CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **22**:1945–1967
Published online 30 March 2010 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1568

Improving grid performance by dynamically deploying applications[‡]

Yongwei Wu[*, †], Gang Chen, Jia Liu, Rui Fang, Guangwen Yang and Weimin Zheng

Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China

SUMMARY

Grid applications are normally deployed on computing nodes beforehand, which may cause the undesirable situation that some of these nodes (with hot applications deployed) are always busy whereas the others are consistently idle. Therefore, the overall performance (e.g. throughput and load balancing) of such a Grid system would be seriously degraded. In this paper, we present the idea of Hierarchical and Dynamic Deployment of Application (HDDA) in Grid to improve the system performance. With HDDA, an application can be dynamically deployed and undeployed when necessary. In order to reduce the overhead caused by HDDA, the Average Latency Ratio Minimum (ALR-MIN) replacement strategy is also proposed. It deploys applications to nodes with minimum ALR of Node (NALR), and evicts applications with minimum increment of ALR. The results of the experiment we conducted on ChinaGrid show that HDDA can achieve 10 and 24% less average complete time (ACT) than the schemes of non-HDDA and Static Deployment of Application (SDA), respectively. Additionally, throughput and load balancing of HDDA are also better than the other two schemas. Results of the simulation performed on a simulator particularly developed for this research show that our ALR-MIN replacement strategy results in 17% less relative delay-time of jobs than the well-known Least Recently Used (LRU)-based strategies in a typical setting. Copyright © 2010 John Wiley & Sons, Ltd.

Received 1 December 2009; Accepted 5 January 2010

*Correspondence to: Yongwei Wu, Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China.
†E-mail: wuyw@tsinghua.edu.cn
‡Part of this paper has been published in the proceeding of ScalCom 2009.

Contract/grant sponsor: ChinaGrid project of Ministry of Education of China
Contract/grant sponsor: Natural Science Foundation of China; contract/grant numbers: 90412006, 90412011, 60573110, 90612016, 60673152
Contract/grant sponsor: National Key Basic Research Project of China; contract/grant numbers: 2004CB318000, 2003CB317007
Contract/grant sponsor: National High Technology Development Program of China; contract/grant numbers: 2006AA01A108, 2006AA01A111, 2006AA01A101

KEY WORDS: grid computing; resource management; dynamic deployment; replacement strategy; system performance

1. INTRODUCTION

Grid computing [1] is to build a computational infrastructure through resource sharing and coordinating among Virtual Organization (VO) participants. Many applications (e.g. services and software) and resources (e.g. computer nodes and storage devices) have already been integrated into various grid projects such as TeraGrid, EGEE, NorduGrid, and ChinaGrid [2]. Normally, an application is statically deployed to a pre-selected subset of computing nodes. This kind of strategy is referred to as Static Deployment of Application (SDA). Nodes with highly demanded applications may receive requests over their capability to handle, in which case it is very easy to encounter the situation that some nodes are always busy, whereas the others are unoccupied. Therefore, the overall performance (e.g. throughput and load balancing) of a grid with the SDA strategy would be seriously degraded. If highly demanded applications could be dynamically deployed on idle nodes, the system's performance would be greatly improved.

Many existing studies (e.g. [3–6]), regarding dynamic application deployment in grids, did not put their focus on improving overall system performance with respect to throughput and load balancing. The approaches proposed in [3–5] deploy Web Services to containers dynamically, whereas an on-demand deployment of scientific applications is implemented in [6] for the purpose of reducing the workload of administrators. DynaGrid [7] and DAG-Condor [8] are two of the studies that have exploited dynamic deployment for the purpose of improving overall system performance. However, DynaGrid [7] cannot undeploy older services when the capacity limitation of nodes is reached. DAG-Condor [8] does not aim to handle multi-domain grids; it only handles the file reuse of workflow scheduling within a single domain. The Application Contents Service Working Group (ACS-WG) in the Open Grid Forum has proposed a specification [9] to standardize application management in a grid environment. It is potentially possible to combine the specifications from ACS-WG and the specifications (e.g. CDDLM-FND, CDDLM-SF, and CDDLM-CDL) from the CDDLM-WG to implement dynamic deployment of applications. However, these specifications do not take the improvement of the system performance into account.

In this paper, we present a schema, named as Hierarchical and Dynamic Deployment of Application (HDDA), to improve the overall performance of a multi-domain grid. With HDDA in use, most applications in scientific research (e.g. bioinformatics and meteorology) can be easily transferred and dynamically deployed, when no instance (i.e. an installed copy of services or software) of the applications can be found on all currently available nodes. In addition, applications with less demand can be undeployed or replaced, when the deployment limitation of all available nodes is reached. In such case, it is unlikely for the scheduler of a grid system to be failed to map jobs to available computing nodes of the grid. Besides, the completion time of jobs can be shortened and the utilization of the nodes cannot be affected by the access frequencies of applications deployed on the nodes and therefore the workloads of the nodes can be well balanced (less unfairness). Furthermore, by placing a local Application Repository in each domain (Section 3.1), a large amount of time can be saved in terms of transferring application packages over networks.

An application replacement strategy is required to realize the HDDA schema and the following aspects influence the design of such an application replacement strategy. (1) The overhead of dynamic deployment of applications. It is perhaps too significant so that the overall performance of the system might be seriously degraded. Especially in the case of applications that have big size, the overhead of dynamic deployment may be too significant to be acceptable. For example, Blast, the well-known bioinformatics application, always needs a protein or nucleotide database whose size is normally several million bytes; therefore it, takes long time to transfer and install the database and software packages of Blast. (2) The capacity of a computing node. It is usually limited and it is simply impossible to keep all applications in a local system consistently. Some applications have to be replaced to make room for a new application and therefore applications may be deployed or undeployed repeatedly.

Most of existing studies (e.g. [3–6,10,11], addressing the problem of dynamic application deployment in grid, do not focus on application replacement strategies. DAG-Condor [8], however, exploits a Least Recently/Frequently Used (LRFU)-based replacement strategy, which does not address the multi-cache problem: a single catch has to be selected before a cache object is placed or evicted. In this paper, we propose a replacement strategy for HDDA, named as ALR-MIN, to reduce the overhead of dynamic deployment. With this strategy, the node on which a new application is about to be deployed and from which the existing applications are about to be evicted are carefully chosen to reduce the Average Latency Ratio (ALR). The strategy contains a set of evaluation functions to predict and compare ALR increment. The node with minimum predicted ALR is chosen to hold a new application and the old applications with minimum predicted ALR on the same node may be selectively swapped out to make room for the new application.

In summary, the following contributions are achieved in this paper:

- A schema for improving system performance by HDDA is proposed in this paper. HDDA has been implemented in ChinaGrid. With this schema, applications can be dynamically deployed to computing nodes, only when no instance of these applications can be found on available nodes. Owing to resource capacity limitation, less-worthy applications may be replaced to release resource for a newly requested application.
- The application replacement problem in HDDA is formalized and two ALR-MIN replacement strategies are proposed to reduce the ALR for both heavy workload and light workload.
- An experiment has been conducted on ChinaGrid to evaluate HDDA by comparing it with the non-HDDA and SDA approaches in terms of system throughput, average completion time of jobs, and load balancing. The experiment results demonstrate that the overall system performance of HDDA outperforms those of non-HDDA and SDA.
- The ALR-MIN strategy is evaluated on a simulator and the simulation results show that with the ALR-MIN strategy jobs take the least relative delay-time to complete, compared with the other two LRU-based strategies.

