J Supercomput (2011) 55: 51–68
DOI 10.1007/s11227-009-0315-4

Automatically constructing trusted cluster computing environment

Yongwei Wu · Chen Gang · Jia Liu · Rui Fang · Xiaomeng Huang · Guangwen Yang · Weimin Zheng

Published online: 9 July 2009

Abstract Trusted Computing is a technology proposed by the Trusted Computing Group (TCG) to solve security problems in computers. A lot of work has been conducted to support Trusted Computing for individual computers; however little has been done for distributed systems (e.g., clusters). If malicious or unqualified applications are deployed on cluster nodes, users may obtain forged results or their data may be leaked out. In this paper, a methodology of the Trusted Cluster Computing (TCC) is proposed to automatically construct user-trustable cluster computing environments. User-specified applications are downloaded from user-specified locations and automatically and dynamically deployed on cluster nodes. To reduce the dynamic-deployment overhead, a novel Heuristics-based Overhead-Reducing (HOR) replacement strategy is also proposed. A highly configurable simulator has been implemented to perform a series of simulations. The simulation results demonstrate that the HOR can produce Average Speedup with up to 14% (light workload), 10% (medium

Y. Wu · C. Gang (✉) · J. Liu · R. Fang · X. Huang · G. Yang · W. Zheng
Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China
e-mail: c-g05@mails.tsinghua.edu.cn

Y. Wu
e-mail: wuyw@tsinghua.edu.cn

J. Liu
e-mail: liu-jia04@mails.tsinghua.edu.cn

R. Fang
e-mail: fangr06@mails.tsinghua.edu.cn

X. Huang
e-mail: huangxiaomeng@gmail.com

G. Yang
e-mail: ygw@tsinghua.edu.cn

W. Zheng
e-mail: zwm-dcs@tsinghua.edu.cn

workload), 8% (heavy workload) higher than that of LRU-based strategies, with a typical setting of the Average Ratio of Deployment time to Execution time (ARDE) being 0.2.

Keywords Trusted Computing · Cluster Computing · Automatic Deployment · Distributed system · Replacement strategy

1 Introduction

Trusted Computing [1] is a technology proposed by the Trusted Computing Group [2] to solve security problems in computers. It is applicable to protect, for example, copyright of music and systems against viruses. Compared with traditional security technologies (e.g., ssh), Trusted Computing requires almost every component (e.g., CPU, memory, disk driver, and OS) of a system to participate in trusted behavior. Another important difference between Trusted Computing and traditional security technologies is that computers (hardware and software) are not fully under the control of owners of the computers. Many projects, including Microsoft NGSCB (Next-Generation Secure Computing Base) [3] and Intel TXT (Trusted Execution Technology) [4], have been conducted to support Trusted Computing.

Although a lot of works have been done to support Trusted Computing for individual computers, little has been done for distributed systems (e.g., clusters), which also face serious security problems. Generally speaking, clusters are placed in a remote computing center, which is usually geographically located far away from the users. The clusters' hardware and software are first installed and configured by the administrators of the clusters. Jobs are then submitted by the users and scheduled to the computing nodes by a cluster management software (e.g., OpenPBS [5]). The users always expect that any computation and data transferring during the process are safe and trusted. However, traditional distributed systems are not completely trustable because 1) applications already deployed on the cluster nodes may be unqualified according to the users' requests (e.g., version mismatch); therefore, the computation results may not be correct, and 2) malicious applications may return forged results or steal key information of the users.

To construct a trusted cluster computing system, we propose a strategy, called Trusted Cluster Computing (TCC), to address the problems we discussed above. In the TCC, a user submits a job together with the description of the applications required to accomplish the job request to a cluster. The Application Manager of the cluster is responsible for checking whether the specified applications have been deployed on the cluster nodes. If not, the Application Manager has to download the specified applications from the user-specified location, and deploy them onto cluster nodes. With this dynamic-deployment mechanism, user-trustable distributed systems can be constructed and therefore malicious or unqualified applications will never get a chance to perform any computation. Notice that this dynamic-deployment mechanism also introduces a certain overhead. On one hand, the capacity of a computing node is normally limited; therefore there is a need to repeatedly swap in and out applications on it. On the other hand, network bandwidth may be low, and additionally the applications often have a considerably large size.

To reduce the dynamic-deployment overhead discussed above, we propose a novel Heuristic-based Overhead-Reducing (HOR) replacement strategy, with which the node being deployed a new user-requested application on and applications to be evicted from this node to accommodate the newly requested application are carefully chosen, so that Average Speedup of the node can be maximized. Speedup of a job here is defined as the ratio of new completion time with some strategy to original execution time. Job speedup of an application here is defined as the Speedup of a job when the application is chosen to be deployed or undeployed. Job speedup of a node here is defined as the Speedup of a job when the node is chosen to deploy or undeploy some applications on it. Average Speedup of a node in this paper is defined as the mean value of Speedup for a group of jobs. Actually, "speedup" does not mean "faster" in this paper.

An evaluation function is designed to predict and compare the Average Speedups of candidate deployment nodes or evicted applications. The node with maximum predicted Average Speedup is chosen to place a new application, and the old applications with maximum predicted Average Speedup are chosen to be swapped out. Compared with traditional and widely implemented Least Recently Used (LRU)-based application replacement strategies, our HOR strategy considers not only access frequency but also size of software packages, number of instances and average execution time of applications.

