

Enable high availability for composite applications

A prototype for delivering variable-requirement high availability to composite applications

Skill Level: Intermediate

Mahesh Viswanathan (mareshv@us.ibm.com)

Senior Technical Staff Member
IBM

Suraj Subramanian (suraj@us.ibm.com)

Senior Integration Architect
IBM

13 Jan 2009

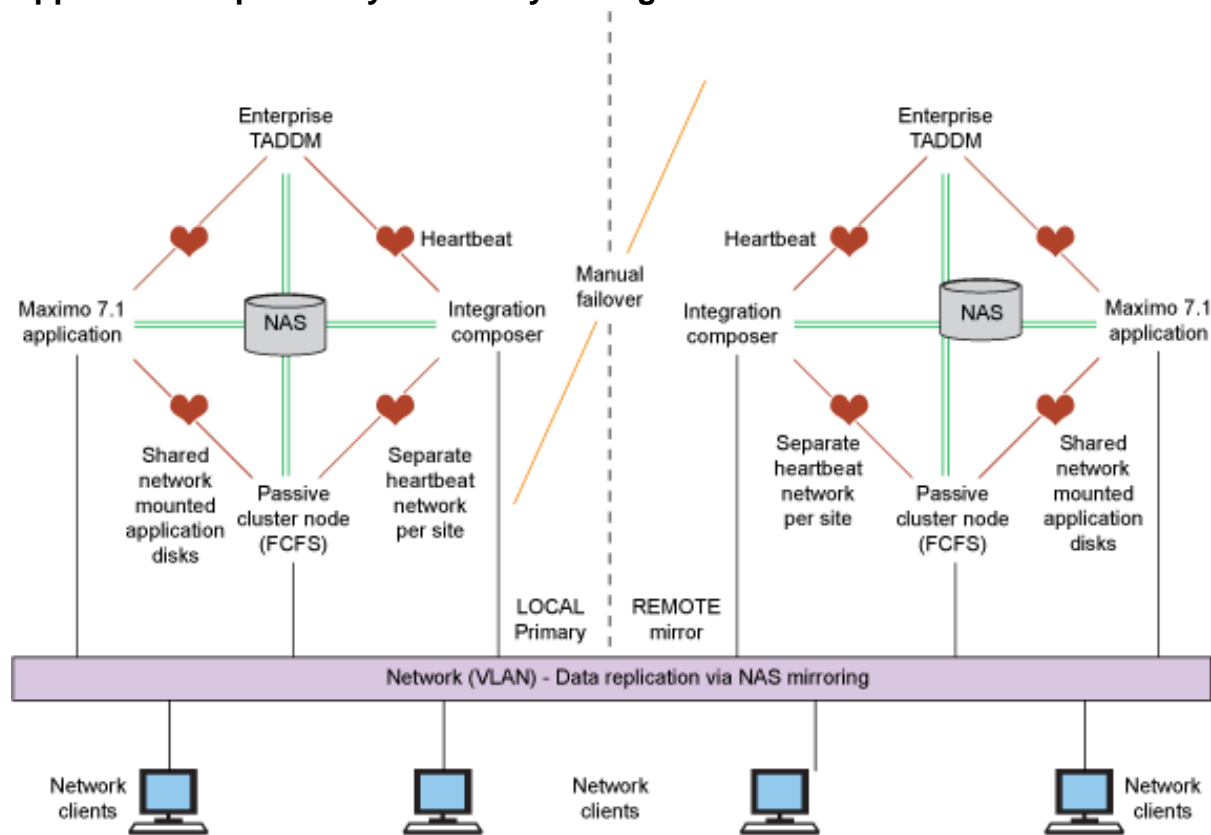
This article describes an implementation of high availability for a composite application using Linux-HA. Delivering high availability to composite applications can be challenging. Because composite applications consist of several distinct applications, each with different availability requirements, configuration is complex. In this article, the authors describe how they designed and implemented a high availability prototype for Tivoli® Maximo®, a composite app. Their configuration script shows how you can provide high availability to a heterogeneous cluster of related applications using a systematic and prioritized failover schedule.

In 2008 we developed a high availability solution for our CMDB implementations (configuration management database), Tivoli Maximo and Tivoli Application Dependency Discovery Manager, for a large online company. This company is embarking on a full-fledged CCMDB (change and configuration management database) implementation of Maximo (now named Tivoli Service Request Manager and Tivoli Asset Manager for IT) that includes a single, enterprise-wide Maximo (CCMDB version). The goal was to use multiple domain CMDB applications to gather information from different company sites and aggregate them into an enterprise CMDB instance. This enterprise CMDB data was filtered and accumulated into a CCMDB instance, Maximo.

Maximo stores both the ideal CMDB state of WISB ("what it should be") of the enterprise as represented by the preset policies *and* the reality of WIRI ("what it really is") as represented by the CI information gathered from thousands of servers and applications deployed across the corporation.

Generally speaking, different functional nodes in a HA hierarchy require different HA designs. For instance, gateway servers, generally Windows® machines, require MSCS. Domain CMDB applications require HA, but using a cold standby is adequate. Maximo, however, was required to be available 24/7, and both Maximo and enterprise CMDB connect to its own individual databases—normally this would be part of the cluster. But in this prototype we focused just on the HA of the applications, so our design is for application availability only. Figure 1 shows our design.

Figure 1. HA disaster recovery (DR) for Maximo and Enterprise Tivoli Application Dependency Discovery Manager: Two 4-node HA clusters



In this article, we describe a method to manage a highly-available, heterogeneous, multi-application cluster of nodes. Within the highly available cluster, each application has a different availability profile; that makes our solution inherently more complex than the simple high-availability solutions of single application classes, such as databases.

