Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Future Generation Computer Systems 20 (2004) 303–313

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/future

An approach to intrusion tolerance for mission-critical services using adaptability and diverse replication

Byoung Joon Min [a,*], Joong Sup Choi [b]

[a] *Department of Computer Science and Engineering, University of Incheon, Incheon, South Korea*
[b] *Korea Information Security Agency, South Korea*

Abstract

In many mission-critical applications, important services should be maintained properly under any circumstances including the presence of compromised components incurred by outside intentional attacks. In the paper, a two-level approach for the intrusion tolerance is presented. At the node level, by means of dynamic resource reallocation within a computing node, the critical services previously selected are to survive even after the occurrence of an attack. If it becomes impossible to find enough resources for the critical services within the node in spite of the adaptive actions taken at the node level, it moves to the system level. The system level mechanism is to deliver the intended services transparently to the clients even when a node fails. An architecture adopting diverse redundant computing nodes is proposed for that purpose. Through the experiments on a test-bed, especially, for web services, the approach turned out very effective to cope with not only denial of service attacks but also confidentiality and integrity attacks. Although the measurement of the timing overhead incurred by the approach represents 50% loss in performance, it seem possible to decrease the cost by optimizing the implementation.

Keywords: Intrusion; Mission-critical services; Adaptability

1. Introduction

Most of the networked information systems adopt intrusion prevention mechanisms such as cryptography and authentication for the purpose of the confidentiality and the integrity of the information. Nevertheless, many successful attacks exploiting various vulnerabilities are found in the systems. Intrusion detection systems can effectively detect pre-defined attacks but have limitations in responding to continuously created novel attacks.

In the mean time, the concept of intrusion tolerance has been presented in recent researches [7]. The purpose of the intrusion tolerance is to maintain critical services at the acceptable level of service quality even in the presence of compromised components in a system due to intrusions. It urges to change the objective of security. The objective of security has been primarily elimination of the intentional threats on the operation of a system by creating barriers between the system and the threats. For complex open systems the barriers may be undesirable and impractical. Now, we need to consider applying some of the fault tolerant techniques to the system security for the construction of the system that can tolerate intrusions. In mission-critical systems, the need for intrusion tolerance is very high. In order to enforce the survivability

* Corresponding author.
E-mail addresses: bjmin@incheon.ac.kr (B.J. Min), jschoi@kisa.or.kr (J.S. Choi).

0167-739X/$ – see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-739X(03)00146-8

of the critical services that have to be maintained under any circumstances, the concept of adaptability and diversity from fault tolerance can be used.

The goal of the research presented in this paper is to enforce the survivability of pre-defined mission-critical services. In order to achieve this goal, two fundamental techniques from the fault tolerance and dependable computing researches are adopted. One is reallocating resources within a computing node, which is to enable the critical services to survive in the node even after the occurrence of a successful system attack. The resource reallocation may accompany the degradation of overall performance and the sacrifice of non-critical services. The other technique is applying diverse redundant servers at the system level. This is to continue mission-critical services transparently to the clients even when a node fails.

When a computing node is attacked, the impacts within the node can be classified into the following three groups in general. An attack called DOS (denial of services) exhausts resources such as CPU power, memory space, and I/O bandwidth. Or, some attacks may create unauthorized files or modify existing files. Lastly, by some attacks, system calls and commands are executed in the node [2].

In general, we need to take the following three major steps to respond to system attacks for the purpose of intrusion tolerance: intrusion detection, isolation of compromised components, and recovery [5]. The basic idea of the approach proposed in this paper is to adopt the mechanisms developed at the node level and at the system level as summarized in Fig. 1.

A two-level approach for the intrusion tolerance is presented in the paper. It aims at enforcing the survivability of critical services even in the presence of compromised component in computing nodes. At the node level, each node tries to adaptively reallocate its resources for the critical services selected by the system manager. A reallocation algorithm within a node is proposed. The algorithm is helpful when enough resources are found in the computing node after the reallocation procedure. If it becomes impossible, delivery of the services should rely on another computing node. For the system level, redundant computing nodes are applied. The redundant architecture proposed in the paper is composed of *N* concurrent active nodes and one hot standby backup node. There is also a surveillance node to control and mediate the computing nodes. The servers running on the computing node are based on diversity. They use different software modules on different platforms with the same design goals. Both acceptance test and vote are used to detect the abnormal situation. Upon the detection, the suspect node is isolated and the backup newly joins as one of the active nodes. When the suspected node is repaired, it becomes a new backup under the supervision of the surveillance node.

