Grid Unit: a Self-Managing Building Block for Grid System

Jianfeng Zhan#, Lei Wang#, Ming Zou#, Hui Wang#, Shuang Gao#, Yulei Ding#

#Key Laboratory of Computer System and Architecture
Institute of Computing Technology
Chinese Academy of Sciences, Beijing 100080, China
jfzhan@ncic.ac.cn

Abstract—Grid system software is inherently complex, hard to build and maintain. In this paper, we propose a self-managing building block: Grid Unit, which facilitates constructing Grid system with higher availability and lower management overhead. We present an agent organization as autonomic management framework, and propose a self-recovering protocol to eliminate most of tough jobs from system administrator’s routines. The system has been deployed on Dawning 4000A since 2004, the biggest node for China Grid system. We have done extensive experiments to evaluate Grid Unit, and the collected log data shows the availability of a Grid parallel process management service, built on the basis of Grid Unit, reaches 99.997%.

I. INTRODUCTION

The Grid system is composed of diversities of heterogeneous resources on Wide Area Network or even Internet scale. The system complexity of Grid software is the one of the main root causes for system crashes. It was reported in the paper [1] that 40% of computer system outages are caused by human operator errors, not because they are poorly trained or do not have the right capabilities.

Though several Grid system architectures [2], [3] have been built, their running depends on the system administrator’s manual configuration and timely intervention to detect, diagnose and restore system. Paper [4] has reported diversities of failure in Grid system. Just depending on administrator’s manual work to deal with these issues is unrealistic and unacceptable for production Grid system, since troubleshooting large-scale distributed needs high-level skill [22]. We need to propose self-managing framework to get rid of those and improve the total system availability and decrease management overload of Grid system.

Some works [1], [5] have proposed the self-organizing mechanism: structure overlay network to adapt to the dynamic changing environment of Grid systems. The system membership service of Vigne [5], a middle-level Grid operating system is based on a structured overlay network. However, our work has different research angle and focuses on proposing a self-managing building block: Grid Unit, with which we could eliminate tough jobs from administrator’ routines, including configuration, deployment, start-up, fault detection, fault diagnose and system recovery. Besides, works [1], [5] assures a flat structure of Grid, while our work pays attention to the fact that the Grid is a hierarchy structure composed of different management domains and their owners prefer much stronger control over their own nodes for the resource provision to different kinds of users.

Our design heuristic is as follows: If we improve the availability of each building block, the whole availability of Grid system will be improved dramatically. If we decrease the management overhead of each building block, the whole management overhead will be decreased dramatically. Our work focuses on self-managing aspect, especially self-recovering. On the basis of building block, we can merge different kinds of interaction frameworks [1], [2], [5], [7], [8] to build high available Grid system with lower management overhead.

The research contribution of this paper can be concluded as: (1) we present an agent organization with six roles as automatic facility for self-configuration and self-recovery in maintaining the Grid Unit system. (2) A self-recovering protocol is proposed into management framework to get rid of most jobs from administrator’s routines. (3) We have done extensive experiments to evaluate the performance of autonomic management facility and analyse its impact on the availability and management overhead of Grid system; the collected log data shows the availability of a Grid parallel process management service, built on the basis of Grid Unit, reaches 99.997%.

This paper is structured as follows. Section 2 gives an overview of related work; Section 3 explains the design elements of Grid Unit; section 4 describes the implementation; section 5 evaluates the system performance. Finally, in section 6, we draw a conclusion.

II. RELATED WORK

In the field of Grid system, Vigne [5], a middle-level Grid operating system proposed membership service based on a structured overlay network. The report [1] envisions making large-scale distributed applications self managing by combining component models and structured overlay networks. Flock of Condor [8] chooses Condor Pool as a management unit and presents peer-to-peer mechanisms which focus on the resource discovery that are self-organizing, fault-tolerant, scalable, and locality-aware, while our work focuses on the self-recovery aspect.