The remainder of the paper is organized as follows. The related work is discussed in Section 2. In Section 3, we describe the HDDA schema, followed by the formalization of the application replacement problem in the context of dynamic deployment (Section 4). Section 5 presents two ALR-MIN strategies for heavy load and light load systems, respectively. The experiment conducted to evaluate HDDA is discussed in Section 6. Section 7 examines the performance of the ALR-MIN compared with two traditional LRU-based strategies. Last, we draw a conclusion in Section 8.

2. RELATED WORK

We discuss the related work from the following two aspects: dynamic deployment of applications and replacement strategies.

2.1. Dynamic deployment

HAND [3] supports dynamic deployment of services in GT4. Pu and Lewis [4] proposed an approach to deploy uniform dynamic service code onto three Web Services containers and two grid services containers. Watson *et al.* described in [5] the Dynasoar project, an infrastructure for dynamically deploying Web Services over a grid or Internet. Some other studies (e.g. GLARE [6]) propose approaches to install and deploy scientific applications with less involvement of administrators. Projects like CGSP2 [2] support traditional legacy software tools to be packaged and wrapped as Web Services so that they can be manually transferred to and invoked on other nodes without reinstallation. Though all these studies more or less address dynamic deployment of applications from different perspectives, none of them have taken the improvement of overall system performance into account.

DynaGrid [7] has a dynamic service deployment mechanism and a service resource migration mechanism for WSRF-compliant applications. These two mechanisms do improve the performance and utilization of a system; however, they cannot facilitate the undeployment of older services to release resource when a capacity limitation is reached.

Condor [12] can transfer input and executing files to an execution machine before a job begins. The approach does not cache these input and executing files but deletes them immediately. This causes excessive overhead of job preparation for a grid system. DAG-Condor [8] exploits a LRFU-based data caching policy to reduce response time; however it considers file reuse in the workflow scheduling within an individual domain, instead of in multi-domains.

2.2. Replacement strategy

A LRFU-based replacement strategy is exploited in DAG-Condor [8] to keep useful files in disk. However, this strategy is not good enough since it does not address the multi-cache problem.

Local cache replacement strategies have been investigated for a long time, especially in virtual storage. Traditionally and frequently used schemes include ARC [13], FIFO, LRU, LFU, LRU-2, 2Q, LIRS, FBR, and MQ. The main drawback of these strategies is that they cannot deal with the case where sizes and miss costs of cache objects are non-uniform. Non-uniform-cost local replacement has also been addressed by some strategies [14], such as BCL, DCL, and ACL proposed by Jaeheon, and Lowest-latency-first [15]. Their main drawback is that two cache objects with the same miss cost but different sizes are treated equally. For applications in this paper, the miss cost of an application is not always proportional to its size. The miss cost of an application is sometimes determined by not only its size but also its deployment or installation time cost. Hence, with same miss cost, applications with bigger size should be swapped out first and thus more space can be freed for new applications.

Other non-uniform-cost schemes (e.g. LRU-Threshold [16]) neglect the fetch cost of a block. GreedDual-Size [17] is only helpful in single cache space. Some other studies (e.g. [18,19]) exploit cooperative strategy to solve multi-cache problems. However, they do not address the problem of where to put the requested data. Data replication strategies (e.g. [20,21]) in P2P Networks are similar to the application replication in our work. However, our study has a different optimization objective: decreasing miss rate.

3. DYNAMIC DEPLOYMENT OF APPLICATIONS

In this section, we discuss and compare the architectures of non-HDDA and HDDA first (Section 3.1). Then, the detailed description of how HDDA dynamically deploying applications within a domain is presented in Section 3.2.

3.1. Non-HDDA and HDDA

To compare with SDA, dynamic deployment of applications is more plausible to balance load in a distributed computing system. For a multi-domain grid, the layout of a dynamic deployment architecture can be as simple as the one shown in Figure 1(a). This architecture has a central Application Repository for all the computing nodes of all the domains to obtain applications from it when necessary and this architecture is referred to as non-Hierarchically Dynamic Deployment of Applications (non-HDDA). However, a grid is a computing system that is composed of domains distributed around national or international areas. It is common to have a big gap between inter-domain bandwidths and inner-domain bandwidths. Since all applications are obtained from a single repository in non-HDDA, large overhead could be caused by package transfer across networks with low bandwidth and big Round-Trip Time (RTT). In order to solve this problem, we propose the HDDA model, which is illustrated in Figure 1(b).

As shown in Figure 1(b), HDDA has a tree-like topology, which is composed of a root domain and a set of offsprings. The inner bandwidth (within a domain) is usually much larger than that the inter-domain bandwidth (between two independent domains). Each domain has its own Application

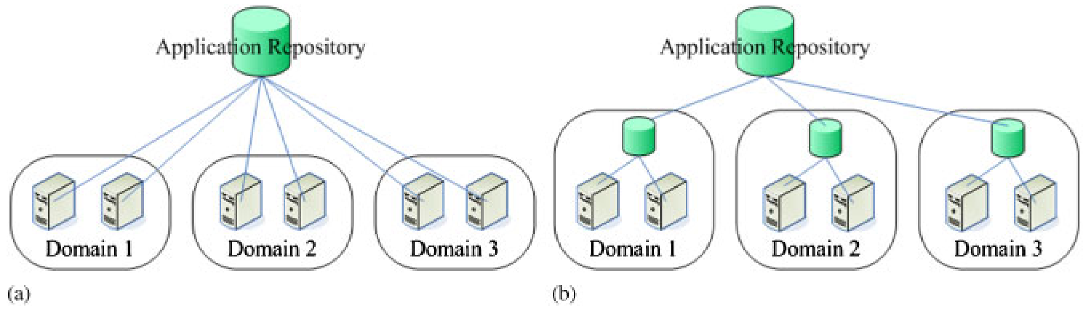


Figure 1. (a) Non-HDDA layout and (b) HDDA layout.

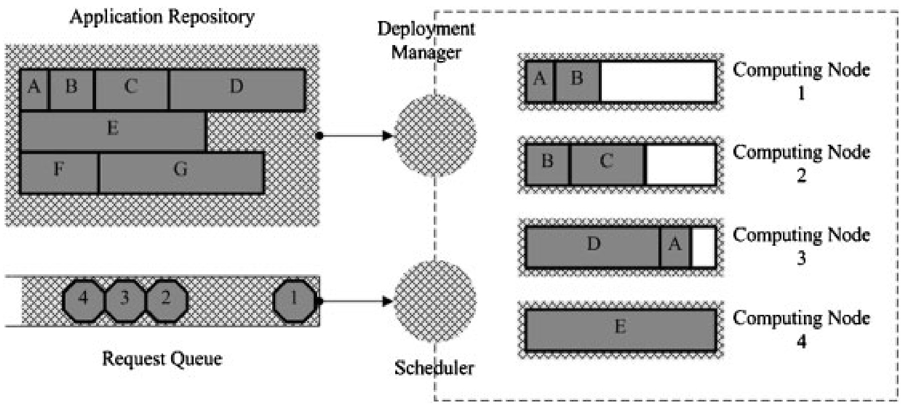


Figure 2. Dynamic deployment of applications.

Repository, in which replicas of all application packages (files used to install applications) are cached. When a new application package is published by the administrator, its replicas are transferred to and stored in the Application Repository of the root domain. If an application needs to be deployed to a new node of a domain, the application package can be obtained from the local Application Repository of the domain so that no package transfer across domains is required. If the application package is not obtainable from the local Application Repository of the domain, the application package will be fetched from the parent or grandparent domain of the domain. The Application Repository is supposed to have infinite capacity so that applications only need to be transferred once to the local repository of a domain.

