Redundant Virtual Machine Placement for Fault-tolerant Consolidated Server Clusters

Fumio Machida, Masahiro Kawato and Yoshiharu Maeno
Service Platforms Research Laboratories, NEC Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211-8666, Japan
{h-machida@ab, m-kawato@ap, and y-maeno@aj}.jp.nec.com

***Abstract*— Consolidated server systems using server virtualization involves serious risks of host server failures that induce unexpected downs of all hosted virtual machines and applications. To protect applications requiring high-availability from unpredictable host server failures, redundant configuration using virtual machines can be an effective countermeasure. This paper presents a virtual machine placement method for establishing a redundant configuration against host server failures with less host servers. The proposed method estimates the requisite minimum number of virtual machines according to the performance requirements of application services and decides an optimum virtual machine placement so that minimum configurations survive at any *k* host server failures. The evaluation results clarify that the proposed method achieves requested fault-tolerance level with less number of hosting servers compared to the conventional N+M redundant configuration approach.**

***Index Terms*—Virtual Machine, Fault-tolerant, Redundant Configuration, Placement Algorithm**

I. INTRODUCTION

Sever virtualization has emerged as a powerful technique for consolidating servers in data centers. Virtualization platforms such as VMware Infrastructure [1] and Citrix XenServer [2] virtualize hardware resources and generate multiple virtual execution environments called virtual machines. Each virtual machine can behave as an independent physical server, and hence multiple OS instances can run concurrently on a hosting server. Data centers and large-distributed enterprise systems have many temporal unutilized servers. By converting these servers to virtual machines and running them on the fewer hosting servers, resource utilizations of the whole data centers improves [3]. The reduction of the number of hosting servers also contributes to cutting back the power consumptions in the data centers [4][5].

A failure of a hosting server becomes a serious problem in consolidated server systems using virtualization. Virtual machines depend on physical devices and virtualization platform on the hosting server. When the hosting server goes down due to any failures of their components, all virtual machines on this server are unable to escape from service down. The more virtual machines the hosting server hosts, the more serious damage a failure of this hosting server causes. Any countermeasures against multiple server downs caused by host server failures are required.

This paper presents a method to make a redundant configuration of virtual machines in anticipation of host server failures in consolidated server systems hosting various online applications. The proposed method estimates the requisite minimum number of virtual machines according to performance requirements of application services and decides an optimum virtual machine placement so that minimum configurations survive at any *k* host server failures. In terms of the cost reduction by server consolidation, the number of hosting server should be minimized. An optimum virtual machine placement for minimizing the number of required hosting server depends on several factors such as the required fault-tolerance level *k*, the capacity of hosting server, the number of applications and their performance requirements. The paper defines this problem as a combinatorial optimization problem and presents an algorithm for determining an optimum virtual machine placement under given conditions. From some evaluation results, we have observed the proposed method achieves requested fault-tolerance level *k* with less number of hosting servers compared to the conventional N+M redundant configuration approach. N+M redundant configuration prepares M redundant components so as to keep N components at any M component failures.

The rest of the paper is organized as follows. Section II describes a configuration and requirements for consolidated server systems using virtualization. Section III provides a problem definition for determining redundant virtual machine configurations while minimizing the number of required hosting servers. Section IV discusses a performance model for estimating required resources to meet performance requirements for applications. In Section V, a method for determining a redundant configuration under the given constraints is proposed. Experiments and evaluations are shown in Section VI, related work is presented in Section VII, and finally the summary of this paper is given in Section VIII.

II. REQUIREMENTS FOR HOSTING SERVER CLUSTER

This section describes the configuration of a hosting server cluster that hosts various online applications, and their performance requirements and level of fault-tolerance.

978-1-4244-5367-2/10/$26.00 ©2010 IEEE

A. Hosting Server Cluster

Data center providers recently provide virtual machine hosting service by introducing server virtualization to own physical server clusters. Application providers, who want to launch application services in the data center, can rent virtual machines and start services quickly by signing a contract with the data center provider. The application services introduced by the application providers often take redundant server configurations for assuring scalability and availability. There are many redundant configuration methods for online applications such as web servers, mail servers and data base servers. Web servers distribute their workloads to multiple servers using load balancing module [6]. Mail servers can improve their performance and fault-tolerance by distributing processes to replication servers using DNS round robin. Data base servers often use clustering method for scalability and high-availability [7]. This paper assumes that each hosted application has own redundant configuration method and focuses on the virtual machine placement issue in data center providers to provide a fault-tolerant hosing server cluster.

B. Performance Requirements

A data center provider and an application provider make an agreement for performance of applications in the service level agreement (SLA). Performance requirements of online applications such as web applications are usually specified average response time of application service. The response time depends on the various factors like resource capacities, utilization limitations and network congestions. The data center provider has to allocate sufficient resources to the applications to keep the requested average response time.

Performance requirements for an application restrict the minimum resource configurations including the number of virtual machines or CPUs for the application. By allocating more computing resources, most of CPU-intensive online applications like web applications improve their processing power and average response time. Since the relation between average response time and the CPU allocation can be modeled using queuing theory, the requisite number of virtual machines or CPUs can be estimated from the requested average response time. The detail of the performance model is described in the Section IV.

