High Availability Server Platform for Operations Support System

Nobuhiro Kimura[†], *Akira Yamada, Hikaru Seshake, and Toshihiro Nishizono*

Abstract

In the near future, operations support systems (OSSs) will be required to have higher availability. To cope with this, we propose three methods: (1) distributed server restart, (2) domain division restart by HA-SP, our high availability server platform, and (3) service recovery by application process functions. Our methods concern the platform of a computer system and require no changes in the application processes, so they are applicable to other kinds of systems. Applying these methods to an OSS for the intelligent network service of IMT2000 (International Mobile Telecommunications 2000) reduced the service down-time by 40%.

1. Introduction

Recently, with the growth of IP networks, the number of network elements (NEs) has increased rapidly and so has the number that one operations support system (OSS) [1] can manage. OSSs used to be used only by network operators to monitor and control the network for network service maintenance. Nowadays, however, not only carrier operators but also network service users control the network to open or change network services or monitor the network to check the service level agreement (SLA) between users and the carrier. This trend is leading to an increase in the number of OSS users, so any interruption of OSS services is becoming as serious as a network service interruption: the availability of OSS needs to be as high as that of the network, as shown in Fig. 1.

These days, computer systems, including OSSs, are composed of commercially available servers, such as workstations or personal computers for the following reasons.

- Systems can be built at a low cost.
- Prevalent servers are made of state-of-the-art technology.

We propose three methods that shorten the service down-time to speed up service offerings and maintain the operation service quality by using an OSS consisting of two or more ordinary servers, based on state-of-the-art technology. In general, clustering technology [2]-[4] is used to shorten service down-time. Our methods minimize the service down-time and enable complete system restoration by improving the clustering technology.

2. Current clustering technology

Clustering technology can be applied to ordinary servers that use state-of-the-art technology. It treats a set of servers as one system. When some processes fail due to the failure of an application process or when all processes on one server fail due to the failure of hardware, this technology detects the failure immediately and restores the failed process.

Suppose that a system is composed of two servers. When trouble occurs in one server, the failed processes are restored in either the same server or the other server as shown in Fig. 2. Trouble that may occur in the system is classified into two types.

- failure of an application process
- failure of hardware, for example the server's CPU or memory

† NTT Network Service Systems Laboratories
Musashino-shi, 180-8585 Japan
E-mail: kimura.nobuhiro@lab.ntt.co.jp

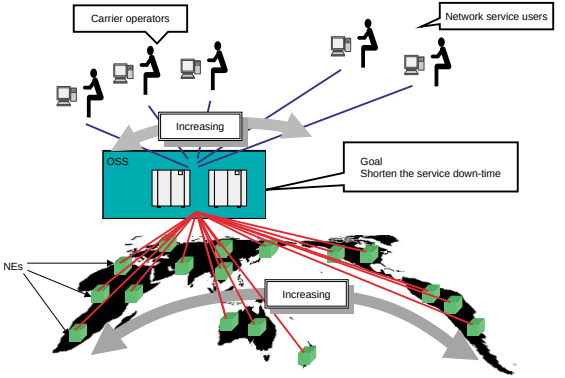

Fig. 1. Background.

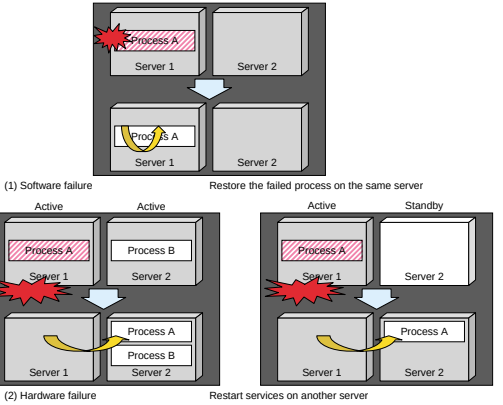

Fig. 2. Current clustering technology.

When a process on server 1 fails, clustering technology restarts it on the same server. On the other hand, when some hardware on server 1 fails, all the processes running on server 1 are restored on server 2 if server 1 is judged to be malfunctioning.

