University of Amsterdam
System & Network Engineering

# Research Project 1

## Failover research for *Bright Cluster Manager*

*Authors:*
Cosmin Dumitru
cdumitru@os3.nl

Niek Timmers
ntimmers@os3.nl

*Coordinator:*
Prof. dr. ir. Cees de Laat
University of Amsterdam

*Supervisor:*
Martijn de Vries
Bright Computing Inc.

February 5, 2010

**Abstract**

A common approach in scientific research is to make use of clusters to help solve advanced computational problems. Clusters used to be difficult to use and manage. Multiple companies have designed cluster managers which led to a more easy to use and productive experience. A high performance computing (HPC) cluster is based on the Beowulf model[1] and consists of a head node and a number of compute nodes.

One important aspect of a cluster is high availability. To assure a cluster is usable at any time, a failover mechanism is implemented. This failover mechanism mitigates the problems that occur when a head node fails. A common approach for a failover mechanism is to have redundant head nodes.

During our research, we investigated the failover mechanism implemented in *Bright Cluster Manager* . We have addressed some issues in the current implementation and we introduced some proposals which could make *Bright Cluster Manager* more efficient.

# List of Figures

# Contents

# 1 Introduction

High Performance Computing (HPC) is commonly used for scientific research but also commercial organizations make use of it. To achieve HPC, either a supercomputer or a cluster of computers is used. The problem with a cluster of computers is developing an easy-to-use interface for its end users. There are several software companies which achieve this in the form of a Cluster Manager (CM) like *Bright Cluster Manager* which is developed by *Bright Computing Inc.* The philosophy of their CM is:

- Easy to install clusters
- Easy to use clusters
- Easy to monitor clusters
- Easy to manage clusters
- Easy to scale clusters

A common approach [1] for the cluster is to have a single head node which is essential for the operation of the cluster. Without this head node, the cluster is completely unusable by its end users. To mitigate this problem, a protocol has to be introduced which takes over when the head node fails. *Bright Cluster Manager* has the option to use an active and a passive head node. When the active head node fails, the passive head node takes over so the cluster can still be usable.

## 1.1 Research

Designing and developing a cluster failover between two head nodes can be a challenging task. Designing a perfect theoretical system is one thing but actually implementing it can be a quite complex task. Usually developers keep things as simple as possible. The research we have done at *Bright Computing Inc.* involves testing the failover mechanism of *Bright Cluster Manager* . The research will be rather focused on the logical implementation and not on the technical implementation. The question we answer by doing this research is:

> *Is the failover mechanism implemented in* Bright Cluster Manager *working as intended and can it be improved?*

The answer of this question will be based on answering the following subquestions:

- How does the failover mechanism work?
- Is the failover mechanism working as intended?
- Is the failover mechanism scalable?

- Is the failover mechanism efficient?
- Is the logical design of the failover mechanism correct?

Researching the failover mechanism of *Bright Cluster Manager* will give us a better insight of how such a mechanism should work. We hope to give *Bright Computing Inc.* also some advice to improve their product.

## 1.2 Experimentation Setup

For our research, we have setup a test configuration of *Bright Cluster Manager* . We did this on our test machines in the OS3 lab. The test setup consisted of three Dell servers which were running *VMWare ESX 4.0*.

**server1(pidgey)** : head node 1, slave node 1, slave node 3

**server2(nidoran)** : head node 2, slave node 2, slave node 4

**server2(sandshrew)** : Router, DNS, RedMine, Subversion

On the first week we focused on installing the infrastructure and on top of that *Bright Cluster Manager* itself. Like the *Bright Cluster Manager* website states, installing the software was fairly easy. We had some problems in getting the failover mechanism to work, but this was due to a bug. Luckily the developers at *Bright Computing Inc.* were able to fix this bug within a day. When the cluster was running we processed some test cases which can be found in the appendix A.

In the following two weeks we mainly conducted research on the current design and thought of proposals to improve the design. This involved a deep research on the current state-of-the-art HPC cluster designs.

# 2 Cluster Computing

## 2.1 Basic Principle

A computer cluster is a group of tightly-coupled computers that share resources. In many aspects the cluster appears to its users as a single computer. [2]

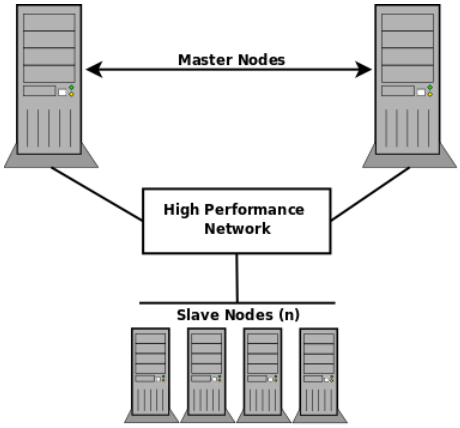


Figure 1: A common cluster setup with two head nodes.

## 2.2 Types of Clusters

### 2.2.1 HA clusters

High-Availability (HA) clusters are employed to increase the availability of services. This is achieved by eliminating single-points-of-failure through the use of redundant nodes and redundant hardware. Usually, these types of clusters make use of a failover mechanism which is described in detail in chapter 3.[2]

### 2.2.2 Load-balancing clusters

Load-balancing clusters are used to distribute the load over multiple machines, thus increasing the performance and scalability of a system. [2]

### 2.2.3 Compute clusters

Compute clusters or HPC clusters are used for computationally intensive (usually parallel) applications. Jobs are divided in smaller tasks which are then submitted to the compute nodes. While the job is running, the nodes exchange information using the high-bandwidth, low-latency network that interconnects them. The result of the computing jobs are collected on a master node which

coordinates the compute nodes. The nodes in a compute cluster usually have the same architecture and the same configuration. [2]

### 2.2.4 Grid computing

Grid computing is a form of large-scale cluster computing. A grid consists of loose connected and geographically dispersed computers (or compute clusters). Grids aim at aggregating distributed computation capabilities and offering them as a service[15].

# 3 Failover

Failover is the action of switching automatically to a standby server in the event of network, hardware or application failure. This increases the overall system availability and therefore, it is commonly implemented in systems where high-availability is required. Ideally the failover should meet the following criteria [2]:

- *Transparent* - Failover should not be more intrusive to clients than a simple system reboot. This does not imply that the outage should take as long as a reboot, but that the actions that the users must take in order to access the services should not differ from the ones performed after a system reboot
- *Quick* - Failover time should ideally take between two and five minutes [2, p. 366]. This can be achieved if the standby server is already booted and running as many of the required processes as possible.
- *Minimal manual intervention* - Ideally the failover process should be fully automated in order to provide the best failover times possible.
- *Guaranteed data access* - Once the failover is completed, the new server should have access to the same data as the original server.

## 3.1 Hardware Failures

Server failures affect the system availability as recovery implies diagnosing the failed component, replacing it and sometimes even reconfiguring the system to its previous state. The time that takes for a failed component to be repaired is called *MTTR - Mean time to recovery.*

One solution for server failures is equipping the server with fault-tolerant (i.e. redundant) hardware. In such a configuration, most of the hardware is redundant and the server can withstand at least one hardware failure. Achieving redundancy depends greatly on the targeted hardware: for disk drives, a RAID (Redundant Array of Independent Disks) setup is nowadays common, while for backplane or motherboard failures are supported only by high-end specialized systems. The most common hardware failures in HPC clusters[5]:

- *Disk drives* - mitigated when using a RAID configuration
- *Network cards* - mitigated by using redundant network connections, e.g NIC teaming
- *Memory modules* - mitigated by using error correcting memory modules

## 3.2 Software Failures

Software failures can have several causes, some more difficult to mitigate than others.

While I/O errors can be easily detected by checking the return values of write or read calls, other failures like improper database reconnection need additional logic or even human intervention. Network failures can have an impact on the applications that are dependent on network connectivity. Improper handling of such situations causes systems to go into states where human intervention is necessary.

Last but not least, internal application failures are the most common cause of application-level failures. Possible causes are memory access faults, memory corruption, hanging processes because of deadlocks or infinite loops. Thorough application design and testing can greatly reduce the occurrence of these events.

A special type of application crashes are service crashes. A service crash that depends by an internal state of the service and on everything that happened before will be fixed by a service restart as the internal state is cleared. Literature [18] calls this type of error a *Non-deterministic error*. A deterministic error [18] occurs when the service crashes regardless of the internal state. This is most likely as a response to data input and automatic mitigation of this type of error is impossible.

## 3.3 Failover Requirements

In [2], the authors present the following items as necessary when designing a basic failover system:

- *Two servers* - an active server and a passive server. In the case when the active server fails, the passive server takes over and the critical applications are migrated. Ideally, the servers should be identical in terms of hardware, operating system and configuration.

- *Network connections* - a network connection dedicated for *heartbeats* that need to be exchanged between the two servers. To provide redundancy, two heartbeat networks should be deployed. The heartbeat networks allow the servers to communicate and monitor each other. Having a separate network for heartbeats assures that there are no network delay and bandwidth problems and in this way messages are guaranteed to arrive in a timely manner.

- *Shared storage* - besides the *unshared disks* that host the operating system and other files, additional shared storage is necessary to host the application critical data. Access to the shared disk is changed back and forth when a failover occurs. The shared disks should be configured in a redundant setup (e.g. RAID). An alternative to shared disks is the *shared nothing* approach where data is permanently synchronized between the two servers. This method increases the overall complexity of the system.

