Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 66 (2006) 1442–1454
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Stochastic modeling and analysis of hybrid mobility in reconfigurable distributed virtual machines

Song Fu, Cheng-Zhong Xu*

Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202, USA

Received 31 July 2005; received in revised form 18 May 2006; accepted 18 May 2006
Available online 12 July 2006

Abstract

Virtualization provides a vehicle to manage the available resources and enhance their utilization in network computing. System dynamics requires virtual machines be distributed and reconfigurable. To construct reconfigurable distributed virtual machines, service migration moves the runtime services among physical servers when necessary. By incorporating the mobile agent technology, distributed virtual machines can improve their resource utilization and service availability significantly. This paper focuses on finding the optimal migration policies for service and agent migrations for high throughput in reconfigurable distributed virtual machines. We analyze three issues of this decision problem: migration candidate determination, migration timing and destination server selection. The service migration timing and destination server selection are formulated by two optimization models. We derive the optimal migration policy for distributed and heterogenous systems based on stochastic optimization theories. Renewal processes are applied to model the dynamics of migration. We solve the agent migration problem by dynamic programming and extend the optimal service migration decision by considering the interplay of the hybrid mobility. We verify the accuracy of our migration decision policy in simulations.

Keywords: Migration decision; Reconfigurable distributed virtual machine; Service migration; Agent migration; Stochastic optimization

1. Introduction

In the span of three decades, computers have become not only increasingly faster and cheaper, but also increasingly connected to other computers both within local area and across wide area networks. With large numbers of high-performance computers being connected to the Internet, resource management for service availability and security is becoming critical.

Resource virtualization provides a vehicle to manage the available resources and enhance their utilization. By virtualizing a system or component at a given abstraction level, its interface and visible resources are mapped onto the interface and resources of an underlying, possibly different, real system. While its roots trace back decades, resource virtualization has drawn renewed attention in recent years. Systems such as DVM [23] and Collective [22] develop techniques to construct a distributed virtual machine (VM) for networked computers or to migrate the state of a VM across a network. Besides, the widely used Java VM (JVM) middleware and PVM software package facilitate the construction of distributed and parallel applications. There are also recent efforts in the development of the agent-oriented resource brokerage infrastructure on JVM. Traveler [32] is such an example that supports application-level distributed virtual machines to execute agent tasks.

A distributed virtual machine can be viewed as a set of virtual servers running on top of multiple physical servers. It provides a powerful layer of abstraction in distributed and heterogeneous computing environments. It is certainly possible to deploy its components as static computing units. However, this abstraction layer would not be fully exploited unless it is instantiated and managed dynamically. Virtual machine migration is one approach to its dynamic reconfiguration. It moves parts of or the entire VM from one computer to another for system adaptivity. By this means, we can eliminate the performance bottleneck of a VM by increasing capacities of its overloaded components. Since VMs are usually created to provide services for programs running on top of them, we use the terms of VM migration and service migration interchangeable. Service

* Corresponding author.
E-mail addresses: song@eng.wayne.edu (S. Fu), czxu@eng.wayne.edu (C.-Z. Xu).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.05.006

migration techniques have been incorporated into many recently proposed systems. Examples include a VM-based architecture for grid computing in [7], *capsule* in Collective [22], M-DSA [11] in Traveler [32], virtual servers in [20] for Peer-to-Peer systems.

In addition to service migration, agent migration is another way to construct dynamic distributed systems [12]. An agent represents an autonomous computation that can migrate in a network to perform tasks on behalf of its creator. It is the autonomy of agents that makes high-performance grid computing different from cluster-wide process migrations. Multi-hop mobility allows agents to move computation across a wide-area network and negotiate with others. It provides a general approach to load-balancing, fault-tolerance, and data locality. MALD [4] is an example that uses mobile agents to balance the workload of distributed web servers. In [32,3,11], we proposed a resource brokerage infrastructure, which assigns task agents to a distributed shared array (DSA), an application-level distributed virtual machine. Load-balancing is achieved via the agent migration. Furthermore, the DSA virtual machine can be reconfigured reactively or proactively by transferring the DSA servers between physical servers.

In a reconfigurable distributed virtual machine system with the hybrid mobility of services and agents, migration decision becomes a crucial performance issue. It deals with three aspects of the problem. First is migration candidate determination, concerning which server should transfer a service and/or which agent should exercise a migration. Another one is migration timing, determining when a migration should be performed. Finally, a destination server should be selected so that the performance gain from a migration will not be outweighed by its overhead. Literature is full of heuristic algorithms for either service or agent migration and lacks formal analyses of this decision problem. This paper provides a stochastic optimization model to obtain the optimal migration policy for hybrid mobility.

The migration decision problem is non-trivial. A distributed VM provides a dynamic environment for the execution of agents. Each agent will carry out its tasks during its life span (corresponding to a *time domain*) and its execution may be performed on different virtual servers due to migrations (referring to this as a *space domain*). Therefore, we have to consider both domains when deriving the optimal migration decision. Their interaction makes the decision process more complicated. Secondly, the capacities of physical servers in a distributed VM may be different, which is typical in Internet-based computing. This capacity of heterogeneity results in different distributions of the agent's workload changes. Their distributions can even alter from one server to another during service migrations. Besides, the agent migration is hard to schedule in such a dynamic environment with server reconfiguration. The impact of agent migration on service migration also has to be carefully handled.

In this paper, we investigate the hybrid migration decision problem in bulk synchronous computation, by assuming the workload change of an agent is a random process with arbitrary probabilistic distributions. We formulate the migration issues as two stochastic optimization models, and obtain the optimal migration policy in both time and space domains. We derive the optimal phase for service migration with the objective of minimizing the migration frequency, and obtain the lower bound of the destination server capacity for an expected target gain value. Since the general stochastic optimization approaches tend to reveal asymptotic or stationary properties of a random process, renewal processes are applied to model the dynamics of migration because a migration operation may be invoked anytime over the course of computation. We solve the agent migration problem by dynamic programming and extend the optimal service migration decision by considering the interplay of the hybrid mobility. The simulation results verify the accuracy of our migration decisions. That is, the decision policy ensures the performance gain due to a migration will not be overwhelmed by its overhead and the overall migration frequency is minimal. It provides a formal guidance to perform hybrid migrations for load-balancing and is complementary to the existing migration mechanisms.

The rest of the paper is organized as follows. Section 2 introduces the system and workload evolution model of a reconfigurable distributed virtual machine. Sections 3 and 4 describe the optimal decisions of service/agent migration, and their interplay. Simulation results for a verification of our decision policy are presented in Section 5. Section 6 presents the related work and Section 7 concludes the paper with remarks on future work.

2. Reconfigurable distributed VM model

An agent-aware distributed VM is composed of a set of virtual servers spreading among distributed physical servers. Each virtual server can accommodate multiple agents to share resource and perform computation and communication. Each physical server can be run in two modes: *dedicated* or *multiprogrammed*. On a dedicated server, only virtual servers of the same application can reside, while a multiprogrammed server allows virtual servers of different applications to share its resource. On the other hand, a VM can be run on a single physical server or a cluster of servers. Correspondingly, on a *dedicated* and *multiprogrammed single-server* system, only service migration can be applied for its reconfigurability. In a *dedicated multi-server* VM, both service and agent migrations are useful to balance workload among multiple physical servers. The most general type is the *multiprogrammed multi-server* VMs. But the asynchrony among different applications makes multiprogrammed servers hard to analyze. In this paper, we confine our discussion to the dedicated multi-server reconfigurable distributed virtual machines.

We consider the bulk synchronous computation [26] as our execution model because of its popularity in scientific and engineering applications. Its computation proceeds in phases that are separated by the global synchronization. During each phase, agents perform calculations independently and then communicate new results with their data-dependent peers. Due to the need of synchronization between steps, the duration or execution time of a step is determined by the most heavily loaded

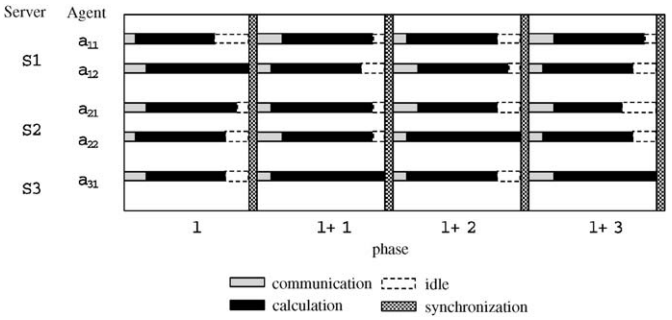

Fig. 1. Bulk synchronous computations of distributed virtual machine.

server, on which the agent having the longest completion time resides. Phase-wise computation may exhibit varying requirements as the computation proceeds and even a static bulk synchronous computation may have non-deterministic computational characteristics. Fig. 1 illustrates an example of bulk synchronous computations over a distributed VM with multiple agents on each server.