Three main goals are achieved in this paper. 1) Our proposed TCC methodology allows an application to be automatically deployed on a computing node only if no instance of the application can be found on available nodes. Due to the limit of resource capacity, some less-worthy applications can be undeployed to release resource for the newly requested application, called application replacement. 2) We formalized the application replacement problem and propose the HOR replacement strategy to maximize estimated Average Speedup. 3) We implemented a highly configurable simulator and conducted a series of simulations to evaluate our HOR strategy by comparing it with two commonly applied LRU-based strategies. The simulation results show that jobs with our HOR strategy take the least relative delay-time to finish.

The rest of the paper is organized as follows. In Sect. 2, we describe the TCC architecture. Section 3 formalizes the application replacement problem in the TCC and presents the HOR strategy. The simulation methodology is presented in Sect. 4. Section 5 examines the performance of the HOR and LRU-based strategies. Related work is discussed in Sect. 6. Last, we draw a conclusion in Sect. 7.

2 Trusted Cluster Computing

In this section, the TCC methodology we proposed to construct a trusted cluster computing system is discussed, including the environment architecture of the TCC (Sect. 2.1), the layout of the TCC (Sect. 2.2), the application deployment of the TCC (Sect. 2.3), and the application replacement of the automated deployment (Sect. 2.4).

2.1 Trusted Computing environment architecture

Figure 1 presents the architecture of the Trusted Computing environment. At the bottom of architecture is the Trusted Platform Module specification, proposed by the TCG. According to this specification, a chipset, named Trusted Platform Module, is required for a computer to enable the Trusted Computing. It is used to keep a pair of RSA (an algorithm for public-key cryptography) public and private keys, which can be used to generate signature or encrypt/decrypt data. For security purposes, it is impossible for anyone to retrieve a copy of the RSA private key from the TPM (Trusted Platform Modules) [6] chipset. The Trusted Computing is heavily based on the safety of the private key.

The hardware level is the level above the TPM level. Many IT manufactories have their own products to support trusted computing. Intel TXT [4] is one of them, which can create a private environment for an application. With TXT, some specific memory area can also be guarded for some specific process and refuses the access from any other applications.

The upper level is the software level. Many features of trusted computing require not only the extension of hardware but also the support of operating systems (OS). Microsoft NGSCB [3] implements a part of trusted computing. Based on the TPM and other Trusted-Computing enabled hardware, the NGSCB provides functions such as storage and memory protection for NGSCB-enabled applications, which interact with the Windows OS through a special interface, named Nexus API.

The top level is for distributed systems. This level is currently neglected by researches on Trusted Computing. However, most of top High Performance Computing systems in the world currently are clusters. The computation therefore should be trusted not only in an individual computer but also in a distributed system like a cluster. What users care about most in a cluster are the applications deployed on it. The users should have rights to validate that the applications on a cluster system are qualified and trustable. If not, the users should be able to reconfigure the cluster and such a reconfiguration support should thus be provided in the cluster system. The Trusted Cluster Computing environment architecture is our solution proposed to satisfy this user requirement, which constructs a trusted cluster by automatically and securely deploying applications to its computing nodes.

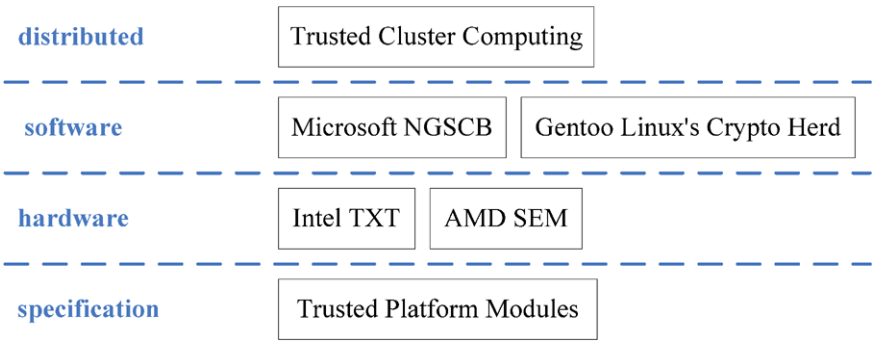


Fig. 1 Architecture of Trusted Computing environment

2.2 Layout of Trusted Cluster Computing

A typical layout of our Trusted Cluster Computing environment is shown in Fig. 2, which is composed of a switch and three computers (i.e., computing nodes of the cluster). Each of them is a traditional trusted computing system consisting of trusted hardware and OS. Therefore, the users can be assured that the hardware has not been changed and the applications have not been cracked. But this is not enough for trusted cluster computing. If the users observe that there are unqualified or malicious applications on some computing nodes, these computing nodes have to be automatically reconstructed. For an individual and personal computer, the users normally have privileges and are able to manually reconfigure the computer; however this is not the case for a cluster. In most clusters, administrators are the ones who own the rights to reconfigure the clusters. Allowing the administrators to reconfigure the clusters is insecure and inefficient.