- *Application portability* - the critical applications must be able to run simultaneously on both servers.

- *No single point of failure* - an ideal highly available system has no single-point-of-failure. Careful hardware and software design is needed in achieving this goal. Additional thorough testing is necessary to make sure that in the case when one component (software or hardware) fails, the system can failover to a working state.

## 3.4 Failover Problems

### 3.4.1 Split-Brain Syndrome

The *Split-Brain Syndrome* [23] occurs in a cluster of paired servers when the servers lose connectivity with each other, but in other aspects remain functional. The servers are no longer in sync and each of them acts individually, without taking into consideration the other server's state. This triggers a takeover procedure by the passive node who tries to acquire access to the shared disks. At this moment, both servers try to write data on the shared disks, leading to instant corruption of the shared application data.

Mitigation methods of the split-brain syndrome include:

- using an additional network connection for heartbeats. Eliminating single-points-of-failure dramatically reduces the chance of split-brain.
- asking a 3rd party to determine the next action that needs to be taken (i.e. a quorum vote)

# 4 Failover Configurations

A failover solution can be configured in multiple ways. There is no rule that states which is the best solution but there are some points to think of while designing a high-availability cluster. This section will explain the types of available failover solutions. Also, this section will cover the advantages and disadvantages of each setup in a HPC context.

## 4.1 Two head node configuration

A common approach for failover is to have two homogeneous head nodes, where the passive node can take over if the active node fails. There are basically two options to configure the head nodes in a two-node failover configuration [20]:

- Active/Passive
- Active/Active

### 4.1.1 Active/Passive

In an *active/passive* failover configuration only the active node is doing all the tasks needed to operate the cluster. The passive node is a dedicated standby, which means that it is ready to take over when the primary node fails. A failure of the active node will result in an unusable cluster. The standby node is there to mitigate the problem of an unusable cluster. A disadvantage of an *active/passive* setup is that the standby node is basically doing nothing. Only in the case of an failover the standby node will render itself useful.[2][24]

In the case of a failover, the configuration of the active node needs to be loaded on the standby node. Before loading this information, the standby node must be sure that the active node is not online any more. In an *active/passive* failover configuration , it is easy to get into problems if both the active and passive node are online. For example, they could start writing to the same shared disk which will lead to data corruption. These are the primary reasons a failover in this configuration could be inefficient. A configuration which probably could resolve this issue is an *active/active* failover configuration.

### 4.1.2 Active/Active

In an *active/active* failover configuration, both nodes provide services [2][24]. If one of the nodes fails, the other node has to take over all the requests. Advantages of an *active/active* configuration over *active/passive* are:

- *load balancing* - Tasks processed by the head node can now be distributed between two machines.
- *faster failover* - Failover can be faster because there is no need to load the configuration as in the standby node.

- *partitioning* - Partitioning can be more effective.

This type of configuration also presents some disadvantages:

- *writing* The head nodes cannot write to the same disk at the same time, thereby it is necessary that both nodes have their own storage or maybe a complex disk sharing mechanism has to be designed.
- *communication* The slave nodes need to communicate with two different machines at the same time, thereby the software they are using needs to be compatible. *Out of the box* software might not work any more and additional configuration might be required.
- *job manager* The job manager which is used needs to be able to cope with the distribution of job queues over multiple machines.
- *complexity* Using multiple nodes for the head node task can be complex. The design must be well thought off, otherwise the complexity will destroy the effectiveness.

In the end it all depends on the given situation and the desired degree of availability.

## 4.2 Multiple Node Configuration

It is possible that one standby head node is not sufficient. The worst case scenario in a two head nodes setup, would be that after the active fails, the passive (which becomes active) fails while the old active is still being repaired. To achieve optimal availability, it is advised to have multiple standby head nodes. This would lead to a more complex setup and higher costs but it will increase the availability of the cluster [2].

## 4.3 Which is best?

Every cluster design has different needs for a failover configuration. Implementing a failover mechanism can be a complex and subtle task, thereby developers tend to keep a failover mechanism as simple as possible. However, keeping the design simple will not necessarily result in a better product. Later in this paper there will be proposals which use different failover configurations.

# 5 Failover Mechanisms

A failover mechanism can be designed in many different ways. While there are no "silver bullet" designs in what regards failover systems, there are techniques which are almost always used. In this section, some of these common techniques will be discussed. We consider a high availability cluster with two nodes in an *active/passive* configuration.

## 5.1 Fencing

It is not desirable to have multiple nodes writing to the same shared resource at the same time because it will sooner or later result in data corruption. When designing the system, there should be a feature that makes sure only one node can access a shared resource at a given time. Fencing is a cluster feature which cuts off the access to a shared resource during a failover, for example. The active node gets cut off by the fencing mechanism from the shared resource, thus giving the passive master node the opportunity to take over the resource, without the active master node still using it. The most effective way of accomplishing this is to make use of a STONITH 5.2 mechanism [6].

## 5.2 S.T.O.N.I.T.H.

STONITH stands for *Shoot The Other Node In The Head* and in the context of a cluster this means that a node has the possibility to "kill" another node. This can be achieved by cutting off the power of the targeted node. Another way, which is more hardware-friendly but less effective is shutting down the targeted node's network interfaces. To check if the targeted node is really "dead", the other node can use a heartbeat 5.3 and a quorum 6.3.4. If the killer node can reason that the targeted node is really "dead", it can initiate the failover. This conclusion is usually easier to make after powering off the targeted node rather than turning off its network interfaces. For both cutting off the power and shutting down the interfaces on the switch, special hardware is needed. For cutting off the power, a power distribution switch with a network interface is required so the master nodes in the cluster can communicate with it. For shutting down the network interfaces a remotely configurable switch is needed [25].

## 5.3 Heartbeat

The master head nodes need to monitor each other's online status. Also, the master nodes must know which slave nodes are offline and which are online. Besides the physical machine being offline, it is also important to know that all the essential services are running. There are several methods and programs that can achieve this. Common types of heartbeats are:

- ping[7]

- snmp[8]
- ipmi[9]

A heartbeat can use several communication channels for gathering its information:

- IP multicast[10]
- IP broadcast[11]
- IP unicast[12]
- serial line[13]

Like in authentication systems, it is always better to use multiple heartbeats. Especially between the master nodes, it is advisable to use, for example, an ICMP ECHO_REQUEST ping over two separate links to make sure that if a link between the master nodes fails, a false positive is not triggered, thus resulting in an undesired failover.

## 5.4 Quorum

If the passive master can no longer detect the active master, it can not sure if the failover is desired. There can be situations where the slave nodes and the active master can communicate. but the passive master cannot. This should not trigger a failover. To mitigate this problem the passive master needs to make sure that the active master is indeed in a failed state.

One method implies that the passive master asks the slave nodes what is their opinion about the current situation. It asks the slave nodes if they want the current active master node to be the active or the passive one. After the vote is made, the passive master node knows if it needs to start a failover or not.[26]

An action is started when the majority of the votes of the slave nodes were for either the current active or passive master node. Before the quorum is started, the passive master node should know at least how many slave nodes are online in order to determine what the majority is. The majority of votes is defined by $(\frac{n}{2}) + 1$ where $n$ is the total amount of online slave nodes.

In the situation where the passive master node detects a failure of the active master, it will initiate a failover sequence. To actually get to the point where it performs a STONITH 5.2 on the active master, it first needs a quorum majority from the slave nodes. A typical quorum process has the following steps:

- Passive master node needs a quorum decision.
- Passive master node sends out vote requests to the slave nodes.
- Slave nodes reply with their vote to the passive master node.

- Passive master node accepts the votes from the slave nodes.
- Passive master node starts the failover when a majority of $(n/2) + 1$ is achieved.

To summarize, a quorum is required in order to make a accurate decision in a cluster and to avoid a "split-brain" 3.4.1 situation.

# 6 Bright Cluster Manager

## 6.1 Goals

The design of *Bright Cluster Manager* is guided by the following goals :[14]

1. Make clusters easy
2. Scale clusters to thousands of nodes
3. Be complete

### 6.1.1 Make clusters really easy

*Bright Cluster Manager* makes sure that the following tasks can be executed as easy as possible by the end user [14].

**Install Clusters** *Bright Cluster Manager* is easy to install even with minimal Linux knowledge.

**Monitor Clusters** *Bright Cluster Manager* can monitor all the nodes in the cluster via a GUI.

**Manage Clusters** The cluster is centrally manageable through a GUI.

**Use Clusters** *Bright Cluster Manager* supports a large collection of compilers, MPI libraries, software development tools and environment modules.

**Scale Clusters** It is easy to add slave nodes to the cluster by plugging them in and let the software take care of the rest. Advanced features such as failover and load-balanced node provisioning are accessible through the GUI.