C. Fault-tolerance of Hosting Servers

A host server failure is a serious issue in the consolidated server systems because it causes the downs of multiple virtual machines on the hosting server. Host server failures are induced by various causes like OS hang up, device failures and unexpected power down. To protect application services from any host server failures, data center provider should configure the hosting server cluster with redundant application instances.

The fault-tolerance level of the data center service against host server failure can be measured by the acceptable number of simultaneous host server failures. The metric indicates a capability of keeping application services survive at server failures in the data center. In the area of the distributed computing systems, a system that can continue services in case of any k components failures is called *k-fault-tolerance* [8]. In order to make a system k-fault-tolerance without virtualization, the data center provider should prepare k additional servers for each application. For the data center using virtualization, a placement of virtual machines is important as well as the preparation of redundant application instances to achieve k-fault tolerance.

The fault-tolerance level can be changed by virtual machine placements. Let us consider an example of redundant configuration of four applications $\{a_1, a_2, a_3, a_4\}$ using three hosting servers $\{s_1, s_2, s_3\}$. Each physical server can run three virtual machines and each application requires at least one virtual machine for minimum configuration. Fig. 1 illustrates two different patterns of virtual machine placements. Fig. 1 (a) indicates that two virtual machines for application a_1 and one virtual machine for application a_4 are placed on the hosting server s_1. The descriptions are same for s_2 and s_3.

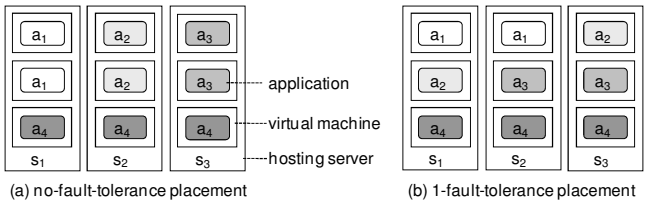

Fig. 1. Redundant configuration of virtual machines using three hosting servers

The difference between two placements appears at a host server failure. When any one of hosting servers fails, the placement (a) violates the minimum configuration of any one of applications (a_1, a_2 or a_3). On the other hand, the placement (b) keeps the minimum configuration of all applications and hence achieves 1-fault-tolerance.

The placement (a) becomes 1-fault-tolerance by adding a hosting server and allocating more virtual machines for application instances as shown in Fig. 2 (c). The more number of hosting servers generally increases the number of redundant application instances and improves the fault-tolerance level. However, the number of hosting servers in the data center should be reduced in terms of the total cost. Data center providers need to find a virtual machine placement that satisfies the required fault-tolerance level, while minimizing the number of hosting servers.

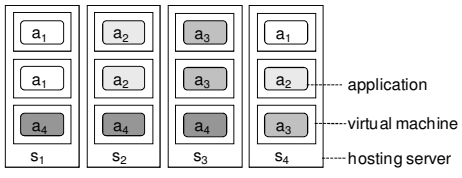

Fig. 2. Redundant configuration of virtual machines using four hosting servers

III. Problem Definition

The problem of virtual machine placement for minimizing the number of hosting server under the specified performance requirements and fault-tolerance level is defined as a combinatorial optimization problem. The assumptions made

here are that all hosting servers have the same capacity and all virtual machines are equal in terms of the required capacity. The assumptions are realistic especially for the simple hosting services charging by the amount of used virtual machines and providing the common quality of the hosting service. For providing the basis of the problem definition, the performance requirements are assumed to specify by the average response time.

A. Redundant Virtual Machine Placement

Let the set of applications hosted in the data center be $A = \{a_1, a_2, .. , a_n\}$, and the set of equipped hosting servers be $S=\{s_1, s_2, .. , s_m\}$. The max number of virtual machines is denoted as p that equals to the number of virtual CPUs on the server if each virtual machine uses one virtual CPU. Virtual machines on the server s_j are denoted as $\{v_{j1}, v_{j2}, .. ,v_{jp}\}$. The virtual machine placement is expressed by the projection function $\phi(v_{xy})$ that indicates the application running on the virtual machine v_{xy}. The goal of the optimization problem is to minimize the number of hosting servers m, while satisfying the fault-tolerance level k and all performance requirements that are specified by the average response time r_i for each application a_i.

Redundant virtual machine placement problem:

Solve a virtual machine placement ϕ so as to minimize the number of hosting servers m under the given values of n, p, k and $r_i (1 \le i \le n)$.

The problem is a combinatorial optimization problem that has objective function m.