Server 2 can be active or standby. If it is on standby, it is started when server 1 fails. In this case, the performance of server 2 is the same as that of server 1. If server 2 is already active, running other processes, and the set of processes A is switched over to server 2, the performance of server 2 will drop because it is running two sets of processes. But when the system returns to normal, system resources can be used effectively on both servers again.

3. Study model

Generally, a service is provided through the cooperation of many processes. Figure 3 shows some relationships between services and processes. Cooperation between processes A and X provides service 1, that between B and X provides service 2, and that between C and X provides service 3. Processes can be classified into two groups: individual processes (processes A, B, and C) and shared processes such as process X.

An individual process only interacts with a shared process, which provides primitive services shared by an individual process. In this model, for example, if process B fails, service 2 cannot be provided until process B has been restored. However, services 1 and 3 can both be provided continuously.

On the other hand, if process X fails, none of the three can be provided because all three processes need to use its primitive services. Moreover, in a large system that has many processes, there is no guarantee that other processes will not also be affected by process X failing. In other words, we cannot predict how the failure of a shared process will affect services in such a system.

4. Requirements

There are two requirements for a high-availability OSS.

1. Complete system restoration.
   However severe the trouble, the system should be restored completely to provide services.
2. Minimal service down-time.
   Failure of the OSS monitoring a network masks

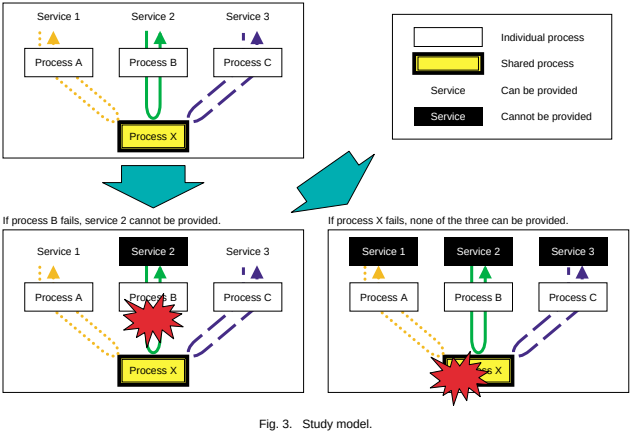

Fig. 3. Study model.

failures of the network. Therefore, OSS service interruption affects not only OSS users but also network users. Therefore the service down-time should be as short as possible.

To meet the above requirements, we propose the following three methods. The first is the "distributed server restart method", which restarts all the processes affected by the failure to enable complete system restoration. The second is the "domain division restart method" to localize the processes to be restarted. These two methods are individually provided in our high availability server platform called HA-SP. HA-SP can localize the process to be restarted. However, some service down-time is inevitable while a failed process is being restored. To shorten the service stoppage, our third method, the "service recovery method", achieves cooperation between the platform and application process functions. This method minimizes the time of service switchover from a failed server to another server. These methods concern the platform of a computer system and do not need any changes to applications.

5. High Availability Server Platform (HA-SP)

5.1 Distributed server restart method

The distributed server restart method can restart either individual processes or all processes as shown in Fig. 4. When an individual process A fails (top left of Fig. 4), only the failed process is restarted. If a shared process fails, individual processes A, B, and C cannot interact with the shared process until it has been restored. Even when a shared process has been restored, some processes may not be able to repeat events that involve interaction with the shared process. Thus, all processes are restarted to reestablish stable relationships between a shared process and individual processes (bottom left of Fig. 4).

On the other hand, when a server fails, the system is restored as follows. If all the processes on a failed server 1 are individual ones, processes on other servers provide services continuously and the failed processes are restarted individually on other servers (top right of Fig. 4). If there is at least one shared process on a failed server 2, not only the processes on the failed server but also processes on other servers are unable to provide services (bottom right of Fig. 4). In this case, the system isolates server 2 and all the processes are restarted on servers 1 and 3.