### 6.1.2 Scale clusters to thousands of nodes

*Bright Cluster Manager* is designed to scale to thousands of nodes. The following features help achieve this:[14]

- Management Daemon
- Load-Balancing Provisioning Nodes
- Synchronized Cluster Management Daemons
- Built-In Redundancy
- Diskless Nodes

### 6.1.3 Be complete

*Bright Cluster Manager* is designed to be complete. There is no use for third party tools. This does not mean that the software is not making use of third party tools. It heavily depends on open source tools but this is fully transparent to the end user. The following features can be found in the software :[14]

- Multiple Linux Distributions Supported
- Intel Cluster Ready Complient
- Cluster Management GUI
- Cluster Management Shell
- Node Provisioning System
- Node Identification
- Software Update Management
- Cluster Monitoring
- User Management
- Parallel Shell
- Workload Management
- Cluster Health Care Management
- Cluster Security
- Compilers
- Debuggers & profilers
- MPI Libraries
- Mathematical Libraries
- Environment Modules

### 6.1.4 Cluster Resources

An HPC cluster using *Bright Cluster Manager* essentially has two type of nodes: master nodes and slave nodes. The slave nodes are responsible for running compute jobs. They are not accessible directly by the cluster users. The master nodes serve as management nodes for the slave nodes and as access gateways (login role) for users. The users can submit jobs from the master nodes. Additionally, master nodes monitor the status of the slave nodes by periodically polling monitoring data from the slave nodes.

To implement a high availability system, a number of two master nodes are used. Besides static IP addresses that are uniquely assigned to them, they also have two shared IP addresses. One IP address is used by the users to connect to the cluster and is configured on the external network, while the other is used by the slave nodes to communicate with the master nodes and is configured on the slave network.

In addition, a shared storage is used by the master nodes to store cluster configuration information, job results and certificates used in authenticating and authorizing cluster commands.

### 6.1.5 Management

*Bright Cluster Manager* offers two options to manage the cluster. One is a shell interface called *Cluster Manager Shell* or in short `cmsh` and the other one is a GUI, *Cluster Manager GUI* or `cmgui`. Most tasks can be accomplished both from the GUI and the shell. The shell can be used to execute automated sequences, while the GUI offers an easy interface to the cluster management tools.

## 6.2 Failover Configuration

*Bright Cluster Manager* has an *active/passive* head node configuration for its failover feature. One could argue that is a *active/active* head node configuration because it is possible to assign certain tasks to the standby master node. It is not possible to run the head node specific tasks on both nodes at the same time. This is the primary reason that makes it an *active/passive* configuration.

The active node is said to be the "active master" and the standby node is said to be the "passive master". Both masters monitor each other via several heartbeats:

- *MySQL* - A local mysql instance status check is being done. The status is reported to both nodes.
- *Ping* - An ICMP ping is made via the slave network between both nodes. The result of this ping is reported to both nodes.
- *SOAP RPC* - A SOAP RPC call via HTTP is made via the slave network between both nodes. The *CMDaemon* can be accessed on port *8080*. The result of this SOAP RPC is reported to both nodes.
- *BackupPing* - In addtion to the default heartbeats, another ping heartbeat can be configured. It is up to the administrator via which network this ping is being made.

Together, these heartbeats form a status. Via the cluster management shell it is easy to query this status. The result of a status query where an additional heartbeat is configured looks like this:

```
[root@master1 ~]# cmha status
Node Status: running in active master mode

Failover status:
master1* -> master2
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
  backupping   [  OK  ]
master2 -> master1*
```

```
mysql        [  OK  ]
ping         [  OK  ]
status       [  OK  ]
backkuping   [  OK  ]
```

During normal operation, the shared resources 6.1.4 are available and configured on the active master. The active master has exclusive access to the shared disk and the shared IPs are configured as aliases on its network interfaces. During failover the cluster resources are migrated to the the passive master.

### 6.2.1 Manual Failover

The failover is intended to be triggered automatically, but manual failover is also possible. This can be used when the active node needs for example maintenance. Without the failover mechanism, this could result in a relatively long downtime. A failover can be manually initiated by simply executing the following command on the passive node:

```
cmha makeactive
```

### 6.2.2 Automatic Failover

During a manual failover, no quorum is needed to decide which of the head nodes should be the master. The decision is made by human intervention. Only an automatic failover needs a quorum to make the right decision. A failover is initiated when all configured heartbeats are failing. This means that a failover is only initiated in the following situations:

- *No Power* - When the "active master" loses power, the "passive masive" detects this. It will initiate a quorum. The nodes will decide if they want a failover or not.
- *No Slave Network* - Only when the additional heartbeat is not configured, a failover will be initiated when the "active master node" loses connection to the slave network. All three default heartbeats are sent via the slave network. The "passive master" will detect this and initiate a failover.

## 6.3 Logical Design Issues

The current design of *Bright Cluster Manager* can be improved in various ways. This section will focus on the areas that can be improved. The design of *Bright Cluster Manager* is now roughly:

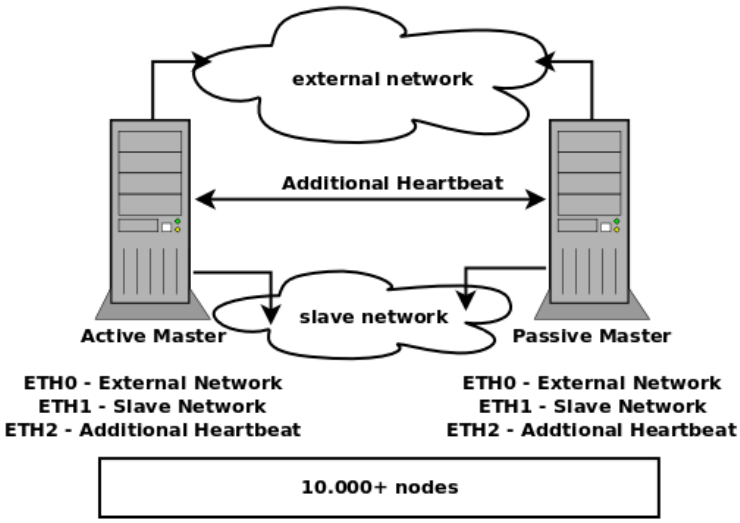


Figure 2: Cluster failover including $n$ slave nodes.

All the information used in the following sections was gathered by communicating with the developers of *Bright Cluster Manager* . This has been done either in person or via email.

### 6.3.1 Additional Heartbeat Link

**Context** In *Bright Cluster Manager* there is the option for an additional ping heartbeat over an alternative network. By default, the ping and SOAP heartbeat are going over the slave network, while the additional heartbeat ping can go over the slave network, external network or an additional network.

**Problem** If the active master node loses connectivity to the slave network, the cluster is rendered unusable, as users cannot submit new jobs to the slave nodes and they can not send job results back to the master node. Clearly, this is an event that should trigger a failover.

In the default setup if, the slave network interface fails, a failover is initiated by the passive master as no heartbeat is detected from the active one.

When using an additional heartbeat network, a failover is initiated only if all heartbeats fail. Therefore. if an administrator configures the additional link over an additional network, no failover will be triggered if the slave network NIC on the active master fails and the backup heartbeat interface is still up.

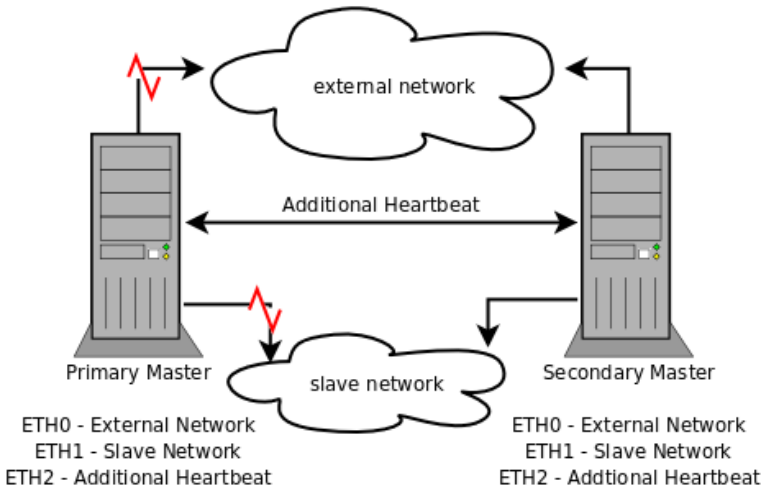


Figure 3: Failover setup with both external and slave network failing.

We reason that the behavior of the failover system is not consistent, as slave network failure can or can not trigger a failover, all depending on the setup of the master node pair. The slave network interface is treated only as a heartbeat link and not as a critical component of the cluster setup.

**Impact** If the slave network connection fails and the additional heartbeat link is still functional, the system does not perform a failover.

**Recommended resolution** We recommend that in the setups where an additional heartbeat link is deployed, the NICs should be connected to the slave network and not to a separate network. Ideally, the backup NICs should be connected to different switches than the main NICs. When the network interface on which the master IP of the cluster is configured fails, the IPs configured on the failed NIC should be migrated to the backup NIC.

Our recommendation eliminates a single-point-of-failure: the slave network NIC.

**Considerations** The solution does not add a lot of complexity to the existing failover system. Extra care should be taken to restart the services that are dependent on the migrated IP addresses.

### 6.3.2 Local Disk Failure

**Context** We consider a local disk failure an event which leads to a state where the volume that stores the current operating system files can not be accessed anymore.

