A Flexible Fault-Tolerance Mechanism for the Integrate Grid Middleware

Stanley Sousa, Francisco Silva

Laboratório de Sistemas Distribuídos
Departamento de Informática
Universidade Federal do Maranhão, Brasil

June, 2007
1. Introduction

2. Fault Tolerance in Grid Environments

3. Architecture Overview

4. Implementation Highlights

5. Experimental Results

6. Conclusions
Motivation

- Computer grids have been used to solve problems in varied areas, empowering the conception of applications that allow combining computations, experiments, observations, and data got in real time;

- The complexity of modern grid environments have turned impracticable its configuration, maintenance and recovery in case of failures solely by human beings, due to:
  
  - Its dynamism, that can be observed in terms of high variation in resource availability, node instability, and workload variations in nodes and network links;
  
  - Its heterogeneity, by aggregating a great amount of computation and communication resources, databases and, sometimes, sensors and specific peripherals;
  
  - Its high scalability, by integrating computational resources spread through several administrative domains.
Challenge

- Providing a greater autonomy to grid systems is one of the greatest challenges for the new generation of grid middlewares;
- The term autonomic computing has been used to denote a system that exhibits functional properties, such as:
  - Context-awareness: the system must be aware of its execution environment and be able to react to environmental changes;
  - Self-Optimization: the system should be able to detect performance degradation and intelligently perform self-optimization actions;
  - Self-Healing: the system must be aware of potential problems, and should be able to reconfigure itself in order to continue to function smoothly and to recover from failures using diverse and adaptive failure-handling techniques;
  - Self-Configuration: the system must have the ability to dynamically adjust its resources based on its state and the state of its execution environment.
The AutoGrid Project

- Main goal: the development of a robust and self-managing autonomic grid system;
- It is based on the Integrade grid middleware, incorporating autonomic mechanisms to its infrastructure in order to make its configuration and administration independent from human intervention;
- This work presents our initial effort toward AutoGrid self-healing infrastructure: the development of a flexible failure handling mechanism that supports multiple fault tolerance techniques;
- The decision about which technique should be applied in each situation must be taken by the grid middleware in an autonomic and automatic way, based on the environment status and previous experiences.
Fault Tolerance in Grid Environments: Motivation

Computer grid environments are highly prone to failures:

1. Grid systems are composed of a wide range of services, software, and hardware components, which need to interact with one another. System failures can result not only from an error on a single component but also from the interaction between components;

2. Grid environments are extremely dynamic, with components joining and leaving the system all time;

3. The likelihood of errors occurrence is exacerbated by the fact that many grid applications will perform long tasks that may require several days of computation.
Grid Fault Tolerance Services

- Failure detection: grid nodes and applications must be constantly monitored by a failure detection service;
- Application failure handling:
  - Retrying: restarts an application from scratch;
  - Replication: submits the same application for execution a number of times, generating various application replicas;
  - Checkpointing: periodically saves the state of the computation in a stable storage during the failure free execution time. Upon a failure, the application restarts from the last saved point.
- Stable storage: stores execution states that will allow to recover the pre-failure state of applications.
Flexible fault tolerance mechanism

- Grid environments are heterogeneous in respect to its tasks (e.g., long running tasks, mission critical tasks, transactional tasks) and its execution environment (e.g., highly reliable execution environments, unreliable execution environments);

- The environment is also highly dynamic, especially on opportunistic grids since nodes are not dedicated;

- The best fault tolerance technique to be applied on a given situation depends on application characteristics and its underlying execution environment;

- Hwang et al. presents a comparative analysis based on simulations among the four main failure handling techniques: retrying, checkpointing, replication, and replication with checkpoint.
The Integrade project is an effort to build a novel grid computing middleware infrastructure to leverage the idle computing power of personal workstations for the execution of computationally-intensive parallel applications;

- The basic unit: cluster, a collection of machines usually connected by a local network;

- Clusters can be organized in a hierarchy, encompassing a large number of machines;

- Each cluster contains:
  - A Cluster Manager node, responsible for managing the cluster computing resources and for inter-cluster communication;
  - Workstations, which export part of its resources to grid users. They can be shared or dedicated machines.
Integrate Main Components

- **Application Submission and Control Tool (ASCT):** allows users to submit applications and control their execution;

- **Application Repository (AR):** stores the code of applications that can be executed on the grid;

- **Local Resource Manager (LRM):** runs in each cluster node. Responsible for monitoring resources usage and for instantiating and executing applications scheduled to the node;

- **Global Resource Manager (GRM):** manages the cluster resources and runs the application scheduler;

- **Execution Manager (EM):** maintains information about each application execution, such as its state, executing node, input and output parameters, submission and conclusion timestamps. Coordinates the application recovery process, in case of failures.
Integrade Application Execution Protocol
The original Integrate fault tolerance mechanism was completely based on application level checkpointing;

A precompiler inserts into the application source code the statements responsible for gathering and saving its state on a stable storage;

It minimizes the overhead by copying the checkpoint data to a buffer and performing the coding and transfer of checkpoints through a separate application thread.
Integrate Replication Mechanism

- We developed on Integrate the support for replication;

- Each replica represents an active instance of the application, running on a resource different than the other replicas;

- When one of the replicas finishes, the grid middleware must discard (or ignore) the others and return the results to the requesting user.
Flexible Fault Tolerance Mechanism

The mechanism allows the user to customize parameters related to the failure handling mechanism, as part of the application submission process:

- Enable or disable the checkpointing;
- Set the time interval between consecutive checkpoints;
- Enable or disable the replication mechanism;
- Set the amount of application replicas to be generated.

