

Fault Tolerant Grid Migration Using Network Storage

Kodai Kagawa, Kazuya Yamada, Tokimasa Kamiya, Motoyasu Nagata
Osaka Kyoiku University
Kashiwara, Osaka 582-8582, Japan
TEL/FAX +81-729-78-3672
nagata@cc.osaka-kyoiku.ac.jp

Abstract—We will present a protocol of fault tolerant grid migration system using network storage. Our proposed grid migration system has capability to recover from fault in autonomic manner. We also implemented a grid migration system using Globus Toolkit4 and the network storage. Performance evaluation will be reported to show effectiveness of the fault recovery.

I. INTRODUCTION

For the mission critical grid computing system, non-stop fault tolerance is required even in the case of fault occurrence. The non-stop execution at alternative node is capable for fault occurrence at one node in the distributed grid system. For this non-stop execution in grid computing system, we are obliged to face problem that means who and when the recovery should be done. For manual recovery, it will highly cost. Therefore, such a grid computing system is considered to be desirable that the distributed system itself can detect the occurred fault and recover in autonomic manner.

We will present an protocol of fault tolerant grid migration system using network storage. Our proposed grid migration system has capability to recover from fault in autonomic manner. We also implemented the grid migration system using Globus Toolkit4 and the network storage. Performance evaluation about experiment for our proposed protocol will be reported to show effectiveness of the fault recovery.

Organization of this paper is outlined. Section 2 presents an protocol of our proposed system. Section 3 presents implementation about fault tolerant grid migration system using network storage. Section 4 reports performance evaluation of our proposed protocol. Section 5 concludes.

II. GRID MIGRATION ARCHITECTURE

The architecture of fault tolerant grid migration system is featured with a network storage and some nodes for computing. Figure 1 illustrates architecture of fault tolerant grid migration system.

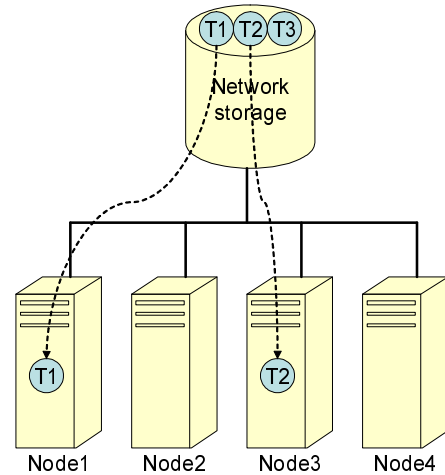


Fig. 1. Architecture

The most important facility of our proposed architecture is task migration. That is, each node can load one of tasks from the network storage. This task migration distinguishes our architecture from fixed allocation of task at assigned node. For modeling this grid migration system, we definite some notations listed below.

Nodes = N_1, N_2, \dots, N_p : A set of nodes

Tasks = T_1, T_2, \dots, T_p : A set of tasks

Execute(N) : A set of tasks being processed on node N

A. Taskgroup

The processes of this grid migration system are characterized by a "taskgroup". A taskgroup is a set of tasks which execute a series of processes. Generally, each taskgroup depends on another taskgroup. Figure 2 shows an example of distributed processes by taskgroups TG1 and TG2. T_d is a task that sends invoke request to the other tasks. The task sends processing result to a process T_d when the task completes its execution. After T_d waits for completion of all the tasks belonging to one taskgroup, the process T_d invokes tasks of the other taskgroup. This is called barrier synchronization. T_d receives all processing results and then sends these results to tasks of TG2, along with starting-up these tasks in concurrent

manner.