TABLE I
NOTATION USED IN THE REDUNDANT VIRTUAL MACHINE PLACEMENT PROBLEM

Symbol	Description
a_i	Application class $(1 \le i \le n)$
s_j	Hosting server $(1 \le j \le m)$
n	Number of application classes
m	Number of hosting servers
p	Max number of virtual machines on a hosting server
k	Required fault-tolerance level
r_i	Required average response time for application a_i
v_{xy}	A virtual machine on the hosting server s_x $(1 \le x \le m, 1 \le y \le p)$
$\phi(v_{xy})$	Virtual machine placement that indicates the application class that runs on the v_{xy}

B. Lower Bound

The number of virtual machines allocated to an application a_i is limited by the performance requirements r_i. Let the minimum number of virtual machines for application a_i be c_i. The lower bound of the number of hosting server m in the redundant virtual machine placement problem can be derived theoretically from the consideration of the number of virtual machines surviving after k host server failures. The number of surviving virtual machines at k host server failures out of m host servers is given by $(m-k)\cdot p$. Since these virtual machines must contain the minimum number of virtual machines c_i for all a_i, the following condition is obtained.

$$(m-k)\cdot p \ge \sum_{i=1}^{n} c_i \quad (1)$$

Because m is an integer value, the lower bound of m is given as follows.

$$m \ge \left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil + k \quad (2)$$

IV. Performance Model

This section describes the performance model for determining the requisite minimum number of virtual machines to satisfy the given performance requirements.

The performance of an online application such as a web application service is often analyzed using queuing models. A simple performance model for single web server which assumes Poisson arrival, general service time and bounded accepted request number of K has been modeled as M/G/1/K queue model [9]. To incorporate the burst request arrival, the extended model MMPP/G/1/K has been presented [10]. For multi-tier application systems, the request arrival rate for each server depends on the load balancing/scheduling algorithms. Several studies used G/G/1 to model the performance of multi-tier web applications [11][12].

Since there is no general performance model that can apply various online applications, this paper introduce M/M/1 queue model as a basic example which assumes Poisson request arrival and exponential service time. These assumptions are not realistic in some online applications systems. The appropriate performance model for each application should be determined through some experiments or observations of real workload. According to M/M/1 model, the average response time r_i of application a_i with a service rate μ_i and a request arrival rate λ_i is modeled as follows [13].

$$r_i = \frac{1}{\mu_i - \lambda_i} \quad (3)$$

When the application gets c times larger computation power by using more virtual machines or virtual CPUs, the average response time is approximated as follows.

$$r_i = \frac{1}{\mu_i - \frac{\lambda_i}{c}} \quad (4)$$

Consequently, given the requested average response time r_i, the requisite minimum number of virtual machines c_i is bounded by following expression.

$$c_i \ge c = \frac{\lambda_i}{\mu_i - \frac{1}{r_i}} \quad (5)$$

Although the value of μ_i depends on the application and available resources, it can be estimated by observing the values of λ_i and r_i from some performance experiments. Expression (5) allows us to determine the requisite minimum number of virtual machines c_i for satisfying the requested average response time r_i. The estimated minimum configuration is used to determine the virtual machine placement method described in the next section.

TABLE II
NOTATION USED IN THE M/M/1 PERFORMANCE MODELS

Symbol	Description
λ_i	Request arrival rate of application a_i
μ_i	Service rate of application a_i
c_i	Requisite minimum number of virtual machines for application a_i

V. VIRTUAL MACHINE PLACEMENT

This section introduces *k*-redundancy method and multiple *k*-redundancy method for determining redundant virtual machine placement. The multiple *k*-redundancy method achieves theoretically minimum number of hosting servers.

A. Approach

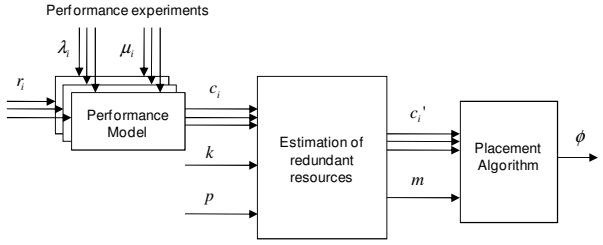

Fig. 3. Procedure to solve the redundant virtual machine placement problem

An overview of the proposed procedure to solve the redundant virtual machine placement problem is depicted in Fig. 3. First, performance model for each application is generated through experimental performance evaluations. Using the generated model, the requisite minimum number of virtual machines c_i is estimated for satisfying the required average response time r_i. Next, to achieve the required fault-tolerance level *k*, the number of redundant virtual machines c_i' is estimated in consideration with c_i and *p*. The number of required hosting servers *m* is solved at the same time. Finally, an algorithm for virtual machine placement determines a placement ϕ by c_i and *m*. The detail of the algorithm is shown in the following section.

Clearly, the estimation step dominates the decision of the required number of hosting servers *m* that is the objective function of the redundant virtual machine placement problem. As the estimation methods, the *k-redundancy method* is described in Section *C* and the *multiple k-redundancy method* is described in Section *D*.

B. Placement Algorithm

We define an algorithm for virtual machine placement so as to achieve *k* fault-tolerance under the given c_i and *m*. The redundant virtual machines that provide the same application service should be distributed to the different hosting servers in order to diversify the risks of service down or performance degradation due to the hosting server failures. Therefore, the placement algorithm for virtual machine placement is designed with the heuristic of distributing the same application instances to different hosting servers.

