Adapting grid applications to safety using fault-tolerant methods: Design, implementation and evaluations

Xuanhua Shi a,*, Jean-Louis Pazat b, Eric Rodriguez c, Hai Jin a, Hongbo Jiang d

a CGCL/SCTS, School Comput., Huazhong Univ. Sci. & Tech., Wuhan, 430074, China
b IRISA/INSA de Rennes, Campus de Beaulieu, 35042 Rennes, France
c CEDRAT, 15 Chemin de Malacher-Inovalée, 38246 MEYLAN, France
d EIE Department, Huazhong Univ. Sci. & Tech., Wuhan, 430074, China

A R T I C L E I N F O

Article history:
Received 2 October 2008
Received in revised form 26 July 2009
Accepted 27 July 2009
Available online 31 August 2009

Keywords:
Adaptive
Safety
Fault-tolerance
Component

A B S T R A C T

Grid applications have been prone to encountering problems such as failures or malicious attacks during execution in recent years, due to their distributed and large-scale features. The application itself, however, has limited power to address these problems. This paper presents the design, implementation, and evaluation of an adaptive framework—Dynasa, which strives to handle security problems using adaptive fault-tolerance (i.e., checkpointing and replication) during the execution of applications according to the status of the Grid environments. We evaluate our adaptive framework experimentally using the Grid5000 testbed and the experimental results have demonstrated that Dynasa enables the application itself to handle the security problems efficiently. The starting of the adaptive component is less than 1 s and the adaptive action is less than 0.1 s with the checkpoint interval of 20 s. Compared with non-adaptive method, experimental results demonstrate that Dynasa achieves better performance in terms of execution time, network bandwidth consumed, and CPU load, resulting in up to a 50% lower overhead.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

In recent years, Grid computing has been very popular in high performance computing due to its capacity of aggregating high-performance computational and large-scale storage resources distributed over the Internet. Grids make high performance computing possible from the resources in one institute controlled by one group of users to resources over the Internet controlled by different groups of users [1]. On the downside, Grids also pose many problems which make dependable Grid computing a great challenge [2]. Since the virtual organization inherently is a dynamic environment, Grid applications have to communicate with untrusted resource or users. Although most Grid infrastructures protect themselves against malicious attacks with portal systems, Grid services are typically maintained behind network firewalls with holes punched through firewalls in order to allow applications, resources, and services to communicate with others [3]: These permanently opening ports potentially allow potential intruders to determine what services are listening on those ports, thus increasing the chance of the system getting attacked. Improving the availability of Grid applications is the object of this paper.

Most applications on Grids are HPC applications, which easily expose them to malicious attacks—one successful attack may cause them to lose days’ or weeks’ results. In view of the dynamic nature of the Grid and the long running times of applications, we present an adaptive framework, Dynasa, that handles malicious attacks during runtime. Dynasa enables applications to execute in an adaptive way in order to handle the security problems using fault-tolerance methods. This feature is desirable for Grid applications because Grid users often have limited control over the computing environment, and system administrators cannot always meet all the requirements from a large number of users and applications. Dynasa is based on the Dynaco component, which is able to modify its behavior during runtime [4]. Dynasa makes adaptive decisions according to changes of security levels, and the adaptive actions are based on fault-tolerance techniques (i.e., checkpointing and replication). For example, when one site is under attack, Dynasa will be able to migrate applications running on that site to another safe site. Dynasa works in a centralized way, thus, Dynasa only targets service Grids, not the P2P Grid [5,6].

This remainder of this paper is organized as follows. Section 2 discusses related work; Section 3 presents the framework of Dynasa; Section 4 describes the implementation of Dynasa; The experimental evaluation and performance analysis are presented in Section 5, and we conclude our work in Section 6.

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
2. Related work

Fault tolerance techniques for HPC have been intensively studied. Among those techniques, rollback recovery is the most studied one [7]. The two main elements of rollback recovery are message-logging and coordinated checkpoints [8]. Most researches have focused on restarting the failed application on the same resources, so the application has to wait for the recovery of the resources. In [9], a LAM/MPI [10] process migration method is presented. The LAM/MPI checkpoint/restart method stores checkpoints images locally, requiring the system to guarantee the security of every node. We adopt the MPICH-V [11] in our implementation, for MPICH-V can store the checkpoint images of every MPI process on the checkpoint servers; if we can keep the checkpoint servers safe, the safety of the whole application can be guaranteed. Replication is another widely researched fault-tolerance technique [12]. Reference [13] presents an architecture for wide-area replication that handles Byzantine faults by constructing logical machines out of the collection of physical machines. Atomic broadcast and consensus are crucial primitives of replication [14]. An adaptive replication middle-ware system is presented by the MEAD research group [15]. Most of the replication techniques for enhancing availability are used in such applications where the scale of data processing is small. In Grid computing, users can reserve a number of nodes for data processing. In [2], it is proved that the replication with checkpoint method provides the highest availability for Grid applications, and Dynasa takes similar methods presented in [2] with adaptive technology to target safety problems in Grid computing.