Thus, an Application Manager, a cluster node responsible for automatically deploying and undeploying applications, is introduced in our TCC environment. The following four requirements should be satisfied in order to guarantee the security of the environment constructed by its Application Manager. First, before the Application Manager deploys a new application, all the chosen nodes have to attest the security of OS, previously deployed applications and the Application Manager itself. This function is supported by traditional trusted computing technologies (without

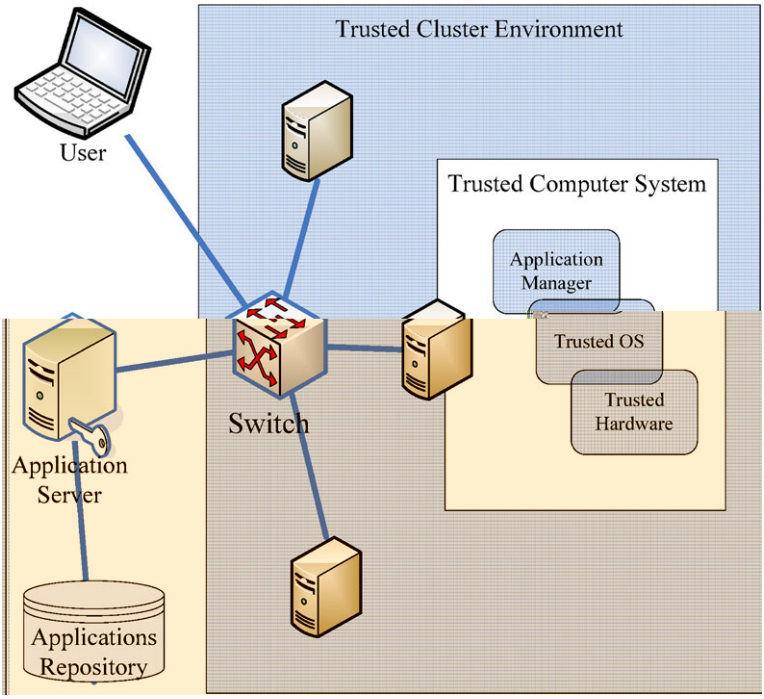


Fig. 2 Layout of the Trusted Cluster environment

Application Managers). Second, the applications to be deployed by the Application Manager should be specified by the users. The application specification can be submitted along with the computation request by the users. Third, the applications to be deployed should be obtained from a trustable location on network. The trustable location should also be specified by the users and authenticated by using the RSA key. Fourth, the applications should be safely deployed without receiving any potential attacks from malicious applications. This action can also be achieved by traditional Trusted Computing technologies, such as sealed storage and memory curtaining.

To support the above second and third requirements, an Application Server is located outside of the trusted cluster. As shown in the left side of Fig. 2, the Application Server keeps its own RSA public and private key, which can be used to authenticate identification of the Application Server itself. Application packages are actually stored in the Application Repository managed by the Application Server.

2.3 Automatically and securely deploy applications for Trusted Cluster Computing

The procedure of the Application Manager automatically and securely deploying an application to its cluster is presented as a UML (Unified Modeling Language) [7] sequence diagram in Fig. 3. Four major participators (User Agent, Application Manager, Application Server and Computing Node) are involved in the procedure. The whole procedure can be roughly divided into three interaction segments. In the first segment, the user agent first requests the computing nodes to attest their security. The main objective of this step is to validate that the Application Manager on each

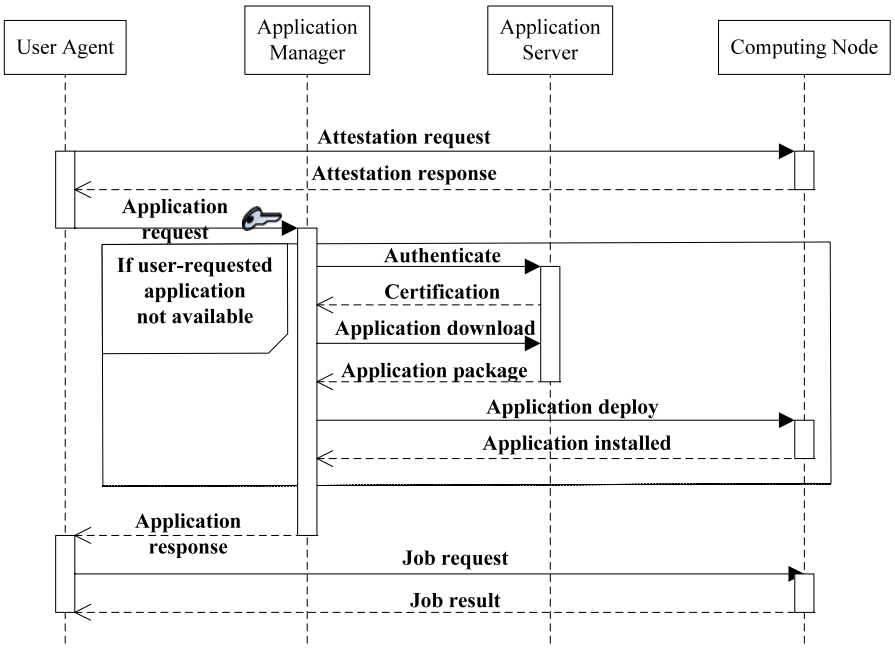


Fig. 3 Procedure of automatically and securely deploying an application

node is original and reliable by using traditional Trusted Computing technologies. Then the user agent submits an application request to the Application Manager of a node. Together with the request, the public key owned by the Application Server is also transferred to the Application Manager the node. In the second segment, if the application required by the user agent is available on this node, the response will be sent back to the user immediately; otherwise, the Application Manager communicates with the user-specified Application Server and authenticates its identification. Then, the application will be downloaded from the Application Server. The application is then deployed on the computing node by the Application Manager. In the last segment, the user agent submits a computation request to the computing node. The computing node calls the corresponding application to accomplish the request.

