Presentation Notes
Paper
Bio
Return to Main Menu

PRESENTATION

T14

Thursday, November 4, 1999
1:30 PM

System Testing Strategies for Highly Available Clustered Systems

Subbarao Jagannatha
Sun Microsystems

International Conference On
Software Testing, Analysis & Review
November 1-5, 1999
San Jose, CA

System Testing Strategies for Highly Available Clustered Systems

Subbarao Jagannatha

Software Engineer

Sun Microsystems, Inc

jagan@eng.sun.com

Overview

- What is a highly available clustered system?
- Test planning for large and complex systems
- Test strategy and methodology
- Fault injection tests
- System configuration testing
- Defect tracking process
- Effective testing practices

What is a Highly Available (HA) Clustered System?

- Consists of two or more server machines
- Recovers from single point hardware or software failure
- Configured for maximum uptime
- Involves complex set of hardware and software
- Widely used in mission-critical enterprise applications

Cluster System Components

- Hardware Components
  - Two or more server machines
  - Cluster interconnect
  - Storage device
  - Public network
- Software Components
  - Cluster software
  - Operating system software
  - Volume manager software
  - Database server software

Test Planning for Large and Complex Systems

- Clearly identify the product features and test areas
- Provide required training to QA Team
- Prepare a master test plan
- Prepare individual feature test specifications
- Explore and develop test tools as needed

Test Strategy and Methodology

- Installation tests
- Basic sanity tests
- Cluster fault injection tests
- Storage fault injection tests
- Load tests
- Configuration tests
- Stability tests

Cluster Fault Injection Tests

- Cluster software daemon failure.
- Operating system failure.
- Hardware failure on the master node.
- Database failure on the master node.
- Cluster interconnect failure
- Public network failure on the master node.

Storage Fault Injection Tests

- Storage array cable failure.
- Storage array controller board failure.
- Storage array hardware failure.
- Storage array individual disk failure.
- Boot disk failure.

Test Cycle

- Sanity Checks

  - Automated tests

- Functional testing

  - Manual tests

  - Automated tests

- Load testing

  - Standard benchmark database workloads

  - Automated tools for load testing applications

System Configuration Testing

- Multiple hardware platforms, OS versions, storage devices, public networks and databases
- Challenge:

Defining the test configuration matrix

Config ID	**HW Platform**	**OS Version**	**Storage Device**	**Public Network**	**Data Base**
1					
2					

Defect Tracking Process

- Clearly define and consistently follow the defect tracking process.
- Assign correct priority and severity for each defect
- Weekly "bug courts" to keep track of the bug status
- At milestones, review the exit criteria of system test
- Make bug reports available online through web pages

Post Mortem Discussion

- Things that went well
- Things that could be improved
- Using the lessons learned in future testing activities

Effective Testing Practices

- Define a test process that involves risk analysis and management.
- Use full testing for critical software
- Prepare a Master test plan for project
- Prepare individual test specification documents for each major area
- Review each of the test plan and test specification documents

Effective Testing Practices (Continued)

- Develop test standards and templates that can be reused.
- Explore the available test tools and application work loads.
- Effective and frequent communication among QA and development teams.

Conclusion

The effective testing of large and complex systems needs:

- Disciplined test process to be followed by all team members.
- Good understanding of the big-picture of the product by QA team
- Effective communication & relationship between development and QA teams
- Follow the suggestions given in "Effective testing practices"

System Testing Strategies for Highly Available Clustered Systems

Subbarao Jagannatha
Software Engineer
Sun Microsystems
jagan@eng.sun.com

Abstract

This paper focuses on the strategies and challenges of testing real-world, large and complex, Highly Available clustered system in the following areas:

- Test planning, strategies and methodologies
- Defect tracking processes
- Cluster and storage fault injection tests
- System configuration tests
- Effective testing practices

The information provided in this paper is aimed at helping test engineers understand many of the quality assurance issues involved in testing large and complex systems.

Keywords:

Testing strategies, Testing practices, Cluster, High availability, fault-injection.

1. Introduction

Highly Available (HA) clustered systems are at the core of many mission-critical enterprise applications today. Testing HA systems is a challenging task because of the complex set of hardware and software involved and the requirement of highly reliable software.