Our solution was to come up an algorithm (protocol) to accommodate a first *and*

second failure of application (or nodes) within the highly available cluster. This protocol provides the precise failover sequence for each of the application nodes in the cluster. It takes into consideration situations where degraded performance is not acceptable, and cases in which multiple applications cannot run on the same machine due to runtime conflicts.

The design and implementation features are as follows:

1. Even though there are three applications in the cluster (Enterprise CMDB, ITIC, and Maximo), Maximo has the highest priority of the three. Therefore, no matter which application-node fails, Maximo must be available for use by clients (client machines will invoke the Maximo API).
2. We deployed auto-failback using unbalanced weighting of nodes so that when a downed server is restored, the application fails back to it.
3. Our design specification calls for managing for two failures in the cluster; however, we were able to design and implement full availability of CCMDB (Maximo) even after three failures.
4. We initially did not design with a quorum server in mind, but we had to use one.
5. Configuring Linux®-HA to behave with this stringent set of failover rules over one, two, and three failures was quite a challenge.
6. We replicated this design at the disaster recovery (DR) site (the remote mirror site in Figure 1). The failover within a site is automatic and manual across sites: automatic HA, manual DR.
7. To optimize the use of machines and to maintain HA, we designed for a four-node cluster at each site: one node per application and a spare node that will serve as the failover node.

HA architecture is used traditionally for single piece of software like a database or Web server. In our example, we show how to achieve high availability for an application like CCMDB, which consists of three individual software components:

- Tivoli Application Dependency Discovery Manager (TADDM): Provides visibility for IT service management by discovering application dependencies and configurations.
- IBM Tivoli Integration Composer (ITIC): Enables rapid integration of Tivoli Asset Management for IT with asset inventory and system management tools.

- **Maximo:** Provides comprehensive asset lifecycle and maintenance management for all asset types on a single unified platform.

Installing HA

The Linux-HA installation is a simple and straightforward process (see [Resources](#) to get the software). Ensure that the systems have the right level of patches to satisfy the heartbeat software prerequisites. The version of Linux-HA used for our demonstration is 2.1.4.

When the installation is complete, reboot the machine. This is an essential step. Follow the steps in Listing 1 for all four machines that will be a part of this cluster.

Listing 1. Installing HA

```
[root@hacluster2 tmp]# rpm -ivh perl-TimeDate-1.16-3_2.0.el5.noarch.rpm
warning: perl-TimeDate-1.16-3_2.0.el5.noarch.rpm: Header V3 DSA signature:
NOKEY, key ID 66534c2b
Preparing ... ##### [100%]
 1:perl-TimeDate ##### [100%]
[root@hacluster2 tmp]# rpm -ivh heartbeat-pils-2.1.4-2.1.i386.rpm
warning: heartbeat-pils-2.1.4-2.1.i386.rpm: Header V3 DSA signature:
NOKEY, key ID 1d326aeb
Preparing ... ##### [100%]
 1:heartbeat-pils ##### [100%]
[root@hacluster2 tmp]# rpm -ivh heartbeat-stonith-2.1.4-2.1.i386.rpm
warning: heartbeat-stonith-2.1.4-2.1.i386.rpm: Header V3 DSA signature:
NOKEY, key ID 1d326aeb
Preparing ... ##### [100%]
 1:heartbeat-stonith ##### [100%]
[root@hacluster2 tmp]# rpm -ivh heartbeat-2.1.4-2.1.i386.rpm
warning: heartbeat-2.1.4-2.1.i386.rpm: Header V3 DSA signature: NOKEY, key ID
1d326aeb
Preparing ... ##### [100%]
 1:heartbeat ##### [100%]
[root@hacluster2 tmp]#
```

Configuring HA

The next step is to create the ha.cf file. Create the following file: /etc/ha.d/ha.cf. The ha.cf file holds the information about which nodes are parts of this setup.

Create ha.cf on the machine assigned as the DC (designated coordinator). Copy this file and the authkeys file to the other machines once you have created them using FTP or plain SCP. Since four nodes (three applications and the spare) are in this cluster, our ha.cf file will look something like this:

Listing 2. HA configuration file, ha.cf

```
node hacluster1.svl.ibm.com hacluster2.svl.ibm.com hacluster3.svl.ibm.com
    hacluster4.svl.ibm.com
bcast eth0
crm on
```

In Listing 2:

- `node` is a directive that lists the nodes that are part of this cluster.
- `bcast` is a directive that means the nodes will communicate and ping each other on this interface.
- `crm` is a directive that specifies whether the heartbeat should run a cluster manager that supports two or more nodes.

In Listing 2, we keyed in the hostnames of the machines that belong to this cluster. Next we needed to add an authentication key to `/etc/ha.d/authkeys`. Listing 3 shows the example we used:

Listing 3. Sample authentication file, authkeys

```
#
#      Authentication file. Must be mode 600
#
#
#      Must have exactly one auth directive at the front.
#      auth      sne authentication using this method-id
#
#      Then, list the method and key that go with that method-id
#
#      Available methods: crc sha1, md5. Crc doesn't need/want a key.
#
#      You normally only have one authentication method-id listed in this file
#
#      Put more than one to make a smooth transition when changing wuth
#      methods and/or keys.
#
#
#      sha1 is believed to be the "best", md5 next best.
#
#      crc adds no security, except from packet corruption.
#      Use only on physically secure networks.
#
auth 1
1 sha1 haclusteringisfun
```

NOTE: The permissions on this file must be 0600. Once these files are created, start heartbeat by issuing the command `/etc/init.d/heartbeat start`.