As shown in Fig. 3, the Application Manager is the key component of the TCC. Its data and memory area should be protected carefully. For example, the application package downloaded from the Application Server should be stored in a secure place like sealed storage. Any access requests from other processes should be prohibited.

2.4 Application replacement in Automatic Deployment

A detailed view of the Automatic Deployment of the TCC is presented in Fig. 4. In this view, the Application Repository and its computing nodes play key roles. The Application Repository stores application packages available to the users in its storage devices. The application packages are composed of either binary or source files, which can be transferred to and then installed on the computing nodes. As shown in Fig. 4, there are seven application packages (designated from 'A' to 'G') 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 application packages. For example, in Fig. 4, 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 Fig. 4 as an example, we describe the procedure of the Application Replacement as follows.

When a request for the application 'F' comes, the scheduler of the system fails to find any instance of the application 'F' on all available nodes, which is referred to as Invoking Miss. A node available but not deploying the requested application is termed a Raw Node in this paper. In the case of Invoking Miss, the scheduler suspends the request for the application 'F'. Since node 1 has sufficient space to place the application 'F', the Application Manager of the system is invoked to transfer the application package of 'F' and install it on node 1. When the transformation and the installation are done, the scheduler resumes the suspended request and schedules it to node 1.

When a request for the application 'G' comes, none of the nodes have enough space to install the application 'G', in which case the scheduler takes the following actions: first, it selects an appropriate node from all available nodes by following the application replacement strategy (described in Sect. 3); second, some of the deployed applications on the selected node are undeployed to release resource of the node for the user-requested application 'G'. For example, if the scheduler selects node 3 as

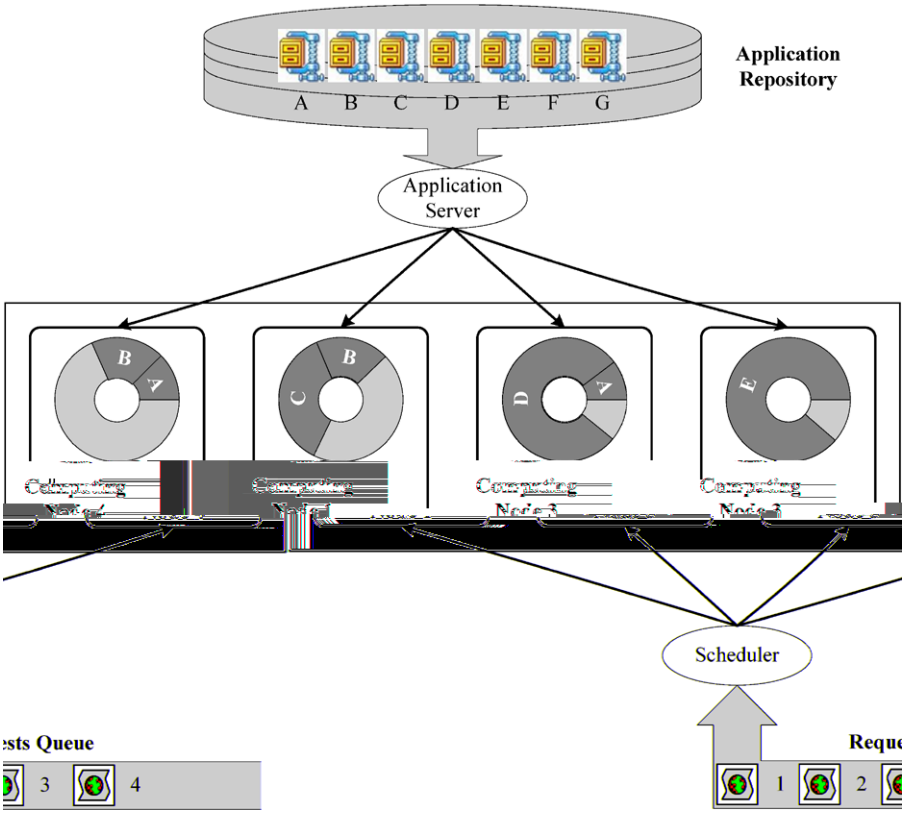


Fig. 4 Application replacement in automatic deployment

the node to deploy the application 'G', then the Application Manager undeploys the applications 'A' and 'D' to make room for the application 'G'. These two actions are key steps of the Application Replacement.

3 Application replacement strategies

As we discussed in the previous section, when none of the nodes has enough space to install a requested application, a strategy is required 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. This application replacement strategy has a significant impact on the overhead of Automatic Deployment. In this section, we formalize the application replacement problem by formally specifying the preliminaries of the strategy (Sect. 3.1), presenting the optimization objective (Sect. 3.2) and giving the Heuristic-based Overhead-Reducing (HOR) replacement strategies (Sect. 3.3).

3.1 Preliminaries

Notation used throughout this paper is as follows:

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 ith application package.

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

B = the bandwidth.

Applications are scattered on the nodes within a cluster. To denote the distribution of applications, a matrix is defined as follows:

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

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

The ith row in C means the array of the deployment status for all applications on the ith node, and the jth column in C denotes the array of the deployment status of the jth 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 entry c_i^j must be either 0 or 1. The number of instances of the jth application on all nodes can be obtained by $c^j = \sum_{i=1}^{n} c_i^j$.