Consider a distributed application in a heterogeneous environment. A distributed virtual machine $V = \langle s_1, s_2, \ldots, s_N \rangle$, formed by N servers out of the available physical servers, is allocated to the application. Let a sequence $C = \langle c_1, c_2, \ldots, c_N \rangle$ denote their computational capacities and each c_i, $i = 1 \ldots N$, is a constant. The computation on the distributed virtual machine comprises multiple agents, each of which executes a number of tasks that collectively determine its workload. We use a dynamic agent distribution sequence $M = \langle m_1, m_2, \ldots, m_N \rangle$ to represent the number of agents assigned to different servers. Due to a many-to-one mapping between agents and servers, we use a sequence $A_i = \langle a_{i,1}, a_{i,2}, \ldots, a_{i,m_i} \rangle$ to denote the set of agents residing on server s_i in the distributed VM.

In the bulk synchronous computation, agents proceed in phases. Let $l = 0, 1, \ldots$, be the phase index of the computation and $w_{i,k}(l)$ denote the workload of agent $a_{i,k}$ on server s_i at phase l. Due to the heterogeneity of server capacities, we use scaled workload, $\tilde{w}_{i,k}(l)$, to denote the workload of agent $a_{i,k}$ normalized by its quota of server capacity. We assume the proportional share scheduling [27] among mobile agents of the same server. Multiple agents share the resource and each of them has an equal quota of the server capacity. That is, for server s_i with capacity c_i, a residing agent $a_{i,k}$ is assigned a portion of capacity as $c_{i,k} = c_i/m_i$. Therefore, we have $\tilde{w}_{i,k}(l) = w_{i,k}(l)/c_{i,k} = m_i w_{i,k}(l)/c_i$. For simplicity in notation, we will use $w_{i,k}(l)$ to denote the scaled workload henceforth.

As the computation proceeds, the scaled workload of each agent may evolve dynamically. Let $\delta_{i,k}(l-1)$ denote the net change of scaled workload $w_{i,k}(l)$ from phase $l-1$ to l due to the workload generation and/or consumption by agent $a_{i,k}$ during the period. The overall scaled workload of server s_i equals to the summation of its agents' scaled workload, i.e. $w_i(l) = \sum_{k=1}^{m_i} w_{i,k}(l)$. Similarly, its scaled workload change is defined as $\delta_i(l-1) = \sum_{k=1}^{m_i} \delta_{i,k}(l-1)$. Let sequences $w(l) = \langle w_1(l), w_2(l), \ldots, w_N(l) \rangle$ and $\delta(l-1) = \langle \delta_1(l-1), \delta_2(l-1), \ldots, \delta_N(l-1) \rangle$ denote the scaled workload distribution at phase l and the scaled workload change distribution from phase $l-1$ to l of the distributed virtual machine, respectively. Then the bulk synchronous computation can be characterized by an additive dynamic system

$$w(l) = w(l-1) + \delta(l-1), \tag{1}$$

where the scaled workload change is independent of the server workload. The scaled workload is described by a Markov chain, which is assumed in many other studies on bulk synchronous computations [19,9,33]. As a result of the agents' workload change between phases, the computation of a server becomes non-deterministic. Although this paper focuses on the additive scaled workload evolution model, our analysis can be extended to other dynamic models as discussed in [31].

During a service migration, the residing virtual servers on a physical server can be transferred as a whole. We treat the virtual servers on each physical server as a set and the two terms, physical and virtual servers will be used interchangeably. When a server decides to perform a service migration to another available server during the computation, the correspondent entry in the capacity sequence C will be replaced by the destination server's capacity. Similarly, agent migration leads to a modification of the agent distribution sequence M, as agents land on or leave from servers of the distributed virtual machine. So the migration decision problem can be tackled based on renewal processes. We use sequences $C^0 = \langle c_1^0, c_2^0, \ldots, c_N^0 \rangle$ and $M^0 = \langle m_1^0, m_2^0, \ldots, m_N^0 \rangle$ to represent the capacities and agent distribution on the *original servers*, i.e. the servers allocated to an application at its initiation. The scaled workload of an agent a_i at phase l and its scaled workload change from phase $l-1$ to l on its original server are denoted by $w_{i,k}^0(l)$ and $\delta_{i,k}^0(l-1)$, respectively. We assume initially agents have equal scaled workload at phase $l = 0$.

To make the migration decision problem tractable, we make some simplifying assumptions.

Assumption 1. The initial scaled workload changes of different agents on a server, $\delta_{i,k}^0(\cdot)$, are i.i.d. random variables and they are distribution-free. Agent autonomy makes the construction of independent agents easier by utilizing data locality and migration.

Assumption 2. Service/agent migration does not affect the probabilistic distribution of the scaled workload change of an agent. But it may have different mean after migration. This is reasonable because the scaled workload change equals to the amount of agent workload change relative to its capacity quota. This quota is a constant during the agent execution on a server with a fair-share scheduling scheme.

Many distributed applications execute in ways that satisfy these two assumptions. For example, simulations of protein folding in the Folding@Home project [8] steal idle CPU cycles of computers in the Internet. Folding@Home follows the SPMD computation model. Clients download and execute independent work sets, which have i.i.d. initial scaled workload changes. Migrating work sets among client computers does not change

the distributions of their scaled workload changes. Document search in large data centers, such as web search by google [17], is another example whose workload change patterns satisfy Assumptions 1 and 2.

3. Service migration decision

A distributed virtual machine is composed of a set of servers executing tasks of their residing agents and synchronizing with each other by performing barrier operations between phases. By the dynamic system in (1), it is expected that the servers' scaled workload will change with time and cause overloaded states. Since the duration of a phase is determined by the heavily loaded servers, the overall system performance may deteriorate in phase. Service migrations aim to eliminate the performance bottleneck of a distributed virtual machine by increasing capacities of the overloaded components. A service migration involves the tasks of transmitting service states and data along with the residing agents to a destination server with a higher capacity, continuing services and resuming agent execution there. Although we do not focus on the details of how to perform service migrations, it is clear that a migration operation would incur significant run-time overhead and the adaptive computation cannot afford frequent migrations.

The objective of service migration decision is to obtain the optimal timing in phase to minimize the migration frequency, and the lower bound of the destination server capacity to guide the server selection. Our migration policy is *optimal* in the following two ways. First, migration operations are performed only when the workloads of servers surpass the unbalance threshold so that the number of migrations is minimized. Second, the lower bound of destination server capacity ensures the performance improvement due to a service migration equals to a target gain apart from the migration overhead. In the following, we suppose there are only service migrations during the execution of a distributed virtual machine. The agent migration decision problem and its interplay with service migration will be tackled in Section 4.1.

3.1. Migration candidate determination and service migration timing

Although servers of a distributed virtual machine need to perform global barrier synchronization between phases, their executions within a phase can be assumed to be independent. This is particularly the case in agent-based systems, where autonomous agents perform their own tasks with local services and data sets. A service migration decision is made by a server based on its overall workload relative to its capacity. In the following, we use a set of simplified notations, in which s denotes any server in a distributed virtual machine, c and m represent its capacity and number of agents, $\langle a_1, a_2, \ldots, a_m\rangle$ is the set of its agents, $w_i(l)$ and $\delta_i(l)$ denote the scaled workload and workload change of a residing agent a_i at phase l.

For a server, its overall workload at a phase is the summation of all its agents' workloads, if we do not consider their overlapping. We define the *expected relative workload* of server s at phase l as

$$r(l) \;=\; E\left[\frac{\sum_{i=1}^{m} w_i(l)}{m}\right]. \tag{2}$$

The expected relative workload represents a server's overall workload, i.e. $\sum_{i=1}^{m} w_i(l)\cdot c/m$, w.r.t. its capacity, c. We use the scaled workload of each agent in order to ensure workload of different servers in a distributed VM are comparable with each other. For example, consider two servers s_1 and s_2 with the same capacity c. Server s_1 has one agent and server s_2 has five agents. Assuming all the agents have the equal workload w. On server s_1, since the agent is running exclusively, its quota is 100% of the capacity c. Its scaled workload is w/c. By contrast, on server s_2, each agent is assigned a quota of 20% of c. Their scaled workload becomes $5w/c$. According to (2), server s_2's expected relative workload is still five times as much as that of server s_1.