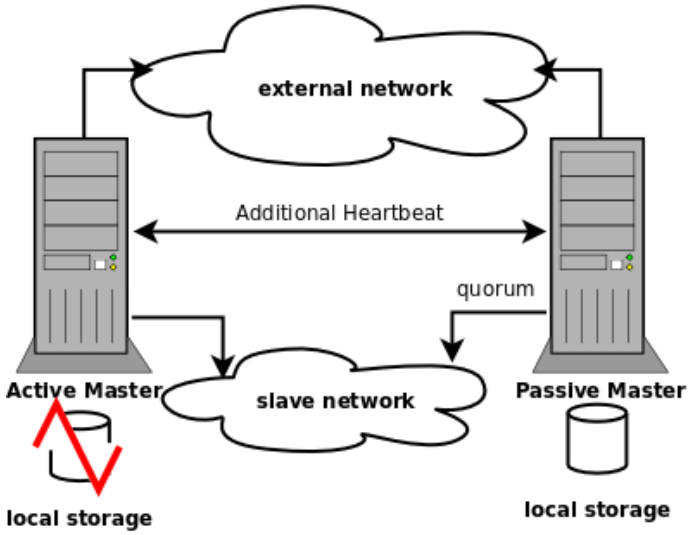


Figure 4: Slave nodes report to masternode

Local disk failures are uncommon in High Availability systems as usually the disks are configured in a redundant RAID. Yet the chances of local disk failure still exist as studies show that disk failures are correlated [17].

In a RAID setup, the disks are connected to a RAID card that performs the logical operations needed by the configured RAID level. This hardware component is a single-point-of-failure as its failure will make all the connected disks unusable.

In addition to hardware failures, research shows that a large percentage of disk failures are caused by human operators [5]. Having this in mind, the probability of local disk failure is no longer negligible.

**Problem** *Bright Cluster Manager* does not handle local disk failures. We have tested the behavior of *Bright Cluster Manager* to local disk failure by several methods. We have simulated a local disk failure by removing the OS disk from the running active master node. This was done by sending a signal to the Linux kernel to remove the SCSI device:[27]

```
echo 1 > /sys/bus/scsi/devices/0:0:0:0/delete
```

At this point, we have observed that the OS on the active master is still functional. Testing the failover status on the passive master indicates some trouble on the active master but a failover is not initiated.

Below is a status output for the failover system:

```
[root@master2 ~]# cmha status
Node Status: running in passive master mode (a.k.a. "failover mode")

Failover status:

(Wed Jan 27 14:28:07 2010) Soap error (cmjob->getJobQueues()):
Unexpected  end of file, no input, or timeout while receiving
data or invalid socket

(connection error)(#-1) master1* -> master2
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
master2 -> master1*
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
```

Figure 5: HA status after local disk failure

The `getJobQueues` command fails as checking the job queue requires disk access but other than that, the health indicators on which *Bright Cluster Manager* relies are indicating that the system is in a valid working state. Because the kernel and the running applications are stored into RAM memory, the network stack is still functional and the processes are still running. The active master continues to reply to ICMP ECHO_REQUEST packets and some SOAP RPC calls still succeed. SSH connections initiated to IP addresses configured on the active master fail. The behavior of the other running services is unpredictable but during our testing *Bright Cluster Manager* did not detect any service failures. This shows that *Bright Cluster Manager* is not checking the state of the mounted disks and it does not detect local disk failures. We have tested the effect of a human error by running the following command on the active master:

```
[root@master2 ~]# rm -fr /
```

The effect was similar to the one described in the previous paragraph.

**Impact** In the event of local disk failure, the cluster is rendered as unusable without a failover being initiated.

**Recommended resolution** We recommend including a periodic checking of the current mounted file systems. The check should rely only on system calls as standard Unix checking utilities can not be used. Our proposal enhances the automatic failover mechanism of *Bright Cluster Manager* by making it more reliable in the event of catastrophic failures.

**Considerations** The overall system complexity is not increased.

### 6.3.3 External Network

**Context** The master nodes are equipped with at least two network cards. One is used for the external network from which clients connect to submit computing jobs and from which management operations are initiated, and the second one is used for the slave network, a dedicated network for the computing nodes.

In addition, the external network interfaces can be used to check the other master node by sending ICMP packets.

External network connectivity is needed also for keeping an accurate time. Accurate time is provided by the `ntpd` service which connects to external time servers.

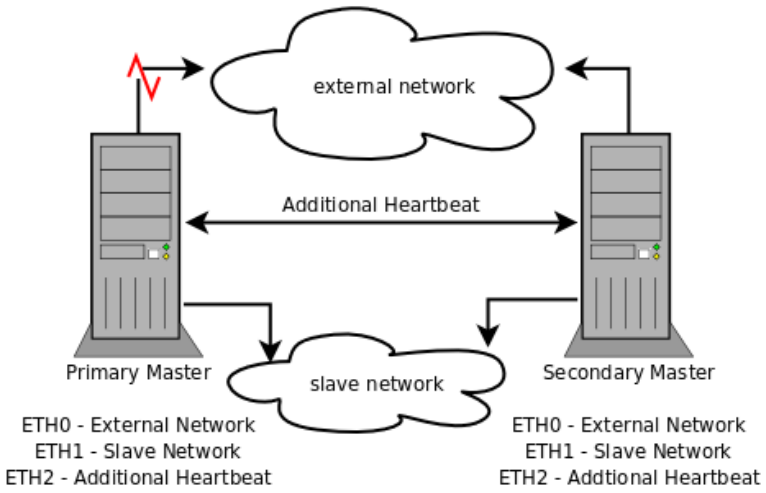


Figure 6: External network link fails.

**Problem** If the external network interface fails, a failover action is not triggered.

**Impact** The failure of the external network interface can affect both the users of the clusters, as they are unable to connect to the active master and cluster services rely on accurate time. In addition, depending on the noticed clock skew, service log correlation on the active and passive master is made difficult if not impossible.

**Recommended resolution** We recommend that *Bright Cluster Manager* should monitor the external network interface and in the event of failure it should initialize a *graceful* failover. In order to detect if the external connectivity has been lost, the active master should send ICMP probes to the external interface of the passive master and to the default gateway. Only if both tests fail, the

failover should be started. In addition, to eliminate a single-point-of-failure, a second standby external network NIC should be installed on the master nodes.

**Considerations** The proposed resolution does not add much complexity to *Bright Cluster Manager* .

### 6.3.4 Large Quorums

**Context** The quorum is initiated when the passive master detects that the active master node is failing. The quorum consists of unicast messages sent to all the active slave nodes. The slave nodes will reply with information regarding whether they want the passive master to takeover the active master. When a majority of $n/2+1$ is reached, where $n$ is the number of active slave nodes, the failover is initiated by the passive master. The power is cut off from the active node and the passive master takes over.

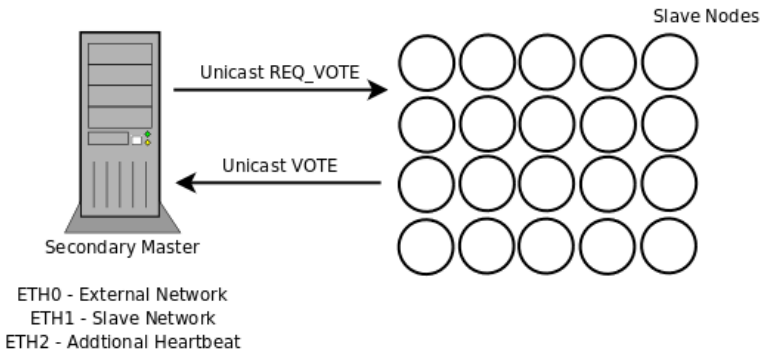


Figure 7: Quorum mechanism initiated by the active master

This quorum mechanism has good performance in relatively small clusters. The quorum mechanism was intensively tested in the test setup 1.2. With four slave nodes at low load the voting process takes less than 5 seconds.

**Problem** In a large cluster with high load, the quorum mechanism might not perform efficiently. In the current setup, the cluster management daemon is running on the slave nodes with normal priority ( `nice 0` ). Slave nodes with high loads on both I/O and CPU can delay the quorum voting response as they must process the voting request, check which of the master nodes are up and provide a response to the node which initiated the voting. This issue is mainly present because the passive master needs to send out all the VOTE_REQ in a unicast manner. Unicast is a sequential operation, therefore a delayed response further delays the voting process. The current implementation uses a multi-threaded voting process which partially circumvents the eventual delays, yet the number of running threads is limited by the number of available CPUs on

the master node. Given the fact that the voting operation becomes essentially sequential, the delay in voting could exceed the desired failover time threshold.

**Impact** Large, high load computing clusters could be affected, when a failover is needed and the load on a large percentage of the slave nodes is exceptionally high.

**Recommended resolution** There are a number of operations that can limit the effects of this issue.

One of them is starting a larger number of threads, which is dependent on the cluster size. The current 10 threads might not be sufficient for a 10.000 node cluster. The optimal number of threads is hard to determine at first sight. Further research needs to be made in order to determine the optimal value. Other possible solutions for this problem can be found in section 8.

In addition, to mitigate the delay in scheduling the cluster management daemon on the slave nodes, the `cmd` process could be made to run with a higher priority on the slave nodes. This would assure that the voting operation takes less time. The Unix `renice` command is capable of altering the priority of a process.

**Considerations** Increasing the cmd's process priority on the slave nodes might affect the raw computing performance. The `nice` value needs to be set to a reasonable setting. The high process priority is useful only in failover scenarios when the voting process is used.