2. What is a Highly Available Clustered System?

A Highly Available (HA) clustered system consists of two or more server machines which are also called *nodes*. The nodes are interconnected through cluster interconnect network. The nodes communicate the heart-beat information through cluster interconnect. The cluster also has storage system for data storage and is connected to a public network. Please refer to figure in the following page for a generic schematic of a two node highly available cluster.

A cluster can withstand single point of hardware or software failure and recovers from the fault with minimum down time. The cluster is configured for high availability and maximum up time.

A HA cluster includes complex set of hardware and software components. The software components are the cluster software, operating system, volume manager and database server software.

Test engineers need to be aware of the mission-critical nature of applications which run on clusters. Problems caused by system failure can cause customer dissatisfaction, lost transactions, lost productivity, lost revenue, lost customers, penalties or fines.

3. Test Planning for Large and Complex Systems

The disciplines of planning and analysis, formal documents, configuration management, measurement, standards, tools and procedures are crucial for the success of testing large and complex systems.[4]

Following are some planning and preparation activities recommended for testing:

- Clearly identify product features and test areas
- Provide required training to test engineers about the product and Quality Assurance process
- Prepare a master test plan and individual feature test specifications for each test area
- Explore, plan and develop test tools as needed

4. Test Strategy and Methodology

Our strategy for testing the HA cluster system involved the following types of tests:

- **Installation tests** - Check that the product can be successfully installed by following the instructions given in the Product Installation Manual.

- **Basic sanity tests** - Check that the product satisfies the entry criteria for starting the system testing.

- **Cluster fault injection tests** - Check the behavior of the cluster in case of single point hardware or software failure.

- **Storage fault injection tests** - Check the behavior of the cluster in case of single point hardware failure in the storage device.

- **Load tests** - Check the behavior of the system under load.

- **Configuration tests** - Validate the product on different hardware and software configurations.

- **Stability tests** - Check the stability of the system by running the system for extended period of time.

5. Cluster Fault Injection Tests

Recovery of the cluster system in case of single point hardware or software failure is tested using the fault injection tests. Any failure on the master node will cause the system to failover to backup node.

Following cluster fault injection tests are performed:

- Cluster software daemons failure can be simulated by killing the daemon process.

- Operating system failure simulated by using standard techniques.
- Hardware failure on the master node simulated by powering off the node.
- Database failure on the master node simulated by killing the database server process.
- Cluster interconnect failure simulated by physically disconnecting one of the cluster interconnect cables.
- Public network failure on the master node simulated by physically disconnecting one of the public network cables.

6. Storage Fault Injection Tests

The data in the storage array and boot disk data are mirrored for high availability. Following storage fault-injection tests are performed:

- Storage array cable failure simulated by disconnecting the cable.
- Storage array controller board failure simulated by physically removing the controller.
- Storage array hardware failure simulated by powering-off the array hardware.
- Storage array individual disk failure simulated by physically unplugging one of the disks.
- Boot disk failure simulated by physically unplugging the boot disk.

7. Test Cycle

A typical test cycle involves sanity tests, functional tests and load tests.

Sanity tests are performed using a set of automated scripts. These are short tests performed to check if the product satisfies the entry criteria of system testing phase.

After the sanity tests are successful, the functional testing begins. The functional tests are performed both manually and with automated tools. All the functional test cases documented in the feature test specification document will be executed during this phase.

After completing the functional tests successfully, load testing are performed using industry standard benchmark workloads with databases. Next, automated tools for load testing applications like web server, mail server, news server and NFS are also performed.

8. System Configuration Testing

As mentioned above, a cluster product includes a complex set of hardware and software components. The product must be qualified on multiple hardware and storage platforms, multiple OS versions and Volume Managers, different types of public networks and cluster interconnects, and multiple applications and databases.

The challenge is in defining the test configuration matrix. The possible system configuration combinations count can be several hundred, which is practically impossible to test. By researching customer requirements and excluding the least likely and impractical combinations, a small number of combinations that can be tested within the scheduled time are distilled. A sample format of configuration matrix is given below.

Config ID	HW Platform	OS Version	Storage Device	Public Network	Data Base
1					
2					

Configuration Matrix

9. Defect Tracking Process

In large and complex projects, using a defect or bug tracking tool is a must. The defect tracking process should be clearly defined and consistently followed by all team members. A group discussion of all team members about the defect tracking process is very useful.