Most of the previous work designs a migration strategy to minimize the degree of overload. Due to the non-negligible run-time overhead in a migration, certain degree of overload must be tolerated. We reformulate the decision problem as "Given a bound of overload, find the minimal migration frequency". Therefore, the optimal migration timing is obtained by finding the maximal phase l^* for a given overload bound R, i.e.

$$r(l) \leqslant R. \tag{3}$$

R reflects the degree to which a server is considered to be overloaded. In general, the server capacity refers to the amount of tasks that can be performed by the server in a time unit. In the multi-phase computations we considered in this paper, a conservative definition of the time unit is the sequential execution time of a phase. Since a multi-phase application may have different execution times in different phases, strictly, the time unit should be the maximum of the execution time in different phases. Other definitions may be possible, but we require that the time unit is fixed and large enough to cover any phase. The workload of an agent changes with phases, and represents the number of tasks to be carried out in a phase. So, each agent's workload occupies a percentage of the server capacity. The overall workload of a server in a time unit is less than its capacity, i.e. $0 < R < 1$.

To find the maximal phase in (3), we need to calculate the expected value of the scaled workload $w_i(l)$ for each residing agent. But the heterogeneity in terms of different server capacities and alteration of an agent's scaled workload due to service migration, add complexity to its calculation. Next, we try to derive the expression of scaled workload $w_i(l)$ with reference to the value $w_i^0(l)$ on the original server.

Consider server s makes a service migration at certain phase k. Let s' denotes the server after migration, in the *space domain*. We use (c', m') to represent its capacity and number of agents, respectively. For a residing agent, say a_i, its scaled workload and workload change after migration are denoted by $(w_i'(\cdot), \delta_i'(\cdot))$. The *workload conservation property* states the workload of an agent remains unchanged at the moment of migration. That is $w_i'(k)c'/m' = w_i(k)c/m$. In the *time domain*,

with the additive scaled workload evolution model in Eq. (1), it is clear $w'_i(l) = w'_i(k) + \sum_{t=k}^{l-1} \delta'_i(t)$, for phases $l > k$. If there is no service migration at phase k, the scaled workload will become $w_i(l) = w_i(k) + \sum_{t=k}^{l-1} \delta_i(t)$, for $l > k$. By Assumption 2, we have $\delta'_i(t)c'/m' = \delta_i(t)c/m$. Thus, we derive $w'_i(l) = w_i(l)cm'/(c'm)$. Together with (c^0, m^0) for the original server and $m = m^0$ due to no agent migration, the scaled workload of agent a_i on server s can be calculated as

$$w_i(l) = \frac{w_i^0(l)}{\tilde{c}}, \tag{4}$$

where $\tilde{c} = c/c^0$ is called the *relative capacity*. We can see that the scaled workload of an agent equals to its initial scaled workload relative to the ratio of capacities between the current server and original one. This is caused by service migrations in the space domain.

Let μ_i be the means of the scaled workload changes $\delta_i^0(\cdot)$, for $i = 1 \ldots m$. According to the additive workload evolution model, the scaled workload of agent a_i on its original server at phase l is $w_i^0(l) = w_0 + \sum_{t=0}^{l-1} \delta_i^0(t)$, where $w_0 = w_i^0(0)$ is the scaled workload at phase 0, i.e. the initial scaled workload of an agent on its original server. Since the scaled workload changes $\delta_i^0(\cdot)$ in different phases are i.i.d. random variables, we have the expected value of scaled workload $E[w_i(l)] = E[w_i^0(l)]/\tilde{c} = (w_0 + l\mu_i)/\tilde{c}$. Thus, the expected relative workload of server s becomes

$$r(l) = \frac{\sum_{i=1}^{m} (w_0 + l\mu_i)}{m\tilde{c}} = \frac{w_0 + l\bar{\mu}}{\tilde{c}}, \tag{5}$$

where $\bar{\mu} = (\sum_{i=1}^{m} \mu_i)/m$ is the average change rate of the scaled workload. If $\bar{\mu} \leqslant 0$, then the overall scaled workload of a server tends to remain unchanged or decrease, as the computational phase proceeds. It means there is no need to perform service migrations any more. So, we only consider the cases with $\bar{\mu} > 0$ in our discussion. With $r(l) \leqslant R$, we have the following theorem to determine the optimal timing for service migrations.

Theorem 3.1. *For any server s with capacity c in a distributed virtual machine, suppose the scaled workload changes of different agents on the original server, $\delta_i^0(\cdot)$, $i = 1 \ldots m$, have means μ_i and their average $\bar{\mu} > 0$. For a given overload bound R, $0 < R < 1$, of the expected relative workload, the optimal service migration timing l^* in phase, under Assumption* 1 *and* 2 *is*

$$l^* = \frac{\tilde{c}R - w_0}{\bar{\mu}}. \tag{6}$$

Intuitively, Theorem 3.1 states that the optimal service migration timing of a server equals to its available server capacity, apart from the initial workload, divided by the average rate of workload increase. Eq. (6) reflects the influence from both the time domain (by the average mean of the scaled workload changes) and the space domain (by the relative server capacity $\tilde{c}$). It shows the timing is proportional to the relative server capacity. This is because both w_0 and each μ_i, are defined relative to the capacity of the original server. To reduce the service migration frequency, we can select the destination server with the highest capacity. But this greedy strategy may allocate more resource to less heavily loaded servers, which leads to resource imbalance in the distributed virtual machine. We will discuss this tradeoff in Section 3.2.

3.2. Destination server selection

Servers in a distributed virtual machine independently decide when to perform service migrations based on Theorem 3.1. At phases of the optimal service migration timing, their services and agents will be transmitted to the destination servers. As we have discussed in the previous subsection, the selection of a destination server is a tradeoff between the local and global performance optimizations. From an individual server's perspective, it is preferable to obtain an available server with the highest capacity as its destination so that its service migration frequency becomes minimal. But from the perspective of the entire distributed virtual machine, it would better allocate more powerful servers to more heavily loaded servers. In this section, we introduce a migration gain function and try to find the lower bound of capacity, that a prospective destination server must have, for a given target gain value.

For server s in a distributed VM, the execution time of its agent a_i at phase l, $t_i(l)$, is proportional to its workload relative to the capacity quota, i.e. its scaled workload, as $t_i(l) \propto w_i(l)$. The duration of a computational phase on a server is determined by its agent with the longest completion time, i.e. $t(l) = \max_{i=1\ldots m} t_i(l)$. Consider, server s makes a service migration to an available server, denoted by s', with capacity c' at the optimal migration timing l^* according to Theorem 3.1. We require $c' > c$, otherwise this migration is meaningless w.r.t. the system performance improvement. Thus, the execution time of each agent on server s' will be reduced. We define the *expected migration gain* for this service migration as

$$g(c') = E\left[\sum_{l=l^*}^{k-1} (t(l) - t'(l))\right] - E[O(w(l^*))], \tag{7}$$

where k is the prospective optimal migration timing for server s' with its supposed capacity c' according to Theorem 3.1. The execution time on server s' is $t'(l) \propto \max_{i=1\ldots m} w'_i(l)$, and $O(\cdot)$ returns the migration overhead for a given scaled workload. The expected migration gain function describes the performance improvement, in terms of agents' execution time on the two servers until the next service migration, compared with the migration overhead. We do not use the execution time of the entire distributed virtual machine as a metric to express performance gain, due to the unpredictable behaviors of other servers in the distributed VM and the independency of server execution. We expect the performance benefits in terms of a value G from a service migration. The target gain, G, is measured relative to the migration overhead. So, the selection of a destination server in a service migration is guided by finding the lower bound of the server capacity for a given non-negative target gain value G of the migration gain function, i.e.

$$g(c') \geqslant G. \tag{8}$$

A special case is to have $G = 0$, which means no benefit is yielded from a migration except to counteract the migration overhead. By increasing G, capacities of the destination servers will become higher accordingly. But to avoid resource allocation imbalance in the distributed virtual machine, we need to choose the value of G appropriately. Since the state information of services on a server is usually maintained by a relatively limited number of variables, the main portion of the service migration overhead is caused by transferring codes and data of services. The size of data set is likely to change little since its initial assignment to a server and the code of services are immutable. So, the migration overhead of a server can be represented by a constant, denoted by H. However, our optimization approach is still applicable with the general overhead function, $O(\cdot)$. Since the execution time of an agent, $t_i(l)$, is solely determined by its scaled workload $w_i(l)$, we will use these two terms interchangeably. Next, we transform the migration gain function to find the minimal capacity of the destination server for a given target gain value.