The placement algorithm is shown in Fig. 4. The function *ha-vm-placement* returns virtual machine placement *placement[]* with input parameters of *c[]* and *m*. This function sorts all of the virtual machines by the application classes, and in this order, allocates them to the different hosting servers in number order.

Algorithm 1

```
 1:# c[] : required VMs for each application
 2:# m   : number of the hosting servers
 3:
 4: ha-vm-placement(c[], m) {
 5:  cs[] = sort(c[]);  # sort by application type
 6:  total = cs[].length;  # total num of VMs
 7:  i=0, y=1;
 8:  while (i=<total) {
 9:    for (x=1; x++; x=<m) {
10:      placement[x, y] = cs[i];  # allocate
11:      i++;
12:    }
13:    y++;
14:  }
15:  return placement[];
16:}
```

Fig. 4. Algorithm for virtual machine placement

C. K-redundancy Method

K-redundancy method allocates *k* redundant virtual machines to each application. To accomplish the *k*-fault-tolerance of the hosting server clusters, at least *k* redundant virtual machines for each application a_i are needed besides the minimum configurations estimated as c_i. In the conventional server clusters without virtualization, *k*-fault-tolerance is accomplished by preparing *k* redundant physical servers. The *k*-redundancy method is established on this conventional approach. The total number of redundant virtual machines for each application a_i, c_i' is expressed as follows.

$$c_i' = c_i + k \tag{6}$$

With the *k*-redundancy method, virtual machines that host the same application instances must not run on the same hosting server. Otherwise a failure of a single hosting server causes multiple downs of the same application instances and leads to SLA violations. Since the placement algorithm allocates virtual machines to the different hosting servers in the order of s_j, this restriction is specified as the following constraint for *m*.

$$m \geq \max_i c_i' = \max_i c_i + k \tag{7}$$

Another constraint for *m* is derived from the fact that the total number of required virtual machines $\sum_{i=1}^{n} c_i'$ is not over the total available virtual machines on the *m* hosting servers. This restriction is expressed as follows and leads to another constraint condition for *m*.

$$\sum_{i=1}^{n} c_i' \leq m \cdot p \tag{8}$$

$$m \geq \frac{1}{p} \cdot \sum_{i=1}^{n} c_i' = \frac{1}{p} \cdot \left(\sum_{i=1}^{n} c_i + k \cdot n \right) \tag{9}$$

From the constraints (7) and (9), the minimum number of hosting servers m_{KR} by *k*-redundancy method is expressed as follows.

$$m_{\mathrm{KR}} = \max\{ \max_i c_i + k, \left\lceil \frac{1}{p} \cdot \left(\sum_{i=1}^{n} c_i + k \cdot n \right) \right\rceil \} \tag{10}$$

The algorithm 1 generates a virtual machine placement using the parameters c_i' and m_{KR} that are decided by the expression (6) and (10). In this virtual machine placement, minimum configurations given as c_i necessarily survive at any k hosting server failures. Since the system is admissible to k host server failures, k-fault-tolerance of hosing server cluster is achieved.

A drawback of the k-redundancy method is that the number of hosting server m_{KR} is not always equal to the theoretical minimum value. For example, the k-redundancy method does not achieve the theoretical minimum number of hosting server in the system n=3, k=1, p=3, and (c_1, c_2, c_3) = (3, 1, 1).

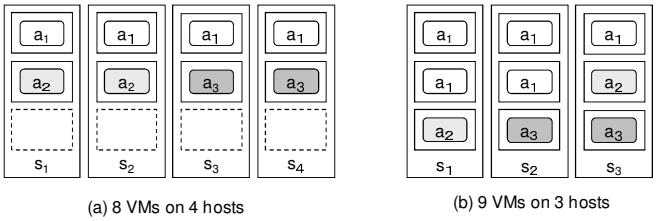

Fig. 5. Redundant virtual machine configuration by k-redundancy method (a) and an alternative (b)

The number of redundant virtual machines (c_1', c_2', c_3') are calculated as (4, 2, 2) by the expression (6). The m_{KR} equals to 4 by the expression (10) and the virtual machine placement is decided as Fig. 5 (a) by the algorithm 1. However, the k-fault-tolerance can be achieved with only three hosting servers, if the placement is provided as Fig. 5 (b) with an additional virtual machine for a_1. In this example, the k-redundancy method requires an unnecessary hosting server. As a result the k-redundancy method minimizes the total number of required virtual machines for k-fault-tolerance, but does not always minimize the number of hosting servers m.

D. Multiple k-redundancy Method

Multiple k-redundancy method improves the k-redundancy method for minimizing the required number of hosting servers.

There is a risk of more than k failures at k hosting server failures, if $c_i + k$ is greater than m. In order to keep c_i virtual machines at k hosting server failures, the multiple k-redundancy method prepares the integral multiple (multiples of x, where x is an integer) of k of redundant virtual machines. More specifically, the total number of redundant virtual machines for each application a_i decided by the multiple k-redundancy method is expressed as follows.

$$c_i' = c_i + \left\lceil \frac{c_i}{m-k} \right\rceil \cdot k \tag{11}$$

The minimum configurations specified by c_i can be sustained in case of any k hosting server failures, if the virtual machine placement is decided by algorithm 1 with c_i' estimated by expression (11). The proof is given as below.