Run the command shown in Listing 4 on all machines in this cluster:

Listing 4. Starting High-Availability services

```
[root@hacluster1 heartbeat]# /etc/init.d/heartbeat start
```

```
Starting High-Availability services:
```

```
[ OK ]
```

```
[root@hacluster1 heartbeat]#
```

If you see this message, it means your installation of Linux-HA is successful. Now it's time to test the high availability of the composite application.

Adding a quorum server

In a two-node cluster, when one fails or the network connection snaps, each node believes that it is the master and starts interacting with the outside world. This is a race condition that is undesirable. We need an outside arbiter to ask one of the machines to stand down.

If one of the machines has crashed, the arbiter will make that machine stand down. This arbiter is called a *quorum server*; it can be any machine that both nodes in the cluster can reach. This quorum server is thusly named because it runs a quorum daemon. You modify the `ha.cf` to add this line to each of the `ha.cf` files:

Listing 5. Identifying a quorum server in `ha.cf`

```
cluster ourcldb
quorum_server hacluster4.svl.ibm.com
```

The quorum server machine is not required to run heartbeat, but we recommend heartbeat be installed in order to get access to all the binaries and directory paths that are created (such as `/etc/ha.d`) when heartbeat is installed. On the quorum server machine, edit the file `/etc/ha.d/quorumd.conf` and add the following lines:

Listing 6. Configuring the quorum server

```
cluster ourcldb
version 2_1_4
interval 1000
timeout 5000
takeover 3000
giveup 2000
```

Then, start the quorum daemon using `quorumd`. Make sure that it starts each time the machine is rebooted. To start `quorumd` automatically, add this to the `inetd`.

Linux-HA uses a configuration file called `cib.xml`, which is created automatically when you start heartbeat on all the nodes in the cluster. This `cib.xml` file stores the application configuration (specifying which application takes higher priority including rules for high availability). You can modify the `cib.xml` by using the GUI tool (`/usr/bin/hb_gui`), which is recommended, or manually.

Cib.xml contains the following:

- Configuration information:
 - Cluster node information
 - Resource information
 - Resource constraints
- Status information:
 - Which nodes are up/down
 - Attributes of nodes
 - Which resources are running where

Because the cib.xml is under the control of the heartbeat process, avoid modifying this file when the cluster is running.

NOTE: Permissions on cib.xml must be 0600 and must be owned by `haclient:hacluster`.

Linux-HA comes with a set of resource agents based on the Open Cluster Framework (OCF), the standard for achieving high availability. In our scenario, since all the applications were customized, we had to build OCF resource agents for the individual software components that were a part of the composite application.

Configuring heartbeat

Listing 7 shows the resource configuration in the heartbeat version we're using. The config file is located at `/var/lib/heartbeat/crm/cib.xml`. Basically, this file specifies the resource(s) for the cluster and where this resource should be executed.

Here is the cib.xml file we developed for our scenario; see [Resources](#) for a link. We've annotated it to make the process easier to follow.

Listing 7. The resource configuration in heartbeat version 2

```
<cib generated="true" admin_epoch="0" have_quorum="true" ignore_dtd="false"
num_peers="2"
  ccm_transition="2" cib_feature_revision="2.0" crm_feature_set="2.0" epoch="3"
dc_uuid="
ad893965-d27d-4908-a2ea-868f1661f644" num_updates="3" cib-last-written="Fri Nov
14
10:14:40 2008">
  <configuration>
    <crm_config>
      <cluster_property_set id="cib-bootstrap-options">
        <attributes/>
      </cluster_property_set>
    </crm_config>
```

```

<nodes> /* Names of the nodes in the cluster */
  <node id="ad893965-d27d-4908-a2ea-868f1661f644"
    uname="hacluster1.svl.ibm.com" type="normal"/> /* Maximo node */
  <node id="5994eb92-0a13-4fc7-ab41-76098672fdbb"
    uname="hacluster3.svl.ibm.com" type="normal"/> /* ITIC node */
  <node id="c14b9082-5b1a-481c-930a-561e926df7c3"
    uname="hacluster4.svl.ibm.com" type="normal"/> /* eCMDB node */
  <node id="827b7884-06db-4d8d-994d-7e743b9bb969"
    uname="hacluster2.svl.ibm.com" type="normal"/> /* Spare node */
</nodes>
<resources>
  <group id="maximo_group">
    /* Assign base parameters for each application - Maximo */
    <primitive class="lsb" id="maximo_id" type="maximo">
      <operations>
        <op id="1" name="monitor" interval="10s"/>
        <op id="2" name="start" start_delay="10s"/>
      </operations>
      <meta_attributes id="063383a7-2c60-4cf0-b3b0-a3670328c3b8">
        <attributes> /* Priority are set as weighting factors. This
determines which
event of
that the
spare.
downed
precedence.
application will be placed on the spare node in the
a second failure. Here, Maximo > eCMDB > ITIC. Note
first failure pushes the downed application onto the
A subsequent failure determines whether the initially
application or the newly downed application takes
*/
        <nvpair name="priority" value="3"
          id="f0c58dd3-43bb-4ale-86cc-8993e58ba399"/>
      </attributes>
    </meta_attributes>
  </primitive>
</group>
  <group id="iticd_group"> /* Assign base parameters for each application -
ITIC */
    <primitive class="lsb" id="itic_id" type="iticd">
      <operations>
        <op id="3" name="monitor" interval="10s"/>
        <op id="4" name="start" start_delay="10s"/>
      </operations>
      <meta_attributes id="cb35cbb3-3241-4bf6-9ab7-f06d6f5baf89">
        <attributes>
          <nvpair name="priority" value="2"
            id="7f7a35eb-aff8-4f7d-8e4d-ec6a0414d6da"/>
        </attributes>
      </meta_attributes>
    </primitive>
  </group>
  <group id="taddm_group"> /* Assign base parameters for each application -
eCMDB */
    <primitive class="lsb" id="taddm_id" type="taddm">
      <operations>
        <op id="5" name="monitor" interval="10s"/>
        <op id="6" name="start" start_delay="10s"/>
      </operations>
      <meta_attributes id="e7baab9a-1460-423f-aabd-f68137d00d42">
        <attributes>
          <nvpair name="priority" value="1"
            id="0448d632-f8e7-4347-90bd-de8222b77bda"/>
        </attributes>
      </meta_attributes>
    </primitive>
  </group>
</resources>