Adaptive computing is not a new concept. For example, an adaptive scheduling method is presented in [16]. This method does not give developers full control of applications. In [17], a framework that allows an easy and efficient execution of jobs in a "submit and forget" fashion is described. This framework automatically performs the steps involved in job submission and also watches over its efficient execution. In [18], a Program Control Language is proposed, which provides a novel means of specifying adaptations in distributed applications. Gorenèr et al. present an adaptive programming model for fault-tolerant distributed computing, which provides upper-layer applications with process state information according to the QoS [19]. In [20], a component-based autonomous repair management service for distributed systems is presented, which provides the adaptive replication ability to manage the subsystems. These prior works have similar concepts to our work; the difference is that our work focuses on providing an adaptive component which targets security threats for HPC applications at the application level using fault-tolerant methods.

Numerous malicious attack countermeasures have been proposed in the literature (e.g., the hop-count filtering in [21], the puzzle auctions method in [22], and the Bitmap Filter in [23]). Those techniques have one thing in common: they all focus on coping with attacks at the network level. For example, in [21], the Hop-Count filtering filter the spoofed IP packets by an IP to hop-count mapping table. Y. Amir et al. propose using accountability graphs to cope with malicious clients [24]. This work is similar to ours. Their work does not consider malicious attacks detection and focuses on finding out what the damages of these attacks are, while Dynasa focuses on enabling the application to run smoothly even in the presence of attacks. Dynasa deals with malicious attacks at the application level using fault-tolerant methods, because the system dependability and the system security share many characteristics [25]. Veríssimo et al. discuss some strategies and mechanisms for intrusion tolerance systems [26], which present a good reference for us. However, Dynasa handles the intrusion for service Grids with a centralized adaptive component.

3. Framework

The Dynasa framework is composed of two parts, the safety sensor, and the adaptive controller, as illustrated in Fig. 1. The functions and adaptive strategies of the framework’s components will be introduced in turn.

3.1. Components of Dynasa

The adaptive controller is a component. It includes Decider, Planner, Executor, Action and Service, which are also components. Their relationships are shown in Fig. 1.

- The Decider decides which strategy should be taken for the planner according to the dynamic information provided by the safety sensor. The information can be sent in two ways: the safety sensor can notify the decider, or the decider can pull information from the safety sensor. The decider chooses strategies based on Policy, which describe component-specific information required to make the decisions. The policy in our framework is a set of rules for keeping applications safe, such as resubmitting jobs and creating replicas. The decider has already been implemented in Dynaco; based on it we add new policies to handle the security problems.

- The Planner is a program that makes adaptation plans, which instruct the component to adopt a strategy given by the decider, and the plan-making is based on the Guide. The planner has been implemented in Dynaco; we add new guides to target the security problems. For example, the guide for “migrate the application” can be: checkpointing process; stopping the application, redeploying the application in another domain, and starting the application with checkpoints.

- The Executor is a virtual machine that interprets the plan provided by the planner by means of Actions. For example, checkpointing is an action. In addition, the executor handles the replica management semantic for secure applications; the ReplicaManager fulfills this function. Compared with Dynaco, the ReplicaManager is a new component, and the ReplicaManager can handle security problems for Grid applications.

- The Service is the program of a Grid application wrapped as a component.

The safety sensor is a component deployed outside the adaptive controller, which provides the subscription and notification interface for the adaptive controller to obtain information about environmental status. These statuses include potential Denial of Service (DoS) attack incoming, malicious content attack incoming, node failure, just to name a few.

3.2. Protocols

There are three different types of security problems in Service Grid applications described in [27]. Accordingly, in this work,
we propose three protocols to guarantee the availability of Grid applications. These protocols are implemented as policies and guides in Dynasa. For illustration, we first present two terms which will be used later.

**Checkpoint wave** refers to a state in which all the MPI processes of the application have completed making a checkpoint in a coordinated way.

**Checkpoint servers** refer to the nodes onto which the checkpoint images of the MPI processes are stored. To save storage space, we only store the checkpoint images of the latest two checkpoint waves.