In the work [15] agent-based self-organization is proposed to perform complementary load balancing for batch jobs with
no explicit execution deadlines. Though our work also adopts the agent model, we focus on the self-recovering mechanism for system software itself. The purpose of [16] is to demonstrate a self-organized grouping (SOG) method for efficient Grid resource discovery. Organic Grid [9] proposes a new design for desktop grids that relies on a self-organizing, fully decentralized approach to the organization of the computation. These works have different research angles from ours.

In the field of Cluster system software, designers used several different structures to deal with high availability issues of cluster system, such as master-slave structure [12],[17], group structure [18],[19]; Master-slave structure is only suitable for small cluster systems with its scalability limit. Several projects propose group membership protocol in cluster management framework, for example Galaxy [18], GulfStream [19] and HACMP [20] project. But when the scale of cluster system reaches thousand nodes, it is unacceptable for all nodes joining a group managed by group membership protocol, as HACMP only contain up to 32 nodes [20]. Our system is a hierarchy structure. The UCLA Fault-Tolerant Cluster Testbed (FTCT) [21] project is focused on developing and evaluating algorithms and implementation techniques for fault tolerant cluster managers. The manager group consists of three active replicas: one primary and two backups, which is different from our system. Besides, our work also pays attention to self-configuring.

### III. THE DESIGN ELEMENTS OF GRID UNIT SYSTEM

#### A. System Model

Our presumed Grid system is different from Desktop Grid or traditional Grid with dedicated high-speed network, whose characteristics are described in the paper [9]. In fact, it’s a mixed Grid system model with the following characteristics:

1. The basic unit is an independent management domain, which we call Grid Unit. It may be a cluster system, e.g. data center or an ordinary office LAN for cycle harvesting.

2. Each Grid Unit may join or leave from the Grid system at its owner’s will. If one Grid Unit joins the Grid, it still reserves its own managing policy on how to provision resource for different kinds of users. If the Grid Unit leaves from the Grid system, it is an independent site.

3. Grid unit system software is responsible for managing itself. From the angle of high-level Grid system, it only need deal with joining, leaving or rejoining of Grid Unit and maintain its availability.

4. For interaction framework among all the Grid Units, it may adopt the structured overlay network proposed in paper [5], [8], or static hierarchy proposed in Globus [2], depending on different application scenarios, as shown in Figure 1.

---

**Fig.1 Building block with different interaction frameworks**

#### B. Agent, Role and Organization

According to [10], an agent is a program that acts in a self-interested manner in its dealings with numerous other agents inside a computer. We introduce agent concept into Grid Unit system for self-configuration and self-recovery. Deployed on each node, agent acts as the representative of its host node and plays at least one role with the specified behavior.

Figure 2 explains the role hierarchy defined in Grid Unit. There are six roles: basic, executor, manager, intermediate, King and Princess, whose behaviors are defined in the Section 3.3. Among them, executor and manager are derived from basic role. The node that plays executor role (we call it executor node for short) is used for running user-specific computing task or business application, while manager node is used for running Grid system services. Intermediate role is a mixed one derived from both executor and manager. When the manager node is OK, the intermediate acts as executor; if the manager is down, the intermediate will promote to be manager and run Grid system services.

From the perspective of multi-agent system [10], the agent organization defines the rule for forming Grid Unit and decides each agent’s role in the organization. For scalability, the agent organization will be divided into more than one group with each agent belongings to only one group. At any time, there are only one manager and one intermediate in each agent group. The manager is responsible for administrating the own group. All managers form a meta-group in a logical ring, among which there are one King and one Prince, as shown in Fig.3. The King and Prince roles are special kinds of manager, which we will describe in the next section.

