ALR-MIN: A Replacement Strategy to Reduce Overhead during Dynamic Deployment of Applications in Grid

Chen Gang, Yongwei Wu, Jia Liu, Rui Fang, Guangwen Yang, Weimin Zheng
Department of Computer Science and Technology;
Tsinghua National Laboratory for Information Science and Technology
Tsinghua University Beijing 100084, China
c-g05@mails.tsinghua.edu.cn, wuyw@tsinghua.edu.cn, liu-jia04@mails.tsinghua.edu.cn,
fangr06@mails.tsinghua.edu.cn, ygw@tsinghua.edu.cn, zwm-dcs@tsinghua.edu.cn

***Abstract*—Many applications of existing grid systems are usually deployed on grid nodes in a static manner. Grid computing nodes with highly demanded applications deployed on are always busy; while the others are idle. If these applications could be dynamically deployed on idle nodes, the system performance would be greatly improved. However, the overhead (e.g., package download) caused by dynamic deployment in existing studies may be so much that the system performance is seriously degraded. It is because existing studies on dynamic deployment of applications do not exploit an application replacement strategy to deploy or undeploy applications. In this paper, we propose an application replacement strategy, the Average Latency Ratio Minimum (ALR-MIN) strategy. With this strategy, an application is either deployed or evicted, considering the minimum ALR of a Node (NALR). A series of simulations have been performed and their results demonstrate that the ALR-MIN strategy results in 17% less relative delay-time of jobs than the well-known Least Recently Used based (LRU-based) strategies with a typical setting.**

Keywords-Grid Computing; Resource management; Dynamic deployment; Replacement strategy; System performance

I. 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, while 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.

The following aspects may influence the design of an application replacement strategy. 1) The overhead caused by dynamic deployment of applications is perhaps too significant so that the overall performance of the system may be seriously degraded because of this. 2) The capacity of a computing node is normally limited. 3) It is impossible to keep all the applications in the local system consistently. 4) Some of applications have to be replaced to make room for a new application and therefore applications may be deployed or undeployed repeatedly. 5) In the case of applications with big size, the overhead of dynamic deployment may be too much 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.

Most of the existing studies (e.g., [3], [4], [5], [6], [7], and [8]), addressing the problem of dynamic application deployment in grid, do not focus on application replacement strategies. [3] proposes an integrated framework, named CINWEGS, for dynamically deployment. [4] presents an architecture to deploy visualization services dynamically. However, they both do not include an application replacement strategy. The approaches proposed in [5], [6], and [7] dynamically deploy Web Services to containers without using a replacement strategy. An on-demand deployment strategy for scientific applications is proposed in [8] in order to reduce administrators' workload. DynaGrid [9] does not require a replacement strategy since the approach cannot undeploy older services when capacity limitation is reached. DAG-Condor [10] exploits a LRFU(Least Recently/Frequently Used)-based replacement strategy, which does not address the problem of multi-cache problem. In multi-cache problem, one of the catches has to be selected before a cache object is placed or evicted.

The Application Contents Service Working Group (ACS-WG) in Open Grid Forum has proposed a specification [11] to standardize application management in a grid environment, which makes it possible to combine ACS-WG and the specifications (e.g., CDDLM-FND, CDDLM-SF and CDDLM-CDL) from the CDDLM-WG to realize dynamic deployment of applications. However, these specifications do not take into account of application replacement strategies.

978-0-7695-3825-9/09 $26.00 © 2009 IEEE
DOI 10.1109/EmbeddedCom-ScalCom.2009.65

In this paper, we propose a replacement strategy, called ALR-MIN, to reduce the overhead of dynamic deployment. With this strategy, the node to deploy an application on and the applications to be evicted for the node are carefully chosen to reduce the Average Latency Ratio (ALR). The strategy contains a set of evaluation functions to predict and compare the increment of the ALR. The node with the minimum predicted ALR is chosen to hold a new application and the old applications with the minimum predicted ALR on this node may be chosen to be swapped out to make room for the new application.

Followings are two differences between ALR-MIN strategy and LRU-based strategy. First, ALR-MIN strategy is capable of identifying both the applications which should be evicted and the node where the new application should be placed. However, LRU-based strategies can only identify the applications to be evicted. Second, ALR-MIN strategy can select node and application according to the information including application access frequency, application packages size, application replicas number and average execution time of application jobs. However, LRU-based strategy does not take these information into account.