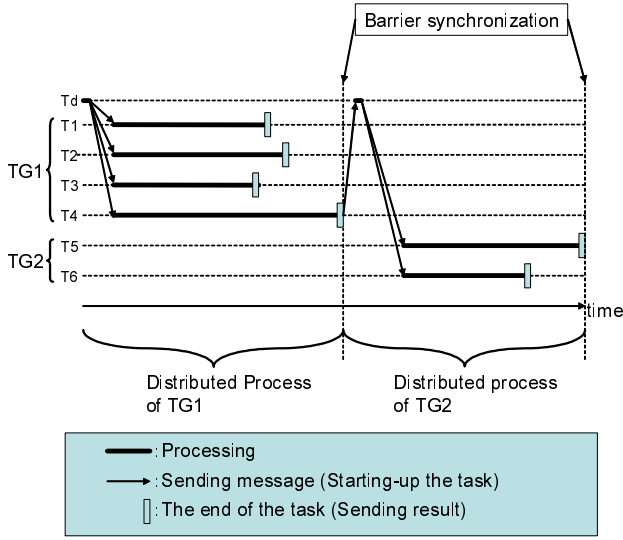


Fig. 2. Process

B. Monitoring

To realize the autonomic recovery, rapid recognition of the fault is very important. Delays of fault reorganization affects mean time to repair (MTTR), and so lessens availability. For monitoring all the nodes, our architecture allocates monitoring tasks to every node. Figure 3 shows situation where the monitoring tasks observe each node according to the pre-determined order. Once the monitoring task recognizes a fault, the recovery process is adapted, which we will describe later. At the same time, the node which had been monitored by the fault occurred node newly becomes monitored by the node which had been monitoring the fault occurred node. Figure 4 illustrates this change of monitoring. Thus, all alive nodes can be kept monitored at any time.

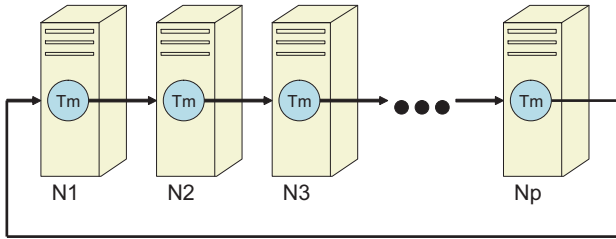


Fig. 3. Monitoring

C. Recovery Process

Once the monitoring task detects fault on a node N_i , the tasks which had been executed on the node N_i , expressed by $Execute(N_i)$, are to be executed on the other alive node. At this time, the recovery algorithm refers message log of the task T_d . Alternative nodes to which task is migrated are determined in pre-determined order. In our proposed protocol, it is no

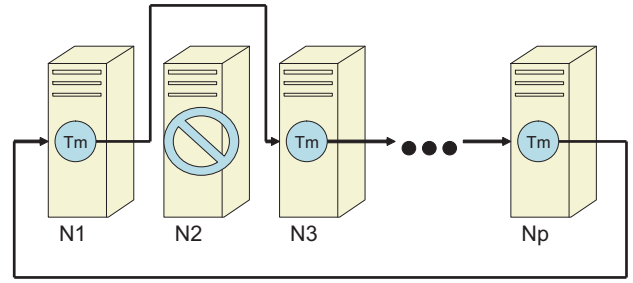


Fig. 4. In the case of failure

need to restart all tasks included in the task group that has a stopped task. As the tasks except for T_d are not assumed to communicate messages to another task, it is enough to restart only stopped task. A recovery algorithm is described below.

```

Obtain Execute( $N_i$ ) from message logs of  $T_d$ 
for  $T \in Execute(N_i)$  do
  Select  $N_j$  ( $j \neq i$ )
  Execute  $T$  on  $N_j$ 
  Send message to  $T_d$ 
end

```

III. IMPLEMENTATION

We will report implementation about our proposed grid migration system.

A. Grid Service

The Globus Toolkit 4, developed by Globus Alliance, is an open source middleware which implements functions to construct the grid environment. We used the Web Service Resource Framework (WSRF) which is a basis for developing the stateful web service, and developed our original grid service container on which tasks are processed.

B. Java Thread

A class-loader of Java helps loading on running program. In our protocol, class files corresponding to tasks are allocated on network storage. Each node can load necessary class files from network storage in dynamical manner, and run the loaded class as Java threads. The actual processes are executed by the above stated grid service. Figure 5 shows the basis of implementation on each node constructing grid environment.

C. Task Invoke

A client sends a request intermediated by the stub when the task is invoked. The stub calls process method, opened public by grid services, on the nodes using Simple Object Access Protocol (SOAP). Then process method loads specified class file assigned by Network File System (NFS) from the network storage. A class file allocated at the network storage is described as the following source code.

