variable \( v_{b,c,k,k'} \) is 1 if the active VM in the AS cluster \( c \) for \( b \) is at server \( k \) and the standby VM is at server \( k' \) in the normal operation state by virtue of these constraints:

\[ v_{b,c,k,k'} + 1 \geq \sum_{l \in K} x_{b,c,A,k',l} + \sum_{l \in K} x_{b,c,S,k,l}, \]

\[ \forall b \in B, \forall c \in C_{AS}(b), \forall k,k' \in K : k \neq k'. \] (17)

The slack variables for the AA clusters are

\[ w_{b,c,k,k'} \in \{0,1\}, \]

\[ \forall b \in B, \forall c \in C_{AA}(b), \forall k,k' \in K : k \neq k'. \] (18)

The variable \( w_{b,c,k,k'} \) is 1 if two VMs in AA cluster \( c \) for \( b \) use server \( k \) and \( k' \) in the normal operation state. This is satisfied by the constraints:

\[ w_{b,c,k,k'} + 1 \geq \sum_{l \in K} y_{b,c,k,l} + \sum_{l \in K} y_{b,c,k',l}, \]

\[ \forall b \in B, \forall c \in C_{AA}(b), \forall k,k' \in K : k \neq k'. \] (19)
As a simple notation to define the constraints on the necessary resources, let

\[ D_{t,k} = \sum_{b \in B} \sum_{c \in C_A} (a_{b,c,t} x_{b,c,k} + s_{b,c,t} x_{b,c,s,k}) + \sum_{b \in B} \sum_{c \in C_A} \left( \frac{p_{b,c,t}}{q_{b,c}} - \frac{p_{b,c,t}}{q_{b,c}} \right) w_{j,k,k'} \leq z_k, \forall t, \forall k, k' \in K : k \neq k' \] (14)

\[ D_{t,k} = \sum_{b \in B} \sum_{c \in C_A} (a_{b,c,t} x_{b,c,k} + s_{b,c,t} x_{b,c,s,k}) + \sum_{b \in B} \sum_{c \in C_A} \left( \frac{p_{b,c,t}}{q_{b,c}} \right) y_{b,c,k,k'} \leq z_k, \forall t, \forall k, k' \in K : k \neq k' \] (15)

for \( t \in T, k \in K \). Notice that \( D_{t,k} \) represents the amount of the resources necessary at server \( k \) at time \( t \).

Inequality (14) represents the limit for the increase of resources at server \( k \) due to the failure of server \( k' \). We refer to these constraints as failure-limiting constraints. In Inequality (14), the increases of CPU resources are from two sources: (i) those from standby VMs turning into active VMs in AS clusters, which is represented as \( (a_{b,c,t} - s_{b,c,t}) v_{b,c,k'} \), and (ii) those from active VMs in AA clusters, which is represented as \( (p_{b,c,t}/q_{b,c} - 1) - p_{b,c,t}/q_{b,c} \) \( w_{j,k,k'} \).

Inequality (15) represents the increases of the amounts of the resources at server \( k \) due to the maintenance work on server \( k' \), called the maintenance-limiting constraints. Their meanings can be understood analogously to the constraints related to the failure of server \( k' \), except that unlike failure, no increase in resources occurs when the VMs are moved.

Finally, the objective function of the MIP formulation for dependable VM allocations is to minimize the sum of the resources, \( \sum_k z_k \), for all \( k \in K \) under the constraints shown in Eqs. (7)–(19), the failure-limiting constraints (14), and the maintenance-limiting constraints (15). In addition, we apply the two-quantile transformation (defined in Section III) to the obtained MIP for a higher level of dependability. The details of the MIP formulation are omitted due to space limitations. Notice that the number of variables and constraints in our MIP formulation are respectively \( O(K^2|V|) \) and \( O(K^2 |T||V|) \), assuming the number of VMs in an AA cluster is at most a fixed constant, where \( V \) is the set of VMs.

### VI. Experimental Results

In this section, we present the results of applying the proposed approaches to an allocation of VMs for a bank we studied, with a focus on the running time of the proposed approaches. Our goal was to find an optimal and dependable allocation of the 20 VMs to four physical servers, where at most 8, 10, 8, and 7 CPUs can be allocated, respectively. The 20 VMs constitute two business applications as shown in Figure 6 and run the workloads that were examined in Section II. All of the computations for the allocation were done on a PC with an Intel Core i5 CPU (M560) running at 2.67 GHz and with 3 GB of RAM. We found that an optimal and dependable allocation was found within a few seconds for systems of the size that are of interest to the bank.