3.2. Dynamic deployment within a domain

An overview of the dynamic deployment of applications is presented in Figure 2, in which the `Application Repository` of the local grid domain and its Computing Nodes play key roles. The `Application Repository` stores application packages available to users in its storage devices. Its contained application packages (e.g. `A`, `B`, and `C`) are composed of either binary or source files, which can be transferred to and installed on the computing nodes. As shown in Figure 2, there are seven application packages (i.e. `A`–`F`) in the storage devices of the `Application Repository`. The width of an application's block (e.g. block '`A`') roughly indicates the size and access frequency of the application. Each computing node has fixed space to be occupied by the application packages. For example, in Figure 2, applications `A` and `B` have been installed in `Computing Node` 1 (two gray blocks), which still has some space unoccupied (i.e. the white area). Using Figure 2 as an example, we describe the procedure of our proposed dynamic deployment as follows:

CASE 1. When a request for application `F` comes, the scheduler of the system fails to find any replica of application `F` on all available nodes (not deploying the requested application) termed as *raw node* in this paper. This situation is referred to as *invoking miss*, in which case the scheduler suspends the request for application `F`. As shown in Figure 2, computing node 1 has sufficient space to place application `F`, the `Deployment Manager` is then invoked to transfer the application package of `F` and install it on node 1. When the transfer and installation are completed, the scheduler resumes the suspended request for application `F` and schedules it to node 1.

CASE 2. When a request for application `G` comes, none of the nodes have enough space to install it since none of the nodes have sufficient space. In such case, the system takes the following actions. First, it selects an appropriate node from all available nodes by following an application replacement strategy. Second, some of the deployed applications on the selected node are undeployed to release resource. For example, if node 3 is chosen as the node to deploy application `G`, then the system undeploys applications `A` and `D` to make room for application `G`. These two actions are the major steps of our application replacement strategy (Section 4).

4. REPLACEMENT PROBLEM

When a application request comes and none of the nodes have enough space to install the requested application (e.g. CASE 2 discussed in Section 3.2), a strategy should be applied to determine which node should be selected to install the application and which installed application(s) of the selected node should be undeployed to make enough space for the newly requested application. The application replacement strategy has a significant impact on the overhead of the system. In this section, we first formalize the application replacement problem by specifying the preliminaries of the strategy (Section 4.1) and then discuss its optimization objective based on the formalization (Section 4.2).

4.1. Preliminaries

Notations used through this paper are defined as follows:

$n =$ the number of computing nodes.
$m =$ the number of applications.
$K =$ the total number of requests.
$S:\{S_1, S_2, \ldots, S_n\} =$ a set of computing nodes.
$A:\{A_1, A_2, \ldots, A_m\} =$ a set of applications.
$R_j:\{R_j^1, R_j^2, \ldots, R_j^{k_j}\} =$ a set of requests for the jth application.
The number of requests for the jth application is k_j.
$V_i =$ the size of the ith application package.
$D =$ the available size of disk space on a node.
$B =$ the bandwidth.
$E_j =$ the average execution time of the requests for the jth application.
$W_j =$ the time cost of the deployment of the jth application.
$P_j =$ the probability of that the jth application is requested.
$M_j =$ the probability of *invoking miss*, when a request for the jth application comes.
$I_j =$ the *average interval time* of user requests for the jth application.
$L_j =$ the ALR of all the requests for the jth application.

The following matrix is defined to denote the distribution of the applications on the computing nodes:

$$
C = \begin{array}{c} \\ S_1 \\ S_2 \\ \vdots \\ S_n \end{array}
\begin{array}{c}
\begin{array}{cccc} A_1 & A_2 & \cdots & A_m \end{array} \\
\begin{pmatrix}
c_1^1 & c_1^2 & \dots & c_1^m \\
c_2^1 & c_2^2 & \dots & c_2^m \\
\vdots & \vdots & \ddots & \vdots \\
c_n^1 & c_n^2 & \dots & c_n^m
\end{pmatrix}
\end{array}
\tag{1}
$$

$$
c_i^j = \begin{cases} 0 & A_j \textit{ has not been deployed on } S_i \\ 1 & A_j \textit{ has been deployed on } S_i \end{cases}
$$

The ith row in the matrix C means the array of the deployment status for all the applications on the ith node, and the jth column in the matrix C means the array of the deployment status of the jth application on all the nodes. Since there is no reason to have more than one copy of the same application on a single node, the value of the entry c_i^j must be either 0 or 1. The number of replicas of the jth application on all the nodes can be obtained by $c^j = \sum_{i=1}^{n} c_i^j$.

4.2. Optimization objective

It is not desirable for grid users to wait too long before their requested applications to be deployed; therefore, the optimization objective of a dynamic deployment strategy is to minimize the deployment time cost. To formalize the optimization problem, we define the *latency ratio* of a request as w/e, where w is the time taken for the completion of the deployment of an application and e is the time cost of the execution of the application. The AVR of all jobs is then defined as

$$
ALR = \sum_{j=1}^{m} (L_j \cdot P_j) \tag{2}
$$

where L_j is the ALR of all the requests for the jth application; P_j is the probability that the jth application is requested. The optimization objective is therefore formulated as

$$
\begin{aligned}
&\text{Objective: Minimize} \quad (ALR) \\
&\text{Strict To: } \sum_{j=1}^{m} (c_i^j \cdot V_j) \le D, \qquad 1 \le i \le n
\end{aligned}
\tag{3}
$$

5. ALR-MIN REPLACEMENT STRATEGIES

Most traditional replacement strategies are LRU-based and they can hardly minimize the ALR of jobs. The following are two differences between our ALR-MIN strategy and LRU-based strategies. First, our ALR-MIN strategy is capable of identifying not only the applications that should be evicted

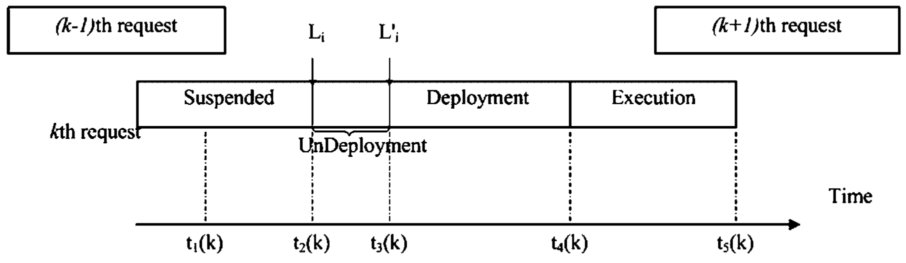


Figure 3. Lifetime of the kth request.

but also the node where a new application should be placed. However, LRU-based strategies can only identify applications to be evicted. Second, the ALR-MIN strategy can select nodes and applications according to the information including application access frequency, application packages size, application replicas number, and average execution time of application jobs. However, LRU-based strategies do not take this information into account.

In this section, we first describe how ALR-MIN works in general and then we propose two ALR-MIN strategies to minimize the ALR for heavy workload systems and light workload systems, respectively.

5.1. Two steps of ALR-MIN

As shown in Figure 3, four periods of processing the kth request in the system with dynamic deployment of applications are: `Suspended`, `UnDepoyment`, `Deployment`, and `Execution`, among which `UnDeployment` and `Deployment` are related to application replacement steps: (1) selecting an appropriate node to deploy a newly requested application and (2) undeploying the selected applications of the same node to make room for the newly requested application.

During the first step, the node with minimum *average latency ratio of node* (NALR) is chosen to place the requested application. $NALR_i$ is defined as the increment of ALR, caused by the undeployment of all applications on the ith node. The node with minimum NALR is probably the node where the least useful applications are deployed. Let L_i and L'_i be the latency ratio at the time of $t_2(k)$ and $t_3(k)$, respectively. Suppose all applications on the ith node are undeployed during the time from $t_2(k)$ to $t_3(k)$. Then $NALR_i$ can then be obtained by $L'_i - L_i$.