The first protocol targets the situation where there are some potential attacks notified by the safety sensor. For example, there may be denial of service attacks existing in the clusters over which the application is running. To handle potential attacks, the adaptive controller creates a passive replica on another site with checkpoint files, as shown in Fig. 2. The adaptive controller sends a ‘create’ message to the running application, which then creates a passive replica, after that the passive replica sends an ‘ack’ message to the running application. Upon receiving the ‘ack’ message, the running application sends an ‘ack’ message to the adaptive controller. By doing so, the replication group is created, and the running application runs as the primary replica. Later, the adaptive controller sends an ‘update’ message to the primary replica, which synchronizes ‘R1’ after receiving the message; and ‘R1’ sends an ‘ack’ message to the primary replica after it has been synchronized. The ‘update’ action is performed for every checkpoint wave of the application. The new replica runs passively, so it only receives synchronization messages and updates its states if it is not activated by the controller.

The second protocol targets the problems that the resources over which the application runs are attacked by malicious users, which can lead to denial of service of those resources. As depicted in Fig. 3, the application first creates two replicas on other sites after receiving the ‘create’ message from the controller. When the replicas are created, they send ‘ack’ messages to controller, and the running application is killed by the controller. Dynasa chooses a primary site, where the application is anticipated to have the shortest execution time, as the primary replica site. Consistent with [28], the execution time $T_E$ can be estimated by: $T_E = (T_T/T_{1}) e^{i(T_{T}/T_{c}) + k_{2} / \lambda}$, where $T_T$ is the failure-free execution time of the application, $T_{T}$ is the checkpoint interval, $T_{T}$ is the checkpoint transfer time from current site to the target site, $T_{c}$ is the checkpoint latency, $T_{s}$ is the restart time with the checkpoint, and $\lambda$ is the dos attack rate. After a checkpoint wave, the primary site on which the new application runs synchronizes the replication states, as explained in protocol 1.

The third protocol targets malicious content attacks that will produce untrustworthy results—it is usually difficult to determine whether or not the results have been tampered with by malicious users. In this paper, we do not target the detailed protocols handling Byzantine faults as illustrated by Schneider and Castro [29,30], we only illustrate what action Dynasa will take according to the attacking events. For this situation, the adaptive component first creates a replica using the checkpoints of the checkpoint wave. This checkpoint wave is the penultimate, instead of the latest checkpoint wave, shown in Fig. 4. The reason the penultimate one is chosen is that we make the certification test before taking checkpoints, and thus we can trust those checkpoints, as shown in Fig. 4. After creating the new replica, the running application is killed. The controller consequently sets the new replica as the primary replica, and sends another ‘create’ message to the new primary replica to create a new replica ‘R2’, as shown in Fig. 4. Later, the replication group is synchronized as in protocol 1.

When the security level of the environment is increased, the adaptive controller will kill one passive replica in order to reduce the overhead of replication management. Because this action only kills the passive replica, the system synchronization is easy to perform. The desirable behavior of Dynasa is that the adaptive controller can make adaptive decisions during the execution of the applications according to any security events introduced above.

4. Implementation

Dynasa consists of several components, as shown in Fig. 5. The safety sensor obtains the safety information with the snort tool (snort is a widely-used open-source tool for intrusion detection/prevention). The adaptive controller gives a replication instruction to the replication manager. The replication manager consists of group communication and synchronization manager.

4.1. Adaptive controller

As mentioned in Section 3, the adaptive controller provides a framework to input policies and guides, which allow the programmer full control of the application. We add policies and
Replication manager guides to deal with security problems in Grid environments. The policies are based on the protocols presented in Section 3.2, given in Fig. 6: (1) when there are some potential attacks presented, the replication manager should create a passive replica on a remote site; (2) when there are some DoS attacks on the running processes, the replication manager should create two replicas, and switch the primary replica site to the new replica site; (3) when there are some malicious content attacks which attack the running processes, the replication manager creates a new replica from the last correct checkpoints, switches the primary to the new replica, and creates another replica based on the new primary one; (4) when the systems increases the security control, the replication manager will kill one replica to reduce replica management overhead. With these policies, the application can change its running states during its execution according to different safety events.