### 6.3.5 Service Monitoring

**Context** *Bright Cluster Manager* monitors the essential services on which the functionality of the cluster is dependent upon. If a service stops or crashes, it is automatically restarted and the event is logged. The event database can be viewed either from the GUI or from the Cluster Manager shell. This mechanism is designed to deal with non-deterministic service errors (see section 3).

**Problem** *Bright Cluster Manager* does not perform any other additional actions when a service fails. If a service fails because of a permanent error, restarting it will not bring the system to a working state.

**Impact** Failing critical services would bring the cluster in an unusable state.

**Recommended resolution** *Bright Cluster Manager* should implement a mechanism that controls the cluster's behavior on unrecoverable service failure. If a certain fail threshold is hit, then the *Bright Cluster Manager* should trigger a graceful failover.

After analyzing the current services that *Bright Cluster Manager* monitors, we suggest that the `sge`, `nfs` and `cmdaemon` services should trigger a failover, as they are essential for master node operation.

The `ntpd, mysql, ssh, named, dhcp` and `posftix` services are not critical and the current behavior of restarting them upon failiure is sufficient. The system administrator should be notified if one of the services fails, but further action is not mandatory. This feature would allow the system adminstrators to cluster other services that would benefit from high availibility.

**Considerations** If a second service failure is encountered after the failover and the fail count is hit, *Bright Cluster Manager* should not perform a second failover . This would put the cluster in a failover loop. The number of failover events might increase.

### 6.3.6 NTP Configuration

**Context** The master nodes run a NTP service which provides accurate global time for the cluster. The slave nodes use the masters as reliable time sources.[28]

**Problem** The ntpd configuration files on both masters include only external servers as time sources. If the cluster has no Internet connectivity, the master nodes are subject to time differences as they are unable to synchronize their time.

**Impact** Service log correlation on the active and passive master is made difficult if not impossible. Clustered services that require the same time on both master nodes are affected.

**Recommended resolution** The ntpd configuration on each master node should include as a last resort time source, the other master node. In this way, even though the time is no longer accurate with the real time, the nodes are not affected.

**Considerations** No complexity will be added to the design of *Bright Cluster Manager* .

### 6.3.7 Failover Toggle switch

**Context** The *Bright Cluster Manager* failover mechanism is triggered whenever a critical failure is detected.

**Problem** There might be moments when even though a failure was induced, a failover is not desired. Such situations can include maintenance operations where services are reconfigured or network topology changes trigger temporary outages.

**Impact** The serviceability of the cluster system is impaired.

**Recommended resolution** *Bright Cluster Manager* should include an option to temporarily disable the failover mechanism. This can simply be a on/off button in the GUI or a command in cmsh.

**Considerations** The temporary disabled state of the failover mechanism should be clearly highlighted in the GUI and logs. Misconfiguring this option would seriously affect the cluster in the event of failures.

# 7 Other Cluster Software

Besides *Bright Cluster Manager* , there are several other cluster managers. Basically what all cluster managers have in common is that their goal is to keep cluster management as simple as possible.

## 7.1 Red Hat Cluster Suite

**Redhat HPC** Redhat offers a bundle of RHEL and Platform Cluster Manager. The bundle is oriented towards small cluster solutions. While Redhat offers a HA clustering solution for regular Unix services, called *Redhat Cluster Manager*, there is no guideline to integrate the Platform Cluster Manager with the Redhat cluster suite.

## 7.2 Platform Cluster Manager

The Platform Cluster Manager combines various open source tools into a single cluster manager interface. This is also how the *Bright Cluster Manager* is designed [30].

| **Feature** | **BCM** | **Platform** |
|---|---|---|
| failover available | ✓ | × |
| failover configuration | active/passive | - |
| max nr. of head nodes | 2 | - |
| quorum type | majority quorum | - |
| resource fencing | STONITH - PDU | - |
| heartbeat | ping<br>SOAP<br>backupping | nagios (snmp) |

**No failover** The *Platform Cluster Manager* has no support for failover. Thereby it is impossible to see this product as complete as it is advertised. An HPC product without HA is simply not complete.

## 7.3 Pacemaker

Pacemaker is a resource manager which uses several open source libraries to glue a cluster manager together [31]. It mainly relies on:

- Heartbeat[32]
- OpenAIS or corosync[33]

| **Feature** | **BCM** | **Pacemaker** |
|---|---|---|
| failover available | ✓ | ✓ |
| failover configuration | active/passive | active/passive<br>$n+1$<br>$n$-to-$n$ |
| max nr. of head nodes | 2 | 2<br>$n$ |
| quorum type | majority quorum<br>using cluster slave nodes | majority quorum<br>using HA cluster members |
| resource fencing | STONITH - PDU | STONITH - PDU<br>STONITH - UPS<br>STONITH - Blade Control<br>STONITH - Lights-Out |
| heartbeat | ping<br>SOAP<br>backupping | *Heartbeat* |

The main difference between Pacemaker and *Bright Cluster Manager* is that the first one is more flexible. There are several options to perform a STONITH5.2 on the active head node. Such options are not yet implemented in *Bright Cluster Manager* . Thereby, one could argue that *Pacemaker* is more flexible than *Bright Cluster Manager* . An interesting feature implemented in *Pacemaker* is that it supports multiple failover configurations.

Pacemaker is not oriented towards HPC, but towards a more generic HA cluster. Regular services can be made highly available using Pacemaker. At the time this was written, there is no HPC project that includes Pacemaker.

## 7.4 PBS Pro

*PBS Pro* is a scalable cluster management solution. It is basically similar to *Bright Cluster Manager* . They both have HA in their design goals.

| **Feature** | ***Bright Cluster Manager*** | **PBS Pro** |
|---|---|---|
| failover available | ✓ | ✓ |
| failover configuration | active/passive | - |
| max nr. of head nodes | 2 | - |
| quorum type | majority quorum | - |
| resource fencing | STONITH - PDU | - |
| heartbeat | ping<br>SOAP<br>backupping | -<br><br>- |

*PBS Pro* did not provide any information about their failover mechanism. Thereby the information provided for this cluster manager is incomplete.

## 7.5 Moab Cluster Manager

| **Feature** | ***Bright Cluster Manager*** | **Moab** |
|---|---|---|
| failover available | ✓ | ✓ |
| failover configuration | active/passive | backup job service |
| max nr. of head nodes | 2 | 1 + backup job host |
| quorum type | majority quorum | - |
| resource fencing | STONITH - PDU | - |
| heartbeat | ping<br>SOAP<br>backupping | -<br><br>- |

The Moab Cluster Manager is a product of Adaptive Software. Its goal is to provide an unified solution across diverse resources and environments in a cluster. It supports most of the popular job management solutions like Sun Grid Engine, PBS, Torque, etc.

The High Availability features are not very advanced. The only HA feature is to provide a backup job management service host if the local job management service fails [37].

## 7.6 Scyld ClusterWare

| **Feature** | ***Bright Cluster Manager*** | **Scyld ClusterWare** |
|---|---|---|
| failover available | ✓ | ✓ |
| failover configuration | active/passive | active/active* |
| max nr. of head nodes | 2 | 2 or more |
| quorum type | majority quorum | - |
| resource fencing | STONITH - PDU | - |
| heartbeat | ping<br>SOAP<br>backupping | -<br><br>- |

The Scyld Clusterware management software supports a limited form of high availability using multiple master nodes. Slave nodes can be assigned to specific master nodes, but at any given moment a slave can be assigned to exactly one master node. Partitioning makes a part of the cluster still available even if one master node has failed. The partitions can be defined statically or dynamically, based on slave nodes load. An additional option enables a slave node to change its master node when it detects a failure. After a reboot, the slave can renegociate its membership in the cluster groups [38]. A serious drawback of this system is that it makes the use of all computing nodes in one job impossible.

## 7.7 Comparison Results

What all cluster managers have in common is that they tend to present their solution as perfect. A lot of all the remarks vendors make about their products are rather marketing slogans than technical facts. All the products have in common the fact that they are running on a Linux distribution and they make heavy use of open standards. They are basically using a common library for the following tasks:

- monitoring (snmp, ssh, nagios, custom)[36]
- message passing (MPI)[34]
- job management (SGE,PBS) [35]

Most of the software companies were not really open about their failover mechanism. Even *Bright Cluster Manager* is quite closed about this. Luckily, the developers provided us some more information. Unfortunately, this was not the case with alternative cluster managers. We believe that a better description of the failover features would be in the customer's interest.

# 8 Proposals

This chapter will describe possible future improvements for the failover mechanism. Some designs are a lot harder to implement but the advantages could mitigate some of the limitations the current design has.

## 8.1 Overlay Network

An overlay network is an network on top of the underlying physical network and is implemented at the application layer. One of the purpose of an overlay network in a HPC cluster would be to establish group memberships between the slave nodes. This would assure that only healthy nodes are included in computing jobs and failed nodes are more easily detected and replaced. In large clusters, this could help in decreasing the quorum size and therefore the duration of the voting process would be also decreased.

A group membership relation assures that there is connectivity between all the members of the group. For HPC this is essential as jobs could be affected by poor connectivity between the nodes.

However, an overlay on top of the underlying network will also introduce some disadvantages. The software that is deployed on all the slave nodes would need additional logic to handle group memberships and a routing algorithm for communicating inside the overlay network. This would increase the complexity of the software and the computing performance of the HPC cluster might be affected by the induced overhead.