- $1 \le c_i < m-k$

  From the expression (11), the number of redundant virtual machines is given by

  $$c_i' = c_i + k \tag{12}$$

  Since $c_i' \le m$ is true in this case, the c_i of virtual machines can survive at any k server failures.

- $(\alpha-1)\cdot(m-k) \le c_i < \alpha\cdot(m-k)$ where α is any integer greater than 1.

  From the expression (11), the number of redundant virtual machines is given by

  $$c_i' = c_i + \alpha \cdot k \tag{13}$$

  Since $(\alpha-1)\cdot m + k \le c_i' < \alpha \cdot m$ is true in this case, k server failures does not cause to more than $\alpha \cdot k$ failures of virtual machines and hence the c_i of virtual machines certainly survive.

From the consideration of the above two cases, it is proved that for all $a_i (1 \le c_i)$, c_i of virtual machines certainly survive at any k hosting server failures.

In the multiple k-redundancy method, there is no restriction that prohibits the existence of multiple application instances on the same hosting server. The number of hosting servers m is bounded by the constraint of the total number of required virtual machines. The total number of required virtual machines is not over the total available virtual machines $m*p$.

$$\sum_{i=1}^{n} c_i' \le m \cdot p \tag{14}$$

By rewriting the c_i' with the expression (11), the total number of required virtual machines are expressed as follows.

$$\begin{aligned} \sum_{i=1}^{n} c_i' &= \sum_{i=1}^{n} \left\{ c_i + \left\lceil \frac{c_i}{m-k} \right\rceil \cdot k \right\} \\ &\ge \sum_{i=1}^{n} \left\{ c_i + \frac{c_i}{m-k} \cdot k \right\} \\ &= \frac{m}{m-k} \cdot \sum_{i=1}^{n} c_i \end{aligned} \tag{15}$$

The expression (14) can be rewritten using the above inequality.

$$\frac{m}{m-k} \cdot \sum_{i=1}^{n} c_i \le m \cdot p \tag{16}$$

Then the constraint condition for m can be expressed as below.

$$m \ge \frac{1}{p} \cdot \sum_{i=1}^{n} c_i + k \tag{17}$$

Consequently, the minimum number of hosting servers m_{MKR} by the multiple k-redundancy method is given by

$$m_{MKR} = \left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil + k \tag{18}$$

By substituting m_{MKR} into the expression (11), the number of redundant virtual machines for application a_i is expressed as

$$c_i' = c_i + \left\lceil c_i \cdot \left(\left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil \right)^{-1} \right\rceil \cdot k \tag{19}$$

The parameters for algorithm 1, c_i' and m_{MKR} are decided by the expression (18) and (19), and the algorithm generates a virtual machine placement that achieves k-fault-tolerance of hosting server clusters. Since m_{MKR} is equal to the theoretical minimum value described in Section III-*B*, the multiple k-redundancy method can minimize the required hosting servers under any given conditions.

VI. EVALUATION

In order to evaluate the proposed redundant configuration method, we conducted performance experiments and simulations studies.

A. Performance Profiling

The proposed approach described in Section 5 uses the performance model for estimating the requisite minimum number of virtual machines from the performance requirements specified as r_i. A parameter of the M/M/1 model μ_i which varies depending on applications and available resources needs to be estimated by some performance experiments. Here, an example of performance model for a web server is considered. The M/M/1 model for the web server is generated from the experimental observation of the relationships among the number of allocated CPUs, request rates and average response times.

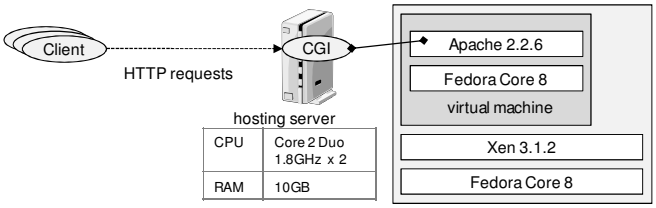

Fig. 6. Configuration of testing environment

Fig. 6 illustrates the configurations of performance test environment. A Xen-based virtual machine runs on the hosting server that has 2 Dual Core Intel Xeon 1.8 GHz processors and 10GB of RAM with Fedora Core 8 and Xen 3.1.2 hypervisor. An Apache 2.2.6 web server runs on top of the test server. The performance experiments are conducted by a test CGI script that receives web requests from another client host.