More specifically, the following contributions are achieved in this paper:

- The application replacement problem is formalized. Based on that, two ALR-MIN replacement strategies are proposed to reduce the ALR for both Heavy Workload and Light Workload respectively.
- The ALR-MIN strategy is evaluated and our simulation results show that the jobs with ALR-MIN strategy took the least relative delay-time to complete, compared with other two LRU-based strategies.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the dynamic deployment of applications, followed by the formalization of the application replacement problem in the context of dynamic deployment (Section 3). Section 4 presents two ALR-MIN strategies for heavy load and light load systems respectively. Section 5 examines the performance of the ALR-MIN and two traditional LRU-based strategies. Related work is discussed in Section 6. Last, we draw a conclusion in Section 7.

II. Dynamic Deployment of Applications

An overview of the dynamic deployment of applications is presented in Figure 1, in which the `Application Repository` of the local grid domain and its `Computing Node`s 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 1, 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 1, the applications `A` and `B` have been installed in the `Computing Node 1` (two grey blocks), which still have some space unoccupied (i.e., the white area). Using Figure 1 as an example, we describe the procedure of our proposed dynamic deployment as follows:

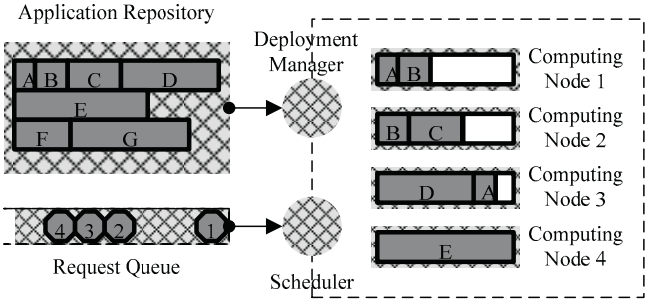


Figure 1. Dynamic deployment of applications.

When a request for the application `F` comes, the scheduler of the system fails to find any replica of the 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 the application `F`. As shown in Figure 1, the 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 the node 1. When the transfer and installation are completed, the scheduler resumes the suspended request for the application `F` and schedules it to the node 1.

When a request for the 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 the application replacement strategy (Section 4); second, some of the deployed applications on the selected node are undeployed to release resource of the node. For example, if the node 3 is chosen as the node to deploy application `G`, then the system undeploys the applications `A` and `D` to make room for the application `G`. These two actions are the major steps of our application replacement strategy.

III. Replacement Problem

As previously discussed, when a request comes and none of the nodes have enough space to install the requested application, then a strategy should be applied to determine which node is selected to install the application and which installed applications of the selected node should be undeployed in order 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 formalize the application replacement problem by formally specify the preliminaries of the strategy (Section 3.A) and the optimization objective (Section 3.B).

A. Preliminaries

Notations used through this paper are defined as follows:

n = number of computing nodes.

m = number of applications.

K = the total number of requests.

S: $\{S_1, S_2, \cdots, S_n\}$ = a set of computing nodes.

A: $\{A_1, A_2, \cdots, A_m\}$ = a set of applications.

$R_j : \{R_j^1, R_j^2, \cdots, R_j^{k_j}\}$ = a set of requests for the j_{th} application. The number of requests for the j_{th} application is k_j.

V_i = the size of i_{th} application package.

D = the available size of disk space on a node.

B = the bandwidth.

E_j = the average execution time of requests for the j_{th} application.

W_j = be the time cost of the deployment of the j_{th} application.

P_j = the probability of the j_{th} application is requested.

M_j = the probability of *invoking miss*, when a request for the j_{th} application comes.

I_j = the *average interval time* of user requests for the j_{th} application.

L_j = the ALR of all requests for the j_{th} application.

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

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

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

The i_{th} row in the matrix C means the array of the deployment status for all applications on the i_{th} node, and the j_{th} column in the matrix C means the array of the deployment status of the j_{th} application on all 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 j_{th} application on all nodes can be obtained by $c^j = \sum_{i=1}^{n} c_i^j$.