During the second step, the application(s) with the minimum increment of ALR on the selected node are chosen to be evicted. Let l_j and l'_j denote the ALR of the jth application just before and right after it is undeployed, respectively. The increment of the ALR caused by evicting the jth application is $l'_j - l_j$.

According to the above definitions and calculations, the following equation can be obtained to formulate the increment of the ALR for evicting all the applications on the ith node. Recall that P_j is the probability that the jth application is requested (Section 4.1).

$$NALR_i = L'_i - L_i = \sum_{j=1}^{m} ((l'_j - l_j) \cdot P_j) \tag{4}$$

where m is the total number of applications.

Let J be the set of applications that have been deployed on the ith node. For $\forall j \notin J$, the number of application replicas in the system does not change during the time from $t_2(k)$ to $t_3(k)$. Hence, it can be considered that $l'_j = l_j$ is approximately true. Then the following equation can be derived from (4):

$$L'_i - L_i = \sum_{j \in J} ((l'_j - l_j) \cdot P_j), \quad J = \{j | c_i^j = 1\}$$
$$P_j = \frac{1/I_j}{\sum_{j=1}^{m} (1/I_j)} \tag{5}$$

where c_i^j is defined in Equation (1). Recall that I_j is the *average interval time* of user requests for the jth application (Section 4.1). Hence, the key problem to get the predicted value of $L'_i - L_i$ is how to calculate the value of $l'_j - l_j$, which will be further discussed in Sections 5.2 and 5.3. The estimated ALR of the jth application is determined by

$$l_j = M_j \cdot \frac{W_j}{E_j} = M_j \cdot \frac{(V_j/B)}{E_j} \tag{6}$$

Here, we assume that the time to transfer an application package is the major part of the time cost of deploying the jth application: W_j. Otherwise, the ratio of the size of the jth application package over the bandwidth V_j/B can be replaced with $(V_j/B) + (\textit{deployment time of application})$.

Notice that the probability of *invoking miss* is generally determined by matrix C (the distribution of the applications on the computing nodes), the sequence of requests, and the workload of the system.

5.2. ALR-MIN for heavy workload

In the case of a heavy workload, M_j (the probability of *invoking miss* of the jth application) is considered approximately proportional to $1 - c^j/n$, when a request for the jth application comes. Since the system is busy, it is common that a request is suspended for a period of time before being scheduled to a node. Obviously, it is a small probability event that two nodes would become idle at the same time. In most cases, a request can only obtain at the most idle node, after waiting for a certain time. Here, the node that becomes idle is referred to as a *released node*. Hence, the value of M_j is approximately the probability of the *released node* being a *raw node*. This probability depends on the workload of each node. Based on the fact of a heavy workload, we assume that each node has a similar workload. Hence, a busy node is considered to be randomly released. The probability of the *released node* being a *raw node* is approximately $1 - c^j/n$. The latency ratio of the current request is

$$l_j = \frac{\left(1 - \frac{c^j}{n}\right) \cdot (V_j/B)}{E_j}$$

If the jth application is selected and replaced, the new latency ratio is

$$l'_j = \frac{\left(1 - \dfrac{c^j - 1}{n}\right) \cdot (V_j/B)}{E_j}$$

The increment of the latency ratio of the jth application is

$$l'_j - l_j = \frac{\dfrac{1}{n} \cdot (V_j/B)}{E_j} \tag{7}$$

5.3. ALR-MIN for light workload

In the case of a light workload, M_j is considered approximately proportional to $(1 - n_{idle}/n)^{c^j}$. n_{idle} is the number of idle nodes when a request comes. We assume that the workload is randomly scattered on the nodes. The probability of each node being busy is approximately equal to $1 - n_{idle}/n$. For the jth application with c^j replicas, the probability of all nodes installed with the jth application being busy at the same time is $(1 - n_{idle}/n)^{c^j}$. The latency ratio of the current request is

$$l_j = \frac{\left(1 - \dfrac{n_{idle}}{n}\right)^{c^j} \cdot (V_j/B)}{E_j}$$

If the jth application is selected and replaced, the new latency ratio is

$$l'_j = \frac{\left(1 - \dfrac{n_{idle}}{n}\right)^{(c^j - 1)} \cdot (V_j/B)}{E_j}$$

The increment of the latency ratio of the jth application is

$$l'_j - l_j = \frac{\left(1 - \dfrac{n_{idle}}{n}\right)^{c^j - 1} \cdot \dfrac{n_{idle}}{n} \cdot (V_j/B)}{E_j} \tag{8}$$

6. EVALUATING HDDA

An experiment has been conducted on ChinaGrid to evaluate our HDDA by comparing it with other two schemas: non-HDDA and SDA. In this section, we discuss the experiment methodology first (Section 6.1), including the environment setting, experiment procedure, and evaluation metrics. Then the experiments results are summarized, analyzed, and compared (Section 6.2).

6.1. Experiment methodology

The experiment was conducted on BioGrid, a sub-project of ChinaGrid. ChinaGrid integrates a variety of distributed and heterogeneous resources, 10 of whose domains were chosen to be our experimental environment. They are located in different cities of China: Beijing (six domains), Lanzhou (one domain), Wuhan (one domain), and Guangzhou (one domain). The relative computational capacities of the 10 domains were approximately {4, 4, 4, 4, 2, 2, 1, 1, 1}. The bandwidth was $100\,\mathrm{kB\,s^{-1}}$ between domains and $10\,\mathrm{MB\,s^{-1}}$ within a domain. One hundred software tools frequently used in bioinformatics were selected to be the applications in this experiment. The size of the selected applications varies from 500 kB to 850 MB.

As shown in Figure 1(a), non-HDDA only has one central Application Repository from which all the nodes in all the 10 domains retrieve applications. As opposed to non-HDDA, HDDA has a local repository in each domain (see Figure 1(b)); a central repository independent of any individual domain is located in Beijing. The simplest scheme of the three is SDA, which has no repository deployed.

Three evaluations were conducted to compare the performance of non-HDDA, HDDA, and SDA. At the beginning of the evaluations, the applications were initiated according to the following rules. The ith application was deployed to the $(i\,MOD\,N)_{th}$ node, where N is the total number of the nodes. This procedure does not stop until the deployment limit (Section 6.2) of the nodes is exceeded or there are copies of all the applications on each node.

During each evaluation, 1000 requests were submitted continuously. The average request arrival rates of different applications followed Zipf's law [22], the arrival rates and the execution times of all the requests were generated according to the trace of CTC from [8]. The job intervals and runtimes were carefully adjusted according to the following equation from Queuing Theory, so that the length of the job queue in the system would not explode:

$$\frac{average\ runtime}{average\ interval} = limit\ of\ parallel\ \ executable\ jobs$$

The performance of the systems during a fixed period of time were obtained and examined with three metrics: throughput, Average Completion Time (ACT) of jobs, and load balancing. Load balancing is the variance of CPU utilization of all the nodes during the period. Since the scheduling strategy is beyond the topic of this paper, a random-based scheduling algorithm was applied.

