2.3 – High-Availability Solutions to Common Software Failures

Frédéric Michaud
Frédéric Painchaud
Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North, Québec, QC, Canada, G3J 1X5

August 16, 2006

Abstract

The Canadian government, especially the Department of National Defence and the Canadian Forces, has a strong need for secure and reliable information systems. Currently used, mass-market systems tend to be very poor with respect to security and reliability, since they routinely contain serious bugs and vulnerabilities. While a redesign of these systems would be a sound long-term solution, an effective short-term solution is an imperative. Mature high-availability products for mass-market operating systems are now available and we believe they could effectively and robustly prevent the effects of some classes of failures. We propose to evaluate this hypothesis.

Introduction

The Canadian government, DND[1], and CF[2] are increasingly dependent on information systems, which need to offer a very high level of security, reliability, and fault-tolerance [1]. However, current information systems are inadequate for many reasons. First, many legacy systems currently in use were designed before the Internet was widespread and were not supposed to be exposed to a world-wide network. Second, they were built with technologies and programming languages that are prone to vulnerabilities and bugs that were not known at the time (e.g., buffer overflows). Since these systems are needed for a foreseeable future, special care should be taken to prevent the

[1] Department of National Defence
[2] Canadian Forces

exploitation of these vulnerabilities. Finally, other systems were built on top of mass-market operating systems and integrated with widely-available applications, which have a poor security and reliability record [3, 4, 5]. These mass-market components are routinely used in contexts that exceeds the level of security and reliability they were designed to offer [2].

We believe that the ideal solution would be a complete redesign of these systems with the use of adapted programming languages and frameworks, aimed at providing better software fault-tolerance, to get rid of the problems at their source. Indeed, better designs with more explicit security and reliability requirements and the use of safe programming languages and technologies not prone to vulnerabilities, such as Java or Ada, would be a very good start. However, systems with critical security and reliability requirements are rather expensive to specify, develop, procure, operate and maintain, because of the time and expertise involved. Fundamental research on subjects related to the survivability of systems is also needed to solve remaining problems and questions [6]. This ideal solution is therefore necessarily a long-term solution.

In the short term, something else must be done. The good news is that mature high-availability products for mass-market operating systems are now available. These products claim that they can "wrap" existing applications and run them in a virtualized environment, allowing their execution to continue even if a fatal error happens. However, we could not find a complete, independent report on their evaluation. Therefore, we need to investigate these products, and this is our proposition, in order to know how good these solutions are and which threats they can mitigate.

The following section presents the family of high-availability solutions that are of interest. Then, section 2 discusses information system threats and how some of them could be mitigated with high-availability solutions. Finally, section 3 details our evaluation's goals and work plan.

1 High-Availability Solutions

High-availability solutions wrap the execution of an existing system and helps it attain a higher level of availability by mitigating the effects of hardware and software errors. This is mainly done by the use of redundancy, where a failed component is replaced with a working one so that the system can continue to offer its service.

Interesting solutions are those that wrap the entire system, not only a single component, as a RAID disk array does. These solutions, called

high-availability clusters [8, 9], generally use clones of the entire system as redundant nodes.

High-availability clusters can work in many modes:

Monitor and Replace Without State Transfer The system is monitored for errors and when a fatal one happens, the failed node is shut down and replaced with a hot standby. As the state of the failed node is not transferred to the new one, the execution cannot continue and the system has to be reinitialized, including remote clients. Therefore, this type of high-availability cluster simply automates the reinitialization process.

Monitor and Replace With State Transfer Again, the system is monitored for errors and when a fatal one happens, the failed node is shut down and replaced with a hot standby. However, the state of the failed node is preserved and transferred to the new node in order to continue the execution as if the failure never happened. A small downtime can occur while the state is transferred from the failed node to the hot standby.