B. Optimization Objective

It is not desirable for grid users to wait long time before their requested applications to be deployed; therefore the optimization objective of dynamic deployment is to minimize the time cost of deployment. To formalize the optimization problem, we define the *latency ratio* of a request as $\frac{w}{e}$, 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 average latency ratio of all jobs is then defined as

$$ALR = \frac{\sum_{j=1}^{m}\left(\frac{W_j}{E_j}\cdot k_j\right)}{K} = \sum_{j=1}^{m}\left(L_j \cdot P_j\right) \tag{2}$$

where L_j is the ALR of all requests for the j_{th} application; e_j^k is the execution time of the k_{th} request for the j_{th} application. The optimization objective is therefore formulated as:

$$\textit{Objective}: \quad \textit{Minimize} \quad (ALR)$$

$$\textit{Strict To}: \quad \sum_{j=1}^{m}\left(c_i^j \cdot V_j\right) \le D \qquad 1 \le i \le n \tag{3}$$

IV. ALR-MIN Replacement Strategies

Most traditional replacement strategies are LRU-based and they can hardly minimize the ALR of jobs. In this section, we propose two strategies to minimize the ALR for heavy workload systems and light workload systems respectively, named as ALR-MIN for Heavy Workload and ALR-MIN for Light Workload.

A. Two Steps of ALT-MIN

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

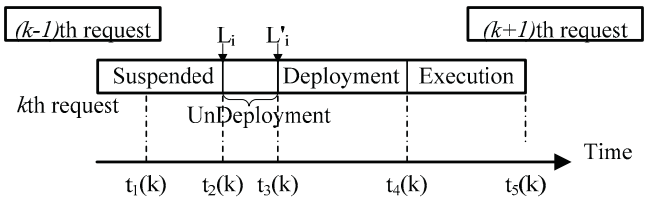


Figure 2. Lifetime of the kth request.

For 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 i_{th} node. The node with minimum NALR is probably the node where the least useful applications deployed. Let L_i be the latency ratio at the time of $t_2(k)$ and L_i' the latency ratio at the time of $t_3(k)$. Suppose all applications on the i_{th} node are undeployed during the time from $t_2(k)$ to $t_3(k)$. Then $NALR_i$ can then be obtained by $L_i' - L_i$.

For the second step, the application(s) with the minimum increment of ALR on the selected node are chosen to be evicted. Let l_j denote the ALR just before the undeployment of the j_{th} application, and let l'_j denote the ALR after the undeployment of the j_{th} application. The increment of the ALR caused by evicting the j_{th} 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 the j_{th} application:

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

Let J be the set of applications that have been deployed on the i_{th} 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 $t_3(k)$. So, it can be considered that $l'_j = l_j$ is approximately true. Then the following equation can be derived from (4):

$$L'_i - L_i = \sum_{j \in J} \left(\left(l'_j - l_j \right) \cdot P_j \right) \qquad J = \{ j \mid c_i^j = 1 \}$$

$$P_j = \frac{1/I_j}{\sum_{j=1}^{m} \left(1/I_j \right)} \tag{5}$$

where c_i^j has been defined in equation (1). So, the key problem to get the predicted value of $L'_i - L_i$ is how to calculate the value of $l'_j - l_j$ (described in section 4.B and 4.C). The estimated ALR of the j_{th} 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 the application package is the major part of W_j. Otherwise, V_j / B can be replaced with $(V_j / B) + (deployment\ time\ of\ application)$.

Note that The probability of *invoking miss* is generally determined by matrix C, the sequence of requests, and the workload of the system. In section 4.B and 4.C, the method to obtain $l'_j - l_j$ will be given.

B. ALT-MIN for Heavy Workload

In the case of a heavy workload, M_j (the probability of *invoking miss* of the j_{th} application) is considered approximately proportional to $1 - \frac{c^j}{n}$, when a request for the j_{th} 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*. So, 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. So, a busy node is considered to be randomly released. The probability of the *released node* being a *raw node* is approximately $1 - \frac{c^j}{n}$. The latency ratio of the current request is $l_j = \frac{(1 - \frac{c^j}{n}) \cdot (V_j / B)}{E_j}$. If the j_{th} application is selected and replaced, the new latency ratio is $l'_j = \frac{(1 - \frac{c^j - 1}{n}) \cdot (V_j / B)}{E_j}$. The increment of the latency ratio of the j_{th} application is