For most of High-availability structure [12], [17], there are only two roles: master and slave. But master-slave structure is only suitable for small cluster systems with its scalability limit. Several projects propose group membership protocol in cluster management framework [18], [19], [20]. But it is unacceptable for all nodes joining a larger group managed by group membership protocol, as HACMP only contain up to 32 nodes [20]. Our system is a hierarchy structure, and we
introduce five roles with different rankings. Executor role is used for running user-specific computing task or business application, while manager role is used for running Grid system services. Intermediate role is proposed for saving resource, since it is a backup for manager, while acting as executor. King, a special subclass of manager, is proposed to simplify the self-recovery protocol, since it is the unique arbitrator for manager’s rejoining and failure diagnose. The Prince is a backup for King. Our system has been applied on 640-nodes Grid Unit with minor impact on scientific computing [11].

C. The Agent’s Behavior in Terms of Role

There are six roles defined in Grid Unit, including basic, executor, intermediate, manager, Prince and King. The role hierarchy is defined in Figure 2, and the derived role will inherit the behavior of its parents. As shown in Figure 4, the role ranking is increasing from executor to King directed by arrow.

![Fig. 2 the Role Hierarchy](image)

The basic role is a virtual one, and any role derived from that has following behavior:

1. Reactively receives the role assignment or role change from administrator or other agent with higher ranking
2. Downloads and deploys Grid system service and applications required by the derived role.
3. Starts and monitors Grid system services and applications required by the derived role.

The Executor inherits behavior from the basic role. In addition, executor sends heartbeats through all network interface cards to its affiliated manager in order to report its states.

The Intermediate is a mixed role between the executor and manager. If the manager is OK, the Intermediate will act as the executor, but software packages required for manager role are deployed. Once the manager is down, the intermediate will promote its role and act as manager.

The Manager inherits behavior from basic role. In addition, other behaviors are as follows:

1. Handles joining or leaving requests from executors or intermediate.
2. Monitors the health states of executors and intermediate, and guarantees their availability;
3. Collaborates with other managers to maintain manager meta-group as shown in Figure 3.
4. Drives the role change of agent with lower ranking in its own group.

The Prince inherits the function from manager. Besides, it is responsible for monitoring King’s health state. Once the King fails, the Prince will promote its role to the King and takeover its work.

The King inherits behaviors from manager with augmented ones:

1. Deals with joining, rejoining and leaving requests from other managers.
2. Handles the failure case of manager;
3. Triggers the role change in manager meta-group.

D. Self-Recovering Protocol

In this section, we introduce the self-recovering protocol for Grid Unit, including failure processing and rejoining of failed manager.

1) Failure Processing:

For Manager:

1. All managers form a meta-group in a logical ring, and each manager will periodically send heartbeats through all network cards of its host to the preceding one as shown in Figure 3.
2. If one manager fails to receive any heartbeat from the successor in the ring after the specified timeout, it will report the suspected case to the King, which is responsible for dealing with the situation.
3. If the suspected failure is validated by the King, this manager will takeover the work of failed one, including
monitoring and diagnosing its subordinate executors and intermediate.

(4) The manager diagnoses the failed one. If it is a process failure, the recovered manager daemon will be restarted; else if it is a node failure, the manager will promote the intermediate, subordinate to failed manager, to be manager.

(5) Once the failed manager rejoined the manager group, it will return the takeover.

(6) If the suspected manager is the Prince, the manager will promote its role as Prince. Once the failed Prince rejoins, the new Prince will return its takeover and degrade to manager role.

Besides,
(7) If no heartbeat is received from the registered Grid system service, the manager will restart it.
(8) If no heartbeat is received from the subordinate executor or intermediate of the manager, it will diagnose the possible failure and takes the right action, for example, restarting process, or promoting one executor to intermediate.

For King:
(1) If the suspected failure of one manager is reported, the King will diagnose it.
(2) If the suspected case is validated, the failed manager is removed.
(3) The King sends meta-group change information to all other meta-group members.
(4) If all other managers respond, the phase of failure processing is over; else the King treats the manager without response as failure and handles it further. The phase of failure processing maybe is nested with limited depth, since there is little possibility for the case that more than two managers fail simultaneously.

For Prince
(1) If the suspected manager is the King, the Prince is responsible for handling the case and diagnoses the suspected failure.
(2) If the failure is validated, the King is removed from the meta-group.
(3) The Prince promotes its role to King and takeover its work and drives the preceding one in the logical ring to be the Prince.