The process monitoring function is required to implement this method. The operating system (OS) assigns a process identity (PID) to a process when it

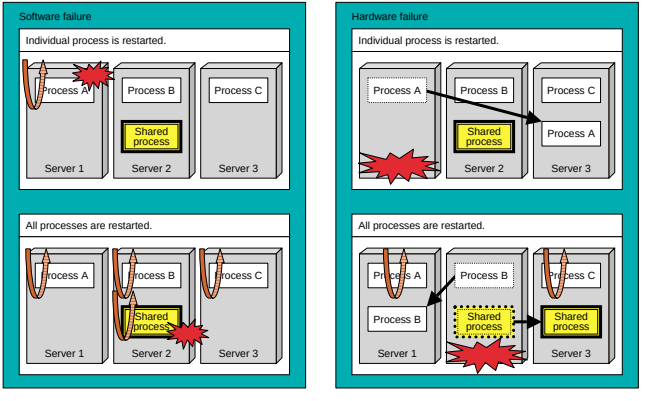

Fig. 4. Distributed server restart method.

starts. And OS manages PID correspondence with the process name until the process fails or stops. If it fails or stops, OS deletes the PID. This OS function enables the process monitoring function to get the PID when the process starts and continuously supervise it to detect process failure. Therefore, this method requires no change in the supervision of the process.

5.2 Domain division restart method

Because the distributed server restart method restarts all processes when a shared process fails, it suspends all the services provided by the system. To minimize service down-time, the processes to be restarted should be localized to isolate the effect of this method.

The domain division restart method divides a system into N domains, each having L servers as shown in Fig. 5. A shared process runs in each domain and processes related to the shared process run in the respective domains. Thus, each service can be processed within one domain. When a shared process in one domain fails, all the processes in that domain are restarted. However, processes in other domains need not be restarted and can provide services continuously. Consequently, the domain division restart method can localize the processes to be restarted within the domain.

5.3 Restart phase

Combining the distributed server and domain division restart methods, we define four restart phases as shown in Fig. 6.

Phase 1 restarts individual processes. It is applied when an individual process fails.

Phase 2 restarts all processes in one domain.

Phase 3 restarts the OSs of all servers in one domain. Either phase 2 or 3 is applied when a shared process fails.

Phase 4 restarts the OSs of all servers in all domains.

The recovery time increases in the order from 1 to 4 and the likelihood of recovery increases in the same order. When a failure occurs, an adequate restart phase that takes the shortest time for restarting should initially be selected to correct the failure. However, if that phase does not restore the failure, the method escalates up the restart sequence. If the system still cannot be restored by the “all restart phase”, this phase is repeated. In almost all cases, this method can restore a system completely and can shorten service down-time.

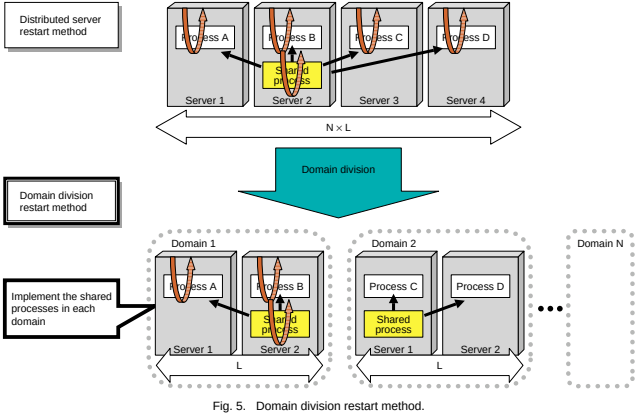

Fig. 5. Domain division restart method.

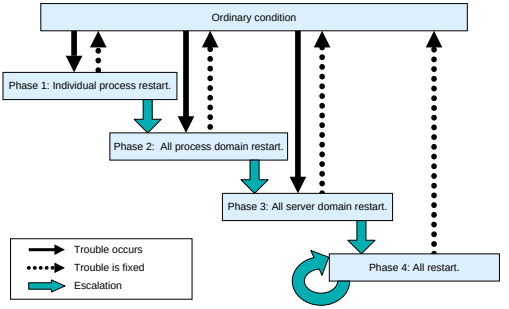

Fig. 6. Restart phase.