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

C. ALT-MIN for Light Workload

In the case of a light workload, M_j here is considered approximately proportional to $\left(1 - \frac{n_{idle}}{n} \right)^{c^j}$. 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 $1 - \frac{n_{idle}}{n}$. For the j_{th} application with c^j replicas, the probability of all nodes installed with the j_{th} application being busy at the same time is $\left(1 - \frac{n_{idle}}{n} \right)^{c^j}$. The latency ratio of the current request is $l_j = \frac{\left(1 - \frac{n_{idle}}{n} \right)^{c^j} \cdot (V_j / B)}{E_j}$. If the j_{th} application is selected and replaced, the new latency ratio is $l'_j = \frac{\left(1 - \frac{n_{idle}}{n} \right)^{(c^j - 1)} \cdot (V_j / B)}{E_j}$. The increment of the latency ratio of the j_{th} application is

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

If the workload is extremely heavy, n_{idle} is then approximately 1. Considering $n > c^j >> 1$, we can obtain:

$$l'_j - l_j \approx \frac{\left(1 - \frac{1}{n} \right)^{c^j - 1} \cdot \frac{1}{n} \cdot (V_j / B)}{E_j} \approx \frac{\frac{1}{n} \cdot (V_j / B)}{E_j} \tag{9}$$

which is the value of $l'_j - l_j$ in equation (7). So, the heavy workload can be considered as a special case of a light workload.

V. Evaluating Two ALR-MIN Strategies

We evaluate our ALR-MIN strategies for application replacement by performing a series of simulations on a well-designed simulator. Our two ALR-MIN strategies (for heavy and light workload systems respectively) are compared to each other and also with other two commonly used LRU-based strategies: Random-LRU and Cooperative-LRU.

In the rest of the section, the two LRU-based strategies to be compared with our strategies are described in Section 5.A. Our simulation methodology is discussed in Section 5.B, including the design of the simulator and the simulation settings. Last, in Section 5.C, the detailed simulation results are discussed.

A. Two LRU-based Strategies

As mentioned in Section 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 steps: 1) determine which node is selected to install the new application, 2) determine which installed applications of the selected node should be undeployed in order to make enough space for the new requested application. To study the performance improvement of applying our ALR-MIN strategies, Random-LRU and Cooperative-LRU are two LRU-based strategies to be compared with our ALR-MIN strategies.

- The Random-LRU strategy 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, LRU is called to choose applications to be evicted. Applications with the oldest (i.e., minimum) Last Access Time (LAT) are chosen to be evicted.
- Cooperative-LRU selects an appropriate node to put the new application on in the first step. The node with the minimum Last Access Time of Node (NLAT) is chosen to deploy the new application. NLAT is defined as the weighted arithmetic mean of application LATs on the node:

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

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, the Cooperative-LRU employs the method of LRU, which is similar to Random-LRU.

B. Simulation Methodology

A simulator has been developed to evaluate the 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.

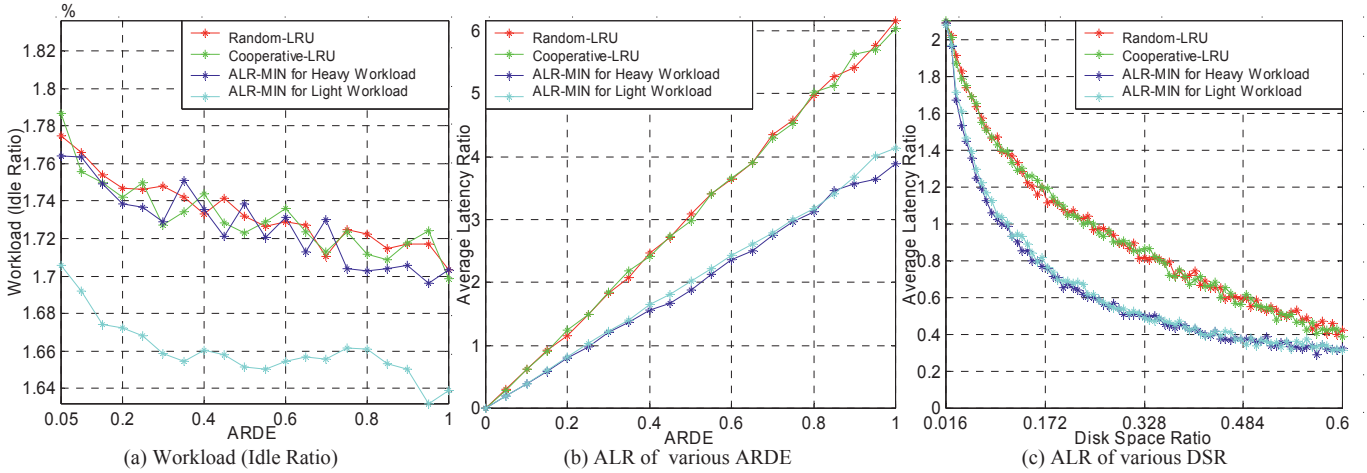
TABLE I. Detailed Setting In Simulations

PARAMETERS	ABBR.	VALUES		
Application Number	M	128		
Total Job Number	-	27000		
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 i_{th} 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 i_{th} application.		