Mask Failures With Virtualization This time, the system runs inside a virtualized environment, made of many nodes that run the system in parallel. When a fatal error occurs in one node, it is simply masked by using the result of another node (or many others), without any downtime or reinitialization. This approach has less drawbacks than other ones, but it is much more complex to implement correctly.

In a nutshell, high-availability clusters can either restart a failed system or mask the failure so that the execution can continue as if nothing happened, sometimes minus a small downtime. Important questions emanate from these observations:

- How useful is restarting a system when a fatal error happens? Is the error going to occur again after a restart?
- Which errors can be masked and which cannot?
- Do applications need to be aware of the cluster or can everything be transparent?
- What kind of applications make the state transfer impractical?

These are some of the questions we would like to answer after our evaluation of high-availability clusters.

1.1 Products of Interest

A short research on the Internet revealed many interesting high-availability products that can be evaluated. Here is a non-exhaustive list with a description from their respective vendor:

Marathon everRun^{FT} *This product synchronizes two unmodified servers to create a virtual application environment that runs on both of them simultaneously. If one server fails, the other server enables the application to continue operating without interruption* [10].

Microsoft Clustering Services *This service provides high availability and scalability for mission-critical applications such as databases, messaging systems, and file and print services. Multiple servers (nodes) in a cluster remain in constant communication. If one of the nodes in a cluster becomes unavailable as a result of failure or maintenance, another node immediately begins providing service, a process known as failover. Users who are accessing the service continue to access the service, and are unaware that it is now being provided from a different server* [11].

Veritas Cluster Server *Veritas Cluster Server can detect faults in an application and all its dependent components, including the associated database, operating system, network, and storage resources. When a failure is detected, Cluster Server gracefully shuts down the application, restarts it on an available server, connects it to the appropriate storage device, and resumes normal operations* [12].

VMware Virtual Infrastructure *VMware High Availability provides easy to use, cost effective high availability for applications running in virtual machines. In the event of server failure, affected virtual machines are automatically restarted on other production servers with spare capacity* [13].

Linux High-Availability Project *It provides monitoring of cluster nodes, applications, and provides a sophisticated dependency model with a rule-based resource placement scheme. When faults occur, or a rule-change occurs, the user-supplied rules are then followed to provide the desired resource placement in the cluster* [14].

On paper, these products seem very promising. Obviously, our evaluation would include thorough testing of these products in order to validate these claims.

1.2 Added Value Provided by High-Availability Solutions

As a side note, it is important to realize that availability offered by these solutions can be leveraged in other contexts and can provide non-negligible added value, such as:

Non-Disruptive Maintenance Upgrades and common hardware and software maintenance can be performed while the system is running, without its users noticing any downtime. Furthermore, if the system's state is extensively logged (which should be the case most of the time), modifications that prove to be erroneous can be rollbacked.

Server Consolidation In order to achieve fault-tolerance, virtualization is often used in high-availability solutions and it can also be used to run many virtual servers on the same hardware. It is thus possible to consolidate many less-used servers on one powerful computer, simplifying management.

Advanced Monitoring & Logging State transfers and virtualization in high-availability clusters need extensive monitoring and logging which is ideal for good forensics. Therefore, if, for whatever reason, a system that uses a high-availability cluster is attacked and fails, the failure has the potential to be deeply analyzed from the logs.

2 Problems & Threats

This section looks at threats that current systems face everyday [7] and how a high-availability solution could help, by either restarting the failed component or by masking the effects of the fault. Restarting a failed component is a sound solution only if the cause of failure is *transient*, or temporary. If the cause of failure is still present after a restart, the component will fail again and will keep being restarted over and over again. Masking the effects of a fault may also not always be possible, if, for instance, all the duplicate nodes fail simultaneously and give the same erroneous output.

2.1 Environmental Faults

Environmental faults are failures that originate from outside the system itself, such as a power failure, a network connectivity loss (cut cable), natural disasters, etc. Since the problem lies outside the system, restarting it will not change much, unless the problem goes away while the system is restarting.