With the expressions of $t(l)$, $t'(l)$ and $w_i(l)$, the expected migration gain (7) becomes

$$g(c') = E\left[\sum_{l=l^*}^{k-1}\left(\max_{i=1\ldots m}\frac{w_i^0(l)}{\tilde{c}} - \max_{i=1\ldots m}\frac{w_i^0(l)}{\tilde{c}'}\right)\right] - H$$
$$= \left(\frac{1}{\tilde{c}} - \frac{1}{\tilde{c}'}\right)\sum_{l=l^*}^{k-1} E\left[\max_{i=1\ldots m} w_i^0(l)\right] - H, \quad (9)$$

where $\tilde{c}' = c'/c^0$. Let μ_i and σ_i^2 be the mean and variance of the scaled workload change $\delta_i^0(\cdot)$. Since $w_i^0(l) = w_0 + \sum_{t=0}^{l-1}\delta_i^0(t)$ and the phase-wise scaled workload changes of each agent $\delta_i^0(\cdot)$ are i.i.d. random variables w.r.t. phases, the central limit theorem of statistics [21] guarantees that as l gets large, the distribution of $w_i^0(l)$ tends to become normally distributed with mean $w_0 + l\mu_i$ and variance $l\sigma_i^2$. Because $\delta_i^0(\cdot)$, for $i = 1\ldots m$, are independent random variables, the scaled workloads $w_i^0(\cdot)$ are also independently distributed. So we have

$$Pr\left\{\max_{i=1\ldots m} w_i^0(l) \leqslant x\right\} = \prod_{i=1}^{m} Pr\{w_i^0(l) \leqslant x\}.$$

Since each $w_i^0(\cdot) \geqslant 0$, it is clear

$$E\left[\max_{i=1\ldots m} w_i^0(l)\right] = \int_0^\infty Pr\left\{\max_{i=1\ldots m} w_i^0(l) > x\right\} dx$$
$$= \int_0^\infty \left[1 - \prod_{i=1}^{m}\int_{-\infty}^{x} f_i(u)\,du\right] dx, \quad (10)$$

where $f_i(u)$ is the probability density function of normal distribution with parameters $w_0 + l\mu_i$ and $l\sigma_i^2$ for the scaled workload $w_i^0(l)$. Therefore, we can derive the lower bound of the destination server capacity, as shown in the following theorem.

Theorem 3.2. *For any server s with capacity c in a distributed virtual machine, suppose the scaled workload changes of different agents on the original server, $\delta_i^0(\cdot)$, $i = 1\ldots m$, have means μ_i and variances σ_i^2. For a given non-negative target gain G of the expected migration gain function, the minimal capacity that a prospective destination server must have during a service migration at the optimal phase l^*, under Assumptions* 1 *and* 2, *is*

$$c^* = \max\left\{c^+, c + \frac{\bar{\mu}}{R}c^0\right\}, \quad (11)$$

where R is the overload bound of the expected relative workload in (3) *and c^+ is the solution to equation*

$$\left(\frac{1}{\tilde{c}} - \frac{1}{\tilde{c}'}\right)\sum_{l=l^*}^{k-1}\left[\int_0^\infty\left(1 - \prod_{i=1}^{m}\int_{-\infty}^{x} f_i(u)\,du\right)dx\right] - H = G, \quad (12)$$

in which the optimal service migration phases $l^* = (\tilde{c}R - w_0)/\bar{\mu}$ and $k = (\tilde{c}'R - w_0)/\bar{\mu}$ by Theorem 3.1.

Proof. By applying $E[\max_{i=1\ldots m} w_i^0(l)]$ in Eq. (10) to the migration gain function, we get the left-hand side part of Eq. (12). And it is clear that the resulting gain function monotonously increases as its parameter c' becomes greater. So, for a given target gain value G, the minimal capacity satisfying $g(c') \geqslant G$ is the solution to equation $g(c') = G$, denoted by c^+.

At the same time, the prospective optimal migration phase k for the new server s' should be greater than phase l^* for server s. Otherwise, a new service migration will occur as soon as the previous one completes. We call this phenomena *idle migrations*. So, we have $k \geqslant l^* + 1$, i.e. $(R\tilde{c}' - w_0)/\bar{\mu} \geqslant (R\tilde{c} - w_0)/\bar{\mu} + 1$ and obtain $c' \geqslant c + \bar{\mu}c^0/R$. Therefore, the minimal capacity of a destination server takes the form in the theorem. □

The expected gain function is hard to calculate by Eq. (10). A special case is when the scaled workload changes $\delta_i^0(\cdot)$, for $i = 1\ldots m$ are i.i.d. random variables with the same mean and variance. This is reasonable for the SPMD applications, in which agents execute the same program on their own data sets and they have similar workload change behaviors. For these applications, the minimal destination server capacity can be determined by the following corollary.

Corollary 3.1. *For any server s with capacity c in a distributed virtual machine, suppose the scaled workload changes of different agents on the original server, $\delta_i^0(\cdot)$, $i = 1\ldots m$, have the same mean μ and variance σ^2. For a given non-negative target gain G of the expected migration gain function, the minimal capacity that a prospective destination server must have during a service migration at the optimal phase l^*, under Assumptions* 1 *and* 2, *is determined by Eq.* (11), *where c^+ is the*

solution to equation

$$\left(\frac{1}{\tilde{c}} - \frac{1}{\tilde{c}'}\right) \sum_{l=l^*}^{k-1} [w_0 + l\mu + \alpha(m)\sigma\sqrt{l}] - H = G, \qquad (13)$$

in which $\alpha(m) = (2 \ln m)^{1/2} - (\ln \ln m + \ln 4\pi)/[2(2 \ln m)^{1/2}] + \gamma/(2 \ln m)^{1/2}$ *and* γ *is Euler's constant* $(0.5772\cdots)$.

Proof. Since the scaled workload change $\delta_i^0(\cdot),\ i = 1 \ldots m$, have the same mean μ and variance σ^2, the scaled workload $w_i^0(l),\ i = 1 \ldots m$ are i.i.d. normally distributed random variables with the same mean $w_0 + l\mu$ and variance $l\sigma^2$. According to the extreme value theory [1], we have

$$E\left[\max_{i=1\ldots m} w_i^0(l)\right] \approx w_0 + l\mu + \alpha(m)\sigma\sqrt{l}. \qquad (14)$$

Therefore, the expected migration gain equals to the left-hand side of Eq. (13). Thus, the minimal destination server capacity is determined according to Theorem 3.2. □

We can see that the lower bound of the destination server capacity ensures the performance improvement due to a service migration equals to a given target gain in addition to the migration overhead. Since servers make their migration decisions independently, it is possible that two or more servers may decide to migrate to the same destination server according to Theorem 3.2. This phenomena is called *destination conflict* in service migrations. To solve the conflict and reduce migration frequency, we allow the server with the largest prospective next optimal migration timing, determined by Theorem 3.1, to take the destination server. Other servers will try to choose those from the remaining available servers with the least sufficient capacities. After selecting a destination server and completing service migration to it, we update the server sequence of the distributed VM and the capacity distribution sequence C. Since then, the renewal process begins a new round of distributed virtual machine computation and each server determines its next migration timing.

4. Hybrid migration decision

The properties of autonomy and multi-hop mobility make mobile agent as a suitable technique to achieve load balancing in distributed computing. Like service migration, there is also a decision problem for agent migration. But the dynamic environment with server reconfiguration and interaction among different agents greatly add to the difficulty of its decision. In this section, we model the agent migration decision problem by dynamic programming and extend the service migration decision to incorporate hybrid mobility.

4.1. Agent migration decision

To adapt to the change of server capacities, agents may need to migrate. The decision question is when to migrate and which server the agent should travel to. A locally optimal strategy can be derived if we assume individual agents make their migration decisions independently. But unlike service migration, different agents may migrate to the same server. The interaction among them may cause this migration strategy useless. So, certain global information about the agent or workload distribution is needed. If we consider the global optimization of agent migration, the decision problem is equivalent to the task scheduling in distributed computing. In general, it is an NP-problem to find the globally optimal task distribution. However, in the bulk synchronous computation model for SPMD, agents proceed their computation in phases and they carry out the same task at each phase. It is possible to find a feasible optimal solution to agent migration with some global information. We relate the coordinated decision problem of agent migration to the remapping problem in parallel computing and use dynamic programming to derive a globally optimal solution.