## 8.2 Large Quorums

We believe that the current quorum system is not optimal for large clusters, as the time needed by a master node to achieve a simple majority during a quorum procedure might be too big. We propose two methods for improving the quorum time in a large cluster setup.

In the current quorum procedure, the passive master (the master node that wants to take over) sends unicast HTTP RPC calls that signal a voting request to all the active slave nodes. The slave node checks if the current active master is reachable and if not, it gives its vote to the passive master. The passive master takes over when a majority of $n/2+1$ of votes is reached (where $n$ is the number of active slave nodes).

The HTTP RPC call makes the quorum procedure sequential, as the nature of HTTP calls is sequential. The current version of *Bright Cluster Manager* starts a variable number of threads that perform the RPC calls. The parallelism of this approach is limited by the number of physical CPUs installed on the master nodes. The complexity of this algorithm is $\frac{O(n)}{no.ofthreads} = O(n)$ ,as all slave nodes must be queried.

### 8.2.1 Resource Groups

We consider organizing the cluster as a tree with height 3. The number of leaves of the tree is dependent on the size of the cluster. As in-field testing needs to be done to determine the optimal values, we will present the general concept without giving actual numeric values.

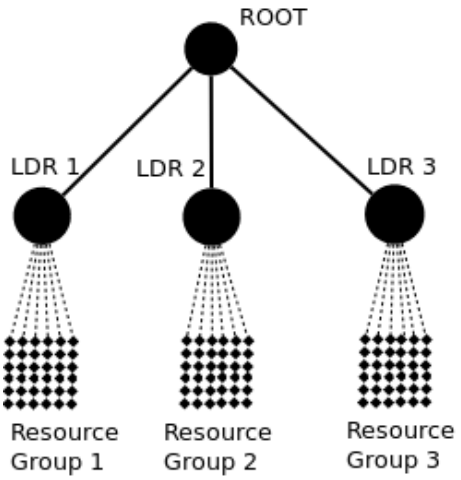


Figure 8: Slave nodes partitioned in resource groups via a tree

The tree structure is described below:

1. root level: members: the master nodes, the active master node is the actual tree root
2. leader level: members: $n/q$ leader nodes, where $q$ is the desired number of groups
3. slave level (members: $n$ amount of slave nodes)

The head nodes have the same functionality as they have now in the current *Bright Cluster Manager* failover setup.

The *leader nodes* are statically defined from either the pool of slave nodes or from other nodes that are member of the cluster (provisioning nodes, login nodes, etc). Besides the regular tasks that are already defined, the leader nodes will perform a number of additional tasks. The leader nodes will monitor a number of slave nodes assigned to them. We define a *"resource group"* as all the slave nodes that are assigned to a particular leader node. The leader node will periodically send the monitor data to the active master node and, if defined, to a node that has the monitoring role. The master nodes will keep track of the active slave nodes using the data sent in by the leader nodes.

If a slave node can not communicate with its assigned leader, it will report monitoring data directly to the master node until the assigned leader will be brought back online.

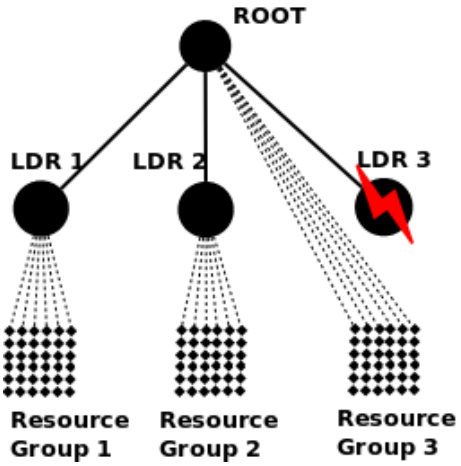


Figure 9: Slave nodes report to the master node

The master nodes will use the list of active nodes to start a quorum in the event of a failure.

The new quorum procedure will run as follows:

1. Passive Master sends leaders a command to start the quorum
2. Leaders start quorum inside group
3. Slaves send vote to master nodes
4. Master node waits a small amount of time to get a majority
5. Master node continues regular quorum procedure on slave nodes that didn't send votes in yet

Our proposal offloads the quorum start to the leader nodes. The current parallelism implemented in the quorum mechanism, based on threads, is extended to multiple hosts who can also start threads to further speed up the process. Step 5 in the algorithm is required to make sure that all the active nodes have the opportunity to vote. Some leader nodes might be unreachable and therefore unable to relay the voting message.

Although the complexity of the algorithm is still $O(n)$, we believe that the performance gain in a real-world setup is significant. As the cluster size increases, additional leader nodes can be added to the tree in order to keep the quorum time as small as possible.

Our proposal is in sync with the current trend of decentralizing cluster management tasks and offloading them to dedicated nodes.

### 8.2.2 Optimized Quorum Sequence

The current quorum sequence gives no priority of low load nodes over high load nodes. An overloaded node could delay the voting response and the thread that issued the voting request will be blocked until a timeout is reached or a vote is obtained.

Our proposal is to create a dynamic list of all the active nodes. We call this list "quorum sequence list". It should be sorted by slave node load for the past 5 minutes. This metric is collected periodically from the slave nodes by the master node. There is no need to poll this information very often. We take into consideration that any monitoring process affects the slave nodes computing performance.

When a quorum is initiated, the master node will start querying the slave nodes in the order specified by the quorum sequence list. Asking idle nodes first helps in obtaining the majority of votes faster. Even if the actual load on some node changes, the number of occurrences of blocking states will be far smaller than in a non-optimized quorum. If the load on all the nodes dramatically changes, then the quorum procedure falls back to the current implementation which does not take into account priorities for nodes.

This proposal is a refinement of the current quorum algorithm and in busy clusters may improve the system response time in the event of failure. We believe that the added complexity is acceptable as its foundation is already implemented in *Bright Cluster Manager* .

## 8.3 Active/Active setup

An *active/active* setup can help overloaded master nodes and it can increase the efficiency of the cluster. We have presented the basics of this type of setup in 4.1.2. The most simple solution of an *active/active* setup implies that the cluster is partitioned and that each master node controls a number of the cluster partitions. In the event of failure of one of the master nodes, the control of the cluster partitions is migrated to the remaining operational master node.

Switching the cluster partitions to a new master requires both changes on the network level (IP & MAC addresses) and on application level, as the job management system must also include the migrated cluster partitions. Supporting more than one job management system would increase the overall complexity of the system.

Another solution for an *active/active* setup implies that services run on both master nodes and they can accept requests and serve content at the same time. Essentially the nodes function as one system. To achieve this, the nodes need to communicate and synchronize data in order to have the same state at any given moment. The main advantage of this approach is that in the event of failure the failover time is nearly 0 and transparent to the users. Also, more than two master nodes can be used, thus achieving load balancing and better redundancy.

The complexity of this type of setup is higher than the traditional approach

of having an active/passive pair. To implement it, the job scheduler needs to be modified. Additional delay and complexity is introduced by the group membership communication protocols employed. In [20], a description of such a system is given. It makes use of a wrapper around the clustered services that ensures that all the nodes in the cluster receive the same information. Additionally, a mutual exclusion manager controls which node performs which operation. The mutual exclusion is needed as replicated events should not be processed by all nodes. To the best of our knowledge, this type of implementation is not present in any of the current state-of-the-art cluster management products on the market.

## 8.4 Automated Failover Test Scripts

During our research, we have found a number of 3 bugs in *Bright Cluster Manager* . The appendix of this document describes the test cases that led to the discovery of the bugs. All of the bugs were triggered by simple actions that would have been made in an usual failover situation. We believe that the failover mechanism should be often tested against common failure scenarios. An automated test suite in a virtual environment would improve the overall quality of the product.

## 8.5 Amazon EC2

Due to the limited time frame, we didn't investigate throughly the option of creating a cluster managed by *Bright Cluster Manager* in the Amazon EC2 cloud environment. Yet, we have identified possible starting points regarding the high availability of such a system.

Amazon offers the option of creating instances in so called "availability zones", which in fact are different physical data centers that might not be subject to simultaneous outages. Placing instances in different availability zones is a good decision for creating a highly available system in the Amazon cloud.

*Bright Cluster Manager* uses a shared IP which is migrated from the active master to the passive master during failover6.1.4. While the IP address of the instances in the Amazon cloud can not be chosen by the user, Amazon offers an option called "Elastic IP", which allows the user to bind the "Elastic IP" to any instance at any given moment. By using the Amazon API, the Elastic IP can be migrated during failover from the active node to the passive. Amazon states that the propagation time of the Elastic IP is within the range of a few minutes.

The performance of a cloud hosted HPC cluster is yet unknown, as the latency and bandwidth inside the cloud is not guaranteed by Amazon [39]. A normal HPC cluster uses high performance networks (like Infiniband) to provide communication between slave nodes. Having this in mind, we believe that offloading to the Amazon cloud is efficient only for batch jobs that require minimal communications with other slave nodes.

## 8.6 Optimized communication

The current communication model in *Bright Cluster Manager* is based on point to point communication (TCP). Scaling this model introduces challenges especially for identical messages that need to be sent to multiple nodes. A broadcast communication model would increase the efficiency. Yet, if each sent message also needs a reply, this might overload the receiver. A backoff algorithm (similar to the one in the Ethernet collision detection) would mitigate this issue. One other major disadvantage of a broadcast model is that the system is no longer synchronous, this meaning that the master node is no longer in total control. This decoupling of nodes might have serious administrative impact and in addition it adds complexity as states need to be maintained for each slave node. We do not see this as a real immediate alternative for the current communication model, yet it is a good future research direction.