```



```

    <constraints>
    <rsc_colocation id="not_same_1" to="maximo_group" from="itcd_group"
run on      score="-INFINITY" symmetrical="false"/> /* Maximo application should not
                                                    the node assigned to ITIC */
    <rsc_colocation id="not_same_3" to="maximo_group" from="taddm_group"
run on      score="-INFINITY" symmetrical="false"/> /* Maximo application should not
                                                    the node assigned eCMDB */
    <rsc_colocation id="not_same_2" to="taddm_group" from="itcd_group"
run on      score="-INFINITY" symmetrical="false"/> /* eCMDB application should not
                                                    the node assigned to ITIC */
    <rsc_location id="location_maximo" rsc="maximo_group">
    <rule id="prefered_location_maximo_1" score="20">
    <expression attribute="#uname" operation="eq"
value="hacluster2.svl.ibm.com"
id="a4b1be4e-4c25-46a6-9237-60a3b7b44389"/> /* Spare node for Maximo
should
                                                    preferred node fail */
    </rule>
    <rule id="prefered_location_maximo_2" score="100">
    <expression attribute="#uname" operation="eq"
value="hacluster1.svl.ibm.com"
id="3ece30c7-0530-4b54-a21b-ace9b127d3e3"/> /* Preferred node for
Maximo
                                                    to run */
    </rule>
    <rule id="prefered_location_maximo_3" score="-INFINITY">
    <expression attribute="#uname" operation="eq"
value="hacluster4.svl.ibm.com"
id="7cb3b3e2-191f-492d-84f9-257b96c02c3c"/> /* Maximo cannot co-exist
with
                                                    eCMDB on this node */
    </rule>
    <rule id="prefered_location_maximo_4" score="-INFINITY">
    <expression attribute="#uname" operation="eq"
value="hacluster3.svl.ibm.com"
id="a78cce70-c717-4c33-910e-11cc39ded186"/> /* Maximo cannot co-exist
with
                                                    ITIC on this node */
    </rule>
    </rsc_location>
    <rsc_location id="location_itcd" rsc="itcd_group">
    <rule id="prefered_location_itcd_1" score="20">
    <expression attribute="#uname" operation="eq"
value="hacluster2.svl.ibm.com"
id="e3de5eee-ee29-4e94-89d7-36fef3d76082"/> /* Spare node for ITIC
should
                                                    preferred node fail */
    </rule>
    <rule id="prefered_location_itcd_2" score="100">
    <expression attribute="#uname" operation="eq"
value="hacluster3.svl.ibm.com"
id="f8153d3d-f821-46e3-b41d-7e869e2960ec"/> /* Preferred node for ITIC
                                                    to run */
    </rule>
    <rule id="prefered_location_itcd_3" score="-INFINITY">
    <expression attribute="#uname" operation="eq"
value="hacluster1.svl.ibm.com"
id="7b7e7ec4-1381-47fb-afdf-732c4b180ba6"/> /* ITIC cannot co-exist
with
                                                    Maximo on this node */
    </rule>
    <rule id="prefered_location_itcd_4" score="-INFINITY">
    <expression attribute="#uname" operation="eq"
value="hacluster4.svl.ibm.com"
id="1fd82c28-982b-462e-b147-6726d655a87f"/> /* ITIC cannot co-exist
with

```

```

eCMDB on this node */
    </rule>
  </rsc_location>
  <rsc_location id="location_taddm" rsc="taddm_group">
    <rule id="prefered_location_taddm_1" score="20">
      <expression attribute="#uname" operation="eq"
value="hacluster2.svl.ibm.com"
      id="b029d8a7-5a40-481f-8ebc-1168d6d76efa"/> /* Spare node for eCMDB
should
preferred node fail */
    </rule>
    <rule id="prefered_location_taddm_2" score="100">
      <expression attribute="#uname" operation="eq"
value="hacluster4.svl.ibm.com"
      id="2cdf690e-e9c7-464e-9148-21be25565161"/> /* Preferred node for
eCMDB
to run */
    </rule>
    <rule id="prefered_location_taddm_3" score="-INFINITY">
      <expression attribute="#uname" operation="eq"
value="hacluster1.svl.ibm.com"
      id="a3065c9e-e253-4890-879f-9cf143d82fed"/> /* eCMDB cannot co-exist
with
Maximo on this node */
    </rule>
    <rule id="prefered_location_taddm_4" score="-INFINITY">
      <expression attribute="#uname" operation="eq"
value="hacluster3.svl.ibm.com"
      id="dfd2f607-a322-483d-af67-b33b6ba3556d"/> /* eCMDB cannot co-exist
with
ITIC on this node */
    </rule>
  </rsc_location>
</constraints>
</configuration>
</cib>
</code>