The rest of the paper is organized as follows. Section 2 summarizes related research results and explains the contribution of the research presented in the paper. In Section 3, a resource reallocation algorithm is presented. It describes the actions to be taken at the node level in order to response the attacks exhausting resources in the node. Section 4 is to explain an architecture adopting diverse replication. It includes the operational steps in detail for the system level response. For the validation of the approach, a test-bed has been implemented and experiments have been conducted. In Section 5, the results of the experiments are presented. Finally, Section 6 concludes the paper.

	Detection	Isolation	Recovery
Node Level • Resource reallocation	Monitoring resource usage	Reallocating resources for critical services	
System Level • Diverse redundant servers • Surveillance through out of band network	Both acceptance testing on results and comparing results of servers	Cutting-off the compromised node by reconfiguration	Rejoining the recovered node under the control of the surveillant

Fig. 1. Basic idea of the proposed approach.

2. Related works

Intrusion tolerance is to be prepared for the still remaining vulnerabilities of a system. Unless identifying all the vulnerabilities of the system, intrusion prevention and intrusion detection cannot be perfectly worked out [1].

One of the important research efforts related to the intrusion tolerance has been conducted in IST (information security technologies) programs at European Commission [9]. It has contributed toward the dependability support to enforce the trust of systems. Another important research activities have been carried out with the support of DARPA.

In [3], the concept of resource reallocation is presented as a security responsive technology. A managing controller of a system decides which action should be taken with a limited amount of communication and computing resources. With the proper resource reallocation, pre-defined critical processes can be protected from a certain type of security attacks. The concrete algorithm and demonstration of resource reallocation, however, have not been fully presented yet.

One of the previous researches on diverse replication is found in HACQIT (Hierarchical Adaptive Control of Quality of service for Intrusion Tolerance) project [6]. In the architecture, a firewall controls the access to the servers. The primary and backup servers provide the desired services. A monitor controls the servers, and has an out-of-bands communication channels so that services can be reconfigured dynamically as needed. The intrusion detection relies on the conventional error detection mechanism. If the results of the two servers are not same, it decides that one of the two is compromised based on a fairly simple comparison of the status codes generated from HTTP servers. The architecture can be easily constructed with COTS (Commercial Off The Shelf) servers. However, the detection mechanism is too weak to handle more complicated situations. Besides, it is hard to extend the architecture once an attack is treated since the goal itself of the architecture is to provide 4 h of intrusion tolerance.

Another example of diverse replication is SITAR (Scalable Intrusion-Tolerance Architecture) [8]. This architecture is composed of proxy servers, acceptance monitors, ballot monitors, an audit controller, and an adaptive reconfiguration module with COTS servers. Proxy servers provide public access points for COTS servers. The acceptance monitors apply certain validity check to the responses, and forward them to the ballot monitors along with an indication of the check results. The ballot monitors serve as representatives for the respective COTS servers and decide on a final response. The adaptive reconfiguration module receives intrusion trigger information from other module, and generates a new configuration according to the evaluation. This architecture is scalable. The configuration is very flexible. However, it seems very hard to implement and impractical because of its cost and complexity.

The contribution of this paper is demonstrating a practical approach to intrusion tolerance. A concrete algorithm for resource reallocation is developed as an adaptive response to the attacks exhausting resources. By combining the adaptability of each server and the diversity of replicated servers, we can reduce the cost incurred by frequent system reconfiguration while considering the aggregated user performance. The approach proposed in the paper aims at handling not only denial of service attacks but also confidentiality and integrity attacks with reasonable amount of timing overheads.

3. Resource reallocation algorithm

In this section, an algorithm for resource reallocation within a node is presented. It is to maintain critical services even after the occurrence of intrusion, especially DOS compromise, in a computing node.

The services are divided into two classes: critical and non-critical. The algorithm is concerned with the critical services. Executing non-critical services in a node is just to obtain better aggregate user performance.