4.2. Replication manager

As shown in Fig. 5, the replication manager is composed of two parts: the reliable group communication, which is based on Spread [31], and the replica synchronization manager. First, the reliable group communication provides three functions: joining/leaving groups, detecting the failures of replica, and reliable multicast. When one new replica is created, it joins the replication management group, and sends its states to the group of replicas. Based on the group message, the replication manager knows whether the replica is running in good states or not. If one replica fails, the replication manager will take some action to correct the faults, such as creating one new replica or making synchronization actions to change the status. Second, the synchronization manager is a service which synchronizes the status of every replica. For the MPICH2 application, the synchronization manager is a program which updates the latest checkpoints images from the primary replica site to other replica sites, and the copying of checkpoint files is performed in parallel to reduce transferring time.

4.3. Fault tolerance replicas

We implement the creation of the replica of the HPC application based on the MPICH-V project. On every replica site, there are three services deployed: incremental logging service, checkpoint service and restart service. The incremental logging service logs the new state of the application and the nodes, which is used by the synchronization service. The checkpoint service is based on an abstract checkpointing mechanism, which provides a unified API to address system-level task checkpointing. The Berkeley Linux Checkpoint/Restart library (BLCR) [32] is used in our work to implement the checkpoint service. We use a coordinated checkpoint protocol, called Pcl [7], to coordinate the checkpoint operation of every MPI process. MPICH2-Pcl is a new protocol in the MPICH-V project. In MPICH2-Pcl, there is a fault tolerant process manager (FTPM), which performs starting, managing, detecting failures, and restarting MPI applications. While MPICH2-Pcl uses mpiexec to perform the checkpointing and restarting tasks, and mpiexec itself cannot be stopped. Here mpiexec is a process to start the MPI application. If mpiexec crashes, all the MPI applications will fail and the application cannot recover. To be able to restart from a specific checkpoint state of the MPI application, we modify the mpiexec and FTPM to support recovering from the crash of the whole MPI program (The modified MPICH-Pcl is available on the website http://mpich-v.lri.fr/index.php). Based on this restarting service, the application can be restarted on any replica site with checkpoint files.

5. Performance evaluation

In this section we present the performance results of the implementation. We conduct all the experiments on the Grid5000 [33] testbed. Grid5000 is a physical platform featuring 13 clusters, connected by the Renater French Education and Research Network. We use 4 clusters with AMD Opteron dual-processors: a 47-node cluster at Nancy, a 216-node cluster at Orsay, a 99-node cluster at Rennes, and a 40-node cluster at Sophia. One major feature of the Grid5000 project is its ability to allow a user to setup his own computing environment including operating system, software versions, libraries, etc. on all the computing nodes reserved for his jobs. We exploit this feature to run all our measurements in a homogeneous environment including the Berkeley Linux Checkpoint/Restart library. All the nodes run Linux 2.6.13.5. The tests and benchmarks are compiled with GCC-4.0.3 (with flag-03), and the adaptive component runs over Sun JVM 1.5.0.11. All experiments are conducted using NAS parallel benchmarks (NPB-2.3) [34] developed by the NASA NAS research center. BT benchmark is one of the HPC application test suite of NPB which is widely used in cluster computing and grid computing [7].

5.1. Cluster-based experiments

We first test the performance on the cluster Paravent at Rennes. Each node is equipped with 2 GHz AMD Opteron 246 dual-processors and 2 GB memory, each node features 20GB of swap and SATA hard drives. All nodes are connected with a Gigabit Ethernet network. We select six different numbers of checkpoints servers (1, 2, 3, 4, 6 and 8). A total of 80 nodes on Paravent were used. We run the BT benchmark of class B with 64 processors (BT.B.64, over
32 dual-processor nodes). 64 nodes are used for creating replicas for NPB computation, and 16 nodes for checkpoint servers. The cluster-based experiments are for the second protocol introduced in Section 3.2 because of its two operations that have the most impact on the application’s performance: transferring checkpoint images and restarting the application from specific checkpoint images. To test the adaptability of the application, we inject DoS attack events to trigger the application's adaptation function with specific mean time to be attacked (MTTA). To inject DoS attack events, we generate ICMP traffic to emulate the flooding traffic from DoS attacks. Although there exist many DoS models, we only inject this type of DoS attack for the purpose of demonstration. We use a single-stream ping with 1500-byte Ethernet packets. Checkpoints are triggered by timeouts.

We choose three different checkpoint intervals (10 s, 20 s, and 30 s), and two MTTAs (40 s, and 50 s). We also measured the execution time with the specific mean time to failure (MTTF) of NPB, and the MTTF is generated by fault injection. To inject faults for NPB, the MPI processes of the NPB computation are killed by another timer-controlled process. We first study the relationship between the MTTF and the execution time for HPC applications. To show the influence between DoS attack and hardware/software failure, we also give a comparison between MTTF and MTTA.