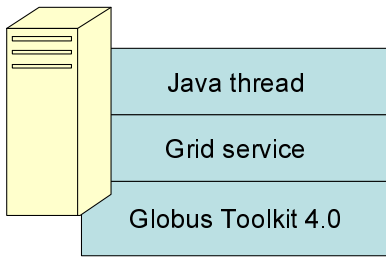


Fig. 5. Container on a node

```

import java.util.*;
import java.net.*;
import java.io.*;
import java.lang.reflect.*;

public class T1 extends Thread{
    public String in;
    public String out;
    public void run(){
        synchronized(this){
            try{

                //TASK

                notifyAll();
            } catch(Exception e){
            }
        }
    }
    public void start(){
        super.start();
    }
}

```

Figure 6 illustrates the procedure when a client makes the task T run at an assigned node. The stubs are allocated at all the nodes on grid, and so each node is able to become the client whenever it is needed.

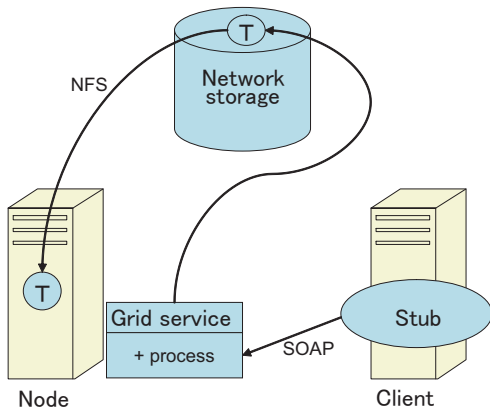


Fig. 6. Starting-up the task

D. Failure Detection

We will present fault detection policy in our protocol system. We supposed the failures occurred by physical factors, e.g. a sudden power failure or network failure. The Globus Toolkit 4 have a good fortune to throw the exceptions for such failures. We implemented our protocol system to request monitored nodes to run dummy tasks periodically. It is designed that if the exception is thrown, the system recognizes a failure on the node. Figure 7 illustrates fault detection policy.

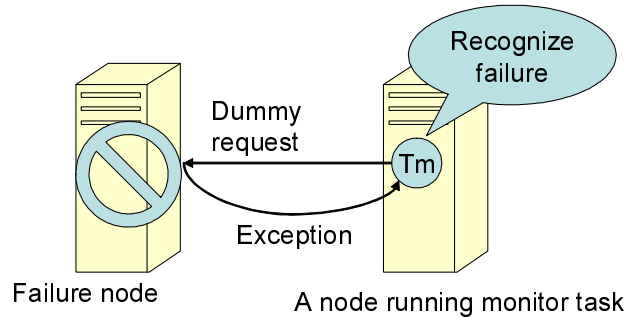


Fig. 7. Recognition policy

E. Message Log

The distribution task, Td, records sending message logs used for distribution of tasks. The message log consists of a set of task-group identifiers, IP addresses and transmitted messages. Figure 8 illustrates organization of message log. Each task has a task-group identifier to which it belongs. When taskgroup completes its execution, records including completed task-group in the message log are deleted. In the message log, number of records equals to the number of task-groups that is on processing. Furthermore, number of IP addresses and sending messages of each record equal to the number of tasks belong to a task-group. The message log enables to recognize $Execute(N_i)$ at failure node, and so to retransmit the message.

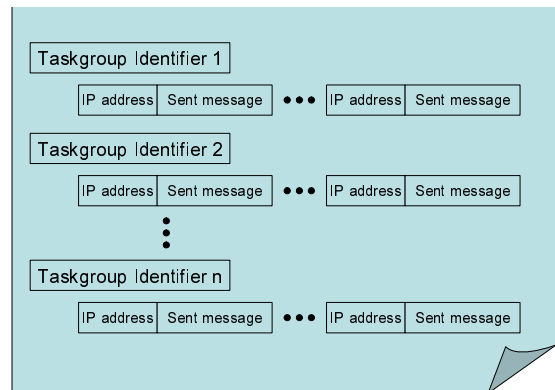


Fig. 8. Message log

IV. PERFORMANCE EVALUATION

We evaluated performance of our proposed fault tolerant grid migration system by executing parallel merge experiment. As an experiment, we allocated objective data on the network attached storage (NAS) and splitted original data file into sixteen files. Two of these splitted files are merged at each node. These eight files that has been merged in such a manner are allocated on network storage again. This merge procedure are repeated, and finally the files are merged to one. As experiment, we intentionally occured fault in the midst of the operation, and observed how the task which had been processed at faulted node is to be migrated. Through experiment, we will consider obtained result about CPU load ratios at nodes to confirm fault tolerance of our proposed prototype, We used NAS and 10 computers as nodes for experiment. The experimental environment is shown below.