The resource reallocation in a node is to provide enough resources for the critical services. This can be achieved by sacrificing non-critical services. One important issue here is how to evaluate the survivability of critical services, that is, whether enough resources for the critical services exist or not. Another important issue is which non-critical service should be terminated first in the reallocation procedure.

For the execution of a service in a computing node, it has to provide a certain level of resources including

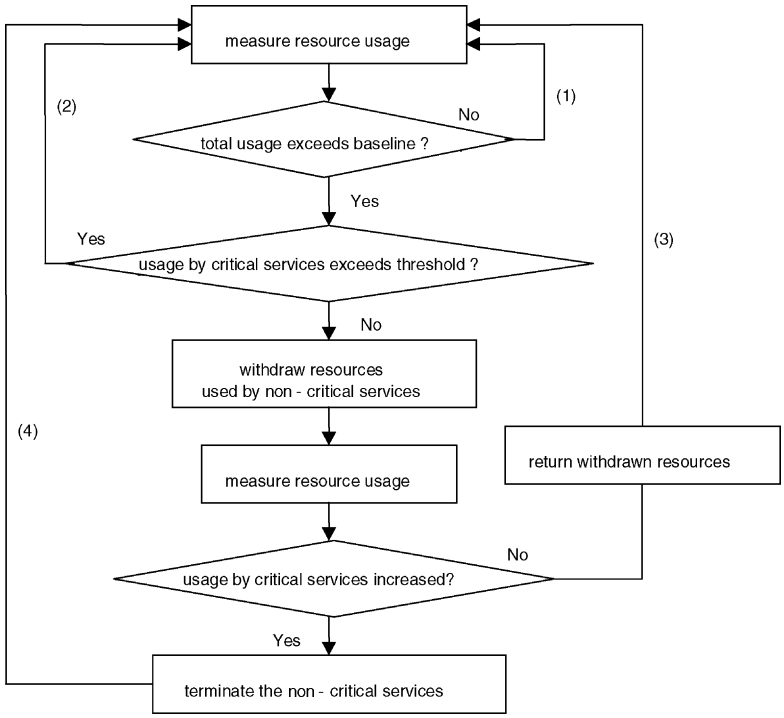

Fig. 2. Resource reallocation in a node.

CPU, memory, and I/O. However, it is not easy to assert the exact level of the required resources for each critical service. Of course, it is impractical to reserve resources in advance for all critical services. In order to resolve the first issue mentioned above, baseline and ATEEB (accumulated time elapsed in excess of the baseline) are used in the algorithm. The baseline of a service implies the minimum amount of resource shares required in order to provide the service at the minimal acceptable level of the service quality. It can be established for each resource, for example, CPU, memory, and I/O. The ATEEB is used to compromise the abruptness that may happen in an instant. It is assumed that the ATEEB of a service should be greater than a threshold when the service is delivered properly.

In order to resolve the second issue mentioned above, the accumulated times not only for critical services but also for non-critical services with default baselines are measured periodically. When a critical service needs more resources, a non-critical service holding most of the resources is sacrificed first.

Fig. 2 depicts the details of the algorithm. The numbers denoted next to the arrows in the figure represent the actions to be taken with the corresponding numbers as follows:

(1) Baseline for the resource usage in total is also established. When the ATEEB for the total usage is below the threshold, which means that there exist plenty of unoccupied resources, the algorithm keeps on monitoring the resource usage.
(2) When the ATEEB for the total usage becomes above the threshold, the amount of resources occupied by each critical service should be checked. If the ATEEB of a critical service is above the threshold, the algorithm keeps on monitoring the resource usage. Otherwise, the algorithm needs to figure out the reason why it cannot obtain enough resources.
(3) In order to recognize the circumstance, the algorithm withdraws some amount of resources intentionally by controlling the priority of the process for the non-critical service occupying the

most of the resources. If the ATEEB of the critical service does not increase even when more resources become available, the algorithm concludes that the critical service does not require more resources, and it returns the withdrawn resources to the non-critical service before gets back to the monitoring stage.

(4) However, if the ATEEB of the critical service increases after taking resources from the non-critical service, which means that they have been competing for the resources, the algorithm tries to terminate the non-critical service by force.