First we partitioned the time horizon into multiple segments using the dynamic programming approach presented in Section IV. In our experiments, we used the 95th percentile points as the input and partitioned the 24-hour time horizon into 6 segments. Changing the start time of the first partition \( t_0 \), our algorithm successfully divided the time horizon into six segments that were divided at midnight (12 am), 8 am, 10 am, 7 pm, 10 pm, and 11 pm. The execution time was less than 0.1 seconds, excluding the precomputation time for \( c_i(t_{k-1}, t_k) \).

Next we formulated the VM packing problem as the full MIP model described in Section V. We used the 95th percentile point as the required CPU resource and the 100th percentile point as the maximum CPU demand for each workload. The time horizon was divided according to the partitions obtained by the dynamic programming. The resulting MIP formulation had 4 integer variables, 368 binary variables, and more than 4,000 constraints. We solved this MIP optimally in 3.20 seconds by using IBM ILOG CPLEX Version 12.3 with the default parameter settings.

### VII. Related Work

To the best of our knowledge, Santos et al. [7] gave the first MIP formulation for server consolidation, and many others have proposed similar MIP formulations [2], [4], [8], [9]. However, those MIP formulations do not take into account server failure or maintenance, and so handle simpler models than ours, which incorporates various types of fault-tolerance mechanisms: the maintenance-limiting constraints and failure-limiting constraints with AS-clusters and AA-clusters. For the failure-limiting constraints with AA-clusters, Machida et al. [10] proposed heuristics (not based on a MIP formulation) to allocate VMs that could survive any fixed number of server failures, although they did not consider the other types of constraints. Our MIP formulation can be extended to cope with multiple server failures by using an idea similar to theirs.

Although there is a large body of the literature on server consolidation, most of the work focused on minimizing the number of servers, unlike our work that minimizes the number of CPUs. An exception is Rolia et al. [4] who considered the minimization of the number of allocated CPUs. There are many papers on VM placement with other objectives. Srikantiah et al. [11] proposes a heuristic algorithm for VM allocation to reduce total energy cost and Ilyas et al. [12] proposes a MIP...
formulation to obtain an optimal VM relocation schedule to reduce the total energy cost. Meng et al. [13] proposes an algorithm to find optimal allocation with respect to network cost.

The periodicity of the CPU demands was also exploited in [2], [3], [4], but all of them partitioned the time horizon with fixed-width time segments. For example, 5 or 15 minutes were used for the width of the time segments in [4]. Meng et al. [14] proposed an algorithm to find pairs of VMs such that the two VMs in a pair have complementary CPU usage behavior over time without partitioning the time horizon, and they packed the VMs into servers so that two VMs in each pair were consolidated onto the same server.

For the (vector) bin packing problem, simple heuristics such as First-Fit-Decreasing (FFD) algorithm are often used, and Ajiro and Tanaka proposed improved heuristics for this problem [15]. However, such heuristics (including FFD) are difficult to apply to a complex problem setting with fault-tolerance mechanisms and temporal variations in the CPU demands. Some prior work [16], [17], [18] addressed the theoretical aspects of the vector bin packing problem, where modeling the resource usage history with random distributions such as Bernoulli-type distributions, Poisson distributions, exponential distributions, and Gaussian distributions.

VIII. CONCLUSION

In this paper, we formulated the virtual machine packing problem as a mixed integer program. Higher dependability is achieved by using two percentiles to characterize each workload and by taking into account the fault-tolerance mechanisms in the MIP formulation. In addition, we proposed an optimal partition of the time horizon into multiple slots of variable length, which is essential both for capturing the patterns of temporal variations in the CPU demands and for reducing the complexity of the MIP formulation.

In our project with a bank, we successfully found an optimal and dependable allocation for the 20 virtual machines by using the partitioning algorithm and the MIP formulation. The number of allocated CPUs in our allocation was 30% less than that of the bank’s expert. After careful examination, the expert agreed that our allocation was indeed adequate to provide the necessary dependability for the business applications under consideration.

In another project with a retail company, we conducted experiments for 116 virtual machines. While the objective was set to minimize the number of physical servers used, and the fault-tolerance concerns were not as severe as the project with the bank, our MIP formulation succeeded in reducing the number of physical servers by 50%.

Our future work will develop more effective heuristics for larger instances, since our current approach relies heavily on the high performance of the state-of-the-art MIP solver.

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their valuable comments. This study was partly sponsored by the Ministry of Economy, Trade and Industry, Japan (METI) under a contract for the next-generation technology for a dependable and an energy saved IT infrastructure R&D Program.

REFERENCES