3.2 Optimization objective

It is not desirable for the users to wait for a long time to get their requested applications deployed in a cluster; therefore the optimization objective of the application replacement strategy is to minimize the time cost of deployment and maximize the Average Speedup. To formalize the optimization problem, we define the Average Speedup of a request as $\frac{e}{e+w}$, 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 Speedup of all jobs is defined as

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

where E_i is the average execution time of requests for the ith application, L_j is the Average Speedup of all requests for the jth application, P_j is the probability that the

jth application is requested, K is the total number of requests, W_j is the deployment time of the jth application, and e_j^k is the execution time of the kth request for the jth application. Then, the optimization objective is formulated as:

$$\begin{aligned} &Obj.\ Maximize\ (AverageSpeedup) \\ &s.t. \quad \sum_{j=1}^{m}\left(c_i^j \cdot V_j\right) \le D, \quad 1 \le i \le n \end{aligned} \tag{3}$$

3.3 Heuristic-based Overhead-Reducing (HOR) replacement strategies

As shown in Fig. 5, processing the kth request is composed of four periods: *Suspended*, *UnDepoyment*, *Deployment*, and *Execution*. Only the periods *UnDeployment* and *Deployment* are related to application replacement. As we already discussed in Sect. 2.4, application replacement contains two key steps: selecting an appropriate node to install a requested application and undeploying selected applications to make room for the requested application.

For the first step, the node with maximum Average Speedup is chosen to place the requested application. $NASpeedup_i$ is defined as the increment of Average Speedup, caused by the undeployment of all the applications on an ith node. Let L_i be the speedup at the time of $t_2(k)$ and L_i' the speedup at the time of $t_3(k)$. Suppose all the applications on the ith node are undeployed during the time period from $t_2(k)$ to $t_3(k)$. Then $NASpeedup_i$ can be obtained by $L_i' - L_i$:

$$NASpeedup_i = L_i' - L_i = \sum_{j=1}^{m}\left((l_j' - l_j) \cdot P_j\right), \tag{4}$$

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

For the second step, the application(s) with the maximum increment of Average Speedup on the selected node is (are) chosen to be evicted. Let l_j denote the Average Speedup just before undeployment of the jth application, and let l_j' denote the Average Speedup after undeployment of the jth application. The increment of the

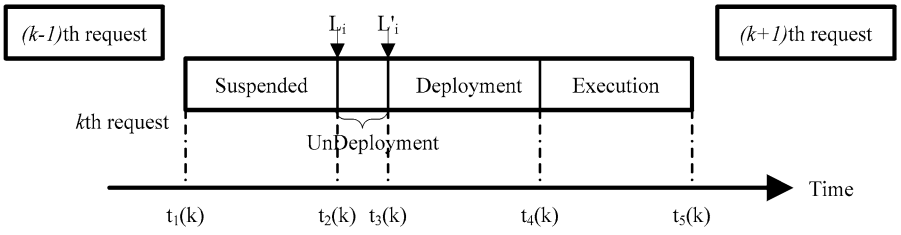


Fig. 5 Lifetime of the kth request

Average Speedup caused by evicting the jth application is $l'_j - l_j$:

$$l'_j - l_j = \frac{(1 - \frac{n_{\text{idle}}}{n})^{c^j - 1} \cdot \frac{n_{\text{idle}}}{n} \cdot E_j}{E_j + (V_j / B)} \tag{5}$$

Here E_j is the average execution time of the requests for the jth application, n is the total number of nodes in the cluster, n_{idle} is the number of idle nodes when a request comes, and c_i^j has been defined in (1).

4 Simulation methodology

Based on the Automatic Deployment model discussed in Sect. 2.4, a simulator was constructed to evaluate our HOR strategy. This simulator is composed of three components: Workload Generator, Virtual Resource and Job Scheduler. The Workload Generator generates a sequence of synthetic requests. The Virtual Resource records the status of virtual resources such as the information of where an application has been deployed and whether a job is running on a specified resource. Since the study of the scheduling strategy is beyond the scope of this paper, the Job Scheduler of the simulator always randomly schedules the current job to some resource and the requests are invariably processed with the order of their arrival times. When the requested application has not been deployed on any idle node, the Job Scheduler exploits the application replacement strategies discussed in Sect. 3 to choose a proper node and deploy it.

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 to all nodes. According to the value of idle ratio, the workloads of the system are classified into three levels: heavy, medium, and light. By adjusting the application arrival rate (AAR), all these three workload levels can be achieved. AAR is the mean arrival rate of requests for an application, and should be multiples 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. 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. The other simulation settings, such as application number, total job number, and node number, are all fixed. Finally, the number of jobs permitted to run concurrently in a node is not more than one. Table 1 shows the detailed information of the system workload of the simulations.

Table 1 Workload setting in the simulations

Workload (Idle ratio)	Maximum value	Minimum value	Average value
Heavy	1.84%	1.63%	1.72%
Medium	49.85%	35.32%	45.39%
Light	95.84%	93.26%	95.23%

5 Evaluating HOR strategy

We evaluate our HOR strategy for application replacement by performing a series of simulations using the simulator we designed. The HOR strategy is compared with two commonly used LRU-based strategies: R-LRU and C-LRU. In the rest of the section, the two LRU-based strategies are described in Sect. 5.1 and the detailed simulation results are discussed in Sect. 5.2.