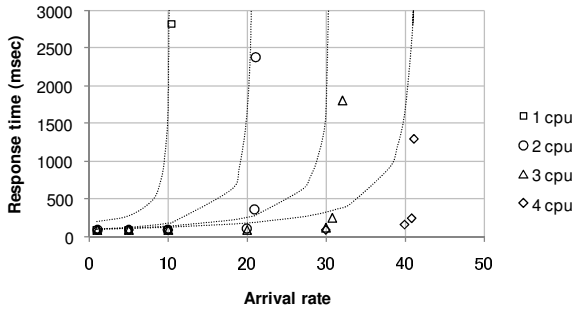

Fig. 7. Observations and model for request response times

Fig. 7 shows the change of relations between the request rates and the average response times by varying the number of virtual CPU allocation from 1 to 4. The average response time is always under 100 msec, unless the request rate over a certain point. When the request rate exceeds the point the average response time increases steeply. The value of the inflection point of request rate is proportional to the number of allocated virtual CPUs. The approximation curves by M/M/1 queue model with μ_i =10.6 are also figured as dotted lines in the Fig. 7. Although the shape of the curve does not completely fit to the observed values, the model gives a good indication for the boundary value of the request rate corresponding to the number of allocated virtual CPUs.

By conducting similar performance experiments, performance model for each application can be generated. The requisite minimum number of virtual machines or virtual CPUs is estimated from the generated performance model specified as expression (5) and required average response time r_i. For example, to keep the average response time of this application below 500 msec under the condition λ_i = 25, the performance model reveals that it requires more than 3 virtual CPUs.

B. Allocation Methods

As described in Section II, the number of required hosting servers for achieving the *k*-fault-tolerance under any given conditions changes depending on the virtual machine placement. In this section conventional approaches for virtual machine placement are described for comparison.

The problem of virtual machine placement for minimizing the number of required hosting server is formulized as a bin packing problem [14][15]. The bin packing problem is known to an NP-hard problem which is difficult to solve completely in the realistic time, thus some heuristic algorithms are used to cope with this problem. First-Fit decrease (FFD) is a well known powerful heuristic approach to the bin packing problem [16]. The FFD is an effective solution to virtual machine placement where each application instance on the virtual machine requires different size of resources (e.g. the number of CPUs). However, since the cluster configured using FFD does not have redundancy, additional *k* clusters that has the same configuration are required to accomplish the *k*-fault-tolerance. The required number of the hosting servers is shown as below.

♦ FFD

The FFD decides a virtual machine placement of a minimum cluster that satisfies all performance requirements of hosted applications, while minimizing the number of required hosting servers as much as possible. By preparing *k* backup clusters of this minimum cluster, the system can achieve *k*-fault-tolerance. Let the required number of hosting servers for the minimum configuration by the FFD be m_{FFD}. The value of m_{FDD} is bounded as follows.

$$m_{\mathrm{FFD}} \geq \left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil \tag{20}$$

In addition, an upper bound of m_{FFD} is given by

$$m_{\mathrm{FFD}} \leq \frac{11}{9} m_{\mathrm{OPT}} + 1 \leq \frac{11}{9} \left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil + 1 \tag{21}$$

where m_{OPT} is an optimum solution of the bin packing problem [16] and the value of m_{OPT} is more than $\left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil$.

Then the best and worst of the required number of hosting servers by FFD are given as follows.

$$m_{\mathrm{FFD\text{-}best}} = \left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil \cdot (k+1) \tag{22}$$

$$m_{\mathrm{FFD\text{-}worst}} = \left(\frac{11}{9} \left\lceil \frac{1}{p} \cdot \sum_{i=1}^{n} c_i \right\rceil + 1 \right) \cdot (k+1) \tag{23}$$

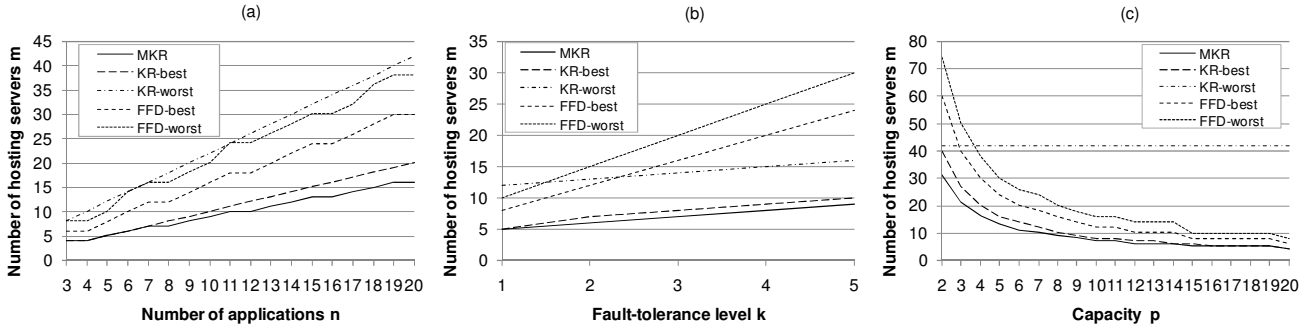

Fig. 8. Comparison results by varying (a) the number of applications n, (b) the required fault-tolerant level k, and (c) the capacity of hosting server p.