```

Testing the scenario

The example we used here is a three-application, three-node system with one spare node. For the algorithm, we used a four-machine cluster that includes Maximo, eCMDB (shorthand for Enterprise TADDM server), IC (shorthand for Integration Composer), and one passive spare, all identically equipped hardware-wise. Furnish the remote site with a similar set of four machines. The passive spare can run any of Maximo, eCMDB, or IC applications. The algorithm follows this logical path:

1. The first machine or application that fails gets to run on the spare. And it automatically fails back if the original node is functioning again.
2. In the event of a second failure, the application precedence on the spare machine is as follows: Maximo, then eCMDB, followed by IC. Maximo is the most critical application since client applications expect to invoke it at all times.

3. Observe same priority at both sites: local and remote.

We require only one spare for three distinct applications. These applications have a built-in priority. While all three are expected to be highly available, there is a pecking order for that availability, and all machines and applications must respect it.

If we have one application that must be available 24x7, and the other applications don't have such a stringent HA requirements, then one spare is the least we need. If we have two such machines with equal HA profiles, then we need two spares, and so on. All of the stated conditions are true if we cannot have applications coexist on the spare node. That is, there is no conflict that prevents the three applications from being installed on the same (spare) machine. [Figure 1](#) illustrated the three-application, four-node cluster example. The figure on the right side is a mirror-image of the left side and is used for disaster recovery purposes. The latter is used only if all four machines on the left side are down.

In the tables below, Maximo (= A), eCMDB (= B), and ITIC (= C) run on separate machines. Let's ignore machine names since they don't matter; each takes on the identity of the application that runs on it.

The machines do not run any other significant applications except the designated ones. The passive spare machine (= O) has Maximo, eCMDB, and ITIC installed, but can only support one application in execute mode.

Remember, the application priority is Maximo > eCMDB > ITIC; in our HA design, and O is designated coordinator (spare node).

Table 1. The first failure

Application that fails	App configuration AFTER first failure
A	A => O; B; C
B	A; B => O; C
C	A; B; C => O
O	No strategy