6. Service recovery method

The server should initialize all the processes in a domain when restoring services by applying phase 2 or 3 in the domain division restart method. In that case, the recovery time becomes long depending on the number of processes. Moreover, the server reboots the OS in phase 3. The recovery time becomes long with the reboot time depending on the number of processes, or the characteristics of the OS.

Here, we propose a rapid service recovery method that reduces the influence of the number of processes and hardware and OS characteristics.

Figure 7 shows a system composed of two domains, each having two servers. An active process and a standby process are located at the same time in different domains. The active process actually provides services while the standby process provides services only when the active one fails. Generally, before processes provide services, they have to be started and initialized. We defined the active and standby processes as follows.

- Active process: a process that has been started and initialized
- Standby process: a process that has been started but not initialized

This method simultaneously starts an active process and a standby process in the different domains. While active process A provides service in domain 1, this method can complete the OS reboot and start the standby process in domain 2. If the server in domain 1 fails, the server in domain 2 restores failed process A rapidly just by initializing standby process A. The server can skip the OS reboot and can start processes while restoring process A. Therefore, switching from active to standby processes hardly depends on the number of processes or OS characteristics.

Another merit of our proposal is low usage of system resources. Because the standby process is only started but not initialized, it consumes memory but no CPU resources. However, if more rapid service recovery is needed, active processes for the same services can be implemented in each domain.

7. Example of restart

These three methods enables complete and rapid service recovery. When a shared process fails in domain 1, all or a group of processes are restarted to achieve complete system recovery by the distributed server and domain division restart methods, as shown in the upper half of Fig. 8.

Thus, the more processes running in domain 1, the longer the restart takes. Therefore, these two methods should successfully shorten the down-time of services in domain 1. When the domain division restart occurs in domain 1, all standby processes (processes A and B) in domain 2 become active via the server recovery method. In turn, processes A and B in domain 1 restart as standby processes as shown in the lower half of Fig. 8.

Unlike the general clustering method, which

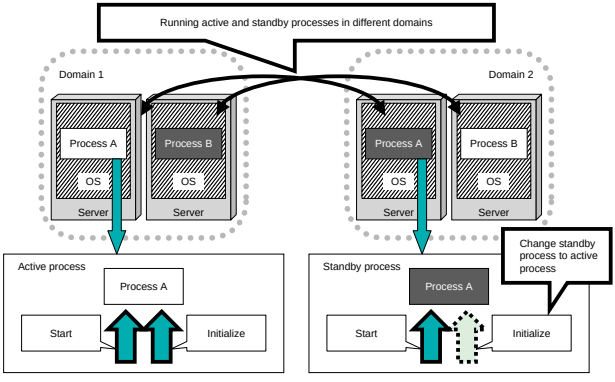

Fig. 7. Service recovery method.

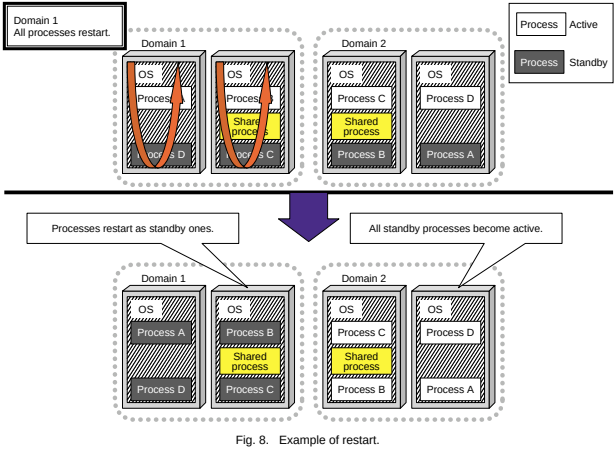

Fig. 8. Example of restart.

restores service but does not repair the failed domain or server, our domain division restart method restores domain 1 automatically. For this reason, if a failure occurs in domain 2 afterward, domain 1 can restore it without interruption.