With this algorithm, resources for critical services can be provided. When a critical service selected by the system manager depends on other services, they also should be protected as critical services. This algorithm assumes that there is no threat within the critical services. If an attack sneaks into a critical service component to exhaust resources, the algorithm would not be able to allocate enough resources for other uncompromised critical services. This is the limitation of the algorithm. Nevertheless, it will be a cost-effective protection mechanism within a node for critical services against DOS attacks.

4. Replicated server architecture

When it becomes impossible to find enough resources for the execution of critical services within a node, the critical services should rely on another computing node prepared in the system. This section presents the architecture for the system level response.

In general, we can divide the replication into two classes according to the status of the replicated node. One is active replication and the other is passive replication. In the active replication, a set of active nodes executes the request concurrently, and majority is voted as the result. The number of replicated nodes depends on the fault model and the execution conditions. A failed node may roll back to a safe state called checkpoint and retry the execution for the recovery. In the passive replication, one primary node is in active while other backup nodes are in passive. The primary node that has passed its acceptance test delivers the result. If the primary fails its acceptance test, one of the backup nodes takes the role of the primary. Again, the status of the backup nodes can be either hot standby or cold standby. A hot standby backup node is always ready to be in the active state, while a cold standby backup node needs some amount of time to make its proper status.

An architecture based on $N + 1$ replication using both acceptance test and vote is proposed in this paper. It has the following characteristics:

- The architecture consists of N active nodes and one hot standby backup node. A front-end node provides clients with the access point on behalf of the server nodes. A surveillance node is connected with the other nodes through a separate out-of-band network.
- Since the backup node is always ready to be in active, not only the fast reconfiguration is possible but also consecutive attacks can be treated with the architecture.
- By utilizing both acceptance test and vote for the intrusion detection triggering, it is possible to reduce the latency incurred by the replication and to increase the detection coverage with the reduced dependency of acceptance test on the application.
- In order to raise the aggregate performance of the replicate nodes, synchronization among active nodes is very restricted. Most of the time, each computing node asynchronously responds to the requests as they arrive in the node. Only the responses of critical services are collected and forwarded by the surveillance node.

Using both acceptance test and vote is also to overcome their limitations. In general, two aspects have been tested by the acceptance test, one is testing logical reasonableness of the result and the other is testing on the bound of the execution time. Since the test requires a certain level of understanding of the application, it is hard to generate a high coverage acceptance test in a systematic way. In the vote, since it waits all the results from the replicas, the system performance is tied up with the slowest node. As we adopt both acceptance test and vote for the intrusion detection, the acceptance test does not have to be complicated to produce the high coverage and at the same time the voter has little chance to wait in vain for the result from a failed node.

The number N depends on the number of intrusions to be tolerated during the execution time of a request. According to the Byzantine algorithm in [4],

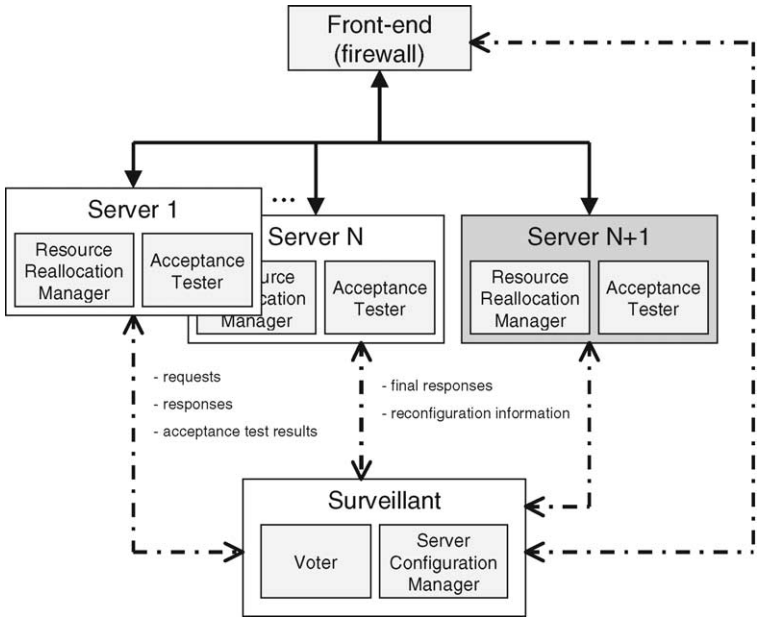

Fig. 3. The replication architecture.

the following formula satisfies: $3t < N$, where t is the number of nodes which could go wrong in an arbitrary manner except disturbing the execution of other normal nodes. With this assumption, if two thirds of N nodes have passed their acceptance with the same result on a request, the surveillance node can deliver the result to the client without waiting other responses from the rest.