♦ k-redundancy method

The system without virtualization has to prepare k redundant servers to achieve k-fault-tolerance. This kind of redundant configuration technique is called as N+M redundant configuration. The k-redundancy method applies this approach to the virtual machine configurations. The required number of the hosting servers by the k-redundancy method is provided as m_{KR} as shown in Section V-C. The value of m_{KR} depends on the maximum value of c_i which varies within the following range.

$$\frac{1}{n}\sum_{i=1}^{n} c_i \le \max_i c_i \le \sum_{i=1}^{n} c_i - n + 1 \qquad (24)$$

Then the best and worst of the required number of hosting servers by k-redundancy method are given as follows.

$$m_{\mathrm{KR\text{-}best}} = \max\{\frac{1}{n}\sum_{i=1}^{n} c_i + k, \quad \left\lceil \frac{1}{p}\cdot\left(\sum_{i=1}^{n} c_i + k\cdot n\right)\right\rceil\} \qquad (25)$$

$$m_{\mathrm{KR\text{-}worst}} = \max\{\sum_{i=1}^{n} c_i - n + 1 + k, \quad \left\lceil \frac{1}{p}\cdot\left(\sum_{i=1}^{n} c_i + k\cdot n\right)\right\rceil\} \qquad (26)$$

♦ Multiple k-redundancy method

The multiple k-redundancy method improves the k-redundancy method to achieve k-fault-tolerance with lower number of the hosting servers. This method prepares the integral multiple of k of redundant virtual machines. The required number of hosting servers by multiple k-redundancy method is provided as m_{MKR} as shown in Section V-D.

A. Simulation Studies

The required number of hosting servers by the multiple k-redundancy method is evaluated under various input parameters with comparison to the FFD based approach and k-redundancy method.

First, in order to observe the effects of the change of the number of applications n, the required number of hosting servers are evaluated with the given parameters p=4,k=1, and $\overline{c_i} = \left\lceil \frac{1}{n}\cdot\sum_{i=1}^{n} c_i \right\rceil = 3$ (the average number of c_i). Comparison results shown in Fig. 8 (a) indicate the proposed multiple k-redundancy method is prior to any other methods independent to the value of n. The greater n, the more advantage the multiple k-redundancy method has. The main drawback of the k-redundancy method is the variation of the results depending on the combination of the number of application instances for each application class. The result marginally worse than the multiple k-redundancy method in the best case, however, in the worst case, the k-redundancy method requires more than twice number of the hosting servers compared with the multiple k-redundancy method.

Next, in order to observe the effects of the change of the required fault-tolerance level k, the required number of hosting servers are evaluated with the given parameters p=4, n=5, and $\overline{c_i} = 3$. Comparison results shown in Fig. 8 (b) indicate the multiple k-redundancy method achieves best performance in the given requirements k compared to other methods.

Furthermore, in order to observe the effects of the change of the capacity of each hosting server p, the required number of hosting servers are evaluated with the given parameters k=1, n=20, and $\overline{c_i} = 3$. Comparison results shown in Fig. 8 (c) indicate the multiple k-redundancy method is prior to any other methods independent to the value of p. According to the increase of the capacity p, the difference between the multiple k-redundancy method and the best case of the k-redundancy method become small. However, the worst case of the k-redundancy method does not improve even if the capacity increases.

VII. Related Work

Although there has been a lot of works on high-availability techniques for server clusters, the combination of high-availability techniques and virtualization is one of the emerging issues [17]. To make application systems running on virtual machines high-available, VMware provides a VMware HA (High Availability) [18]. In the event of a host server failure, VMware HA restarts virtual machines automatically on the other hosting server. Since the VMware HA is a reactive approach, a temporal service down or performance degradation is inevitable. Contrarily, a proactive approach based on the failure perdition of the hosting server has been proposed for Xen virtualization platform [19]. The method predicts a hosting server failure by resource monitoring, and evacuates virtual machines onto the different hosting server before the occurrence of any failures. Some failures are predictable by monitoring the status of server resources like CPU, memory, fan and disk logs. However, it is difficult to predict all failures by monitoring in advance. The proposed method is categorized as a proactive approach and has an original advantage that keeps the minimum

configuration of application service at any *k* hosting server failures. A simple redundant configuration method for virtual machines on multiple hosting servers is presented in [17]. In contrast, our study defines a redundant virtual machine placement problem as a combinatorial optimization problem and provides an optimum solution.

Dynamic resource allocation problem has been studied well in the cluster systems, grid computing and utility computing. Cluster Reserves presented a resource allocation mechanism for isolating the performance of clustered web services [20]. Cluster-On Demand presents an automated framework to manage resources in a shared hosting platform [21]. For the multi-tier web application systems, the dynamic resource provisioning method based on G/G/1 performance model has been proposed [22]. Since the existing resource allocation and provisioning methods mainly focus on the optimization of performance and resource utilization in systems, they do not take into account the fault-tolerance criteria. Few works incorporate the requirements for reliability and availability of application systems in the resource allocation algorithm [23][24]. However the virtual machine placement problem corresponding to host server failures is not formulated and optimum solution has been not discussed well. This paper defines a virtual machine placement problem with the condition of required fault-tolerance level and shows an algorithm to find an optimum solution.