8. NE supervision of OSS

Figure 9 shows an OSS example applying the three methods. The OSS is used for controlling and supervising NEs in an intelligent network service of IMT2000 (International Mobile Telecommunications 2000). The NE supervising function consists of a communication process (process X) and supervision processes (processes A and B). The communication process establishes connections between the NEs and the OSS. It also delivers messages from an OSS application process to an NE, and vice versa. The supervision process performs actions such as testing an NE, according to the messages it receives. In this model, NE and OSS communicate with each other by common management information protocol (CMIP) and events from NE are filtered by an event forwarding discriminator (EFD) [5], [6] on the NE. In our system, NEs send an M-event-report after the OSS has set the discriminator condition in EFD using the M-set requirements.

In this example, the OSS supervises 40 nodes. Process A (which manages NEs 1 to 20) runs in domain 1 and process B (which manages NEs 21 to 40) runs in domain 2 to balance the load. And their standby processes run in different domains so the service recovery method can be used. During initialization, active processes A and B send M-set requirements to EFD on the NE and are ready to receive events from NEs. However, the standby processes, which are not initialized, do not receive events from NEs. Process X is duplicated and runs in both domains 1 and 2 because the supervision processes share it.

If domain 1 fails, active processes A and X in domain 1 stop. And standby process A in domain 2 is initialized immediately and it sends the M-set requirement to EFD in NEs 1 to 20. Using this example system, we measured the service down-time before and after applying our three methods. The results are shown in Fig. 10.

In this OSS, there are about 80 processes for downloading firmware, updating NE files, and so on. Before we applied our three methods, there was only one shared process (process X) located in one of the servers. Each supervision process was also single and located in each server. When the shared process failed, all the processes were started and initialized.

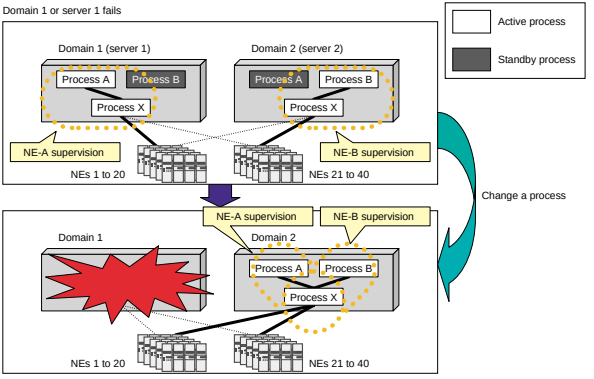

Fig. 9. OSS applying our three methods.

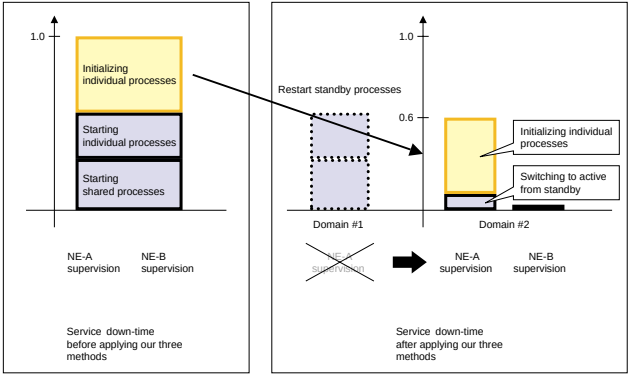

Fig. 10. Service down-time.

The services were interrupted until all restoration had finished. As shown in the left of Fig. 10, the service down-time consists of the starting of shared process, starting of individual processes, and initializing of individual processes. After we applied the three methods, the service down-time was shorter, as shown in the left of Fig. 10. The shared process failure in domain 1 did not affect the services for NE-B supervision because the shared process existed in domain 2. This means that the service down-time for NE-B was zero. For NE-A, compared with the left of Fig. 10, the down-time for starting the individual and shared processes was eliminated and replaced by the shorter time for switching the process from standby to active. As a result, the down-time was reduced by 40%.

While switching the process from standby to active in domain 2, the three methods restore domain 1. Therefore, if a failure occurs in domain 2 afterward, domain 1 can restore it without interruption. The time to restart standby process in domain 1 is shown as dashed line in the right of Fig. 10. This time don't affect the service down-time for NE-A and NE-B.