Fig. 7 shows the execution times of the BT benchmark of class B with different MTTFs and MTTAs. In Fig. 7, the curve labelled ckptxRy represents the case with checkpoint interval x seconds and MTTF y seconds; the curve labelled ckptxDy represents the case with checkpoint interval x seconds and MTTA y seconds. We can make the following observations from Fig. 7:

1. For one checkpoint server, the execution time with any MTTF is very long. Because the checkpoint server has to wait to transfer and write the checkpoint image. Although the network is gigabit Ethernet, the disk I/O performance is the bottleneck of checkpoint servers. For more checkpoint servers, the execution time is much shorter, as shown in Fig. 7. However, we should highly secure the checkpoint servers to make the result trustworthy, the more checkpoint servers we take, the more overhead we have.

2. For different checkpoint intervals, a shorter checkpoint interval leads to longer execution time, because of the checkpoint overhead. When handling the DoS attacks or faults, things are different. In Fig. 7(a), for 3 checkpoint servers, the execution time with MTTF 40 s and checkpoint interval 10 s is shorter than that with MTTF 40 s and checkpoint interval 20 s, while for 2 checkpoint servers, the latter is shorter. The reason for this can be explained as follows: if the checkpointing overhead is low enough, a shorter checkpoint interval will lead to a shorter execution time, but if the checkpointing overhead is so high that network transfer becomes the dominant factor of execution delay, the execution time will be longer even for a short checkpoint interval.

3. Comparing curve ckptxRy and ckptxDy, we find that the execution time difference between handling failures and handling DoS attacks is not significant when we have more than 4 checkpoint servers, indicating that the DoS handling method is efficient when the network and the I/O performance is good.

4. Comparing Figs. 7(a) and (b), we can find that the MTTA has less impact on system performance than the number of the checkpoint servers and the checkpoint intervals.

To test system scalability, we compare two BT benchmarks, shown in Fig. 8. One is a BT benchmark of class A with 9 processors (BT.A.9) with one checkpoint server, and the other is a BT benchmark of class B with 64 processors with 8 checkpoint servers. We still take the execution time to present the availability of the HPC application. If the same HPC application can be executed in a shorter time on one situation, that shows the availability of the HPC application is higher on this situation. Because BT.A.9 is a smaller HPC application than BT.B.16, the execution time for BT.A.9 is less than BT.B.64. D. Buntinas, et al. [7] already proved that a blocked MPI is scalable, if we can prove that Dynas works linearly with a blocked MPI, we can prove Dynas is scalable with HPC application. In Fig. 8, the bar labeled ckptA means the execution time for BT.A.9 without DoS attacks interruption, the bar labeled ARx means the execution time for BT.A.9 with MTTF of x seconds (The execution time for different MTTF shows the
execution time for a blocked MPI), the bar labeled ADx means the execution time with MTTA of x seconds. Similarly, the bar labeled B in its name refers to the execution time for BT.B.64. From Fig. 8, we can observe that: (1) For a specific checkpoint interval, the time difference between failure-free executions and executions with DoS attacks increases when the application scales up, but the increasing rate is less than the increasing rate of application’s scale. This demonstrates that Dynasa is scalable for the HPC applications. (2) The execution time with specific MTTAs is longer than that with the same MTTF. However, the difference in time between with MTTA and with MTTF is almost the same when the application scales up. That shows that Dynasa is scalable under DoS attacks: it is capable of handling a DoS attack using fault-tolerance methods with a small extra overhead.

Dynasa takes fault tolerance method to handle malicious attacks, and fault tolerance sometimes cannot handle the fault if the fault happens too frequently. So Dynasa also has this problem if malicious attacks happens too often. We conduct some experiments to find the shortest time interval for malicious attacks which Dynasa can deal with. After five days’ testing on Paravent, we find that the shortest DoS interval that Dynasa can handle varies with the numbers of checkpoint servers and checkpoint intervals.