Fig. 3 shows the replication architecture proposed in the paper. In the figure, initially, Sever 1 to Server N are concurrent active nodes and Server $N + 1$ is a hot standby backup node. Since the surveillant is connected to other nodes through a separate network as denoted with dotted lines, the clients including malicious attackers cannot access the surveillant. The operational scenario is as follows:

(1) The clients' requests to critical services are delivered to N active nodes through the front-end node. Active servers execute the same requests to critical services autonomously. Results of requests to the critical services are forwarded to the surveillant along with the acceptance test results. On the other hand, a request to a non-critical service is responded by one of active nodes and the response is delivered to the client through the front-end node.
(2) Intrusion detection can be triggered either by an acceptance tester in each node or by the voter in the surveillant. Once an intrusion is detected, the compromised node is isolated from the system by the surveillant, and the backup node replaces it immediately. In order to do that, the backup node has kept on updating its status with the help of the surveillant.
(3) When the compromised node is repaired, it rejoins the system as a new backup node. Again this backup node updates its status with the coordination of the surveillant.

5. Implementation and experiments

This section presents the results of the experimental work on a test-bed implemented in order to validate the two-level approach proposed in the paper.

5.1. Implementation of a test-bed

For the diversity, the IIS (Internet Information Services), Apache, and Tomcat servers on either Windows or Linux platforms for HTTP-based web

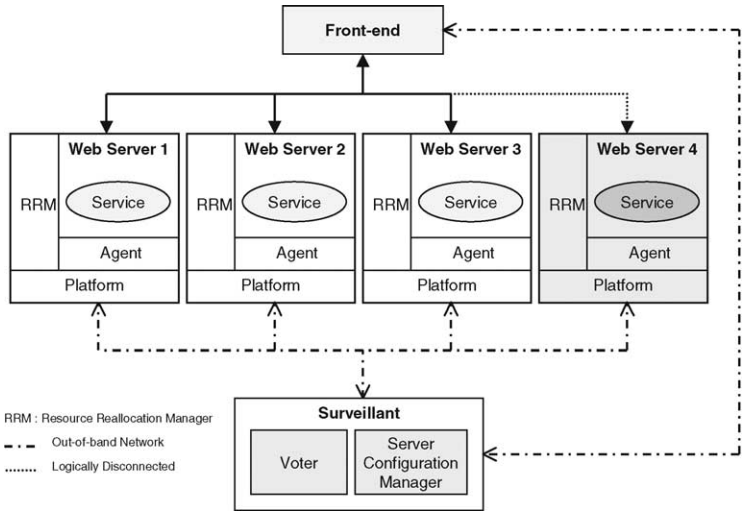

Fig. 4. Test-bed structure.

applications have been chosen in the implementation. They are the most commonly used web servers in the Internet and many add-on software modules have been developed. The server has two types of codes: one is to simply present contents, and the other is to provide functional services. Since the latter can be the threat to the system, service requests containing script codes with data write operations are considered in the experiment.

Fig. 4 shows all the components developed for a test-bed. It consists of four server nodes including an initial backup node represented as web Server 4 in the figure, a front-end node, and a surveillance node.

5.1.1. Implementation of resource reallocation

The algorithm for resource reallocation described in Section 3 has been implemented as the RRM (resource reallocation manager) module in Fig. 4. The RRM implemented as an application object provides a user interface to establish critical services with the baseline and the ATEEB. The RRM uses services internally provided by two threads called HMT (health monitor thread) and SET (survivability evaluation thread). The internal relationship among these components is described in Fig. 5. The HMT collects performance related counters from the operating system through API (application program interface). The information is forwarded to the SET that is actually executing the testing part of the algorithm in Fig. 2. The result of the evaluation is reported to RRM. Then the RRM uses the API to control the priorities of services or to terminate services. The RRM reports its status periodically to the SCM (server configuration manager) in the surveillant.