# 9 Conclusions

During our research we found some issues in *Bright Cluster Manager* . In our opinion these issues need to be addressed in order to achieve a more efficient cluster manager. Additional to possible solutions for mitigation the issues in *Bright Cluster Manager* , we have provided some ideas in order to make the current failover mechanism more efficient.

**Logical Design** In our opinion, the current logical design of the failover mechanism is not efficient in large clusters. Although *Bright Computing Inc.* has not yet installed clusters with more than 1.000 nodes, there is no doubt that this will happen in the near future. We think the proposals we have presented could mitigate some issues which will occur when *Bright Cluster Manager* will be installed on top of a cluster with more than 1.000 nodes. Thereby, these proposals should be taken into consideration as one of the goals of *Bright Cluster Manager* is to scale to 10.000+ computing nodes.

**Monitoring** An issue we have addressed multiple times in this paper is how monitoring is currently implemented. We think monitoring can be improved a lot between the head nodes. If both head nodes have more information about their current health, a failover decision can be more accurate, thus improving high availability.

**Focus on Hardware** Our research shows that the current failover implementation of *Bright Cluster Manager* is more oriented towards hardware failure rather than software failure. The current mitigations against software failure are not complete and should be improved to assure a high level of availability. In terms of hardware failure, the current version performs very well in the most common cases. We have identified additional possible single-points-of-failure that were not taken into consideration.

**Complexity** Software developers tend to keep their software as simple as possible. The main reason is that higher complexity induces more issues. The proposals we have made introduce some complexity, but in our opinion the advantages of some of them weight more than the added complexity.

**Terminology** The terminology used in the configuration of the *Bright Cluster Manager* needs to be more clear. A better distinction between the heartbeat network and slave network is necessary, as we believe that the current one could confuse future customers of *Bright Cluster Manager* . The critical components that trigger a failover should also be clearly outlined. Therefore, the used terminology is not concise.

**Software Testing** During our research of the failover mechanism, we have found and reported a number of 3 software bugs in *Bright Cluster Manager* . All the bugs were easy to reproduce. This shows that the software testing process could be improved. We recommend that an automated suite containing the common failure tests should be used in the development process of *Bright Cluster Manager* .

## 9.1 Future Research

Our four week research raised new ideas and is leading to other future research directions.

**Distributed Cluster Manager** In the current logical design, there is a distinction between the head nodes and computing nodes. A failure of a head node is a bigger problem than the failure of a computing node. To mitigate the failure of a head node, which renders the cluster unusable, a standby head node is configured to take over when the primary head node fails. It would be better if any node could take over from the current head node. Thereby, a homogeneous design is preferred, not only on hardware level, but also on software level. A problem in this design, which needs to be researched, is that the computing nodes will get more tasks, thus affecting the available computing performance.

**Amazon** Sometimes, an user of a cluster needs more computing power for a short amount of time. The option to add more nodes from, for example, a public cloud like Amazon, could result in some extra computing power at very low costs. There would be no need to buy more physical equipment. Although this method seems plausible, there are some limitations. Slow networks and high latency might negatively affect the performance of a cluster. Future research should determine the most efficient approach.

**Geographic Scalability** An interesting feature would be the option to distribute the cluster over multiple locations. In our opinion, this is not really feasible in the current logical design. Problems that need to be addressed are how to place the head nodes in different locations and what are the implications if the slave nodes would be in different places which may be thousands of kilometers away. The advantage of a geographically distributed computing cluster is that it can mitigate disasters.

# 10 Acknowledgements

Our research could not have been done without the help of *Bright Computing Inc.*. Thereby, we want to thank our supervisor Martijn de Vries and lead developer Koen de Raedt. Because we did not do our research at *Bright Computing Inc.* but in the OS3 Lab, we heavily relied on email communication on which they replied always fast. There was always time for discussion about certain issues and proposals which led to better results.

# References

[1] Beowulf (computing) http://en.wikipedia.org/wiki/Beowulf_%28computing%29 Retrieved on 25th of January 2010

[2] E. Marcus & H. Stern, *Blueprints for High Availability*, Wiley Publishing, Inc., 2nd edition, 2003.

[3] Red Hat, Inc. *Cluster Suite for Red Hat Enterprise Linux 4.5* http://www.centos.org/docs/4/4.5/SAC_Cluster_Suite_Overview/index.html Retrieved on 25th of January 2010

[4] *Platform Cluster Manager* http://www.platform.com/ Retrieved on 25th of January 2010

[5] Bianca Schroeder & Garth A. Gibson, *A large-scale study of failures in high-performance computing systems*, Proceedings of the International Conference on Dependable Systems and Networks (DSN2006),Philadelphia, PA, USA, June 25-28, 2006.

[6] *Red Hat Cluster* http://sources.redhat.com/cluster/wiki/FAQ/Fencing Retrieved on 25th of January 2010

[7] *ping send ICMP ECHO_REQUEST packets to network hosts* http://fgouget.free.fr/bing/ping_src-man.shtml Retrieved on 25th of January 2010

[8] *Simple Network Management Protocol (SNMP)* http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol Retrieved on 25th of January 2010

[9] *Intelligent Platform Management Interface* http://en.wikipedia.org/wiki/Ipmi Retrieved on 25th of January 2010

[10] *IP multicast* http://en.wikipedia.org/wiki/IP_multicast Retrieved on 25th of January 2010

[11] *IP broadcast* http://en.wikipedia.org/wiki/Broadcast_ip_address Retrieved on 25th of January 2010

[12] *IP unicast* http://en.wikipedia.org/wiki/Unicast Retrieved on 25th of January 2010

[13] *Serial line*
`http://en.wikipedia.org/wiki/Serial_line`
Retrieved on 25th of January 2010

[14] *The Goals of Bright Cluster Manager*
`http://www.brightcomputing.com/Bright-Cluster-Manager.php`
Retrieved on 25th of January 2010

[15] *Grid Computing Info Centre*
`http://www.gridcomputing.com/gridfaq.html`
Retrieved on 26th of January 2010

[16] *Linux HPC Cluster Installation*
`http://www.redbooks.ibm.com/abstracts/SG246041.html`
IBM Corporation, 2009

[17] Bianca Schroeder & Garth A. Gibson, *Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?* , FAST'07: 5th USENIX Conference on File and Storage Technologies, San Jose, CA, Feb. 14-16, 2007.

[18] James E.J. Bottomley *Implementing Clusters for High Availability*, Proceedings of the FREENIX Track: 2004 USENIX Annual Technical Conference

[19] Bill Highleyman, Paul J. Holenstein, Bruce Holenstein *Breaking the Availability Barrier II: Achieving Century Uptimes with Active/Active Systems* AuthorHouse, 2007

[20] Christian Engelmann, Stephen L. Scott, Chokchai (Box) Leangsuksun *Symmetric Active/Active High Availability for High-Performance Computing System Services*, Journal Of Computers, VOL. 1, NO. 8, December 2006

[21] Bianca Schroeder & Garth A. Gibson, *The Computer Failure Data Repository (CFDR)* , Workshop on Reliability Analysis of System Failure Data (RAF'07) MSR Cambridge, UK, March 2007

[22] Bianca Schroeder, Wolf-Dietrich Weber ,Eduardo Pinheiro *DRAM Errors in the Wild: A Large-Scale Field Study*, Workshop on Reliability Analysis of System Failure Data (RAF'07) MSR Cambridge, UK, March 2007

[23] *Managing Computers with Automation*
`http://techthoughts.typepad.com/managing_computers/2007/10/split-brain-quo.html`
Retrieved on 21st of January 2010

[24] *Differences between active-active and active-passive*
`http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.ha_avail/html/ha_avail/ha_avail3.htm`
Retrieved on 21st of January 2010

[25] *Shoot The Other Node In The Head (STONITH)*
http://linux-ha.org/STONITH
Retrieved on 19th of January 2010

[26] *Understanding How Cluster Quorums Work*
http://www.windowsnetworking.com/articles_tutorials/Cluster-Quorums.html
Retrieved on 19th of January 2010

[27] *Non-RAID Hot Swapping Help Needed*
http://blog.nixternal.com/2008.07.10/non-raid-hot-swapping-help-needed/
Retrieved on 21st of January 2010

[28] *Network Time Protocol*
http://en.wikipedia.org/wiki/Ntpd
Retrieved 17th of January 2010

[29] *Red Hat Cluster Suite*
http://www.redhat.com/cluster_suite/
Retrieved 22nd of January 2010

[30] *Platform Cluster Manager*
http://www.platform.com/cluster-computing/cluster-management
Retrieved 22nd of January 2010

[31] *Pacemaker* http://www.clusterlabs.org/wiki/Main_Page
Retrieved 22nd of January 2010

[32] *Linux-HA* http://www.linux-ha.org/wiki/Main_Page
Retrieved 22nd of January 2010

[33] *OpenAIS Standards Based Cluster Framework*
http://openais.org/doku.php
Retrieved 22nd of January 2010

[34] *Message Passing Interface*
http://en.wikipedia.org/wiki/Message_Passing_Interface
Retrieved 22nd of January

[35] *Sun Grid Engine*
http://www.sun.com/software/sge/
Retrieved 22nd of January