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 AAR, DSR and ARDE can approximately represent the setting of the simulations. So, we define the triple set of [AAR, DSR, ARDE] to represent the simulation setting.

Table 1 shows the detailed settings of the important system parameters of the simulations. The expression of [X:Y:Z] in Column 3, Table 1 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 is obtained by calculating the weighted arithmetic mean of the last ten requests for a specified application.



(a) Workload (Idle Ratio) (b) ALR of various ARDE (c) ALR of various DSR

Figure 3. Performance of heavy workload.

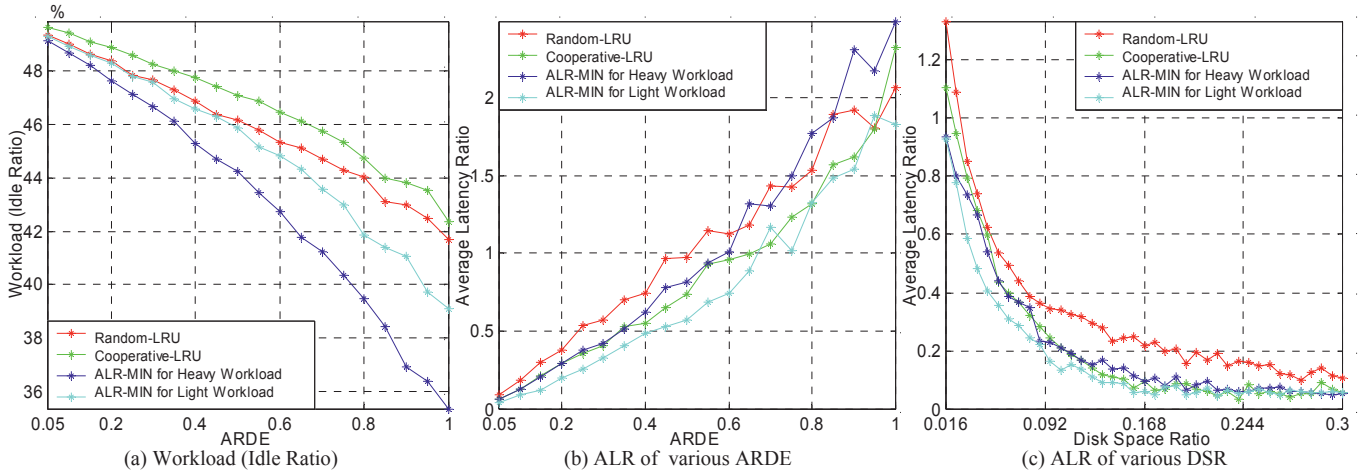
C. Simulation Results

Figure 3 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 with the MIMPA being 10. Figure 3(a) and Figure 3(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 3(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 3(b) shows the *average latency ratio* (ALR) with the same setting in Figure 3(a). The vertical axis is the ALR. We can observe from this figure that the performance 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 3(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 1). When the space becomes more sufficient and therefore less replacement is required, the performances of the four strategies are closer to each other.

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 4 shows the performance of the four strategies in the situation of a medium workload with the MIMPA being 200. Figures 4(a) and 4(b) have the same setting of [200, 0.064, *]. As shown in Figure 4(a), the percentage of idle nodes is within the range from 35% to 50%. It can be observed from Figure 4(b) that the ALR of ALR-MIN for Light Workload has the lowest percentage. In Figure 4(c),

the details of the ALR of the medium workload are presented with the setting of [200, *, 0.2]. When the *disk space ratio* is less than 0.244, ALR-MIN for Light Workload always outperform other three strategies. For a medium workload, the ALR-MIN for Light Workload is better than LRU-based strategies.