6.2. Experiment result

Figure 4 shows the average completion time of successfully completed jobs during a fixed time period across the three schemas. The horizontal axis is the deployment limit—the maximum number of applications allowed to be deployed on an individual node. This figure shows that the ACT of HDDA is always the least when the deployment limit is smaller than 70. SDA results in a much larger ACT with a deployment limit smaller than 60, compared with HDDA and non-HDDA. The average ACTs of HDDA, non-HDDA, and SDA in Figure 4 are 212, 236, and 280, respectively. Hence, the performance improvement by HDDA should be 10 and 24%, compared with non-HDDA and SDA, respectively. The performance improvement is due to the characteristics of HDDA that applications can be dynamically deployed and undeployed. It is impossible that the scheduler failed to map a

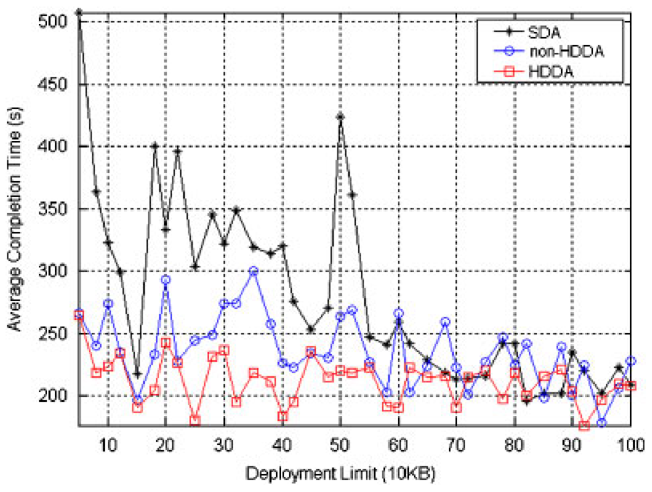


Figure 4. Average Completion Time of successfully completed jobs during a fixed period of time.

job when there are computing nodes available; jobs can be immediately mapped whether or not the corresponding application is deployed on the currently available nodes. Additionally, the local Application Repository in HDDA decreases the time required to transfer application packages over networks; therefore, jobs can be executed earlier in HDDA than non-HDDA. When the deployment limit is larger than 70, most applications can stay in the local disks of nodes, without being evicted. The overhead of package transfer among repositories and computing nodes is small and similar to each other. The completion times of jobs for three schemas also tend to be similar.

The system throughput of the three schemas during a fixed period of time is presented in Figure 5. The figure shows that SDA results in a much lower throughput than HDDA and non-HDDA, when the deployment limit is less than 60. The throughput of non-HDDA is always comparable with HDDA. The reason is as follows. Normally, there exists a large gap between access frequencies of different applications. For very popular applications, the number of requests may be much larger than the number of application replicas. In SDA, application requests usually have to wait for the end of previous jobs so that many jobs are blocked in the waiting queue. However, in HDDA, jobs are processed immediately if idle nodes are available and therefore the queue length in HDDA will never explode. In other words, more jobs can be completed in HDDA within a fixed period of time. When the deployment limit is larger than 60, most applications can stay in the local disks of nodes, without being evicted. The overhead of package transfer among repositories and computing nodes is small and similar to each other. The throughput for three schemas also tend to be similar.

Figure 6 shows the load balancing of all nodes across the three schemes. When the deployment limit is less than 80, the load variance of nodes in non-HDDA is obviously greater than that in HDDA. SDA results in very bad load balancing among computing nodes when the deployment limit is less than 50. The load balancing of non-HDDA is even worse than that of SDA when the deployment limit is between 65 and 80. The load unfairness in SDA is caused by the different access frequencies of applications. The nodes where more popular applications are deployed are always busier than

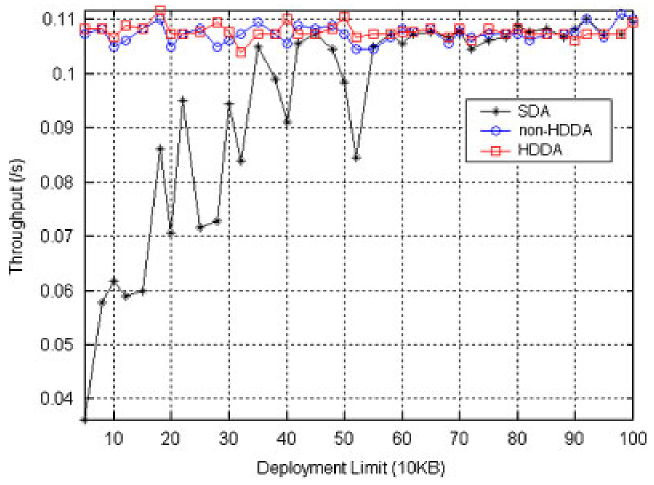


Figure 5. System throughput during a fixed period of time.

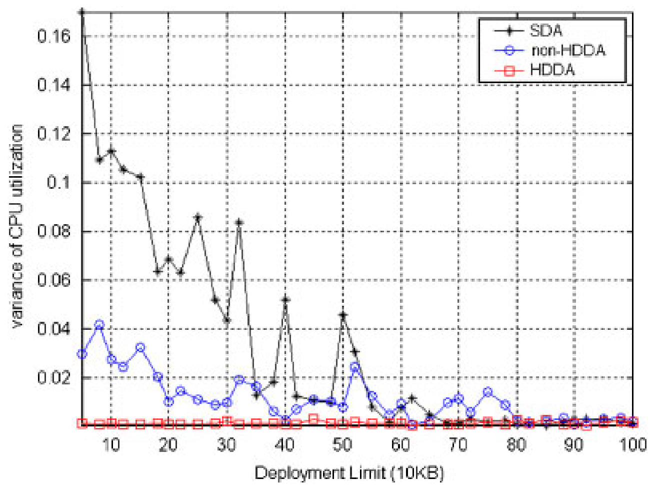


Figure 6. Load balancing of all nodes during a fixed period of time.

those nodes where less popular applications are deployed. The dynamic deployment strategy of HDDA can dynamically balance the distribution of applications with different access frequencies; therefore, the workload is able to be balanced. For non-HDDA, the unfairness of utilization is mainly caused by the transfer of big size applications over networks. Nodes or processors have to wait until the transfer is completed. When the deployment limit is larger than 80, most applications can stay in the local disks of nodes, without being evicted. The overhead of package transfer among repositories and computing nodes is small and similar to each other. The load balancing of all nodes for three schemas also tend to be similar.

From Figures 4–6, we can see that HDDA can yield a smaller delay of jobs, larger throughput, and better load balancing when the deployment limit is smaller than half of the total application number. SDA performs much worse than non-HDDA and HDDA. This is because a requested application may not be deployed on idle nodes. The non-HDDA scheme sometimes requires transferring application packages across networks with low bandwidth; therefore, its overhead is definitely larger than that of HDDA. When the deployment limit is near the total application number, most applications can stay in the local disks of nodes, without being evicted. The overhead of package transfer among repositories and computing nodes is small and similar to each other. Therefore, no significant difference can be identified among the three schemes in terms of system throughput, ACT, and load balancing.

7. EVALUATING TWO ALR-MIN STRATEGIES

A series of simulations have been conducted on a well-designed simulator to evaluate our two ALR-MIN strategies for heavy and light workload systems, respectively. They are compared with each other and also compared with other two commonly used LRU-based strategies: Random-LRU and Cooperative-LRU.

In the remainder of the section, two LRU-based strategies are described in Section 7.1. Our simulation methodology is discussed in Section 7.2, including the design of the simulator and the simulation settings. Last, in Section 7.3, the detailed simulation results are discussed.

7.1. Random-LRU and cooperative-LRU

As mentioned in Section 3.2, when a request comes and none of the nodes have enough space to install the requested application, a application replacement strategy should take the following actions to determine: (1) which node is selected to deploy the new application, and (2) which installed applications of the selected node should be undeployed in order to make enough space for the newly requested application. To study the performance improvement of ALR-MIN strategies, they are compared with two commonly used LRU-based strategies: Random-LRU and Cooperative-LRU strategies.