We define the *state* of a distributed VM as the workload distribution of different servers at certain phase. Assume the choice of agent migration while in state v incurs a cost $C(v, u)$, where u is a binary decision: migrate or not. $C(v, u)$ may be random. The decision process then passes into another state. The probability $p_{vq}(u)$ of passing into state q from v is dependent on the action chosen in state v. The expected total cost of a decision policy is the expected sum of the costs incurred at each decision step. An optimal decision policy minimizes the expected total cost.

Let $\mathcal{J}(v)$ be the expected total cost of a distributed VM which starts in state v, and which is governed by the control decisions. Then

$$\mathcal{J}(v) = \min_u \left\{ C(v, u) + \sum_{q \in I} p_{vq}(u) \cdot \mathcal{J}(q) \right\}, \qquad (15)$$

where I is the set of states that the distributed VM may enter from state v due to agent migrations. For any decision process state v, let $next(v)$ denote the set of states reachable from v in one phase and $travel(v)$ be the entry state of an agent migration. Each decision state $r = \langle r_1, \ldots, r_N, l + 1\rangle \in next(\langle v_1, \ldots, v_N, l\rangle)$ has a transition probability

$$Pr\{r|v\} = \prod_{i=1}^{N} Pr\{r_i|v_i\},$$

where $Pr\{r_i|v_i\}$ is the probability of chain i passing from state v into state r_i in one phase. The execution time of the distributed VM at state v is determined by the most heavily loaded server, i.e. $t(v) = \max_i \{v_i\}$. Each v_i equals to the scaled workload of the agent with the longest completion time. That is $v_i = \max_k \{w_{i,k}(l)\}$. Eq. (15) can be re-written as

$$\mathcal{J}(v) = \min_u \begin{cases} t(v) + \sum_{v' \in next(v)} Pr(v'|v)\mathcal{J}(v'), \\ t(v) + C(v, u) + \sum_{v' \in next(travel(v))} Pr(v'|v)\mathcal{J}(v'), \end{cases} \qquad (16)$$

where the bottom equation on the right-hand side is the cost function associated with agent migration, and the top equation is associated with no migration.

According to the theory of Markov decision processes, the optimal decision to make from state v is the decision which minimizes the right-hand side of Eq. (16). If the number of phases that an agent takes is some random variable with finite mean, then the system of equations given by (16) can be solved. The iteration algorithm proposed in [33] can be applied to solve the equation.

4.2. Interplay between service and agent migrations

With the introduction of agent migration, the set of residing agents on each server of a distributed virtual machine will not be immutable any more. An agent may land on or leave from a server at any phase. So the service migration decision problem is quite different from the discussion in Section 3.

An agent may decide to migrate to a server that will perform a service migration to another server at the same phase. So the information, e.g. server capacity, for the agent to make decision may be incorrect at the moment of its migration. To avoid this problem, we require service migrations are decided and performed first, and after their completion, agents decide whether to migrate or not with the new system information. Now the agent distribution sequence $M = \langle m_1, m_2, \ldots, m_N \rangle$ of a VM is no longer a constant. Instead, we treat it as a sequence of random variables whose distributions are determined by the dynamic agent migration strategy discussed in Section 4.1. At certain computational phase, the residing agents on server s are denoted by $\langle a_1, a_2, \ldots, a_m \rangle$. They may come from different servers via agent migrations. These agents are assigned to their original servers, represented by $\langle s_{(1)}, s_{(2)}, \ldots, s_{(m)} \rangle$, at phase $l = 0$. Entries in this sequence may be identical, which reflects some agents come from the same server. The optimal service migration timing for a server in Theorem 3.1 will be extended by considering the dynamic composition of its agent set. As we can see, it is quite complicated to make optimal decisions with both service and agent migrations for general applications. Here we consider a simplified case where agents on their original servers have the same capacity quota. It can be realized by the task scheduler at the system initiation. The optimal service migration timing with agent migration is described in the following theorem.

Theorem 4.1. *For any server s with capacity c in a distributed virtual machine, its m agents may migrate from different servers. Suppose these residing agents are, at the initial phase, have equal initial capacity quota q_0. Their scaled workload changes, $\delta_i^0(\cdot)$, $i = 1 \ldots m$, have the same mean μ. For a given overload bound R, $0 < R < 1$, of the expected relative workload, the optimal service migration timing l^* in phase, under Assumptions 1 and 2, is*

$$l^* = \frac{1}{\mu}\left(\frac{cR}{q_0 E[m]} - w_0\right). \tag{17}$$

Proof. For a residing agent a_i on server s, suppose it originally resides on server $s_{(i)}$, whose capacity and initial number of agents are $c_{(i)}^0$ and $m_{(i)}^0$, respectively. Its initial capacity quota $q_0 = c_{(i)}^0/m_{(i)}^0$. According to the previous discussion, its scaled workload at phase l is $w_i(l) = w_i^0(l)c_{(i)}^0 m/(cm_{(i)}^0) = w_i^0(l)q_0 m/c$. Since $w_i^0(l)$ tends to have a normal distribution with mean $w_0 + l\mu$, we have $E[w_i^0(l)q_0] = (w_0 + l\mu)q_0$. Similarly, the initial scaled workloads of other agents on server s are also normally distributed. The m random variables $w_1^0(l)q_0, \ldots, w_m^0(l)q_0$ are i.i.d. with the same mean $(w_0 + l\mu)q_0$. Since m is also a random variable, we have

$$\begin{aligned} r(l) &= \frac{1}{c} \cdot E\left[\sum_{i=1}^{m} w_i^0(l)q_0\right] = \frac{1}{c} \cdot E\left[E\left[\sum_{i=1}^{m} w_i^0(l)q_0 \,\middle|\, m\right]\right] \\ &= \frac{1}{c} \cdot E[mE[w_i^0(l)q_0]] = \frac{1}{c} \cdot E[m]E[w_i^0(l)q_0] \\ &= \frac{1}{c}\, q_0(w_0 + l\mu)E[m]. \end{aligned}$$

By applying the above equation to $r(l) \leqslant R$, the optimal service migration timing l^* of server s can be derived as in the theorem. □

According to Theorem 4.1, we can see the optimal service migration timing is reversely proportional to the expected value of agent number m. This can be explained that as the number of agents on a server increases, its overall workload tends to become greater and this leads to a reduced migration interval. Compared with Theorem 3.1, Eq. (17) is additionally related to the initial capacity quota of each agent due to agent and service migrations. The destination server selection policy in Section 3.2 can also be extended when incorporating agent migration.

5. Simulation results

To verify and analyze results of the migration decision according to our optimal model, we performed experiments to simulate the random processes of agent execution in a reconfigurable distributed VM with 30 agents on each server. Each data point of the simulation results was an average of 200 runs. The 95% confidence interval for each simulation result is also presented to demonstrate the robustness of the estimates. The distribution functions of the agent workload evolution characterize the workload dynamics in three of various applications.

Case 1: In the first experiment, we were to find and verify the optimal computational phase before a service migration according to Theorem 3.1 with the objective of minimizing the migration frequency. The agents on a server were divided into three groups with 10 agents in each one. They might be run in different phases and their initial scaled workload changes obeyed the following distribution functions respectively:

$$\delta_1^0(l) = \begin{cases} 0.1 & \text{w.p. } 0.50, \\ 0 & \text{w.p. } 0.50, \end{cases} \quad \delta_2^0(l) = \begin{cases} 0.1 & \text{w.p. } 0.50, \\ 0 & \text{w.p. } 0.25, \\ -0.1 & \text{w.p. } 0.25, \end{cases}$$

$$\delta_3^0(l) = \begin{cases} 0.1 & \text{w.p. } 0.25, \\ 0 & \text{w.p. } 0.50, \\ -0.1 & \text{w.p. } 0.25, \end{cases}$$

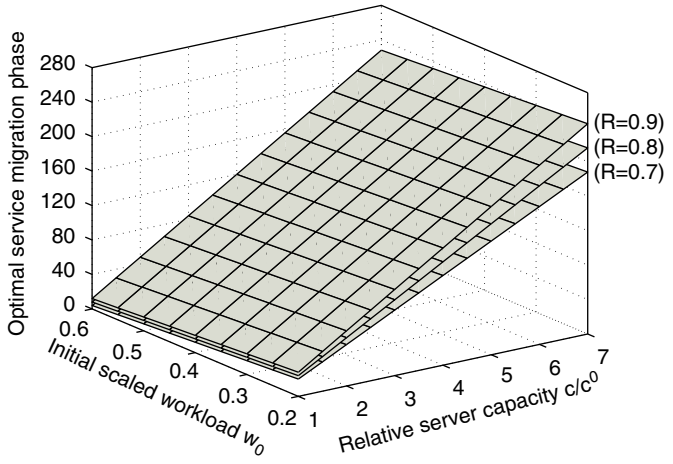

Fig. 2. Optimal service migration timing (numerical results).