If the system cannot deal with a temporary outage of a specific kind (it cannot progress from a failed state to a correct state), restarting it may be necessary after the outage. Masking the fault would require a distributed system with geographically-distant nodes that cannot be subject to the same environmental faults.

We see a limited use of high-availability solutions for these kinds of faults.

2.2 Hardware Faults

Hardware faults occur when a hardware component of the system stops working correctly and reports an error. Examples include a crashed disk, failed parity checking while reading memory, burned power supply, etc.

High-availability solutions were first designed to handle hardware faults and are generally considered to work well in that context.

2.3 Software Faults

Software faults occur when the execution of a program diverges from "what it should be", i.e., its specification. For instance, a program could write data outside the bounds of one of its buffers. This leads to memory corruption and can be the cause of a crash. Errors in computation can also emerge if the program reads outside the bounds of one of its buffers. Another example of a program crash is if the program attempts a division by zero.

Software fault-tolerance is a vast domain being intensively researched right now. All problems and questions are not yet solved and answered. We want to start by investigating what currently-available high-availability solutions have to offer in handling software fault-tolerance of existing systems. The answer to "What can be done when a system crashes because of a software fault?" is not easy to find. However, we believe that high-availability solutions could be useful when a transient software error occurs.

2.4 Malicious Acts

Systems can also crash because of malicious acts like denial-of-service and code injection attacks. High-availability solutions cannot mitigate code injection attacks because if a system is vulnerable to code injection, making it more available will not solve the problem. However, systems wrapped by high-availability solutions can be more resilient to denial-of-service attacks because if the attack slows down one of the nodes of the system, load balancing could be used to redistribute resources and achieve better performance. If the attack crashes the system, it could be automatically restarted.

3 Goals & Work Plan

To sum up, our main goals for our evaluation are as follows:

1. Evaluate monitors' error-detection performance.
   (a) Which faults can be detected?
   (b) Which monitoring approach do they use?
2. Assess how useful is a restart and for which kind of faults.
3. Assess how useful and feasible is to mask faults.
4. Determine if high-availability solutions are transparent to the applications.
5. Determine the limits of each product in general.

And our draft work plan:

Summer and Fall 2006 : Feasibility study and state-of-the-art report.

Spring 2007 : Preparation (development of tests, setup, etc.).

Summer 2007 : Evaluation.

Fall 2007 : Final report.

References

[1] Martin Croxford, Roderick Chapman, *Correctness by Construction: A Manifesto for High-Integrity Software*, CrossTalk, December 2005.

[2] Gregory Slabodkin, *Software glitches leave Navy Smart Ship dead in the water*, Government Computer News, July 13, 1998.

[3] SANS Institute, *The SANS Top 20 Internet Security Vulnerabilities*, http://www.sans.org/top20/

[4] SecurityFOCUS, *BugTraq Mailing List Archive*, http://www.securityfocus.com/archive/1

[5] McAfee, *McAfee Threat Center – Security Vulnerabilities*, http://www.mcafee.com/us/threat_center/vulnerabilities.html

[6] Peter G. Neumann, *Practical Architectures for Survivable Systems and Networks*, Computer Science Laboratory, SRI International, June 2000.

[7] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, January-March 2004 (23 pages).

[8] Gregory Pfister, *In Search of Clusters, 2nd Edition*, Prentice Hall, 1998, 608 pages.

[9] Evan Marcus, Hal Stern, *Blueprints for High Availability, 2nd Edition*, John Wiley & Sons, 2003, 624 pages.

[10] http://www.marathontechnologies.com/

[11] http://www.microsoft.com/windowsserver2003/technologies/clustering/default.mspx

[12] http://www.symantec.com/Products/enterprise?c=prodinfo&refId=20

[13] http://www.vmware.com/vinfrastructure/

[14] http://www.linux-ha.org/