Figure 2. First failure

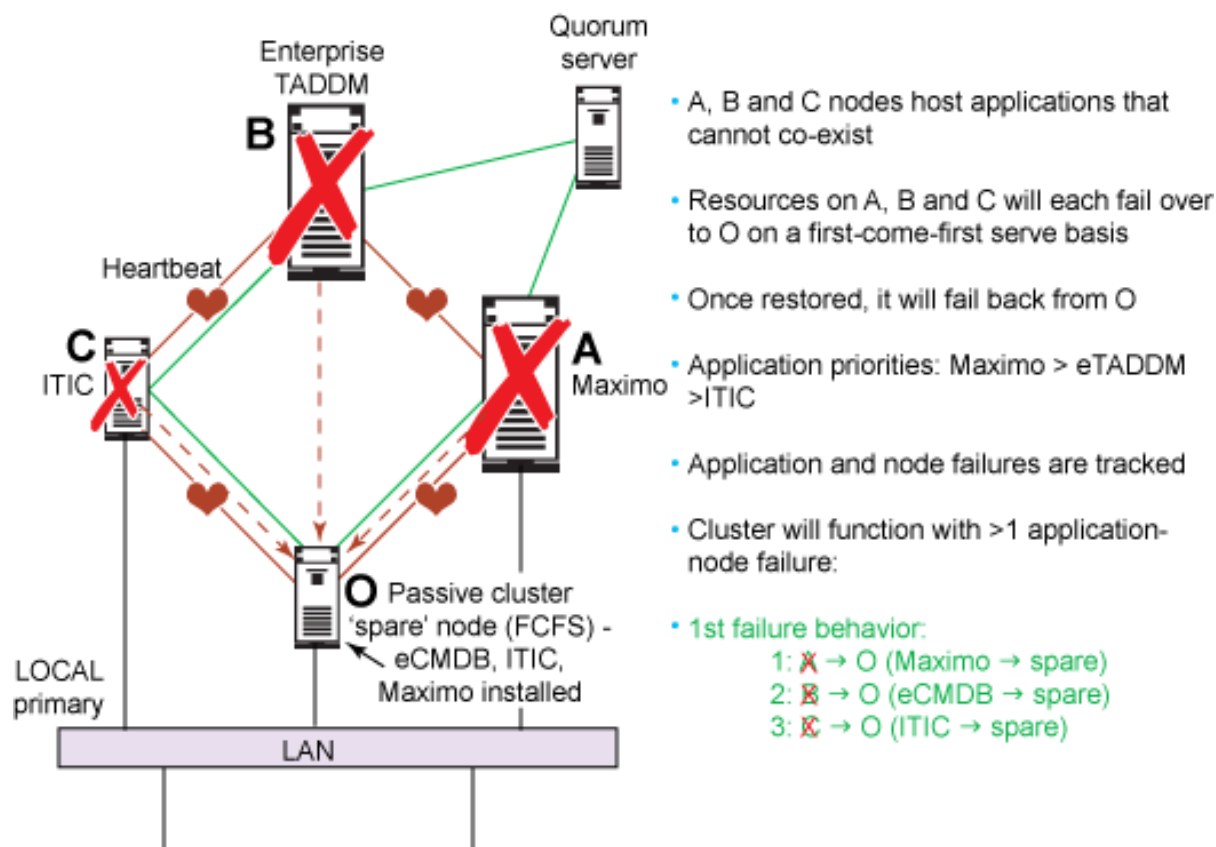
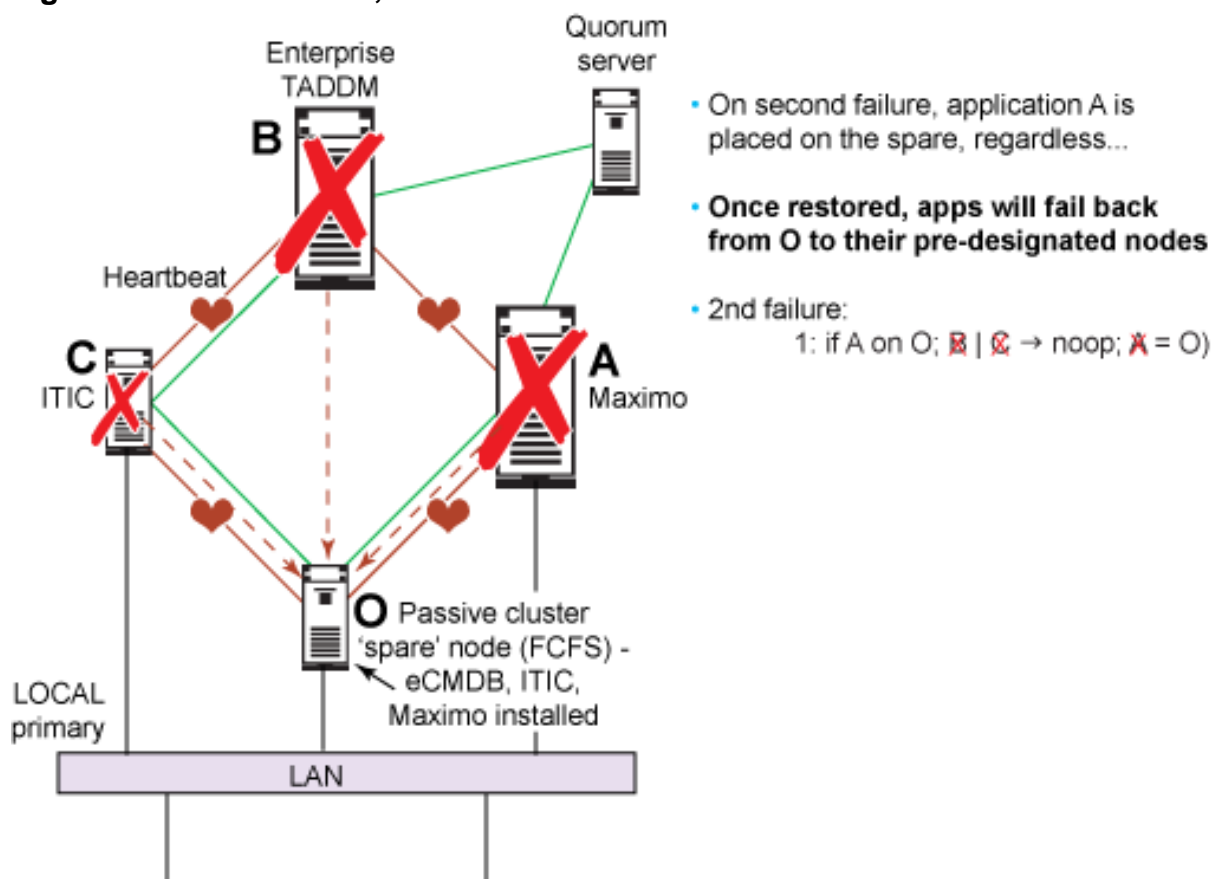


Table 2. Second failure (A = O means A is already running on O after first failure)