- Random-LRU exploits a simple approach of randomness to select a node in the first step. When an invoking miss occurs, a node is chosen randomly from the pool of idle nodes to place the new application. In the second step, applications with the oldest/minimum Last Access Time (LAT) are chosen to be evicted.
- Cooperative-LRU selects an appropriate node to deploy the new application during the first step. The node with the minimum Last Access Time of Node (NLAT) is chosen to deploy the new application. $NLAT_i$ is defined as the weighted arithmetic mean of the LATs of all the applications on node i:

$$NLAT_i = \sum_{j=1}^{m} (c_i^j \cdot V_j \cdot LAT_j)$$

where j is the identifier of the application, and i is the identifier of the specified node to deploy the new application. In the second step, Cooperative-LRU employs a LRU method similar to Random-LRU.

7.2. Simulation methodology

A simulator has been developed to evaluate our ALR-MIN strategies, which is composed of four components: Workload Generator, Virtual Resource, Job Scheduler, and Deployment Manager. The Workload Generator generates a sequence of synthetic requests. The Virtual Resource records the status of virtual resources: providing the information of where an application has been deployed and whether a job is running on a specified resource. Since the study of scheduling strategy is beyond the scope of this paper, the Job Scheduler of the simulator always randomly schedules the current job to resources and the requests are invariably processed with the order of their arrival times. If the requested application is not deployed on any idle nodes, the Deployment Manager then applies the ALR-MIN strategies.

Our simulator is highly configurable. First, the workload of the simulator is synthetic and configurable. The *Idle ratio* of the system is defined to be the average ratio of idle nodes over all the nodes, which is then used to classify the system's workload into three levels: heavy, medium, and light. By adjusting the *Application Arrival Rate* (AAR), all three workload levels can be achieved. AAR is the mean arrival rate of requests for an application and it should be integer times of the *Mean Interval of Requests for the Most Popular Application* (MIMPA). Second, the *Disk Space Ratio* (DSR) of the nodes is configurable. It equals the ratio of space available on each node to the total size of all applications. Third, the *Average Ratio of Deployment Time to Execution Time* (ARDE) of all applications is configurable. It represents the relative overhead of application deployment. The combination of MIMPA, DSR, and ARDE can approximately represent the setting of the simulations. Hence, we define the triple set of [MIMPA, DSR, ARDE] to represent the simulation setting.

Table I shows the detailed settings of the important system parameters of the simulations. The expression of [X:Y:Z] in Column 3, Table I indicates that values of a parameter varies from X to Z with the interval of Y. The values of the parameters were selected according to the following principles. First, some are from our experiences on actual research environments, including Application Number, MIMPA, Workload (*idle ratio*), Application Size, Application Average Execution Time, and Node Number. Second, some of them (e.g. Total Job Number) are maximized to the tolerable limit in the simulation. Third, some numbers are set to vary within a certain range to show the scalability of proposed policies, such as DSR, Application Average Request Interval, and ARDE. Fourth, some parameters are assumed to follow a specific distribution. For example, the Application Arrival Rate is assumed to follow Zipf-like distribution, because the access frequency of an entity is usually considered to be inversely proportional to its rank in the frequency table. Finally, the *mean interval time* and *mean execution time* of the requests are obtained by calculating the weighted arithmetic mean of the last 10 requests for a specified application.

7.3. Simulation results

Figure 7 shows the performance of the four strategies (i.e. Random-LRU, Cooperative-LRU, ALR-MIN for Heavy Workload, and ALR-MIN for light workload) in the situation of heavy workload

Table I. Detailed setting in simulations.

Parameters	Abbr.	Values		
Application number	M	128		
Total job number	—	27 000		
Node number	N	32		
Workload (*idle ratio*)	—	Heavy [0-0.1]	Medium [0.1–0.9]	Light [0.9–1]
Mean interval of requests for the most popular application	MIMPA	10	200	2000
Disk space ratio	DSR	[0.016:0.008:0.64]	[0.016:0.008:0.32]	[0.016:0.008:0.16]
Application average request interval	I	[MIMPA : MIMPA : MIMPA*M]		
Application arrival rate	AAR	Followed a Zipf-like distribution		
Average ratio of deployment time to execution time	ARDE	[0.05:0.05:1]		
Application size	—	Followed a random distribution with mean 500		
Application average execution time	E	Followed a random distribution with mean 500		
Job interval distribution of the same application	—	Followed an exponential distribution with mean I_i, which is the *average request interval* of requests for the ith application		
Job execution time distribution of the same application	—	Followed an exponential distribution with mean E_i, which is the *average execution time* of requests for the ith application		

with the MIMPA being 10. Figures 7(a) and (b) have the same setting of [10, 0.16, *], where * means that the ARDE varies from 0 to 1. The *idle ratio* of the four strategies is presented in Figure 7(a) with ARDE as the horizontal axis. For four strategies, we can observe that the percentages of idle nodes are all about 1.7%. Figure 7(b) shows the ALR with the same setting in Figure 7(a). The vertical axis is the ALR. We can observe from this figure that the performances of Random-LRU and Cooperative-LRU are similar and consistently and significantly worse than that of ALR-MIN for heavy workload and light workload. Figure 7(c) presents the ALR of various DSR with the setting of [10, *, 0.2], where * means that the DSR varies from 0.016 to 0.64 (the range is specified in Table I). When the space becomes more sufficient and therefore less replacement is required, the performances of the four strategies are closer to each other.

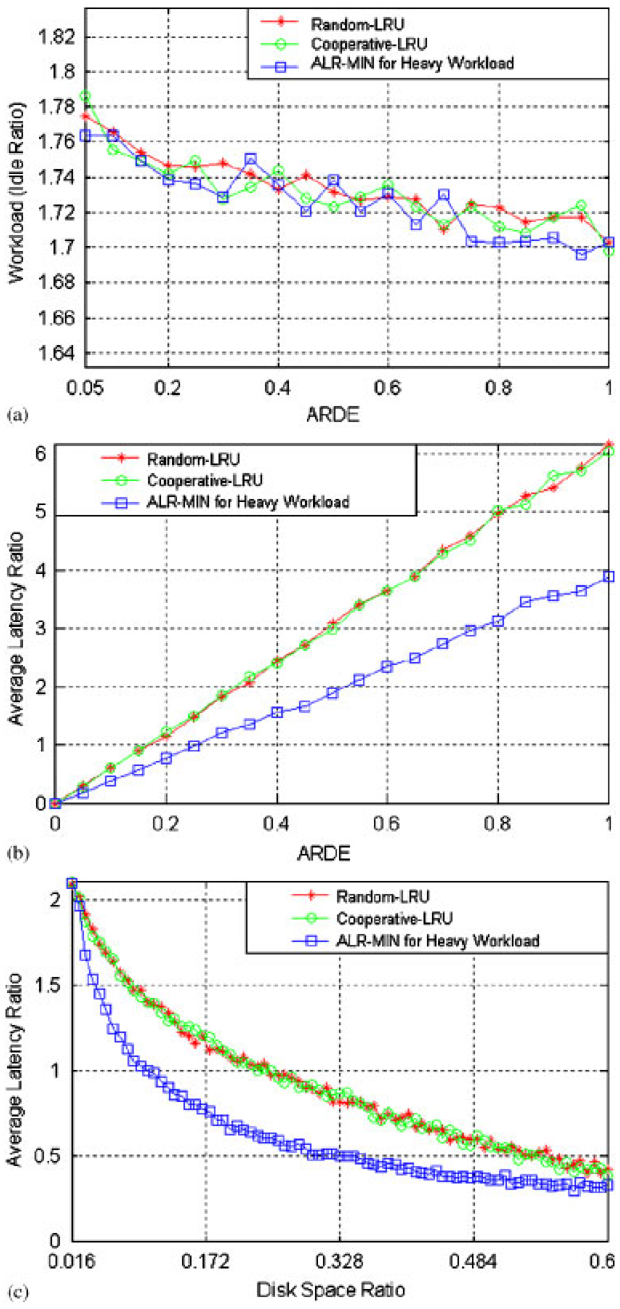


Figure 7. Performance of heavy workload: (a) workload (idle ratio); (b) ALR of various ARDE; and (c) ALR of various DSR.