5.1 Two LRU-based strategies

As previously discussed in Sect. 2.4, when a request comes and none of the nodes have enough space to install the requested application, an application replacement strategy should take the following steps: 1) determine which node is selected to install the application and 2) determine 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 applying our HOR, two commonly applied LRU-based strategies (R-LRU and C-LRU) are used to compare with our HOR.

R-LRU strategy: The R-LRU strategy exploits a simple approach of randomness to select a node in the first step. When an Invoking Miss (discussed in Sect. 2.4) is encountered, 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 with the oldest (i.e., minimum) Last Access Time (LAT) to be evicted.

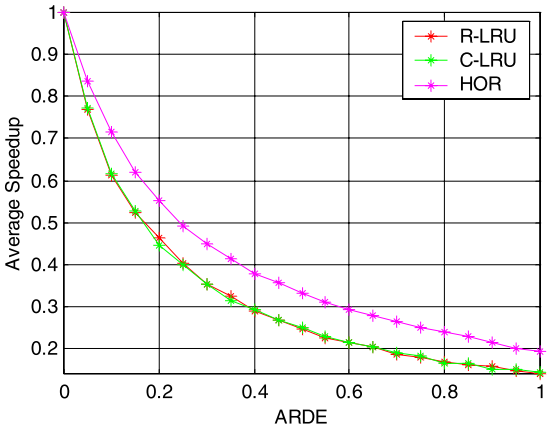
C-LRU strategy: In the first step, C-LRU selects an appropriate node, a node with the minimum Last Access Time of Node (NLAT), to deploy the new application. NLAT is defined as the weighted arithmetic mean of application LATs on the node:

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

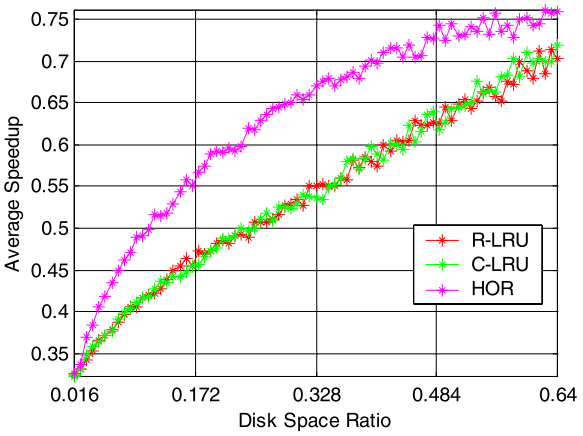
where j is the ID of the application and i is the ID of the specified node. In the second step, the C-LRU employs the method of LRU, which is similar to R-LRU.

5.2 Simulation results

Figure 6 shows the performance of the three strategies (i.e., R-LRU, C-LRU and HOR) in the case of a heavy workload with the MIMPA being 10. Figure 6(a) shows



(a) Speedup of various ARDE.

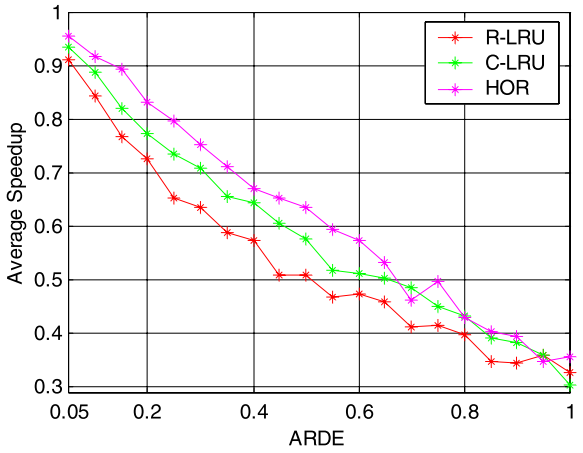


(b) Speedup of various Disk Space Ratio.

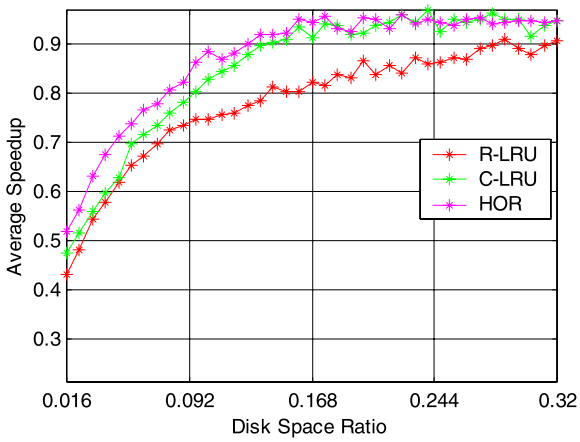
Fig. 6 Performance of HOR with heavy workload

the Average Speedup with the setting of [10, 0.16, *], where * means that the ARDE varies from 0 to 1. The first line in Table 1 gives its idle ratio. It can be observed that the percentage of idle nodes is only about 1.7%. The vertical axis denotes the Average Speedup (Sect. 3.2). The performance of R-LRU and C-LRU are similar and significantly worse than that of HOR. Figure 6(b) gives a detailed presentation of the Average Speedup with the setting of [10, *, 0.2], where * denotes that the DSR varies from 0.016 to 0.64. When the space becomes more sufficient and less replacement is required, the performances of the three strategies are closer.