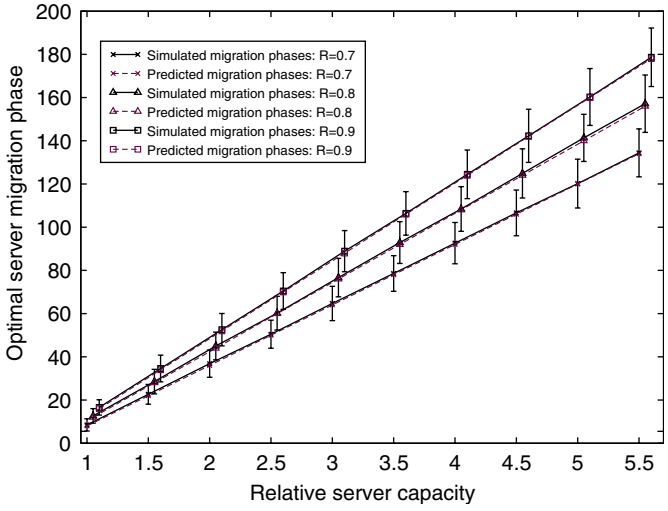

Fig. 3. Optimal service migration timing with $w_0 = 0.5$ (simulation results).

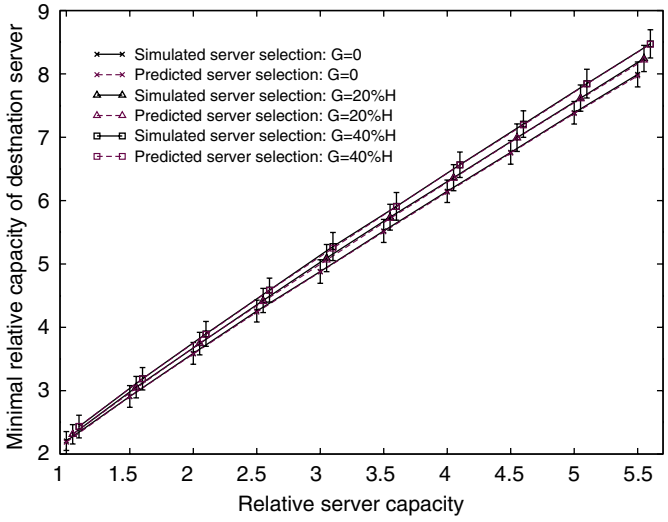

Fig. 4. Destination server selection.

where w.p. means "with probability". The distribution of scaled workload change $\delta_1^0(l)$ implies the total workload of an agent keeps increasing. This is valid in the early phase of some search algorithms, such as the branch-and-bound method [16]. The branching procedure recursively expands a (sub)problem into subproblems such that the size of subproblem set is on the steady increase. The second distribution $\delta_2^0(l)$ indicates the bounding procedure computes the expanded subproblems, which completes part of agent's workload. But the agent tends to generate new subproblems with higher probability as the computation proceeds. In the third phase $\delta_3^0(l)$, the rates of subproblem generation and completion tend to be equal so that the agent's expected workload remains unchanged for a certain period.

We varied the initial scaled workload w_0 from 0.2 to 0.6 and the capacity of each server relative to the original server of its residing agents, $\tilde{c}$, from 1 to 7. The corresponding optimal migration timing changed from the 4th to 244th phase as shown in Fig. 2. The bound of overload, R, indicates the degree to which certain overload must be tolerated by a server to avoid frequent migration. Its value should be set according to the runtime overhead of service migration. By varying the overload bound R from 0.7 to 0.9, we can see the migration timing increases. Even with $w_0 = 0.5$, $R = 0.7$ and $\tilde{c} = 4$, the optimal migration phase is the 100th, at which much computation has been performed. So service migration is a feasible approach to realize distributed virtual machine reconfiguration. Fig. 3 shows the migration phases simulated by random processes when the initial scaled workload w_0 is 0.5. To prevent the marks from overlapping, we move the curves corresponding to $R = 0.8$ and 0.9 to the right along x-axis by 0.05 unit. This does not affect the experiment results. We can see our migration decision provides an accurate lower bound in determining the service migration timing. The difference between simulated and predicted results tends to decrease as the overload bound R increases. The small confidence interval indicates the robustness of the estimates.

Case 2: The purpose of the second experiment is to verify the accuracy of the destination server selection by Theorem 3.2 for a service migration. The agent composition and distribution functions of scaled workload change were the same as those in Case 1. The initial scaled workload of agents and overload bound were set as $w_0 = 0.5$ and $R = 0.8$, respectively. The service migration overhead was $H = 50$, as 100 times greater than the initial scaled workload. It confirms to our measurements of the runtime overhead of service migration in M-DSA [11].

Fig. 4 plots the minimal capacity that a prospective destination server must have. We varied the capacity of a server from 1 to 5.5 relative to the original server and measured the minimal relative capacity of destination server. We can see the predicted value is either equal to or slightly smaller than the simulated result. This is because our migration decision finds the lower bound of the destination server capacity. The difference is caused by approximating the scaled workload of an agent by a normal distribution according to the central limit theorem. From the figure, we can also see that the minimum value of destination capacity increases at a sublinear rate. The target

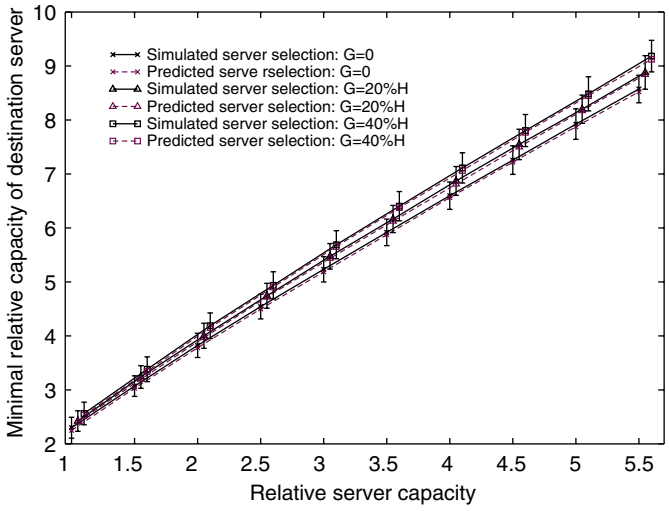

Fig. 5. Destination server selection with extreme value theory.

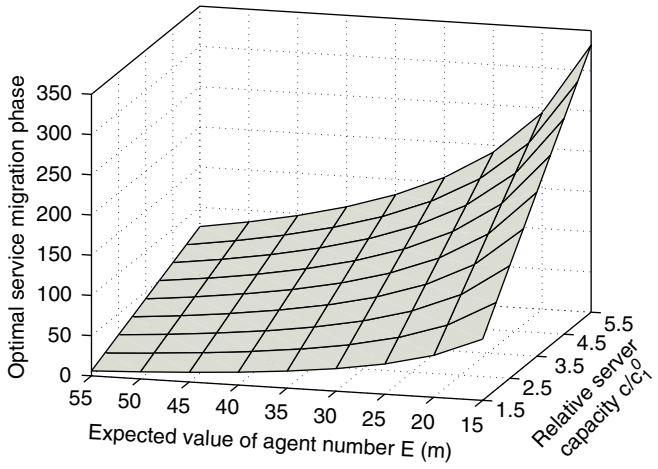

Fig. 6. Optimal service migration timing for hybrid mobility (numerical results).

gain value G represents the expected benefits from a service migration and it is measured relative to the migration overhead. As shown in the figure, if the target gain G is set to 20% of the migration overhead and the relative capacity of a server is 1, the minimal relative capacity of a destination server is about 2.3. Then during the next optimal migration timing, this service will be further migrated to an available server whose capacity should be at least 4.0 times greater than the original server of its residing agents. After migration, the process will be renewed and agent tasks will be executed until the next migration.