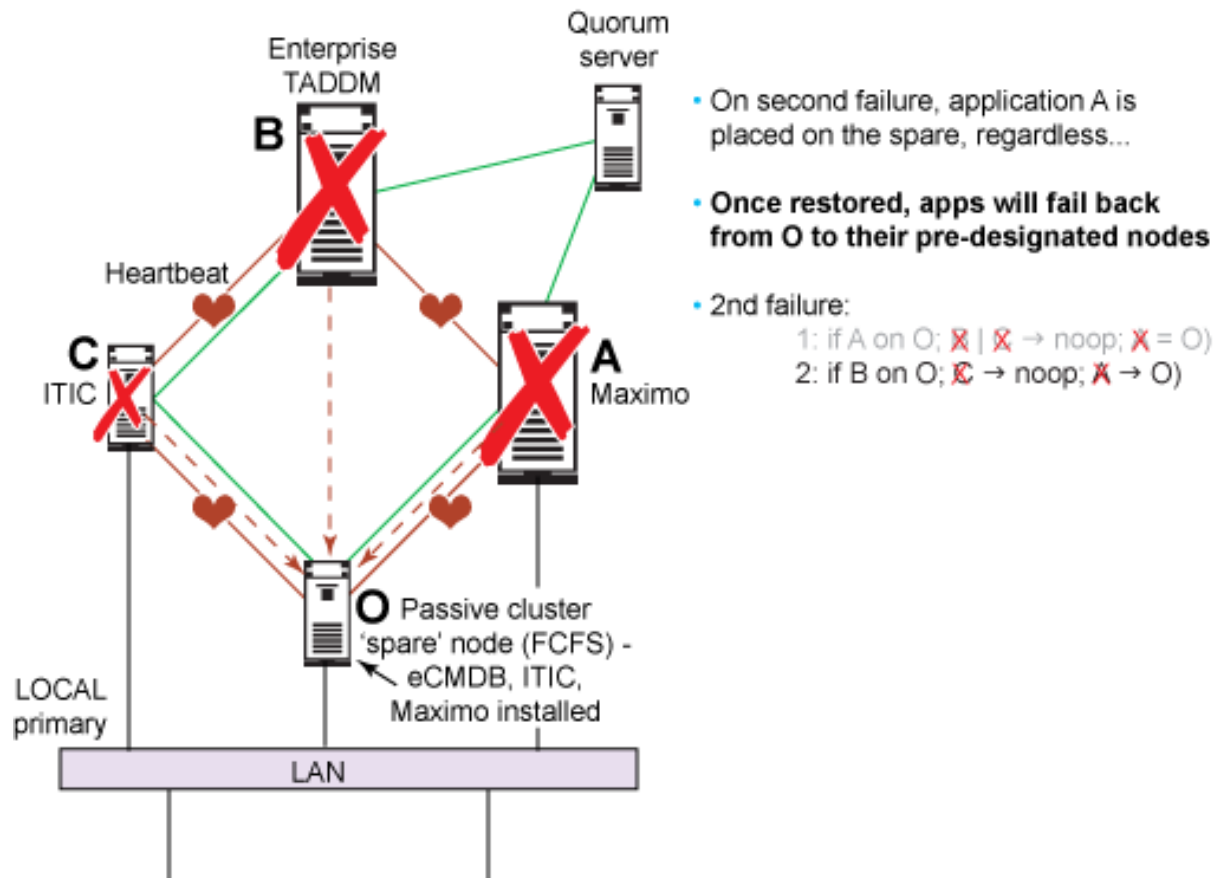
Identity of first failure	Configuration BEFORE second failure	Second failure	Configuration AFTER second failure
A	A => O; B; C	B	A = O; B unavail ; C;
		C	A = O; B; C unavail
B	A; B => O; C	A	B exits; A => O; C; B unavail
		C	A; B = O; C unavail
C	A; B; C => O	A	C exits; A => O; B; C unavail
		B	C exits; A; B => O; C unavail

For the second failure, case 1: If A fails, it is placed on the spare node. If ITIC or eCMDB should fail next, nothing happens. Maximo is continuously available on the spare node.

Figure 3. Second failure, case 1

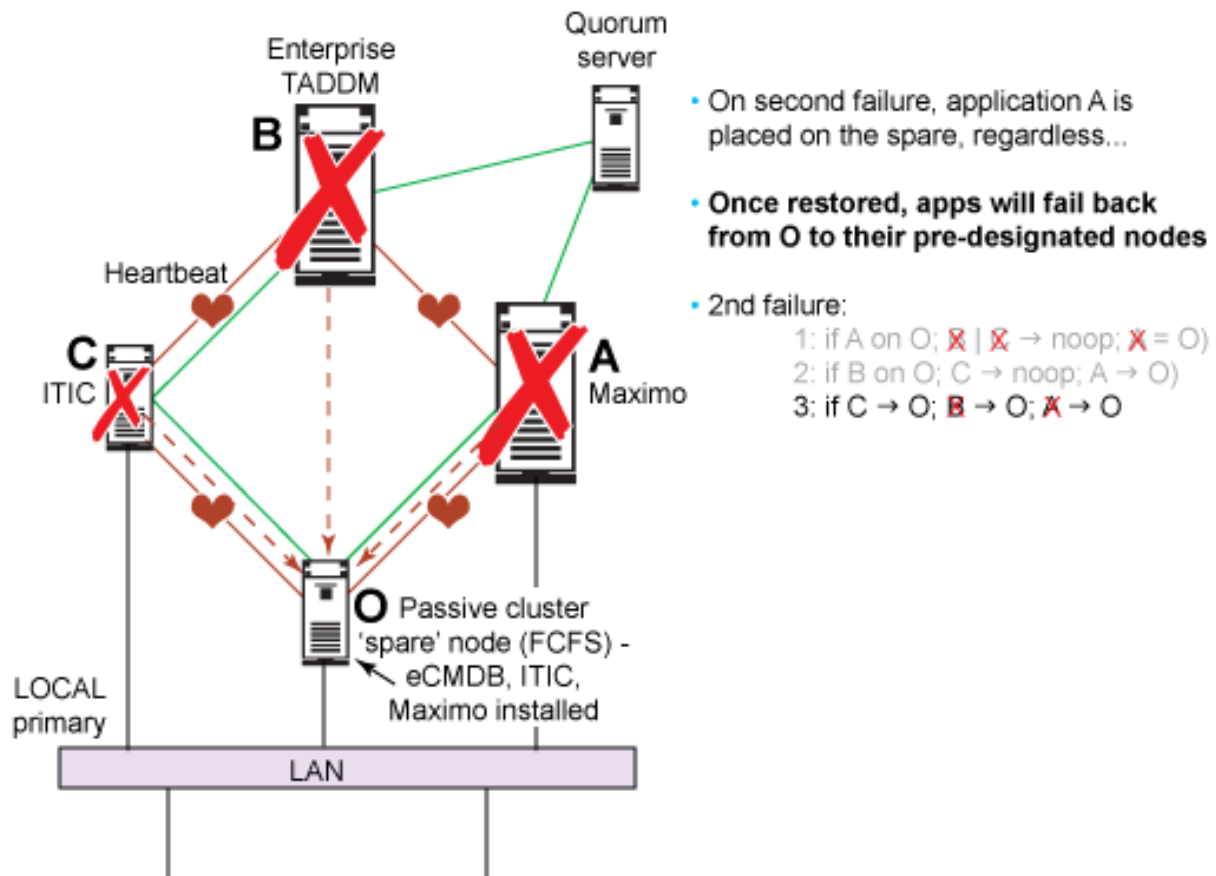
For the second failure, case 2: If eCMDB is on the spare node, then the subsequent failure of ITIC will not change the status quo; however, Maximo will bump eCMDB from the spare node.