R-LRU only takes the access frequency of applications into account. C-LRU neglects the number of replicas and the average execution time of applications. Different application sizes produce different time costs of deployment. The number of replicas affects the probability of deployment of an application. The average execution time of an application is one of the major factors determining the Average Speedup. HOR can evaluate the result of choosing different nodes and applications during the de-



(a) Speedup of various ARDE.

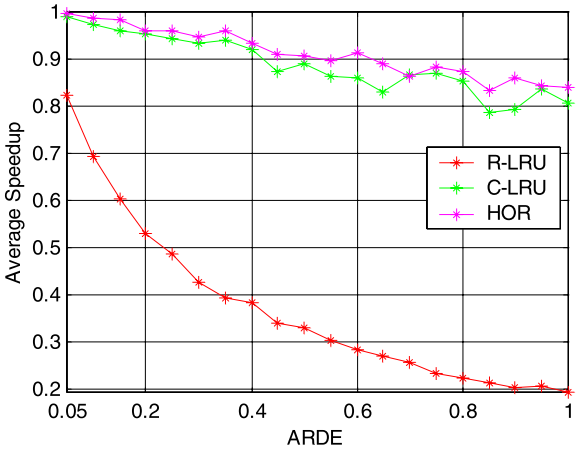


(b) Speedup of various Disk Space Ratio.

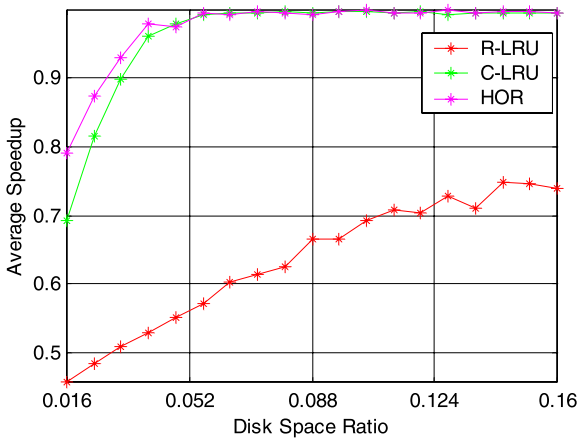
Fig. 7 Performance of HOR with medium workload

ployment and undeployment periods according to the access frequency, size, number of replicas, and average execution time. Thus, the HOR strategy can select the best node or application so that the decrease of Average Speedup can be minimized.

Figure 7 shows the performance of the three strategies in the case of a medium workload with the MIMPA being 200. Figure 7(a) has the setting of [200, 0.064, *]. As shown in the second line of Table 1, the percentage of idle nodes is within the range from 35% to 50%. It can be observed from Fig. 7(a) that HOR has the highest



(a) Speedup of various ARDE.



(b) Speedup of various Disk Space Ratio.

Fig. 8 Performance of HOR with light workload

Figure 8 presents the performance of the three strategies in the case of a light workload with the MIMPA being 2000. Figure 8(a) has the setting of [2000, 0.04, *]. As shown in the third line of Table 1, the percentage of idle nodes is more than 90%. Figure 8(a) shows that HOR for light workload yields the highest Average Speedup, and R-LRU gives the lowest. Figure 8(b) presents the detailed Average Speedup with a setting of [2000, *, 0.2]. It can also be observed from Fig. 8(b) that HOR is the best.

Based on the results presented in Figs. 6, 7, and 8, we can conclude that our HOR strategy can result in a shorter average delay-time of jobs than two LRU-based strategies in most cases. HOR can always give the highest or near-highest Average Speedup among the three strategies. With a typical setting of ARDE being 0.2, HOR can produce an Average Speedup with up to 14% (light workload), 8% (heavy workload) and 10% (medium workload) higher than LRU-based strategies.

6 Related works

6.1 Trusted Computing

Trusted Computing has been supported and adopted by many manufactories, including Intel and Microsoft. However the currently proposed technologies are only applicable for individual computers. Trusted Computing in distributed system has not been well explored.

The TPM [6] specification, proposed by TPG [2], envisioned a standard PC platform equipped with a TPM chip. A hardware private key is stored in the TPM chip. This key can be used to sign the specific system status including deployed software. Then the signature can be used to check whether the current running environment is the same as the original one.

Intel proposed the Trusted Execution Technology (TXT) [4] to support trusted computing. TXT extends the hardware of processors to provide a much safer computing environment. It consists of several hardware enhancements including a TPM. Isolated execution environments can be created to prevent malicious attack. Key data can be signed and guarded by the TPM. AMD also has their technology for trustworthy computing, named Secure Execution Mode (SEM) [8].

Next-Generation Secure Computing Base (NGSCB) [3] is the solution for trusted computing from Microsoft. It is based on the TPM 1.2 specification, Intel's TXT and AMD's SEM. In NGSCB, there are two software components: the Nexus, which is a security kernel that is part of the Operating System, and Nexus Computing Agents (NCAs), which are trusted modules within NGSCB-enabled applications.

The above technologies can support the construction of a trustworthy computer. Remote Attestation, Sealed Storage, Memory Curtaining and Secure I/O are implemented and ensured to support trusted computing on an individual computer. However, they cannot support trusted computing for distributed systems, which have aroused wide concern recently. This paper makes effort to construct an environment of Trusted Cluster Computing (TCC).