Maker	OS	HD
Logitec	Windows Storage Server 2003	160GB

Specifications for the grid computing are illustrated below.

Host name	CPU	HDD	Memory
NODE-1	Intel Celeron 2.4GHz	80GB	512MB
NODE-2	Intel Celeron 2.4GHz	80GB	512MB
NODE-3	Intel Celeron 2.4GHz	80GB	512MB
NODE-4	Intel Celeron 2.53GHz	80GB	256MB
NODE-5	Intel Celeron 2.53GHz	80GB	256MB
NODE-6	Intel Celeron 2.53GHz	80GB	256MB
NODE-7	Intel Celeron 2.53GHz	80GB	256MB
NODE-8	Intel Celeron 2.53GHz	80GB	256MB
NODE-9	Intel Celeron 2.53GHz	80GB	256MB
NODE-10	Intel Celeron 2.53GHz	80GB	256MB

The environment factors which is common to all nodes is shown below.

OS	Developping environment
Fefora Core 3	Java jdk1.5.0

In this experiment, task allocation, monitoring and task migration are executed in the loop order according to the above table. For example, eight nodes processing the first step of the procedure are from NODE-1 to NODE-8. Furthermore, adjacent NODE-9 plays a role as distribution task Td, in turn. So, in the next step of the procedure, nodes after NODE-10 must be processing machines. Furthermore, destination node of the migration in fault occurrence is NODE[X+1], where NODE-X is the node to which the task with the lowest priority among the task group is allocated.

Figure 9 illustrates parallel merge in the experiment. Figure 10 shows observed result about CPU load ratios at each 10 nodes. At first, we will consider an aspect of the experiment without fault occurrence. The phases (i), (ii), (iii), (iv) in Figure 9, corresponding to periods (i), (ii), (iii), (iv) in Figure 10, stand for the taskgroups. From Figure 10, we can observe that the CPU load ratio at each node transits to the other node at several points. For example, at boundary between periods (iii) and (iv), the CPU load ratio at NODE-6 and NODE-7 transits to NODE-9. We can say that parallel processing is

correctly executed in the case without fault occurrence.

Secondly, we will consider an aspect of the experiment with fault occurrence. We caused an intentional fault to NODE-3 in the midst of phase (ii). The fault occurred at a point around 47 seconds in Figure 10. Immediately after the fault, the task of NODE-3 was migrated to NODE-4, and the whole operation was continued with success. On NODE-1, the monitoring screen had been displaying, so we observed a little CPU load at every times.

Among the processing nodes during phase (i), only NODE-8 showed excessive delay for starting of task processing. We guess this is caused by a slight network failure. But even in this case, the other nodes during phase (i) wait for completion of NODE-8 before switching to phase (ii). This is due to the barrier synchronization.

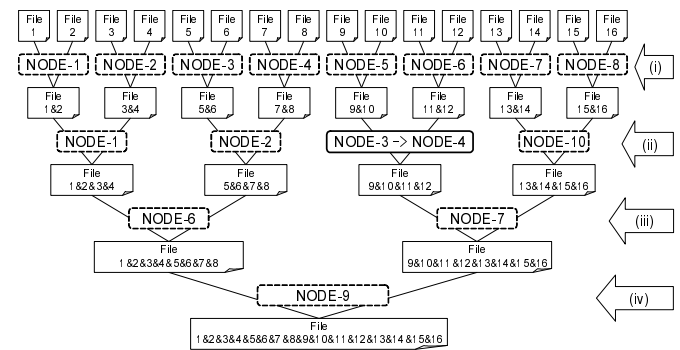


Fig. 9. Parallel merge

V. CONCLUSION

We proposed a grid migration system with fault tolerance that recovers from failure autonomically, and then implemented and evaluated our proposed prototype. We obtained effective result of task migration for fault occurrence in the grid computing.