Case 3: In the third experiment, we repeated the second experiment, but assuming all agents of a server had the same distribution function of the scaled workload change as

$$\delta^0(l) = \begin{cases} 0.1 & \text{w.p. } 0.50, \\ 0 & \text{w.p. } 0.25, \\ -0.1 & \text{w.p. } 0.25. \end{cases}$$

According to Corollary 3.1, the problem of finding the lower bound of the destination server capacity can be approximated by using the extreme value theory.

Fig. 5 shows the accuracy of the prediction approximation. The results in solid line are due to simulation measurements. Compared with the predicted values in Case 2, Corollary 3.1 is a little less accurate in the destination server selection. It is because the extreme value theory provides an upper bound approximation of the expected value for maximal jointly distributed random variables. But, in the figure, we can see the predicted results are still very close to the simulated ones. The advantage of applying Corollary 3.1 is to simplify calculation of the expected migration gain function.

Case 4: The decision problem of agent migration has been modelled by dynamic programming in Section 4.1. It can be solved by an approach proposed in [33]. To analyze the interplay between agent and service migrations, we conducted the fourth experiment. We applied Theorem 4.1 to extend the optimal service migration decisions in Case 1 and conducted simulations to verify the results. To meet the premise of the theorem, we assigned 30 agents to each server and ensured each agent was allocated the same initial capacity quota. All other settings were equivalent to those in Case 3.

Fig. 6 plots the optimal service migration timing by incorporating agent migration. We varied the capacity ratio between a server and the original one of the residing agents. For each ratio value, we measured the optimal migration timing corresponding to different expectations of the agent number, $E[m]$. This expected value represents the dynamics of agent migration. From the figure, we can see the migration timing decreases as the expectation of agent number increases. An extreme was reached when $E[m] = 55$ and server capacity ratio was 1.5. Its migration phase was about 6. This was resulted from many agents landing on this server and the overall workload rapidly overwhelming its capacity. As computation proceeded, agents might decide to migrate to other light-loaded servers. Thus, the server's next migration interval became expanded accordingly. The expected value of agent number for each server can be determined dynamically by the history information of agent membership during the computation. Fig. 7 presents the simulation results. We measured the migration timing for the capacity ratio equal to 1.5, 3.0 and 4.5. We can see the difference between simulated and predicted phases tends to be small around $E[m] = 30$, which is the initial number of agents on each server.

6. Related work

Research interests in applying the classical virtual machines to grid computing are recently revived, with the Denali [29], DVM [23], and Terra [13]. However, these systems are not easily reconfigurable, due to the enormous execution contexts. In contrast, application-level distributed virtual machines can be tuned to become more efficient for specific applications, and it is much easier to migrate these lightweight services in a network. The Global Object Space in JESSICA2 [34], and the distributed shared array [3] in Traveler [32] are such examples.

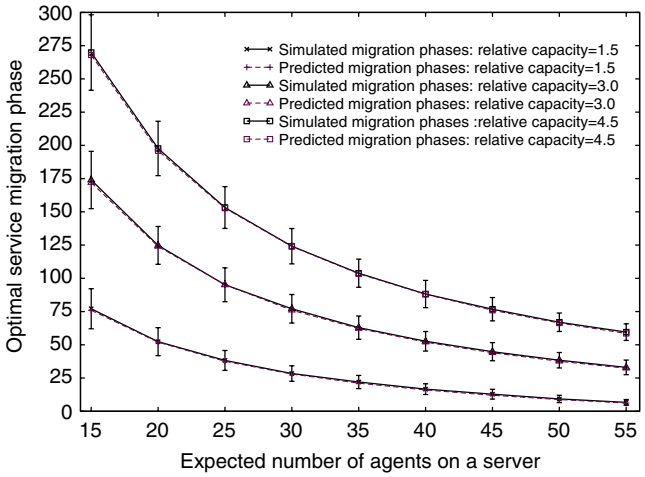

Fig. 7. Optimal service migration timing for hybrid mobility with $w_0 = 0.5$ and $R = 0.8$.

Figueiredo et al. [7] proposed an architecture for grid computing based on dynamic virtual machines. It allows a VM to be created on any computer that has sufficient resource to support it, and enables service migrations among computers to pursue resource locality. With the OS and middleware level support, SODA [15] constructs a distributed VM for an application service on demand. Each virtual service node provides runtime support for grid computation. Sapuntzakis et al. [22] tackled the problem of how to migrate OS-level VMs across the Internet. A virtual machine monitor is utilized to encapsulate the state of a running computer into a data structure, *capsule*. By serializing and transferring capsules between physical machines, a VM can resume running after migration. To support distributed virtual machine reconfiguration in Traveler [32], we designed a mobile distributed shared array (M-DSA) [11], which supports the migration of a DSA [3] service to a new physical server when the original one is overloaded or going to fail. Preliminary experimental results showed that service migrations at the application level with underlying middleware support could significantly reduce the migration overhead.

The distributed virtual machine mechanism and service migration technique have also been applied to peer-to-peer systems. Chord [24] was one of the first who used the notion of virtual servers to improve load balance. By allocating $\log N$ virtual servers on each real homogenous node, Chord ensures with high probability the number of objects per node is within a constant factor from optimal. CFS [5] accounts for node heterogeneity by ensuring the number of allocated virtual servers at each node is proportional to the node capacity. System adaptation is realized by deleting and creating virtual servers according to the load of each physical server. In [20] , Rao et al. presented three heuristic algorithms to achieve virtual server migration in structured peer-to-peer systems. Their simulation results indicated that virtual server migration was able to balance the system load to some extent even with a simple migration scheme.

By utilizing mobile agents, a load scheduling and resource management scheme for network-centric applications was proposed in Traveler. In that scheme, mobile agents can autonomously roam the network to find virtual servers with appropriate capacities to run on. A different approach was introduced in MALD [4], where web servers dispatch mobile agents to retrieve the system load information and redistribute load on all servers. In the grid environment, a similar approach was incorporated into the GrADS [14] system, where applications are migrated to manage grid dynamics [25]. To leverage the existing applications, legacy code migration has been investigated in some systems. For example, HPCM [6] provides middleware support to migrate pre-processed legacy code in heterogenous environments.

Although all of the works dealt with reconfigurable VMs, they focused on how to realize service and agent migrations by providing different mechanisms. In this paper, we concentrate on the migration decision in support of hybrid mobility. Although there are many reports in literature on mobile agent systems (see [28] for a review), there are few on agent scheduling for mobility. Moizumi and Cybenko [18] formulated a travelling agent problem to find optimal agent migration sequence which minimizes the expected completion time of a task with unreliable networks and unpredictable nodal services. In [35], Zhuang et al. proposed an approach to optimize the number of agent migrations for a given task by utilizing compiler optimization techniques. They both focused on finding the optimal agent itinerary to complete its tasks. However, we are more interested in the decision problem as when and to which server an agent migration will be exercised. The objective is to balance system workload in a dynamic distributed environment supporting server reconfiguration.

In this paper, we relate the service/agent migration timing in reconfigurable distributed VMs to a dynamic remapping scheduling problem in parallel computing. Dynamic remapping is to redistribute the workload of processors at runtime for efficient execution of dynamic applications [30]. Most recently, we studied the periodic remapping policy in [9], analyzing the impact of remapping frequencies. We formulated the problem as the sequential stochastic optimization and derived the optimal remapping frequencies with a priori known statistical behavior of the workload. We further modelled the optimal remapping problem as a vector-valued Markov chain and formulated it as a binary decision process in [33]. The optimal strategy was developed by employing the optimal stopping rules in stochastic control. This approach can only be applied to small systems, because computation of the transition matrix increases dramatically as the size of a system becomes larger. Both approaches only considered the time domain in deriving the optimal solutions. The optimization approaches are based on a global knowledge of the system workload distributions, which is almost impossible to obtain in wide-area distributed computing.

7. Conclusions

In this paper, we have studied the decision problem of service and agent migrations for a reconfigurable distributed virtual machine. The migration timing and destination server selection of service migration are formulated as stochastic optimization

models. We have derived the optimal service migration policy in both time and space domains for a given overload bound and a target gain value. The optimal service migration timing of a server equals to its available server capacity, apart from the initial workload, divided by the average rate of workload increase. It reflects the influence from both the time and space domain. It shows the timing is proportional to the relative server capacity. The lower bound of the destination server capacity ensures the performance improvement due to a service migration equals to a given target gain in addition to the migration overhead. We have solved the agent migration decision problem by dynamic programming and analyzed the interplay of the two migration problems.