VIII. CONCLUSION

This paper presents a method to make a redundant configuration of hosting server cluster for multiple applications using virtual machines. Consolidated server systems using server virtualization involves serious risks of host server failures that causes multiple downs of virtual machines. The proposed multiple *k*-redundancy method minimizes the number of required hosting servers to keep the minimum configuration for satisfying the performance requirements for all applications at any *k* host server failures. The method allocates integral multiple of *k* of redundant virtual machines to each application and places them into different hosting server using the simple placement algorithm. The advantage of the multiple *k*-redundancy method is shown through the experimental results of comparison with the conventional N+M redundant configuration approach and the FFD-based virtual machine placement approach. Consolidated server systems become more reliable with low cost by using the proposed multiple *k*-redundancy method.

REFERENCES

[1] VMware Infrastracture: http://www.vmware.com/products/vi/

[2] Citrix XenServer: http://citrix.com/English/ps2/products/product.asp?contentID=683148

[3] G. Khanna, K. Beaty, G. Kar, and A. Kochut, Application Performance Management in Virtualized Server Environments, In Proceedings of 10th IEEE/IFIP Network Operations and Management Symposium (NOMS2006), pp. 373-381, 2006.

[4] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, Power and Performance Management of Virtualized Computing Environments Via Lookahead Control, In Proceedings of the 5th IEEE International Conference on Autonomic Computing (ICAC2008), pp. 3-12, 2008.

[5] L. Hu, H. Jin, X. Liao, X. Xiong, and H. Liu, Magnet: A novel scheduling policy for power reduction in cluster with virtual machines, In Proceedings of the 2008 IEEE International Conference on Cluster Computing, pp. 13-22, 2008.

[6] Apache Module mod_proxy_balancer, http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html

[7] MySQL Cluster NDB 6.X/7.X Reference Guide, http://dev.mysql.com/doc/#cluster

[8] A. S. Tanenbaum, and M. van Steen, Distributed Systems Principles and Paradigms, Prentice Hall Inc., pp. 370-371, 2002.

[9] J. Cao, M. Andersson, C. Nyberg, and M. Kihl, Web Server Performance Modeling using an M/G/1/K*PS Queue, In Proceedings of the 10th International Conference on Telecommunications (ICT 2003), 2003.

[10] M. Andersson, J. Cao, M. Kihl, and C. Nyberg, Performance Modeling of an Apache Web Server with Bursty Arrival Traffic, In Proceedings of the International Conference on Internet Computing (IC), 2003.

[11] Z. Cai, Y. Chen, V. Kumar, D. Milojicic, and K. Schwan, Automated Availability Management Driven by Business Policies, In Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Management (IM2007), pp. 264-273, 2007.

[12] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, Dynamic Provisioning of Multi-tier Internet Applications, In Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC2005), pp. 217-228, 2005.

[13] G. Tesauro, N. Jong, R. Das, and M. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation, In Proceedings of the 3rd IEEE International Conference on Autonomic Computing (ICAC2006), pp. 65-73, 2006.

[14] G. Jungy, K. R. Joshiz, M. A. Hiltunen, R. D. Schlichting, and C. Pu, Generating Adaptation Policies for Multi-Tier Applications in Consolidated Server Environments, In Proceedings of the 5th IEEE International Conference on Autonomic Computing (ICAC2008), pp. 23-32, 2008.

[15] N. Bobroff, A. Kochut, and K Beaty, Dynamic Placement of Virtual Machines for Managing SLA Violations, In Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Management (IM2007), pp. 119-128, 2007.

[16] B. Korte, J. Vygen, Combinatorial Optimization: Theory and Algorithms, Japanese Edition 2005, Springer, 2005.

[17] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, Leveraging virtualization to optimize high-availability system configurations, IBM System Journal, Vol. 47, No. 4, 2008.

[18] VMware High Availability: http://www.vmware.com/files/pdf/ha_datasheet.pdf

[19] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, Proactive fault tolerance for HPC with Xen virtualization, In Proceedings of the 21st annual international conference on Supercomputing (ICS07), pp. 23-32, 2007.

[20] M. Aron, P. Druschel, and W. Zwaenepoel, Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers, In Proceedings of the ACM SIGMETRICS Conference 2000, pp. 90-101, 2000.

[21] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, Dynamic virtual clusters in a grid site manager, In Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), pp. 90-100, 2003.

[22] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal and T. Wood, Agile Dynamic Provisioning of Multi-tier Internet Applications, ACM Transactions on Autonomous and Adaptive Systems (TAAS), Volume 3(1), pp 1:1-1:39, March 2008.

[23] N. R. Gottumukkala, C. Leangsuksun, R. Nassar, and S. L. Scott, Reliability-Aware Resource Allocation in HPC Systems, In Proceedings of the IEEE International Conference on Cluster Computing 2007 (Cluster2007), pp. 312-321, 2007.

[24] F. Machida, and Y. Maeno, Availability-Constrained Virtual Machine Placement for HA Clusters, In the Fast Abstract of The 39th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN2009), 2009.