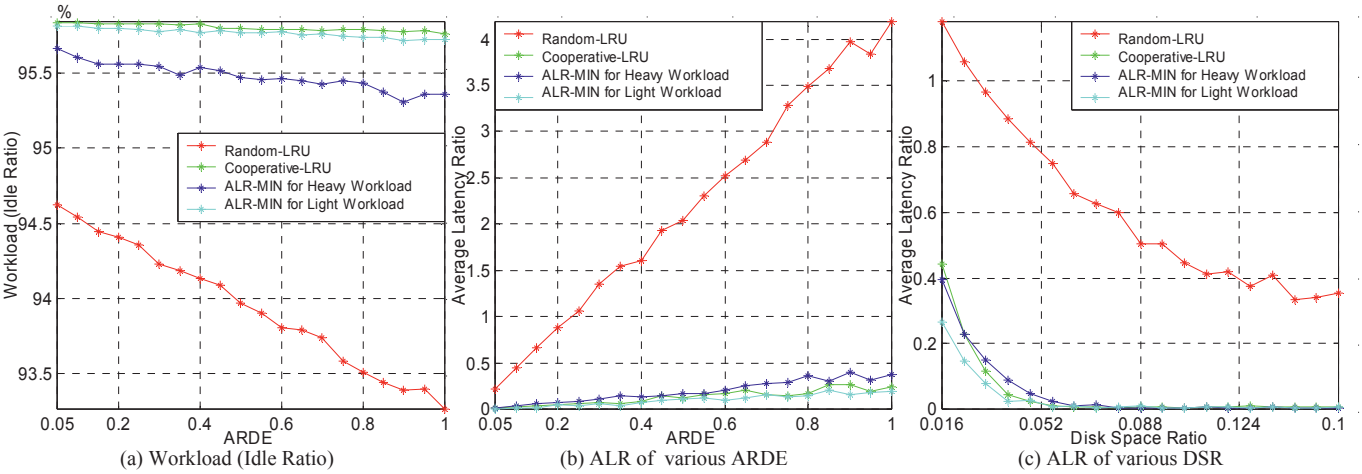


(a) Workload (Idle Ratio) (b) ALR of various ARDE (c) ALR of various DSR

Figure 4. Performance of medium workload.

Figure 5 shows the performance of the four strategies in the situation of a light workload with the MIMPA being 2000. Figures. 5(a) and 5(b) have the setting of [2000, 0.04, *]. As shown in Figure 5(a), the percentage of idle nodes is more than 90%. Figure 5(b) shows that ALR-MIN for Light Workload yields the lowest ALR, and Random-LRU gives the highest. Figure 5(c) presents the detailed ALR with a setting of [2000, *, 0.2]. It can also be seen from Figure 5(c) that the ALR-MIN for Light Workload is the best.

Based on the results presented in Figure 3, Figure 4, and Figure 5, 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 average latency ratio 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%.



(a) Workload (Idle Ratio) (b) ALR of various ARDE (c) ALR of various DSR

Figure 5. Performance of light workload.

VI. Related Works

Existing studies about dynamic deployment of applications in grid have been discussed in Section 1. In this section, we discuss what the existing studies have done in term of application replacement.

A LRFU-based replacement strategy is exploited in DAG-Condor [10] to keep useful files in disk. However, LRFU 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 [12], 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 nonuniform. Nonuniform-cost local replacement has also been addressed by some strategies, such as BCL, DCL and ACL proposed by Jaeheon [13], and Lowest-latency-first [14]. Their main drawback is that two cache objects with the same miss cost but different size 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 may be determined by not only its size but also the deployment or installation time cost. So, for applications with same miss cost, those applications with bigger size should be swapped out first. Thus, more space can be freed for new applications.

Other nonuniform-cost schemes (e.g., LRU-Threshold [15]) neglect the fetch cost of a block. GreedDual-Size [16] is only helpful in the case of single cache space.

Some other studies (e.g., [17], [18]) exploit cooperative strategy to solve multi-cache problems. However, they do not address the problem of where to put the requested data.

The data replication strategies (e.g., [19], [20]) in P2P Networks is similar to the application replication in our work. However, the optimization objective in our study is to decrease the miss rate, which is different from their objective.

VII. CONCLUSION

The traditional way of deploying applications - statically deploying applications to a pre-selected subset of computing nodes, in some cases, may utilize resources unbalancedly and lead to the overall performance of the system very low. Dynamic deployment is therefore a desirable way to handle this problem. In this paper, we propose two ALR-MIN replacement strategies, for heavy workload and light workload respectively to reduce the overhead caused by such a dynamic deployment approach. These two strategies select the node with the least estimated NALR to deploy a new application and select the old applications with the least estimated increment of ALR to be evicted for the node to make room for the new application.