The user combines different failure handling techniques:

- Retrying (without checkpoint or replication);
- Checkpointing;
- Replication (without checkpointing);
- Replication with checkpointing.
Fault Tolerance Components

- Execution Manager (EM): maintains information about each application submission and coordinates the reinitialization of applications in case of failures;
- Checkpointing library (ckpLib): provides the functionality to periodically generate checkpoints containing the application state;
- Autonomous Data Repositories (ADR): a distributed stable storage;
- Cluster Data Repository Manager (CDRM): manages the available ADRs on a cluster and the location of each checkpoint data;
- Application Replication Manager (ARM): manages the replicas of a single application submission. Instantiated on demand by the Global Resource Manager (GRM).
Experiments

- We performed several experiments that measure the tradeoff among failure handling techniques implemented on Integrade considering several different execution environments scenarios;

- Parameters:
  - Failure-free execution time ($F$): execution time of a task in the absence of failures;
  - Failure Rate ($\lambda$): random variable representing an arrival rate of failures;
  - Mean Time to Failure (MTTF): mean interval between adjacent failures;
  - Downtime ($D$): average time following a failure of a task before it is up again;
  - Number of replicas ($N$): number of replicated tasks, each running on a different machine;
  - Checkpoint Interval ($C$): time interval between two consecutive checkpoints in failure-free runs. If $K$ checkpoints are created during $F$, then $C = F / K$. 
The application is a generic method to estimate the variability in statistics;

- Failure-free execution time $F = 157.13$ seconds;
- Downtime $D = 0$ seconds;
- We varied the MTTF to 20, 40, 60, 120 and 180 seconds;
- Number of replicas $N = 3$, whenever replication was used;
- Checkpoint interval $C = 5$ seconds.
Experimental Results

Bootstrap Application: Results

<table>
<thead>
<tr>
<th>MTTF</th>
<th>$R_t$</th>
<th>$C_k$</th>
<th>$R_p$</th>
<th>$R_pC_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>$\infty$</td>
<td>184.55</td>
<td>324.78</td>
<td>162.16</td>
</tr>
<tr>
<td>40</td>
<td>$\infty$</td>
<td>172.30</td>
<td>175.81</td>
<td>158.72</td>
</tr>
<tr>
<td>60</td>
<td>899.24</td>
<td>167.07</td>
<td>160.26</td>
<td>157.81</td>
</tr>
<tr>
<td>120</td>
<td>408.89</td>
<td>161.36</td>
<td>157.14</td>
<td>157.76</td>
</tr>
<tr>
<td>180</td>
<td>229.62</td>
<td>159.87</td>
<td>157.13</td>
<td>157.51</td>
</tr>
</tbody>
</table>

- For $MTTF = 20$ and 40 seconds, checkpointing and replication with checkpointing presented a better performance than the other two;
- However, for a $MTTF \geq 120$ seconds ($\frac{MTTF}{F} \geq 0.76$), replication and replication with checkpointing were lightly better than checkpointing and considerable better than retrying.
Experimental Results

Bootstrap Application: Second Experiment

- Downtime value was set to $D = 79$ seconds ($\approx \frac{F}{2}$);

<table>
<thead>
<tr>
<th>MTTF</th>
<th>$Rt$</th>
<th>$Ck$</th>
<th>$Rp$</th>
<th>$RpCk$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>$\infty$</td>
<td>899.06</td>
<td>527.24</td>
<td>280.82</td>
</tr>
<tr>
<td>40</td>
<td>$\infty$</td>
<td>450.85</td>
<td>241.01</td>
<td>181.67</td>
</tr>
<tr>
<td>60</td>
<td>1117.59</td>
<td>408.61</td>
<td>180.12</td>
<td>158.22</td>
</tr>
<tr>
<td>120</td>
<td>626.02</td>
<td>286.75</td>
<td>157.11</td>
<td>157.70</td>
</tr>
<tr>
<td>180</td>
<td>319.55</td>
<td>246.81</td>
<td>157.06</td>
<td>157.61</td>
</tr>
</tbody>
</table>

- When the downtime increases, replication and replication with checkpointing outperformed the other two techniques.
Matrix Multiplication Application

- Each matrix consists of $1700 \times 1700$ elements of type float;
- Application failure-free execution time $F = 115.75$ seconds;
- Objective: evaluate the impact of the checkpoint data size, since the matrix multiplication application generates a huge checkpoint of 67.8 MB, in contrast to the bootstrap application, whose checkpoint data has only 274.82 KB.
Experimental Results

Conclusions

Considering the objective of minimizing the application response time, we can conclude:

1. In environments with high fault rates (low MTTF), the use of checkpointing presented the best results, specially when combined with replication;

2. If the execution environment presents a low fault rate (high MTTF), the use of replication becomes more attractive;

3. In environments with low fault rate and low downtime, the use of checkpointing can be considered a good choice for applications with small checkpoint size, since the response time is very close to the one obtained with replication, with the advantage of using less grid resources;

4. As the application checkpoint size or the environment downtime increase, replication becomes more attractive.
Conclusions

We presented a flexible fault-tolerance mechanism implemented on Integrate grid middleware. Flexibility is achieved by:

1. Allowing the customization of different failure handling parameters;
2. By letting the user combine different failure handling techniques.

We evaluated the benefits of our flexible mechanism by performing several experiments that sustains the conclusion that grid middlewares can benefit from providing different failure handling strategies;

Towards our objective of developing an autonomic grid middleware, we are currently implementing the support for automatic decision of the failure handling technique to be applied in case of failure.