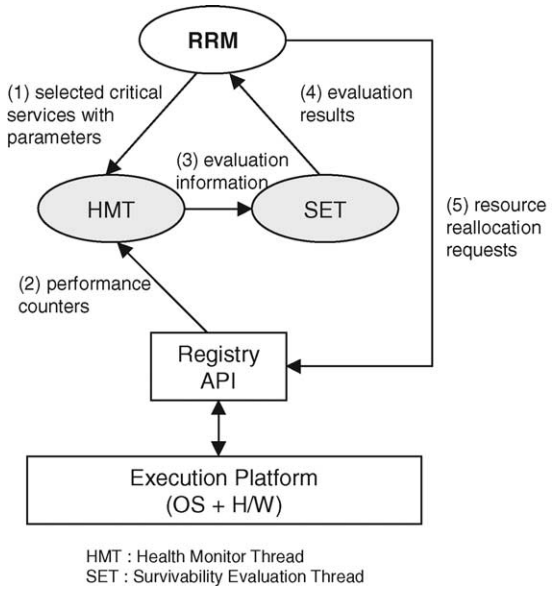

Fig. 5. Internal structure for resource reallocation.

```
class ReqRespMessage
{
   long seq_id;              // ID of request
   String hostIP;            // IP address of the client
   String time               // time when the request arrived
   byte[ ] request;          // request data
   byte[ ] response;         // response data
   MemberInfo member;        // role and status of each member node
}

class MemberInfo
{
   String nodeIP;            // IP address of member node
   byte role;                // role of each node,
                             // 0:front-end, 1:active node, 2: backup node,
                             // 3: surveillant
   String service_name;      // service name
   byte test_result;         // acceptance test result
   byte node_status;         // node status, 0:normal, 1:failure, 2:rejoin
}
```

Fig. 6. Data structure of message.

5.1.2. Implementation of replicated server architecture

The architecture presented in Section 4 has been implemented as the front-end, the agent, and the surveillant in Fig. 4:

(1) *Implementation of front-end*. The front-end receives requests from clients. First, it recognizes the service class. For the request to a critical service, it puts a unique identification number and the arrival time to the request before forwards the request to all active nodes. In order to process multiple requests simultaneously, it uses a pool of connection objects. A message format described in Fig. 6 is used to forward the information to the agent in each active node. ReqRespMessage class of the message contains not only the request and response data but also the information on each node's role and status. On the other hand, a request to non-critical service is distributed to one of active nodes in a Round Robin style.

(2) *Implementation of agent*. The agent receives the message containing the client's request as shown in Fig. 6. After the request is executed, it conducts acceptance test on the result. Test is based on the specification of the web application. Another function of the agent is to forward the execution and acceptance test results to the surveillant. Before forwarding the message to the surveillant, the agent unifies various URL extensions caused by diversity of web servers and platforms.

(3) *Implementation of surveillant*. As the surveillant receives messages from agents of actives nodes, it conducts vote. The vote is completed with the first two identical responses arrived. The voted response is stored in its own database and also forwarded to the backup node and the front-end node.

SCM controls the reconfiguration of the system. When intrusion detection is triggered, it isolates the suspected active node from the system. The isolated node is replaced with the backup node.

5.2. Experimental results

Experiments have been conducted in order to verify the ability of the system to respond to simulated attacks and to measure the timing overheads or the loss in throughput introduced by the mechanisms explained so for.

For the experimentation, various web servers including IIS and tomcat were used on diverse platforms as shown in Fig. 7.

On the platforms, the following three types of service requests were considered:

- Type A: Simple read of HTML file or DB data only.
- Type B: Script execution, mainly simple computation.
- Type C: DB write operation included.

	Server 1	Server 2	Server 3	Server 4
CPU	P3-600MHz	P4-1.7GHz	P3-833MHz	Duron-800MHz
Memory	128MByte	256MByte	256MByte	256MByte
Operating System	Linux 7.0	Windows2000	Linux 7.0	WindowsXP

Fig. 7. Platforms used for the experiments.