We note that although the assumption of scaled workload changes remaining the same distributions during migration was assumed in literature, determination of a distribution is non-trivial. Armstrong et al. [2] experimented with the NAS benchmarks on different machines to determine the distribution types of the execution time. They found that running the same job on different machines led to different execution time means and different distribution types. This poses a new challenge to the migration decision problem. The present solution to the destination conflict problem is still preliminary. New collaborative approach will be developed for the globally optimal decision. To apply our migration policy to online decision, the workload change distribution needs to be calculated based on the past workloads of agents. We have being proposing a stochastic learning framework to obtain this distribution from the existing application performance data. With it, we can make the optimal migration decisions at runtime based on our migration policy.

Acknowledgments

This research was supported in part by US NSF Grants ACI-0203592, CCF-0611750, and NASA Grant 03-OBPR-01-0049. The authors also thank Dr. Kai Hwang and anonymous reviewers for their constructive comments and suggestions. An early version of this work was presented in the Proceedings of ICPP'04 [10].

References

[1] A.H.-S. Ang, W.H. Tang, Probability Concepts in Engineering Plannig and Design, vol. II. Rainbow Bridge, 1984.

[2] R. Armstrong, D. Hensgen, T. Kidd, The relative performance of various mappping algorithms in independent of sizable variances in run-time predications, in: Proceedings of the Seventh Heterogeneous Computing Workshop, 1998.

[3] R. Basharahil, B. Wims, C.-Z. Xu, S. Fu, Distributed shared array: an integration of message passing and multithreading on SMP clusters, J. Supercomput. 31 (2) (2004) 161–184.

[4] J. Cao, Y. Sun, X. Wang, S.K. Das, Scalable load balancing on distributed web servers using mobile agents, J. Parallel Distrib. Comput. 63 (10) (2003) 996–1005.

[5] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, Wide-area cooperative storage with CFS, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Chateau Lake Louise, Banff, Canada, October 2001.

[6] C. Du, X.-H. Sun, K. Chanchio, HPCM: a pre-compiler aided middleware for the mobility of legacy code, in: Proceedings of the International Conference on Cluster Computing, 2003.

[7] R. Figueiredo, P. Dinda, J. Fortes, A case for grid computing on virtual machines, in: Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS), May 2003.

[8] Folding@home: to understand protein folding, misfolding, and related diseases. Available: ⟨http://folding.stanford.edu/⟩, 2000.

[9] N.-T. Fong, C.-Z. Xu, L. Wang, Optimal periodic remapping of bulk synchronous computations on multiprogrammed distributed systems, J. Parallel Distrib. Comput. 63 (11) (2003) 1036–1049 Preliminary version appeared in Proceedings of the 14th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2000, pp. 103–108..

[10] S. Fu, C.-Z. Xu, Migration decision for hybrid mobility in reconfigurable distributed virtual machines, in: Proceedings of the 33rd International Conference on Parallel Processing (ICPP), August 2004.

[11] S. Fu, C.-Z. Xu, Service migration in distributed virtual machines for adaptive grid computing, in: Proceedings of the 34th International Conference on Parallel Processing (ICPP), June 2005.

[12] A. Fuggetta, G.P. Picco, G. Vigna, Understanding code mobility, IEEE Trans. Software Eng. 24 (5) (1998) 342–361.

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, D. Boneh, Terra: a virtual machine-based platform for trusted computing, in: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), October 2003.

[14] GrADS project homepage. ⟨http://nhse2.cs.rice.edu/grads⟩.

[15] X. Jiang, D.-Y. Xu, SODA: a service-on-demand architecture for application service hosting utility platforms, in: Proceedings of the International Symposium on High Performance Distributed Computing, 2003.

[16] E.L. Lawler, D.E. Wood, Branch and bound methods, a survey, Oper. Res. 14 (1966) 699–719.

[17] J.D. Luiz Barroso, U. Hoelzle, Web search for a planet: the google cluster architecture, IEEE Micro. 23 (2) (2003) 22–28.

[18] K. Moizumi, G. Cybenko, The travelling agent problem, Math. Control, Signals Systems 14 (3) (2001) 213–232.

[19] D.M. Nicol, J.H. Saltz, Dynamic remapping of parallel computations with varying resource demands, IEEE Trans. Comput. 37 (9) (1988) 1073–1087.

[20] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, I. Stoica, Load balancing in structured p2p systems, in: Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), February 2003.

[21] S.M. Ross, Introduction to Probability Models, sixth ed., Academic Press, New York, 1997.

[22] C.P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M.S. Lam, M. Rosenblum, Optimizing the migration of virtual computers, in: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI), December 2002.

[23] E.G. Sirer, R. Grimm, A.J. Gregory, B.N. Bershad, Design and implementation of a distributed virtual machine for networked computers, in: 17th ACM Symposium on Operating Systems Principles (SOSP), December 1999, pp. 202–216.

[24] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of the ACM SIGCOMM '01 Conference, San Diego, California, August 2001.

[25] S. Vadhiyar, J. Dongarra, A performance oriented migration framework for the grid, in: Proceedings of the Third International Symposium on Cluster Computing and Grid (CCGrid), May 2003.

[26] L.G. Valiant, A bridging model for parallel computation, Commun. ACM 33 (8) (1990) 103–111.

[27] C.A. Waldspurger, W.E. Weihl, Lottery scheduling: flexible proportional-share resource management, in: Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI), November 1994, pp. 1–11.

[28] D. Wang, N. Paciorek, D. Moore, Java-based mobile agents, CACM 42 (3) (1999) 92–102.

[29] A. Whitaker, M. Shaw, S. Gribble, Denali: lightweight virtual machines for distributed and networked applications, in: Proceedings of the USENIX Technical Conference, Monterey, CA, June 2002.
[30] C.-Z. Xu, F.C. Lau, Load Balancing in Parallel Computers: Theory and Practice, Kluwer Academic Publishers, Dordrecht, 1997.
[31] C.-Z. Xu, L. Wang, N.-T. Fong, Stochastic predication of execution time for dynamic bulk synchronous computation, J. Supercomput. 21 (1) (2002) 91–103 Preliminary version appeared in Proceedings of the 15th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2001, pp. 44–49..
[32] C.-Z. Xu, B. Wims, A mobile agent based push methodology for global parallel computing, Concurrency: Practice Experience 14 (8) (2000) 705–726.
[33] G. Yin, C.-Z. Xu, L. Wang, Optimal remapping in dynamic bulk synchronous computations via a stochastic control approach, IEEE Trans. Parallel Distrib. Systems 14 (1) (2003) 51–62 Preliminary version appeared in Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2002, pp. 50–57.
[34] W. Zhu, C.-L. Wang, F.C.M. Lau, JESSICA2: a distributed java virtual machine with transparentthread migration support, in: Proceedings of the Fourth International Conference on Cluster Computing (CLUSTER 2002), September 2002.
[35] X. Zhuang, S. Pande, Compiler scheduling of mobile agents for minimizing overheads, in: Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS), May 2003.

Song Fu received the B.S. degree in computer science from Nanjing University of Aeronautics and Astronautics, China, in 1999, and the M.S. degree in computer science from Nanjing University, China, in 2002. He is currently a Ph.D. candidate in computer engineering at Wayne State University. His research interests include the resource management, security, and mobility issues in wide-area distributed systems.

Cheng-Zhong Xu received the B.S. and M.S. degrees in computer science from Nanjing University in 1986 and 1989, respectively, and the Ph.D. degree in computer science from the University of Hong Kong in 1993. He is an Associate Professor in the Department of Electrical and Computer Engineer of Wayne State University. His research interests lie in distributed are distributed and parallel systems, particularly in resource management for high performance cluster and grid computing and scalable and secure Internet services. He has published more than100 peer-reviewed articles in journals and conference proceedings in these areas. He is the author of the book *Scalable and Secure Internet Services and Architecture* (CRC Press, 2005) and a co-author of the book *Load Balancing in Parallel Computers: Theory and Practice* (Kluwer Academic, 1997). He serves on the editorial boards of J. of Parallel and Distributed Computing, J. of Parallel, Emergent, and Distributed Systems, J. of High Performance Computing and Networking, and J. of Computers and Applications. He was the founding program co-chair of International Workshop on Security in Systems and Networks (SSN), the general co-chair of the IFIP 2006 International Conference on Embedded and Ubiquitous Computing (EUC'06), and a member of the program committees of numerous conferences. His research was supported in part by the US National Science Foundation, NASA, and Cray Research. He is a recipient of the Faculty Research Award of Wayne State University in 2000, the President's Award for Excellence in Teaching in 2002, and the Career Development Chair Award in 2003. He is a senior member of the IEEE.