Random-LRU only takes the access frequency of applications into account and Cooperative-LRU neglects the number of replicas and the average execution time of applications. Different sizes produce different time costs of deployment. The number of replicas affects the probability of deployment for an application. The average execution time of an application is one of the major factors that determine the ALR. ALR-MIN can evaluate the result of choosing different nodes and applications during deployment and undeployment according to the access frequency, size, number of replicas, and average execution time. Thus, the ALR-MIN strategy is able to select the node or application that can minimize the increment of ALR.

Figure 8 shows the performance of the four strategies in the situation of a medium workload with the MIMPA being 200. Figures 8(a) and (b) have the same setting of [200, 0.064, *]. As shown in Figure 8(a), the percentage of idle nodes is within the range from 35 to 50%. It can be observed from Figure 8(b) that the ALR of ALR-MIN for light workload has the lowest percentage. In Figure 8(c), the details of the ALR of the medium workload are presented with the setting of [200, *, 0.2]. When the DSR is less than 0.244, ALR-MIN for light workload always outperforms other three strategies. For a medium workload, the ALR-MIN for light workload is better than LRU-based strategies.

Figure 9 shows the performance of the four strategies in the situation of a light workload with the MIMPA being 2000. Figures 9(a) and (b) have the setting of [2000, 0.04, *]. As shown in Figure 9(a), the percentage of idle nodes is more than 90%. Figure 9(b) shows that ALR-MIN for light workload yields the lowest ALR, and Random-LRU gives the highest. Figure 9(c) presents the detailed ALR with a setting of [2000, *, 0.2]. It can also be seen from Figure 9(c) that the ALR-MIN for light workload is the best.

Based on the results presented in Figures 7 Figure 8 and Figure 9, we can conclude that our ALR-MIN strategies can result in a shorter average delay-time of jobs than the two LRU-based strategies in most cases. ALR-MIN for heavy workload is suitable for heavy workload systems. ALR-MIN for light workload can always give the best or near-best ALR among the four strategies. With a typical setting of ARDE being 0.4 and DSR being 0.04, the ALR-MIN can reduce the ALR by 18% (light workload), 14% (heavy workload), and 19% (medium workload), compared with LRU-based strategies. Thus, the average ALR improvement of all workloads can be 17%.

8. CONCLUSION

Clearly, the importance of the overall performance of a Grid system has been well-recognized; however insufficient research has been conducted to address the issue of how to improve Grid system performance through dynamically deploying applications. Applications for a Grid system are usually statically deployed to a pre-selected subset of the computing nodes of the system and the overall performance of the system is unavoidably degraded in this sense. A dynamic application deployment schema is required to improve the system performance from the aspects of its throughput, average completion time of jobs, and load balancing.

In this paper, we propose a schema, named as HDDA, to improve the overall system performance of Grid. By dynamically deploying and undeploying applications, the applications with different access frequencies and workload can be balanced among all computing nodes. The completion time of jobs can be shortened. More jobs can be completed within a fixed period of time. By placing

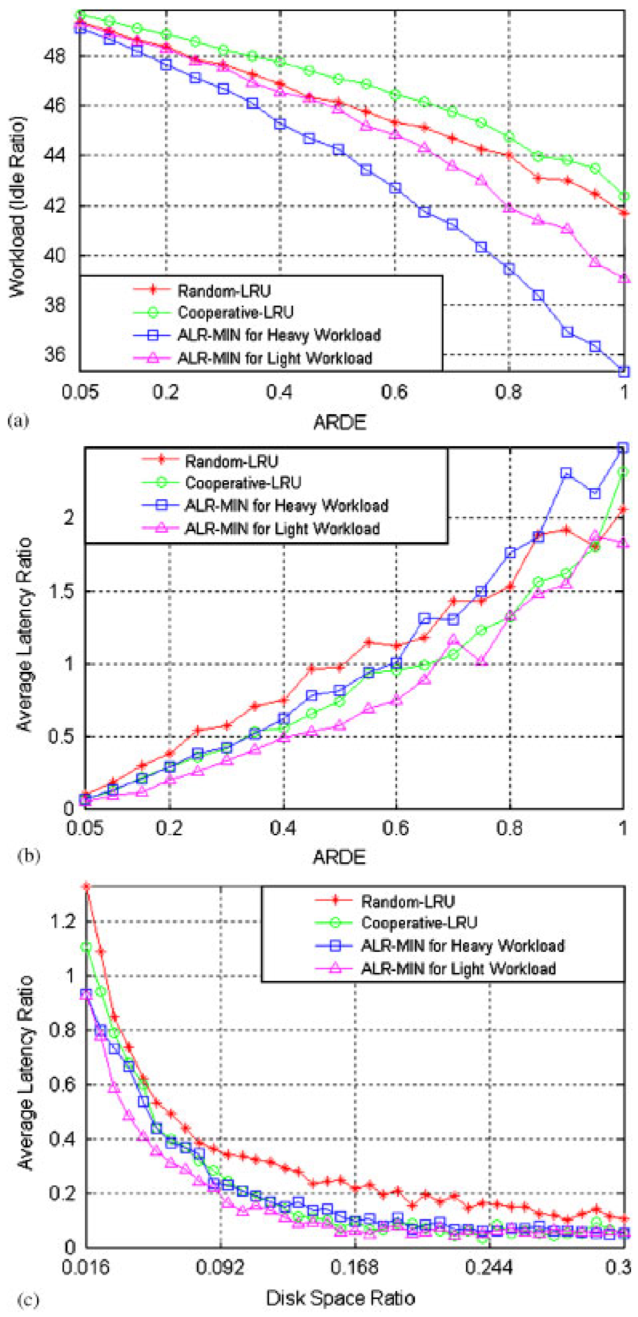
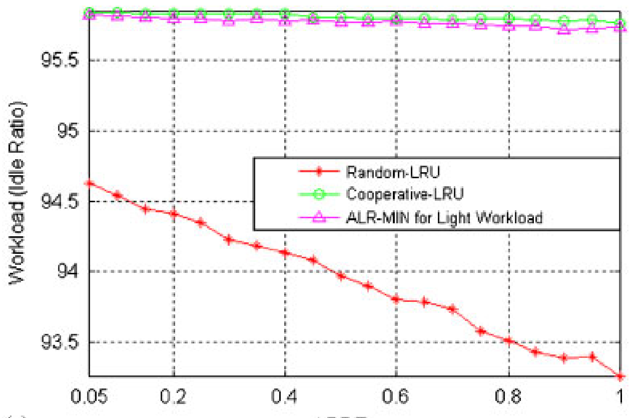
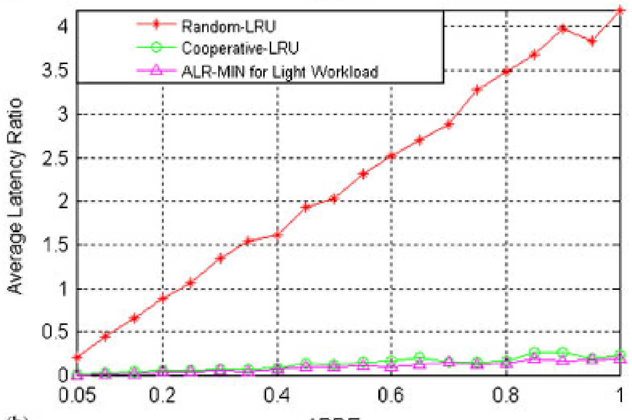