Figure 4. Second failure, case 2



For the second failure, case 3: If ITIC is on the spare node, then the subsequent failure of eCMDB will bump ITIC from its perch. Should Maximo then fail, it will bump eCMDB from the spare node. If no node but the spare is available in the cluster, Maximo will be the only application running.

Figure 5. Second failure, case 3



Summary of implementation

The implementation follows these rules:

1. Four-node cluster with one passive spare.
2. The three applications run on dedicated nodes.
3. Applications cannot be placed together in execution mode (they are mutually exclusive).
4. On application failure, the application will be restarted two times, and then it fails over to the spare node.
5. On node failure, the application fails over to the spare node. The application fails back to its pre-designated node once the node is restored.
6. Application failover takes approximately 45 seconds. TADDM takes longer.

7. Application is placed on the spare on a first-come/first-served basis.
8. Availability hierarchy is observed at all times: Maximo > TADDM > ITIC.
 - Second application/node failure will make sure that Maximo is always available. If ITIC or TADDM is already on the spare, it will be bumped in favor of Maximo.
 - A complete set of round-robin tests is performed to make sure this is the case.
9. A hand-crafted and tested configuration file was tuned for the above behavior.

Disaster recovery

The matrix on the remote site is exactly the same, but none of the applications are in running state. Only the common external disk is replicated from the primary. After site failover, the process is:

1. Stop all primary machines (including databases).
2. Detach VIP from primary machines.
3. Pass control (manually) to remote site.
4. Designate network disk in remote site as Master.
5. Start heartbeat using our cluster control script (which will start the applications in the same sequence and priority as the primary).
6. Sync application with network disk and databases.
7. Assign same VIP from primary to remote network.
8. Respond to client invocation (as before).

Summary

This article describes our HA implementation for a composite application using Linux-HA, based on our experience with a customer's requirements. Our HA task involved multiple applications with different pecking orders within the same cluster. It might have been simpler to add a spare for each application, but that solution is not economical. For most real-world applications of HA, you have to deal with the reality of economics, as well as redundancy—and they are often mutually exclusive.

Acknowledgments

Alan Robertson, Linux-HA wizard, was our consultant and sounding board. We simply could not have done this without his encouragement and generous assistance.

Resources

Learn

- This site presents a [simple and common resource configuration](#) (the one the cib.xml example from this article is based on).
- Learn more about [heartbeat](#) in these Webcasts and documents; there are even [tutorials](#) to teach you.
- Let an expert take you on a tour of the [Linux-HA Quorum API](#) and then [even more about the subject](#).
- [OCF standards](#) can lead you to resource agents (mentioned in this article) or to a world of other Open Cluster Framework APIs including more resource services, as well as node, group, and lock services and external interfaces.
- "[Set up a Web server cluster in 5 easy steps](#)" (developerWorks, August 2007) shows you how to get up and running with the Linux Virtual Server and Linux-HA.org's Heartbeat.
- The [Installing a large Linux cluster series](#) (developerWorks, started in December 2006) describes cluster computing with Linux; the first in the series introduces HA and heartbeat software.
- Learn about the latest in [IBM Tivoli software](#).
- In the [developerWorks Linux zone](#), find more resources for Linux developers (including developers who are [new to Linux](#)), and scan our [most popular articles and tutorials](#).
- See all [Linux tips](#) and [Linux tutorials](#) on developerWorks.
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- Grab your [Linux-HA software](#): The CRM is now maintained as an independent project called Pacemaker; Heartbeat 2.1.4 was the last release to contain the CRM.
- With [IBM trial software](#), available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the [developerWorks community](#) through blogs, forums, podcasts, and spaces.

About the authors

Mahesh Viswanathan

Mahesh Viswanathan is a Senior Technical Staff Member in IBM Global Technology Services. He is Chief Architect for the Express Remote Managed Infrastructure Services, a managed services product for remote delivery of infrastructure services. Mahesh was a Research Staff Member at the T.J. Watson Research Center. His interests include remote services delivery, service-enabled information systems, high-availability of composite applications, human machine interfaces in cars, speech recognition and synthesis, audio/video search and retrieval, and document image analysis. He has a Ph.D. in Electrical, Computer, and Systems Engineering from Rensselaer Polytechnic Institute, specializing in image analysis. He has more than 30 technical publications and 20 international patents and is an IBM Master Inventor and a senior member of the IEEE.

Suraj Subramanian

Suraj Subramanian is a Senior Integration Architect at the Banking Center of Excellence, HiPODS, located in San Jose, CA. He is responsible for rapid prototyping and proofs-of-concept that demonstrate integration with \ IBM's software products to banks and other IBM customers. Previously, Suraj spent 5 years in IBM's Global Services and was the Client IT Architect for eBay. His interests include Enterprise Integration Architecture with a focus on high availability and performance. He has a Bachelor's Degree in Electronics Engineering from Mumbai University, India.