We define this smallest interval as Dynasa’s Minimum Tolerable DoS Attack Interval (MTDAI). Table 1 shows the average MTDAI under various checkpoint intervals (represented by interval) and numbers of checkpoint servers (represented by CS). When the DoS attack interval is smaller than that number, the MPI application will start over and run from the beginning again and again. Generally, when the checkpoint interval is smaller, the framework can better handle a DoS attack, because there are always new checkpoint images created before a DoS attack, and the system can create new replicas with the new checkpoint images, thus the application can move forward. If the DoS Interval is smaller, the application will keep restarting from the beginning, because there are no new checkpoint images available, or the application cannot restart with the new checkpoint images, thus the execution time of the application will be infinite. Different checkpoint intervals have a different impact on the MTDAI. For example, the MTDAI for two checkpoint servers with a checkpoint interval of 10 s is 28.8 s. 18.8 s longer than the checkpoint interval, while for two checkpoint servers with a checkpoint interval of 20 s, the MTDAI is 36.2, which is 16.2 s longer than the checkpoint interval. For two checkpoint servers with a checkpoint interval of 30 s, the difference between MTDAI and the checkpoint interval is 19.2. Besides the checkpoint interval, there are two additional parameters that affect the MTDAI: the checkpoint overhead, and the restarting time from the checkpoint image; the checkpoint overhead always changes with different checkpoint intervals, and so does the restarting time.

We carry out experiments to test the overhead of adaptation with BT benchmark of class B with 64 processors. The checkpoint interval is fixed to 20 s and the numbers of checkpoint servers varies. We find that the starting time of the adaptive component is less than 1 s, and the time for taking the adaptive action is less than 0.1 s. The overhead of transferring checkpoint images of one checkpoint wave and the application restarting is shown in Fig. 9. It is noted that the overhead of adaptation is relatively high if the number of checkpoint servers is small, but when enough checkpoint servers are used, the overhead becomes very small in comparison with the application’s execution time.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Dynasa’s minimum tolerable DoS attack interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval(s)</td>
<td>CS</td>
</tr>
<tr>
<td>10</td>
<td>47.1</td>
</tr>
<tr>
<td>20</td>
<td>55.8</td>
</tr>
<tr>
<td>30</td>
<td>53.4</td>
</tr>
</tbody>
</table>

5.2. Wide-area experiments

The wide area experiments are conducted on Grid5000. We use 40 nodes at Orsay (Gdx), 40 nodes at Rennes (Paravent), 40 nodes at Sophia (Azur) and 40 nodes at Nancy (Grillon). All the clusters are interconnected with internet links. We execute the BT benchmark of class B with 64 processors over these four clusters, and we can deploy four replicas over them. In order to evaluate the performance, we first measure the raw performance of this platform among these four sites using NetPIPE [35]. This is a ping-pong test with various message sizes that are slightly perturbed. During the tests, there is up to a 20 times difference between intra- and inter-cluster data transferring speeds. The results with various MTTFs and checkpoint intervals are shown in Fig. 10. As mentioned before, the DoS attack will lead to node unavailability, and the node will take some time to recover from it; the time for the recovery is called downtime. For simplicity, we set the downtime of nodes to 0. We first study how Dynasa works over wide-area Grid environments. To illustrate this, we show different execution times on different sites (local execution), and we show the total execution time for the HPC application which runs on a different site dynamically with Dynasa.

In Fig. 10(a), (b), and (c), the bar marked with PlaceCSx describes the execution time on the cluster at site Place with x checkpoint servers. For example, RennesCS4 means the cluster at Rennes with 4 checkpoint servers. In Fig. 10(d), the bar marked with Int×Dy describes the execution time over the wide area with checkpoint interval x seconds and MTTF y seconds. Because the checkpoint overhead is higher than the MTTF on Azur, Fig. 10(b) and (c) do not present the execution time on Azur with DoS attacks. Because the adaptive component always chooses the more powerful site as the primary site, the Azur can work as a passive replica site to enhance system availability for the NPB computation. Fig. 10(a) shows that the failure-free execution time among different sites does not change too much, especially between Nancy and Rennes. While in Fig. 10(b), the execution time with an MTTF of 40 s changes a lot compared with the execution time under failures. This shows that even fault tolerance and security problems have a lot in common; the same parameters can have different impacts on the performance. Comparing (a) and (b), we find that MTTF has less impact on the execution time compared with checkpoint overhead. Because the results are collected within the same clusters, the overhead of the adaptation arises from the I/O overhead of BLR and restarting from checkpoints images.

From Fig. 10(d), we find that the number of checkpoint servers has more impact on system performance than checkpoint interval.
and MTTA even over the wide area network, because more checkpoint servers improve the internet bandwidth utilization, and the transfer of checkpoint images becomes a major factor in the total execution time. In Fig. 10(d), the bars marked with Int.30D40 and Int.30D50 have a significant difference, suggesting that over the wide-area, MTTAs have a great impact on the performance, especially when only a small number of checkpoint servers are available (for example, four checkpoint servers as shown in Fig. 10(d)).