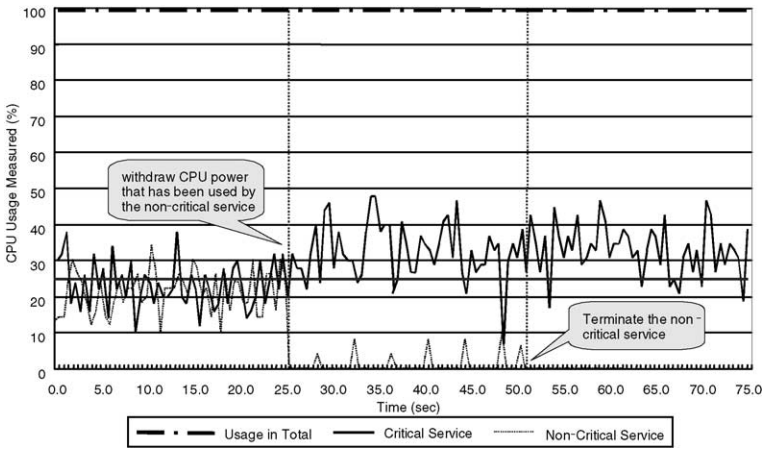

Fig. 8. An example of resource reallocation experiment results.

In order to check the responsiveness of the system, the following two types of attacks have been simulated in the experiments.

(1) *DOS attack*. The attack is simulated by introducing a heavily resource consuming process into a computing node, such as video streaming. By triggering the process as a critical and/or non-critical service in the middle of normal operation, a DOS attack can be simulated.
(2) *Integrity attack*. By inserting a program that modifies files or data in the database, integrity attack can be simulated.

Fig. 8 shows an example of results of experiments on resource reallocation. For the experiment, on Server 2 of Fig. 7, a video application as a non-critical service is triggered to operation in order to simulate a DOS attack. In the figure, horizontal axis represents time and the vertical axis represents CPU usage. The sampling of monitored information is done in every 0.5 s. The RRM makes decision to control the priorities of services in every 25 s in the example. The baseline of the critical service is set to 25%. The amount of CPU resources used by the critical service is denoted by darker line. The thinner dotted line represents the resource used by a non-critical service. At the point of 25 s, since ATEEB is down below the threshold, the RRM puts lower priority on the non-critical service. During next 25 s, the resource usage of the critical service has increased. Therefore, RRM decided to terminate the non-critical service at the point of 50 s.

The sampling rate affects the timing overheads. The RRM decision period should be selected carefully. If it is too short, RRM may make a misjudgment inducing unnecessary termination of non-critical services. On the other hand, it reacts very slowly if it is too long.

In order to measure the timing overhead incurred by the resource reallocation and diverse replication, the average execution time (turn around time) of above three types of requests on a typical clustered

Request Pattern	Server 1	Server 2	Server 3	Server 4	Average
A	24.5	14.9	21.9	26.3	21.9
B	12.1	10.1	7.5	19.6	12.3
C	33.3	24.8	27.4	26.9	28.1

(unit : msec)

Fig. 9. Turn around time without the approach.

Request Pattern	Time Elapsed in Each System Component			Total Time	Overhead (Overhead / Total Time)
	Front-end	Server including RRM	Surveillant		
A	1.7	41.6	1.4	44.7	22.8 (51%)
B	1.9	18.8	1.0	21.7	9.4 (43%)
C	1.8	49.2	1.3	52.3	24.2 (46%)

(unit : msec)

Fig. 10. Timing overhead measured.

server architecture first. Data consistency is maintained, and the requests are distributed in the Round Robin fashion. Fig. 9 represents that the average turn around times for request types A–C when the proposed approach is not applied are 21.9, 12.3, and 28.1 ms, respectively. The numbers were obtained through more than 10 times measurements during 120 s interval each.

With the same condition of the request generation, average turn around times on each request type were measured on the test-bed implemented. The results are summarized in Fig. 10. As shown in the figure, most of the timing overhead is introduced in the execution of the server nodes, which consist of RRM, agent, and service objects.