6.2 Replacement strategy

Local cache replacement strategies have been investigated for a long time, especially in virtual storage. Traditionally and frequently used schemes include ARC [9], FIFO, LRU, LFU, LRU-2, 2Q, LIRS, FBR, LRFU 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, ACL proposed by Jaeheon [11], and Lowest-latency-first [12]. Their main drawback is that two cache blocks with the same miss cost but different size are treated equally. Other nonuniform-cost schemes (e.g., LRU-Threshold [13]) neglect the fetch cost of a block. GreedDual-Size [14] is only helpful in the case of single cache space.

Some other studies (e.g., [16, 17]) exploit cooperative strategy to solve multi-cache problem. However, they do not address the problem of where to put the requested data.

The data replication strategy [18, 19] in P2P Networks is similar to the application replication in our work. [10], [15] and [20] proposed by Keqiu Li mainly target to solve the problems on multimedia and web caching. However, the optimization objective in our study is to decrease the miss rate, which is different from their objective.

7 Conclusion

Trusted Computing in individual computers has been supported and adopted by many organizations and manufactories; however, little work has been done to construct trusted computing for distributed systems like clusters. Trusted Cluster Computing is proposed in this paper to construct trusted computing environment for distributed systems. In Trusted Cluster Computing, user-specified applications are automatically and securely downloaded from a user-specified location and deployed to cluster computing nodes. Thus, users can obtain a trusted cluster environment and be protected from unqualified or malicious applications.

To reduce the overhead of dynamic deployment of the TCC, a novel HOR (Heuristic-based Overhead-Reducing) replacement strategy is also proposed. It considers not only access frequency like traditional LRU-based strategies but also the size of software package, the number of application copies and the average execution time of applications. A simulator is designed, implemented and used to perform a set of simulations. The simulations show that the HOR can produce an Average Speedup with up to 14% (light workload), 8% (heavy workload) and 10% (medium workload) higher than LRU-based strategies, with a typical setting of ARDE being 0.2.

Acknowledgements This work is co-sponsored by Natural Science Foundation of China (60573110, 90612016, 60673152, 60773145), National High-Tech R&D (863) Program of China (2006AA01A101, 2006AA01A106, 2006AA01A108, 2006AA01A111, 2006AA01A117), and National Basic Research (973) Program of China (2003CB317007, 2004CB318000).

References

1. Pearson S (2002) Trusted computing platforms. Prentice Hall International, Englewood Cliffs
2. Buyya R, Abramson D, Giddy J, Stockinger H (2002) Economic models for resource management and scheduling in Grid computing. Concurr Comput-Pract Exp 14(13–15):1507–1542
3. Peinado M, Chen Y, England P, Manferdelli J (2004) NGSCB: a trusted open system. In: Lecture notes in computer science, vol 3108. Springer, Berlin, pp 86–97
4. Intel Trusted Execution Technology (2009) http://download.intel.com/technology/security/downloads/315168.pdf
5. OpenPBS home page (2009) http://www.openpbs.org/
6. Trusted Platform Module (TPM) Specifications (2009) https://www.trustedcomputinggroup.org/specs/TPM/
7. OMG Unified Modeling Language Specification (2009) http://www.omg.org/docs/formal/00-03-01.pdf
8. Trusted Computing Group (2009) https://www.trustedcomputinggroup.org/
9. Megiddo N, Modha DS (2004) Outperforming LRU with an adaptive replacement cache algorithm. Computer 37(4):58
10. Li K, Shen H (2005) Coordinated enroute multimedia object caching in transcoding proxies for tree networks. ACM Trans Multimed Comput, Commun Appl (TOMCAPP) 5(3):289–314

11. Jeong JH, Dubois M (2006) Cache replacement algorithms with nonuniform miss costs. IEEE Trans Comput 55(4):353–365
12. Wooster RP, Abrams M (1997) Proxy caching that estimates page load delays. Comput Netw ISDN Syst 29(8–13):977–986
13. Abrams M, S Dept of Computer, I Virginia Polytechnic, State U (1995) Caching proxies: limitations and potentials. In Proc of 4th international world wide web conference, 1995
14. Cao P, Irani S (1997) Cost-aware WWW proxy caching algorithms. In: Proceedings of the USENIX symposium on Internet technologies and systems, Monterey, CA. USENIX Assoc, Berkeley, pp 193–206
15. Li K, Shen H, Chin FYL, Zhang W (2007) Multimedia object placement for transparent data replication. IEEE Trans Parallel Distrib Syst 18(2):212–224
16. Zhu YW, Hu YM (2007) Exploiting client caches to build large Web caches. J Supercomput 39(2):149–175
17. Dahlin MD, Wang RY, Anderson TE, Patterson DA (1994) Cooperative caching: using remote client memory to improve file system performance. In: Proc of first symposium on operating systems design and implementation, pp 267–280
18. Tewari S, Kleinrock L (2006) Proportional replication in peer-to-peer networks. In: Proceedings of 25th IEEE international conference on computer communications (INFOCOM), pp 1–12
19. Cohen E, Shenker S (2002) Replication strategies in unstructured peer-to-peer networks. In: Proceedings of ACM SIGCOMM'02, 2002
20. Li K, Shen H, Chin FYL, Zheng SQ (2005) Optimal methods for coordinated enroute Web caching for tree networks. ACM Trans Internet Technol (TOIT) 5(3):480–507