2) The rejoining protocol of recovered manager:
For King:
(1) It deals with the rejoining request from the recovered manager. If the request is valid, it sends node, meta-group and registered Grid system service information to the recovered manager; else discards the message.
(2) It modifies the local information and sends new meta-group information to all members.
(3) If the responses have been received from all members, then rejoining phase is over; else it treats the member without response as failure and handles the case further.

For the recovered manager:
(1) It sends rejoining request to the King.
(2) If no response is from the King after the specified timeout, the recovered manager fails to rejoin, then retry after a sleep; else the recovered manager restores the normal works, including monitoring executors, intermediate and the registered Grid system services.

For other managers
(1) If the meta-group change information is received from the King, it responds and modifies its local meta-group information.
(2) If the rejoined manager is the preceding one in the logical ring, the manager will return its takeover.

E. The forming rule for agent organization

To improve the stability and robustness of Grid unit, we simplify the forming rule of agent organization and take the following presumptions:
(1) Node operating system and agent daemon are deployed on each node.
(2) The Grid Unit Administrator uses a GUI named Control Center for system configuration and first-time system startup.
(3) All initial nodes are statically assigned to different groups with specified roles. To achieve the high availability, the Grid unit is divided into at least two groups. The managers in the first and second group are assigned the role of King and Prince.

To simplify first-time startup, we take the following strategies:
(1) The whole Grid Unit startup process is divided into two phases: booting agent organization and booting registered Grid system service (or Grid application).
(2) We treat each booting phase as a transaction. E.g. for booting manager organization, if one manager fails to join meta-group, then we restart all members again. This makes the startup process simpler in case of complex cases, while the probability of system startup success is quite high.
(3) Once the two phases are OK, the state of Grid Unit system is stable with self-recovering.

IV. THE IMPLEMENTATION

Since the focus is self-recovering building block for Grid system, in this section, we give out the implementation of a Grid parallel process management service (GPPMS) [7] on the basis of Grid Unit, which is used to simultaneously initiate processes on hundreds of nodes of different Grid Units and redirect the standard output, error and exit code of these processes to the originator node.

A. The System software Overview

The system software includes two subsystems: GPPMS and Grid Unit system software. The Grid unit system software
includes agent daemon, Deployment Service, Configuration service and GUI: Control Center, shown in the Figure 5.

1. Agent daemon is deployed on each node of grid unit. It cooperates with other modules to configure, deploy, boot, and maintain the system.
2. Control Center is a Java GUI with which user click the button to boot system.
3. Deployment Service provides program packages for agents to download according to its roles.
4. Configuration Service provides the interfaces of querying and storing system configure information.

![Fig.5 Components of Grid Unit System Software](image)

The Grid parallel process management service includes two subcomponents: process executor and parallel process server as shown in Figure 6. E stands for process executor, which is subordinate service affiliated to the specific PPS (Parallel Process Server); R stands for service request, which is the client of PMS.

![Fig.6 the Architecture of GPMS in Grid Unit.](image)

In the case of parallel process service, there are five roles in the Grid unit system:

1. Executor: It will deploy agent daemon and process executor daemon;
2. Manager, King and Prince: It will deploy agent daemon and parallel process server daemon;
3. Intermediate: It will deploy agent daemon, process executor and parallel process server daemon;

Figure 7 is the Snapshot of Parallel Process service GUI.

![Fig.7 the Snapshot of GUI of GPPMS](image)

B. The Implementation and Related API

Agent daemon is developed on the basis of ACE5.2.1 [14] and Phoenix Kernel’s API [11]. The Control Center is written in Java. Process Executor and Parallel Process Server are written in C++ on the basis of ACE 5.2.1. ACE is a freely available, open-source object-oriented (OO) framework that implements many core patterns for concurrent communication software.

We choose a FTP server as the Deployment Service, and Agent daemon can access or download program packages through ftp protocol. Mysql-4.0.13 is used as database for the configuration service.