Since the Java language was used for the implementation, the execution speed was dragged down especially for the monitoring in RRM. Another obstacle is from the difference of the platforms. The agent wastes times to maintain connections with diverse servers. It seems the limitation in utilizing COTS components in the implementation.

In order to make the approach more practical some other aspects should be considered in further study. Front-end can be implemented in hardware to reduce both timing overhead and security vulnerabilities. Coordination with IDS will be desirable. A smart vote is needed rather than a brute force comparison of the result data.

6. Conclusion

An approach to intrusion tolerance has been presented in this paper. The approach functions in two levels. At the node level, the dynamic resource reallocation algorithm helps the critical services previously selected by the system manager survive even after the occurrence of DOS attacks. If an attack sneaks into a critical service component to exhaust resources, the algorithm would not be able to allocate enough resources for other uncompromised critical services. In such a case, a system level mechanism is needed to keep on delivering the intended critical services transparently to the clients. The $N + 1$ replication architecture has been proposed in this paper to provide a system level support. As long as we can assume that only one computing node out of three nodes with diversity is compromised during one execution cycle, concurrent diverse replicas can continue the mission. By utilizing both acceptance test and vote for the intrusion detection triggering, it is possible to reduce the latency incurred by the replication and to increase the detection coverage and accuracy with the reduced dependency of acceptance test on the application.

Experimental results obtained on a test-bed reveal the validity of the proposed approach. Especially, for web applications, it can handle not only DOS attacks but also confidentiality and integrity attacks with reasonable amount of timing overheads. By combining the adaptability of each server and the diversity of replicated servers, we can reduce the cost incurred by frequent system reconfiguration. In order to make the approach more practical, we will need to consider a tradeoff between the throughput of the system and the prompt responsiveness to attacks in the further study.

Acknowledgements

This work is supported by University IT Research Center Project, by RRC at the University of Incheon,

and by Korea Information Security Agency Research Project.

References

[1] M.A. Hiltunen, et al., Survivability through customization and adaptability: the cactus approach, in: DARPA Information Survivability Conference and Exposition, 0-7695-0490-6/99, 1999, pp. 294–306.
[2] A.P. Moore, Attack modeling for information security and survivability, Technical Node No. CMU/SEI-2001-TN-001, CMU 2001.
[3] Securing the US Defence Information Infrastructures: A Proposed Approach, Technical Report, National Security Agency, Defence Advanced Research Projects Agency, Office of the Assistant Secretary of Defence, 1998.
[4] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, J. ACM 27 (2) (1980) 228–234.
[5] B. Randell, Dependability—Unifying Concept, Computer Security, Dependability and Assurance: From Needs to Solutions, 1998. ISBN 0-7695-0337-3/99.
[6] J. Reynolds, et al., The design and implementation of an intrusion tolerant system, in: Proceedings of the International Conference on Dependable Systems and Networks, Washington, DC, June 2002, pp. 258–290.
[7] V. Stavridou, et al., Intrusion tolerant software architectures, in: Proceedings of the DARPA Information Survivability Conference and Exposition, Anaheim, June 2001. ISBN 0-7695-1212-7/01.
[8] F. Wang, et al., SITAR: a scalable intrusion-tolerant architecture for distributed services, in: Proceedings of the 2001 IEEE Workshop on Information Assurance and Security US Military Academy, West Point, NY, June 2001, pp. 38–45.
[9] M. Wilikens, et al., Defining the European dependability initiative, in: Dependability and Assurance: From Needs to Solutions, Report of the Workshop on Dependability of Critical Systems and Services in the Information Society, Italy, December 1997.

Byoung Joon Min received his PhD degree in electrical and computer engineering from the University of California, Irvine, in 1991. He is currently an Associate Professor at the University of Incheon, South Korea. Before joining the university in 1995, he was a research staff at Samsung Electronics and at Korea Telecom. His research interests are in dependability of distributed systems and networks.

Joong Sup Choi received his BS degree from University of Incheon in 1993, his MS and PhD degrees in computer science from Soongsil University, South Korea, in 1995 and 2000, respectively. He is currently a senior member of technical staff at Korea Information Security Agency. From June 1995 to January 1996, he worked at National Computerization Agency, South Korea, as a member of technical staff. His current research interests include intrusion tolerance technologies, embedded real-time systems, and distributed computing systems.