Comparing 10(b), (c) and (d), we can see that the execution time over the wide-area network is nearly twice the execution time of running on a single site when we have 8 checkpoint servers. This shows that, when the network bandwidth is not a bottleneck, the adaptive component can handle the DoS attacks efficiently over the wide-area network.

The second experiment over wide-area network compares Dynasa with a non-adaptive method with four events, potential attacks, DoS attacks, malicious content attacks, and the security level increased. We try to study the different performances between the adaptive method and non-adaptive method in this evaluation. Except for DoS attacks, we do not generate all the real attacks or increase the security level during the experiment; we only notify the adaptive controller that such events exist. We run experiments over four sites (40 nodes from Orsay Gdx, 40 nodes from Rennes Paravent, and 40 nodes from Nancy Grillon). We execute the BT benchmark of class B with 64 processors with 8 checkpoint servers. The checkpoint interval is set to 10 s. We input the four events in the sequence potential attack, DoS attack, malicious content attack, security level enhanced, and the interval between two events is 20 s. For the non-adaptive method, the application has static replicas (1, 2, and 3), while Dynasa creates or kills replicas with the protocols presented in Section 3. We measure network traffic, execution time, and CPU load for the three sites. The traffic and CPU load are average numbers of the system data during the whole period of the application’s execution. For the non-adaptive method, when the application runs for more than 120 s, we activate another passive replica if there is another replica. We take the threshold as 120 s, because the longest failure-free execution time over these three sites for the application is less than 120 s. The results are shown in Table 2. The network traffic is in terms of data transmission rate between the checkpoint servers of the replicas. For example, if there are two replicas over Rennes and Nancy, the network traffic in Table 2 means the network traffic between checkpoint servers over Rennes and Nancy. The CPU load is the average usage of the computational node of all the replicas. Several observations can be made from Table 2:

<table>
<thead>
<tr>
<th>Data</th>
<th>Execution time (s)</th>
<th>Network traffic (MB/S)</th>
<th>CPU load (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-adaptive and Dynasa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Over Rennes</td>
<td>1075.48</td>
<td>0</td>
<td>97.6</td>
</tr>
<tr>
<td>Over Rennes &amp; Nancy</td>
<td>748.87</td>
<td>61.03</td>
<td>73.15</td>
</tr>
<tr>
<td>Over Rennes &amp; Orsay</td>
<td>887.08</td>
<td>63.09</td>
<td>77.31</td>
</tr>
<tr>
<td>Over Rennes &amp; Orsay, Nancy</td>
<td>631.23</td>
<td>49.11</td>
<td>63.17</td>
</tr>
<tr>
<td>Dynasa</td>
<td>374.27</td>
<td>37.31</td>
<td>35.39</td>
</tr>
</tbody>
</table>

Table 2
Adaptive and non-adaptive performance comparison over three sites.

Fig. 10. Wide area experiments with 160 nodes over four sites.
(1) The execution time of Dynasa is much shorter than the non-adaptive method. The execution time stems from three sources: NPB computation, restarting application, and checkpointing. For one replica, the checkpointing latency is vast when there are DoS attacks. For two or three replicas, the synchronization of checkpoint images also lead to a longer latency of checkpoint creation. For static replication, when the application running over the new sites, we input the new attack event which also results in checkpoint latency. For malicious content attack, the non-adaptive method will take a long time to recover. Moreover, the non-adaptive method has to restart the application from the beginning, because the non-adaptive replication synchronizes the checkpoint images all the time, one is not sure which checkpoint images is correct. Dynasa creates and synchronizes the replica at the right time, and when the security threat increases, the replica number is reduced, thus the extra overhead is reduced.

(2) The network traffic for the non-adaptive method is much higher than Dynasa. Most of the network traffic comes from the checkpoint image synchronization. For the events we input, Dynasa needs much less synchronization than the non-adaptive method.

(3) The computational overhead for Dynasa is much lower than the non-adaptive method. With three sites, the average CPU load is around 35%, which means that approximately only one site's computation power is needed, while for the non-adaptive method, the average CPU load is above 60%, which is considerably higher than Dynasa's overhead.