A configurable simulator was developed and a set of simulations were conducted to evaluate our ALR-MIN application replacement strategies by comparing them with two commonly applied LRU-based strategies. The simulation results show that our ALR-MIN strategies (both for heavy and light workload) can result in a lower relative delay-time of jobs, compared with two traditional LRU-based strategies. ALR-MIN results in 17% less delay-time of jobs than the two LRU-based strategies with a typical setting of ARDE being 0.4 and DSR being 0.04.

ACKNOWLEDGMENT

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] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, vol. 15, p. 200, 2001.

[2] Y. W. Wu, S. Wu, H. S. Yu, and C. M. Hu, "Introduction to ChinaGrid support platform," in Parallel and Distributed Processing and Applications - Ispa 2005 Workshops. vol. 3759 Berlin: Springer-Verlag Berlin, 2005, pp. 232-240.

[3] D. Diwakar and S. Diwakar, "CINWEGS-an integrated Web and grid services framework for collaborative solutions," 2008 4th International Conference on Next Generation Web Services Practices, pp. 21-7, 2008.

[4] L. FuQiang, G. Bin, X. Cheng, and M. Yan, "Dynamic visualization service deployment in grid scientific workflow," 2008 Seventh International Conference on Grid and Cooperative Computing, pp. 201-5, 2008.

[5] L. Qi, H. Jin, I. Foster, and J. Gawor, "HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4," in Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on, 2007, pp. 155-162.

[6] L. Pu and M. J. Lewis, "Uniform Dynamic Deployment of Web and Grid Services," in Web Services, 2007. ICWS 2007. IEEE International Conference on, 2007, pp. 26-34.

[7] P. Watson, C. Fowler, C. Kubicek, A. Mukherjee, J. Colquhoun, M. Hewitt, and S. Parastatidis, "Dynamically deploying Web services on a grid using Dynasoar," in Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on, 2006, p. 8 pp.

[8] M. Siddiqui, A. Villazon, J. Hofer, and T. Fahringer, "GLARE: A Grid Activity Registration, Deployment and Provisioning Framework," in Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference, 2005, pp. 52-52.

[9] E. K. Byun and J. S. Kim, "DynaGrid: A dynamic service deployment and resource migration framework for WSRF-compliant applications," Parallel Computing, vol. 33, pp. 328-338, 2007.

[10] S. Shankar and D. J. DeWitt, "Data Driven Workflow Planning in Cluster Management Systems," in Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing Monterey, California, USA, 2007.

[11] "Application Contents Service Specification 1.0," http://www.ogf.org/documents/GFD.73.pdf

[12] N. Megiddo and D. S. Modha, "Outperforming LRU with an adaptive replacement cache algorithm," Computer, vol. 37, pp. 58-+, 2004.

[13] J. H. Jeong and M. Dubois, "Cache replacement algorithms with nonuniform miss costs," Ieee Transactions on Computers, vol. 55, pp. 353-365, 2006.

[14] R. P. Wooster and M. Abrams, "Proxy caching that estimates page load delays," Computer Networks and Isdn Systems, vol. 29, pp. 977-986, 1997.

[15] M. Abrams, S. Dept. of Computer, I. Virginia Polytechnic, and U. State, "Caching Proxies: Limitations and Potentials," in Proc. of 4th International World Wide Web Conference, 1995.

[16] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in Proceedings of the USENIX Symposium on Internet Technologies and Systems Monterey, CA,: USENIX Assoc, 1997, pp. 193-206.

[17] H. P. Hung and M. S. Chen, "On designing a shortest-path-based cache replacement in a transcoding proxy," Multimedia Systems, vol. 15, pp. 49-62, 2009.

[18] Y. W. Zhu and Y. M. Hu, "Exploiting client caches to build large Web caches," Journal of Supercomputing, vol. 39, pp. 149-175, 2007.

[19] E.Cohen and S.Shenker, "Replication strategies in unstructured peer-to-peer networks," in In Proceedings of ACM SIGCOMM'02, 2002.

[20] S. Tewari and L. Kleinrock, "Proportional Replication in Peer-to-Peer Networks," in INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, 2006, pp. 1-12.