Each of the defect needs to be assigned the correct *priority* and *severity*. Priority defines how soon the bug needs to be fixed and Severity defines the level of impact to the customer. Also the required information in the defect report should be well defined and standardized. This will help the development team in getting all the required information for a given defect. This is specially important for large and complex systems where in it will be difficult to obtain any additional information later.

Weekly “bug courts” to keep track of the bug status attended by both QA and development teams greatly helps in effective communication.

At each major milestone, check if we satisfy the system test entry and exit criteria based on the number of open bugs. Also periodically send the list of open bugs to the team and make all the bug reports available online through web pages.

10. Effective Testing Practices

Below are some practical suggestions for effective testing:

1. Define a test process that involves risk analysis and management. Continuously streamline the testing process to make it more efficient and less time-consuming.[5]

1. Use full testing for critical software or any software that will have heavy and diverse usage by a large user population. Use partial testing for small, non-critical software products with small, captive user population.[4]
2. Prepare a Master test plan for the project with high level details. In addition, prepare individual test specification documents for each major area of testing. Each test case should have a purpose, prerequisites, test execution procedure and expected result clearly and concisely defined.

1. Review each of the test plan and test specification documents and involve development in the review process. Make sure that development team and other team members clearly understands and agrees to the entry and exit criteria in the system test plan.

1. Develop test standards and templates that can be reused. Consider using an automated test tool for repetitive tests once the test process has been defined and you know what you need to test.[5] Use automated scripts for sanity testing or any tests which need to be run very frequently.

1. Explore the available test tools and application work loads before you start developing tools from scratch. It can be very costly and time-consuming to develop your own tools. As far as possible, use standard test tools. Do not reinvent the wheel!

1. Effective and frequent communication among QA and Development teams is essential in a large project environment. Also effectively communicate test status frequently among team members.

1. The test engineers need to be aware of problems experienced in the field. They should be involved in mailing list aliases which are discussion forums on the product. Test plans, test cases and test tools should be constantly updated, based on the new problems seen in the field.

1. Test engineers should participate in the reviews of requirements specification, functional design specification, test plans and official product documents.

1. Conducting post-mortem discussion, wherein each test engineer discusses three things that were done well in the project and at least one area which could be improved. A summary of the feedback from each engineer should be shared with all the team members and upper management. The information about the areas for improvement should be used for planning future testing activities.

11. Conclusion

The effective and efficient testing of large and complex systems needs a disciplined test process to be followed by all team members.The test engineers need a good understanding of the big- picture of the product and they need to be adequately trained.

There should be clear division of responsibilities among test team members. Everyone should be made aware of the benefits of delivering the high-quality and the pitfalls and consequences of delivering a poor quality product.

The suggestions given in section 10 about “Effective testing practices” has been successfully used in our project. This can be effectively used by other quality assurance groups for testing large and complex systems.

References:

1. Beizer, Boris. 1991. *Software Testing Techniques.* Second Edition. Van-Nostrand-Reinhold Company. New York, NY.

2. Pfister, G. 1998. *In Search of Clusters* 2nd Edition. Prentice Hall, Upper Saddle River, NJ.

3. Perry, William. 1995. *Effective Methods for Software Testing.* John Wiley & Sons, New York, NY.

4. Kit, Edward. 1995. *Software Testing in the Real World Improving the Process.* Addison-wesley.

5. Perry, William. Randall, Rice. 1997. *Surviving the top ten challenges of software testing.* Dorset House, New York, NY.

SUBBARAO JAGANNATHA

Mr. Subbarao Jagannatha is a software engineer at Sun Microsystem's Availability Laboratory in Menlo Park, California. He has been involved in testing highly available clustered systems for three years. His current focus is on researching and testing computer systems reliability, availability, and serviceability (RAS).

Prior to joining Sun Microsystems, Mr. Jagannatha worked for five years at Bell Laboratories as a member of the technical staff in the software testing and development areas. He has led the quality assurance group responsible for testing a 5ESS switch application.

He received a Master of Science degree in computer engineering from Rutgers University, New Jersey, and a bachelor's of Engineering degree from Bangalore University, India.