To guarantee the high availability of deployment service and configuration service, we choose Duplex machines HA solution ([http://linux-ha.org/](http://linux-ha.org/));

The related API for agent is as follows:
- `register_app`: register applications and services with passive monitoring style
- `remove_app`: remove registered applications and services with passive monitoring style.
- `register_hb`: register applications and services with proactive monitoring style
- `remove_hb`: remove applications and services with proactive monitoring style
- `send_hb`: send heartbeat

The related API for parallel process service is as follows:
- `exec_single_host`: execute a command on a single host
- `seri_exec_cmd`: execute a command serially on several hosts
- `para_exec_cmd`: simultaneously execute a command on several hosts

V. THE EVALUATION

A. The Experiments

The test bed is as follows: 136 nodes interconnected with two network systems in Dawning 4000A [13]. The node OS is SuSE Linux 8.0. The agent organization is divided into 8 groups, and there are 15 executors, 1 manager and 1 intermediate for each group.
From Fig. 8 to Fig. 10, the number in X axis indicates different tests, and the Y axis is time in terms of second.

The following experiments include four parts. In the experiments, for node failure, we power off the specified node; for network failure, we use “ifconfig down” command to disable the specified network; for process failure, we just kill the specified process: agent. For registered service test, parallel process server is chosen.

1) Executor test:

This experiment is to test performance overhead of detecting, diagnosing and recovering executor node. The Executor sends heartbeats through two network interface cards to its affiliated manager in order to report its states. From Figure 8-1 to Figure 8-3, we respectively set the heartbeat interval of executor as 10, 5 and 3 seconds. In these Figures, three cases are tested, including process failure (1), node failure (2) and network failure (3), and these numbers are marked under the x axis; the overhead includes three parts: detection, diagnose and recovery. From Figure 8-1 to Figure 8-3, we can conclude: the total overhead of detecting, diagnosing and recovering executor node is less than \((n+3)\) seconds (n is heartbeat interval for executor in terms of second).

For agent with the role of executor, the recovery of failed process is very short. For node failure and network failure, we ignore the recovery time, because most of which need the manual work and depend on administrator’s expertise.

2) Manager test:

This experiment is to test the performance overhead of detecting, diagnosing and recovering manager node. All managers form a meta-group in a logical ring, and each manager will periodically send heartbeats through all network cards of its host to the preceding one. From Figure 9-1 to Figure 9-3, we respectively set the heartbeat interval of manager as 10, 5 and 1 seconds. In these Figures, three cases are tested, including process failure (1), node failure (2) and network failure (3), and these numbers are marked under the x axis; From Figure 8-1 to Figure 8-3, we can conclude: the total overhead of detecting, diagnosing and recovering manager node is less than \((n+3)\) seconds (n is heartbeat interval for manager in terms of second).

Because the test platform has two networks, the situation that only one network card fails only partly affect the running of manager agent. So we ignore the recovery time for only network failure, and, treat the case of two networks failures as same as node failure. From Figure 8-1 to Figure 8-3, the network recovery time is 0. For node failure, we will promote the intermediate node to manager and migrating Grid system service to new manager node.
3) Registered service test:

This experiment is to test performance overhead of detecting, diagnosing and recovering registered service. The registered service sends heartbeat to agent, and here is parallel process server, which is explained in Section 4.1. In this test, network failure means nothing, since the registered service resides on the same node as manager. From Figure 10-1 to Figure 10-3, we respectively set the heartbeat interval of executor as 10, 5 and 1 seconds. In these Figures, two cases are tested, including process failure (1) and node failure (2), and these numbers are marked under the x axis; From the Fig. 10-1 to Fig. 10-3, we can conclude: the total overhead of detecting, diagnosing and recovering process failure is less than (n+3) seconds, and overhead of detecting, diagnosing and recovering node failure is less than (2*n+4) seconds (n is heartbeat interval in terms of second).