6. Conclusion and future work

This paper presents the design, implementation, and evaluation of an adaptive component framework — Dynasa, which targets the Grid safety problems with fault-tolerance methods. Dynasa makes adaptive actions according to different kinds of safety problems. To demonstrate the adaptive method, we have modified the MPICH2-Pcl with BLCR to support MPI process migration, and tested the performance with NAS parallel benchmarks over Grid5000. The experimental results show that the adaptive component can handle the safety problems efficiently with low overheads. In the near future, we will add new policies to further improve performance, for example, by adjusting checkpoint intervals on-the-fly according to different safety situations. It is expected that the system performance will improve significantly if we could effectively reduce the checkpoint overhead. We will also integrate the Dynasa component with some Grid resource management middle-ware, in order to make adaptive decisions over the different Grid resources with the help of resource management.

Acknowledgments

The authors would like to thank Guang Tan from INRIA for helpful comments, and the authors would like to thank the anonymous reviewers who provided valuable feedback that helped improve this paper. This work is partly funded by the ARASIA SafeScale project of ANR, France and the National Science Foundation of China under Grant No. 60603058. Experiments presented in this paper were carried out using the Grid5000 testbed.

References

Xuanhua Shi received his Ph.D. degree in computer engineering from Huazhong University of Science and Technology (China) in 2005. From 2006, he worked as an INRIA Post-Doc in PARIS team at Rennes for one year. Currently he is an associate professor in Service Computing Technology and System Lab (SCTS) and Cluster and Grid Computing Lab (CGCL) at Huazhong University of Science and Technology (China). His research interests include cluster and grid computing, fault-tolerance, web services, network security and grid security, virtualization technology. Contact him at xhshi@hust.edu.cn.

Jean-Louis Pazat was born in Paris on May 23, 1959. He graduated from the University of Rennes, France and obtained a Ph.D. Thesis (Thèse d’Université) at University of Bordeaux, in 1989. He obtained a “Habilitation à diriger des recherches” in 1997. From 1989 to 2004 he has been assistant professor at the CS Dept. of INSA of Rennes and a researcher at IRISA. Since 2004, he has been a professor at INSA. The main scientific interest of Jean-Louis PAZAT is software tools for Grids. He studied the mapping problem and distributed debugging during his Ph.D. He has been working on compilation techniques (e.g. HPF) for clusters. He is now investigating Service Oriented Architectures, Software Components, Security and Dynamic Adaptation Techniques for Parallel Components.

Eric Rodriguez is an associate engineer in the project team Grand Large of INRIA, working on grid computing and fault tolerance in large parallel systems (MPICH-V project). Currently, he is a scientific computing engineer in Cedrat-Group, software solution for electrical and mechatronic engineering applications.

Hai Jin is a Professor of Computer Science and Engineering at the Huazhong University of Science and Technology (HUST) in China. He is now the Dean of the School of Computer Science and Technology at HUST. He received his Ph.D. in computer engineering from HUST in 1994. In 1996, he was awarded German Academic Exchange Service (DAAD) fellowship for visiting the Technical University of Chemnitz in Germany. He worked for the University of Hong Kong between 1998 and 2000 and participated in the HKU Cluster project. He worked as a visiting scholar at the University of Southern California between 1999 and 2000. He is the chief scientist of the largest grid computing project, ChinaGrid, in China, and he is the director of the Key Lab of Service Computing Technology and System, MOE (Ministry of Education). Also, he is the chief scientist of National 973 Basic Research Program, Basic Theory and Methodology of Virtualization Technology for Computing Systems. Jin is a senior member of IEEE and a member of ACM. He is the associate editor of International Journal of Computer and Applications, International Journal of Grid and Utility Computing, International Journal of Web and Grid Services, Journal of Computer Science and Technology. He is an IASTED technical committee member on Parallel & Distributed Computing and Systems. He is the steering committee chair of International Conference on Grid and Pervasive Computing (GPGC). He is the steering committee member of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid) and International Conference on Grid and Utility Computing, and served as program chair of GCC’04, SCC’05, HS’05, NPC’05, and UI3W’05, program vice-chair of the CCGrid’01, PDCAT’03, NPC’04, EUiC’05, e-Science’05, AINA’06, and CCGrid’06. He served as conference chairs for International Workshop on Internet Computing and E-Commerce from 2001 to 2003. He also served on the program committee for more than 200 international conferences/workshops. He has co-authored 10 books and published over 200 research papers. His research interests include cluster computing and grid computing, peer-to-peer computing, network storage, network security, and virtualization technology.

Hongbo Jiang received the B.S. and M.S. degrees from Huazhong University of Science and Technology, China. He received his Ph.D. from Case Western Reserve University in 2008. After that he joined the faculty of Huazhong University of Science and Technology as an associate professor. His research concerns computer networking, especially algorithms and architectures for high-performance networks and wireless networks.