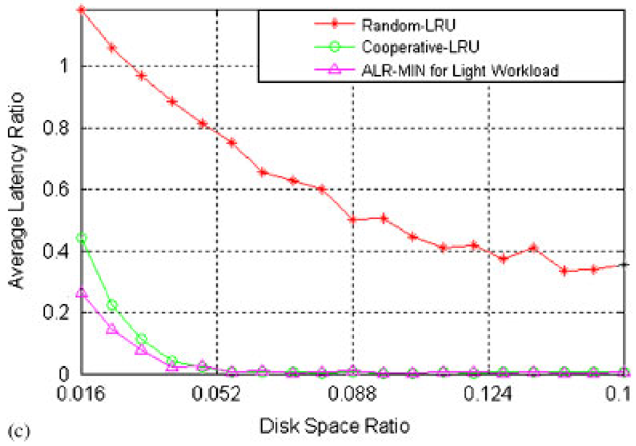
Figure 8. Performance of medium workload: (a) workload (idle ratio); (b) ALR of various ARDE; and (c) ALR of various DSR.



(a)



(b)



(c)

Figure 9. Performance of light workload: (a) workload (idle ratio); (b) ALR of various ARDE; and (c) ALR of various DSR.

a local Application Repository in each domain (one of the characteristics of HDDA), much less time is required to transfer application packages over networks and therefore the completion time of jobs and load balancing can be further achieved.

In order to reduce the overhead caused by dynamic deployment of the HDDA schema, we also proposed two ALR-MIN replacement strategies in this paper. With these strategies, a node with least estimated NALR is selected to deploy a new application, and the older applications with least estimated increment of ALR might be selected to be evicted if insufficient space is available on the selected node to deploy the newly requested application.

An experiment was carefully designed and conduced on ChinaGrid to evaluate HDDA by comparing it with the non-HDDA and SDA schemas. The experiment results show that HDDA can improve the throughput of the system, reduce the average completion time of jobs, and balance the workload more effectively than non-HDDA and SDA. More specially, HDDA can achieve 10 and 24% less Average Complete Time (ACT) than non-HDDA and SDA, respectively. Besides, HDDA can achieve better throughput and load balancing than the other two schemas.

A series of simulations were performed on a well-designed simulator and the results show that our ALR-MIN application replacement strategies can produce a lower relative delay-time of jobs with either heavy workload or light workload, compared with LRU-based strategies. ALR-MIN results in 17% less relative delay-time of jobs than the well-known LRU-based strategies with a typical setting.

ACKNOWLEDGEMENTS

This work is supported by ChinaGrid project of Ministry of Education of China, Natural Science Foundation of China (90412006, 90412011, 60573110, 90612016, 60673152), National Key Basic Research Project of China (2004CB318000, 2003CB317007), and National High Technology Development Program of China (2006AA01A108, 2006AA01A111, 2006AA01A101).

REFERENCES

1. Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications* 2001; **15**(3):200.
2. Wu YW, Wu S, Yu HS, Hu CM. Introduction to ChinaGrid support platform. *Proceedings of Parallel and Distributed Processing and Applications—Ispa 2005 Workshops*, Nanjing, China, 2–5 November 2005; 232–240.
3. Qi L, Jin H, Foster I, Gawor J. HAND: Highly available dynamic deployment infrastructure for Globus Toolkit 4. *Proceedings of 15th EUROMICRO International Conference on Parallel*, *Distributed and Network-based Processing*, Naples, Italy, 7–9 February 2007; 155–162.
4. Pu L, Lewis MJ. Uniform dynamic deployment of Web and Grid Services. *Proceedings of IEEE International Conference on Web Services* (*ICWS 2007*), Salt Lake City, Utah, U.S.A., 9–13 July 2007; 26–34.
5. Watson P, Fowler C, Kubicek C, Mukherjee A, Colquhoun J, Hewitt M, Parastatidis S. Dynamically deploying Web services on a grid using Dynasoar. *Proceedings of Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, Gyeongju, South Korea, 28 April 2006.
6. Siddiqui M, Villazon A, Hofer J, Fahringer T. GLARE: A grid activity registration, deployment and provisioning framework. *Proceedings of the ACM/IEEE Conference on Supercomputing*, Seattle, Washington, U.S.A., 12–18 November 2005.
7. Byun EK, Kim JS. DynaGrid: A dynamic service deployment and resource migration framework for WSRF-compliant applications. *Parallel Computing* 2007; **33**(4–5):328–338.
8. Shankar S, DeWitt DJ. Data driven workflow planning in cluster management systems. *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing*, Monterey, CA, U.S.A., 25–29 June 2007; 127–136.

9. Application Contents Service Specification 1.0. Available at: http://www.ogf.org/documents/GFD.73.pdf [01-01-2008].
10. Diwakar D, Diwakar S. CINWEGS—An integrated Web and grid services framework for collaborative solutions. *Proceedings of Fourth International Conference on Next Generation Web Services Practices*, Seoul, Korea, 22–26 August 2005; 21–27.
11. FuQiang L, Bin G, Cheng X, Yan M. Dynamic visualization service deployment in grid scientific workflow. *Proceedings of Seventh International Conference on Grid and Cooperative Computing*, Shenzhen, China, 24–26 October 2008; 201–205.
12. Parallel workloads archive. Available at: http://www.cs.huji.ac.il/labs/parallel/workload/index.html [03-03-2008].
13. Megiddo N, Modha DS. Outperforming LRU with an adaptive replacement cache algorithm. *Computer* 2004; **37**(4):58–65.
14. Jeong JH, Dubois M. Cache replacement algorithms with nonuniform miss costs. *IEEE Transactions on Computers* 2006; **55**(4):353–365.
15. Wooster RP, Abrams M. Proxy caching that estimates page load delays. *Computer Networks and ISDN Systems* 1997; **29**(8–13):977–986.
16. Abrams M, S. Dept. of Computer I, Virginia Polytechnic, and U. State. Caching proxies: limitations and potentials. *Proceedings of the Fourth International World Wide Web Conference*, Boston, MA, U.S.A., 11–14 December 1995.
17. Cao P, Irani S. Cost-aware WWW proxy caching algorithms. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, U.S.A., 8–11 December 1997; 193–206.
18. Hung HP, Chen MS. On designing a shortest-path-based cache replacement in a transcoding proxy. *Multimedia Systems* 2009; **15**(2):49–62.
19. Zhu YW, Hu YM. Exploiting client caches to build large Web caches. *Journal of Supercomputing* 2007; **39**(2):149–175.
20. Cohen E, Shenker S. Replication strategies in unstructured peer-to-peer networks. *Proceedings of ACM SIGCOMM'02*, Pittsburgh, PA, U.S.A., 19–23 August 2006.
21. Tewari S, Kleinrock L. Proportional replication in Peer-to-Peer Networks. *Proceedings of 25th IEEE International Conference on Computer Communications*, Barcelona, Spain, 23—29 April 2006; 1–12.
22. Power law distributions in real and virtual worlds. Available at: http://www.isoc.org/inet2000/cdproceedings/2a/2a_2.htm [18-05-2008].