9. Conclusion

We proposed a high-availability distributed OSS that uses three new methods. The distributed server restart method can restart any process. The domain division restart method divides a system into N domains to limit the effect of the distributed server restart method. The domain service recovery method runs an active process and a standby process in different domains, and switches the process in another domain from standby to active if an active process fails.

These methods have been integrated into a high availability server platform called HA-SP that is applicable to various systems. The distributed server and domain division restart methods are independent of applications. The service recovery method is implemented through application functions. They were first applied to the OSS for the intelligent network of IMT2000. In this system, we could reduce the service down-time by 40%. We also applied HA-SP to the OSS for VoIP (Voice over IP) service provisioning [7], [8] and to the OSS for equipment management of an MPLS-based IP network. We were able to make these OSSs highly available without modifying the application programs. These methods enable a system to be restored by only restarting the failed process.

References

[1] Ishikawa, "High reliable switching node operation support system based on server," APCC/OECC, Oct. 1999.

[2] http://www.sun.com/service/products/suncluster/index.html

[3] http://www.hp.com/techservers/clusters/

[4] http://www.veritas.com/vcsguided/index.html

[5] ITU-T Recommendation X.734.

[6] H. Seshake, T. Oimatsu, and N. Akiyama, "DATA Communication Platform in Distributed Operation System Based on TMN," IEEE, Vol. 2, pp. 15-19, Apr. 1996.

[7] T. Ichijo, K. Yamane, M. Aso, and Y. Hibino, "The OSS architecture for VoIP service ordering based on the MSF network," APSITT2001, pp. 259-263, Kathmandu, Nepal/Atami, Japan, Nov. 2001.

[8] T. Furukawa and K. Yamane, "A Study on Service Order of VoIP Service Including SLA Information," APNOMS 2002, Jeju Island, Korea, pp. 120-131, Sep. 2002.

Nobuhiro Kimura

Research Engineer, the First Promotion Project, Network Software Service Project, NTT Network Service Systems Laboratories.

He received the B.E. and M.E. degrees in materials engineering from Nagoya University, Nagoya, in 1993 and 1995, respectively. In 1995, he joined NTT Network Service Systems Laboratories, Tokyo, Japan. He worked on the development of congestion control functions for the system, the OSS for IP networks, and the high-availability system platform. He is currently working on IP control and management service research.

Akira Yamada

Engineer, the First Promotion Project, NTT Network Service Systems Laboratories.

He received the B.E. and M.E. degrees in computer science from Nihon University in 1994 and 2000, respectively. Since joining NTT Network Service Systems Laboratories in 2000, he has worked on OSSs. His research interests include integrated operations systems and recently enterprise information portals.

Hikaru Seshake

Senior Research Engineer, the First Promotion Project, NTT Network Service Systems Laboratories.

He received the B.S. and M.S. degrees in science from Tokyo Institute of Technology, in 1990 and 1992, respectively. In 1992, he joined the Switching Systems Laboratories (now NTT Network Service Systems Laboratories), Tokyo, Japan. He worked on the development of congestion control functions for the system, adaptation technology for enterprise application integration, and the high-availability system platform. He is currently working on IP control and management service research.

Toshihiro Nishizono

Senior Research Engineer, Supervisor, Network Software Service Project, NTT Network Service Systems Laboratories.

He received the B.S., M.S., and Ph.D. degrees in information engineering from Kyushu University, Fukuoka, in 1978, 1980, and 1990, respectively. In 1980, he joined the Electrical Communication Laboratories, Nippon Telegraph and Telephone Public Corporation (now NTT), Tokyo, Japan. He worked on the development of DDX packet switching systems and ATM switching systems. From 1990 to 1993, he worked in Advanced Telecommunication Research Laboratories, Kyoto, Japan, on software specifications and distributed operating systems. From 1998 to 2001, he was engaged in international communication service deployment and IP service development in NTT Communications, Tokyo, Japan. He is currently working on IP control and management service research.