4) Heavy load test:

We use exhauster developed by ourself: a resource exhausting program to simulate the heavy load, which will induce CPU, memory load. Under the simulated heavy load, we test the overhead for manager. We set the heartbeat intervals of registered service and manager as 1 second respectively and the heartbeat interval of executor as 3 seconds. The CPU and memory exhausters reside in one program. In the Table 1, the first column is the number of running exhausters on each manager node, and the last column is the increased overhead compared with the data shown in Figure 8. From the table 1, we can conclude that the manager meta-group can still work under heavy load with a light delay. But under very heavy load in the case of running 40 exhausters, the system fails to function.

<table>
<thead>
<tr>
<th>Exhausters</th>
<th>User-mode CPU %</th>
<th>Kernel-mode CPU %</th>
<th>Total CPU usage %</th>
<th>Memory Usage %</th>
<th>Increased overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>11.9%</td>
<td>88.1%</td>
<td>100%</td>
<td>22.1%</td>
<td>0.469s</td>
</tr>
<tr>
<td>30</td>
<td>8.85%</td>
<td>91.15%</td>
<td>100%</td>
<td>58.9%</td>
<td>1.13s</td>
</tr>
<tr>
<td>40</td>
<td>7.13%</td>
<td>92.87%</td>
<td>100%</td>
<td>67.4%</td>
<td>Mal function</td>
</tr>
</tbody>
</table>

B. The availability analysis of Grid Unit
In this section, we evaluate the availability of one parallel process server. According to the statistics of Dawning 4000A, the Mean Time between Failure (MTBF) of registered service is 200 hours, and the MTBF of node is 4000 hours. The Mean Time to Recovery (MTTR) of node is 1 hour and the MTTR of registered service in the case of process and node failure is about 10 and 23 seconds respectively with the heartbeat interval set as 10 seconds as shown in Fig.7.

We presume that there is one intermediate node up in the case of failure of manager. For the period on the scale of MTBF of node, 4000 hours (about five months), there are about 4000/200=20 times of process failure and one time of node failure. So the MTTR is (20*10+23)/11=20.27 seconds.

Then the availability is MTBF/ (MTBF+MTTR) =200*60*60/ (200*60*60+20.27) =99.997%

From the above analysis, we can conclude the Grid Unit could guarantee the high availability of Grid service: Parallel Process service.

C. The analysis of management overhead

It is very hard to analyze the management overhead, since the administration work is very cumbersome. In this section, we roughly analyze the decreased management overhead with Grid Unit. Configuration: with the Grid Unit, the administrator just need specify the role of each node; the other work is automatically done. The Grid Unit simplifies the configuration process.

System Startup: with Grid Unit, the administrator just need click the button of Control center. Grid Unit treats each booting phase as a transaction and startup process become simple and easy to control.

Failure detecting, diagnosing and recovery: most of failure related works are done automatically without administrator’s intervention.

From the above analysis, we can roughly conclude: our proposed method could decrease the management overhead of each Grid Unit, which will result in the decrease of management overhead of the whole Grid.

In addition, in the paper [7], we have evaluated the performance of parallel process service.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose a self-recovering building block: Grid Unit, on the basis of which, we can construct Grid system with high availability and lower management overhead. We present an agent organization as autonomic facility for maintaining the system. Within the framework, a self-recovering protocol is proposed to eliminate most of tough jobs from system administrator’s routines. Besides, we have done extensive experiments to evaluate the performance of autonomic management facility, and analyze its impact on the availability and management overhead of Grid system.

In the near future, we will propose more accurate availability evaluation and benchmark of management overhead on production Grid system.

REFERENCES

[6] Herrmann, K. etc, Self-Management: The Solution to complexity or just another problem? IEEE distributed system Online, 6(1), 2005
[12] Richard Rabbit, Tom McNeal, Tim Burke, A High-Availability Clustering Architecture with Data Integrity Guarantees, CLUSTER.01, Newport Beach, CA
[17] Chokchai Leangsuksun and Ibrahim Haddad, Building Highly Available HPC Clusters with HA-OSCAR, CLUSTER 2004