REFERENCES

- [1] Japan IBM Systems Engineering Inc., "What is Grid Computing?", Softbank Publishing, 2004
- [2] Andrew S. Tanenbaum, Maarten Van Steen, "DISTRIBUTED SYSTEMS: PRINCIPLES AND PARADIGMS", PEARSON Education Japan, 2003
- [3] Philip A. Bernstein, Eric Newcomer, "Principle of Transaction Processing", Morgan Kaufmann Publishers Inc., 1997.
- [4] Globus Alliance
(URL) <http://www.globus.org/>
- [5] The Globus Toolkit 4 Programmer's Tutorial
(URL) <http://gdp.globus.org/gt4-tutorial/>
- [6] T. Jinno, T. Kamiya, S. Takeda, M. Nagata, "Grid Data Migration Using Service Data", SICE Annual Conference 2005, The Society of Instrument and Control Engineers, pp.1377-1380, 2005

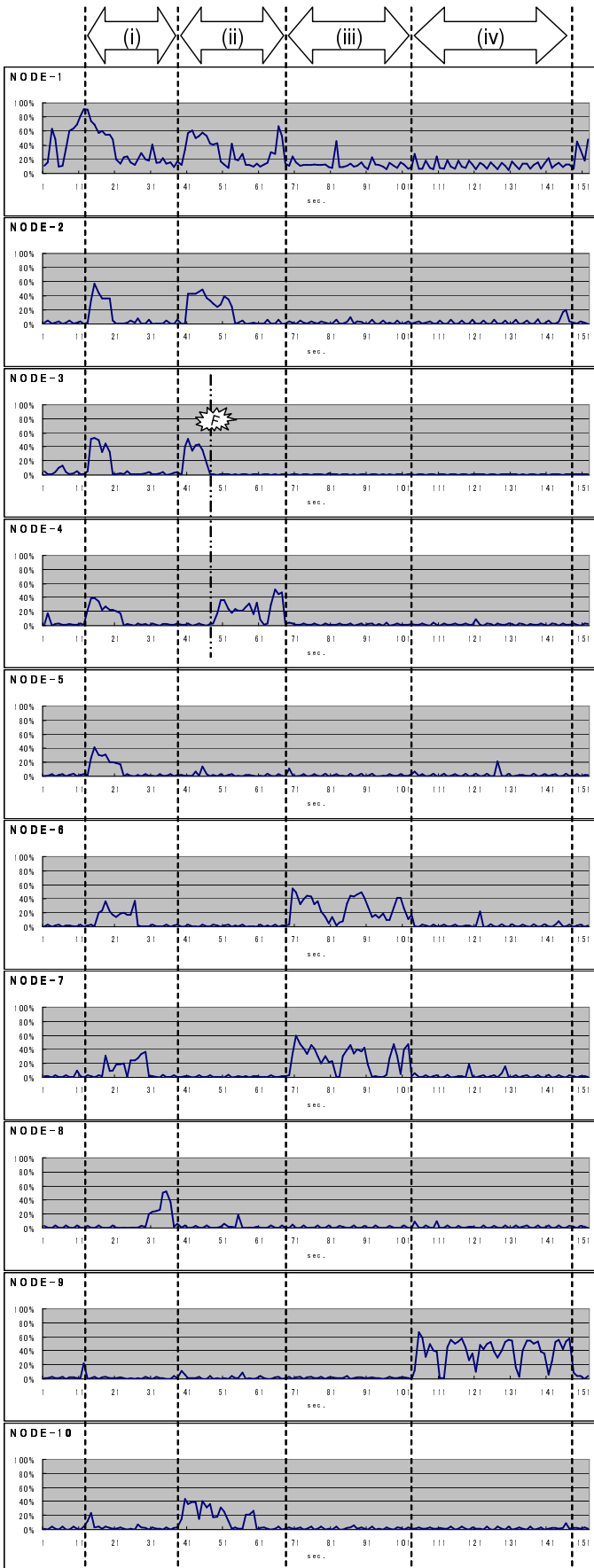


Fig. 10. CPU load ratios at each 10 nodes