[36] *20 Linux System Monitoring Tools Every SysAdmin Should Know*
http://www.cyberciti.biz/tips/top-linux-monitoring-tools.html
Retrieved 22nd of January

[37] *Moab Cluster Manager, version 5.3, Users Guide*
http://www.clusterresources.com/products/mcm/docs/mcmuserguide.pdf

[38] *Scyld ClusterWare HPC, Administrators Guide, Published July 18, 2008* `ftp://ftp.penguincomputing.com/pub/staff/rconnoy/Scyld/docs/4.2.1/administration-guide.pdf`

[39] Alexandru-Dorin Giurgiu, Research Project 1 OS3 SNE *Network Performance in Virtual Infrastructures: A closer look at Amazon EC2*, February 2010

# A Test Cases

## A.1 Hardware

The following tests are done in a situation where master1 is the active master. The status of the cmdaemon is in its default failover state:

```
Failover status:
master1* -> master2
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
master2 -> master1*
  mysql        [  OK  ]
  ping         [  OK  ]
  status       [  OK  ]
```

The following table shows clearly which test cases did not behave as expected:

| action | rslt |
|---|---|
| Power off Master1 | ✓ |
| Power off Master2 | ✓ |
| Disconnect network cable Master1 (slave network) | ✓ |
| Disconnect network cable Master2 (slave network) | ✓ |
| Disconnect network cable Master1 (external network) | ✓ |
| Disconnect network cable Master2 (external network) | ✓ |

### A.1.1 Power off Master1 ✓

*Expected result:* failover is initiated
*Actual result:* failover is initiated

Passive master log:

```
CMDaemon: Info: Failover: no failover was initiatednode slow
CMDaemon: Info: Start quorum
CMDaemon: Info: Started quorum on: https://10.141.0.3:8081
CMDaemon: Info: Started quorum on: https://10.141.0.4:8081
CMDaemon: Info: Started quorum on: https://10.141.0.2:8081
CMDaemon: Info: Node 38654705670 wants me to be master (1 of 3)
CMDaemon: Info: Node 38654705671 wants me to be master (2 of 3)
CMDaemon: Info: Quorum condition reached (2 of 3)
CMDaemon: Info: Quorum decided, taking over
CMDaemon: Info: Failover: power off master1 using custom
CMDaemon: Info: Starting shared storage
CMDaemon: Info: Started shared storage
CMDaemon: Info: Stopping IP
```

```
CMDaemon: Info: Starting IP
CMDaemon: Info: Initialize monitoring database
CMDaemon: Info: Set cmdaemon state: ACTIVE
CMDaemon: Info: I am now master: 1263220570
```

The drbd filesystem gets mounted properly:

```
(Mon Jan 11 15:36:05 CET 2010) DRBD Mount: master2 becoming active
(Mon Jan 11 15:36:06 CET 2010) Setting /dev/drbd0 primary (1)
(Mon Jan 11 15:36:06 CET 2010) Mount /dev/drbd0 /cm/shared (1)
(Mon Jan 11 15:36:07 CET 2010) Setting /dev/drbd1 primary (1)
(Mon Jan 11 15:36:07 CET 2010) Mount /dev/drbd1 /cm/node-installer/certificates (1)
(Mon Jan 11 15:36:08 CET 2010) Setting /dev/drbd2 primary (1)
(Mon Jan 11 15:36:08 CET 2010) Mount /dev/drbd2 /home (1)
```

CMDaemon status:

```
[root@master2 ~]# cmha status
Node Status: running in active master mode

Failover status:
master2* -> master1
  mysql        [  OK  ]
  ping         [FAILED] (40)
  status       [FAILED] (46)
master1 -> [ERROR] Soap error(105)
```

The state of the cluster:

```
[root@master2 ~]# cat /var/spool/cmd/state
ACTIVE
```

### A.1.2 Power off Master2 ✓

*Expected result:* no failover is initiated
*Actual result:* no failover is initiated

Passive Master log:

```
CMDaemon: Info: Provisioning node master2 down, canceling running provisioners.
CMDaemon: Info: FID(4): 38654705667 1263221698
CMDaemon: Info: Failover node: master1 (primary) (active)
CMDaemon: Info: FID(4): 38654705672 1263220570
CMDaemon: Info: Failover node: master2 (passive)
CMDaemon: Info: FID(2): 38654705667 1263221698 1263221698
CMDaemon: Info: FID(2): 38654705672 1263220570 1263220570
CMDaemon: Info: Return code from failover update: 1
CMDaemon: Info: Self configuration updated
CMDaemon: Info: Failover: node slow
CMDaemon: Info: Initializing queue for provisioning node master1 with 10 slots,
```

```
providing all images.
CMDaemon: Info: Initializing queue for provisioning node master2 with 10 slots,
providing all images.
CMDaemon: Info: Successfully reconfigured IPMI interface(s)
CMDaemon: Info: DHCP performing update
CMDaemon: Info: Named performing update
CMDaemon: Info: Postfix performing update
CMDaemon: Info: LDAP performing update
CMDaemon: Info: Updating SGE configuration
CMDaemon: Info: Updating image default-image on provisioning node master2.
CMDaemon: Fatal: Updating image default-image on provisioning node master2 failed:
Rsync exit code 255, signal 0.
```

The drbd daemon also as expected registeres the loss of the peer.

```
master1 kernel: block drbd2: PingAck did not arrive in time.
master1 kernel: block drbd2: peer( Secondary -> Unknown ) conn( Connected ->
NetworkFailure ) pdsk( UpToDate -> DUnknown )
master1 kernel: block drbd2: asender terminated
master1 kernel: block drbd2: Terminating asender thread
master1 kernel: block drbd2: short read expecting header on sock: r=-512
master1 kernel: block drbd2: Creating new current UUID
master1 kernel: block drbd2: Connection closed
master1 kernel: block drbd2: conn( NetworkFailure -> Unconnected )
master1 kernel: block drbd2: receiver terminated
master1 kernel: block drbd2: Restarting receiver thread
master1 kernel: block drbd2: receiver (re)started
master1 kernel: block drbd2: conn( Unconnected -> WFConnection )
```

Because

```
Node Status: running in active master mode

Failover status:
master1* -> master2
  mysql        [  OK  ]
  ping         [FAILED] (47)
  status       [FAILED] (44)
master2 -> [ERROR] Soap error(105)
```

The state of the cluster:

```
[root@master1 ~]# cat /var/spool/cmd/state
ACTIVE
```

### A.1.3 Disconnect network cable master1 (slave network) ✓

*Expected result:* failover is initiated
*Actual result:* failover is initiated

Active master log:

```
CMDaemon: Info: Failover: node slow
CMDaemon: Info: Start quorum
CMDaemon: Info: Waiting 60 seconds for quorum to be decided
CMDaemon: Info: Started quorum on: https://10.141.0.1:8081
CMDaemon: Info: Started quorum on: https://10.141.0.3:8081
CMDaemon: Info: Started quorum on: https://10.141.0.4:8081
CMDaemon: Info: Started quorum on: https://10.141.0.2:8081
CMDaemon: Info: Quorum started
CMDaemon: Info: Quorum decided, index = 0
CMDaemon: Info: Node 38654705668 wants me to be master (1 of 4)
CMDaemon: Info: Quorum decided, index = 1
CMDaemon: Info: Node 38654705669 wants me to be master (2 of 4)
CMDaemon: Info: Quorum decided, index = 2
CMDaemon: Info: Node 38654705670 wants me to be master (3 of 4)
CMDaemon: Info: Quorum condition reached (3 of 4)
CMDaemon: Info: decideQuorumRunning: 2
CMDaemon: Info: Quorum decided, taking over
CMDaemon: Info: Failover: power off master1 using custom
CMDaemon: Info: Quorum decided, index = 3
CMDaemon: Info: Quorum already running
CMDaemon: Info: Failover: setting master
CMDaemon: Info: Quorum already running
last message repeated 3 times
CMDaemon: Info: Failover: Unable to inform https://10.141.255.254:8081
of becoming master, ierr = 105
CMDaemon: Info: Starting shared storage
CMDaemon: Info: Quorum already running
CMDaemon: Info: Started shared storage
CMDaemon: Info: Stopping IP
CMDaemon: Info: Starting IP
CMDaemon: Info: Initialize monitoring database
CMDaemon: Info: Set cmdaemon state: ACTIVE
CMDaemon: Info: I am now master: 1263224807
```

As the "cmha status" shows the two nodes cannot communicate with each other. This means there is at this moment no failover is possible but it also means that the cluster is still accessible by its end users.

```
[root@master2 ~]# cmha status
Node Status: running in active master mode

Failover status:
master2* -> master1
  mysql        [  OK  ]
  ping         [FAILED] (63)
  status       [FAILED] (72)
master1 -> [ERROR] Soap error(105)
```

Master2 changed its state to active:

```
[root@master2 ~]# cat /var/spool/cmd/state
ACTIVE
```

### A.1.4 Disconnect network cable master2 (slave network) ✓

*Expected result:* no failover is initiated
*Actual result:* no failover is initiated

Passive master log:

```
CMDaemon: Info: Failover: node slow
CMDaemon: Info: Initializing queue for provisioning node master1 with 10 slots,
providing all images.
CMDaemon: Info: Initializing queue for provisioning node master2 with 10 slots,
providing all images.
CMDaemon: Info: Successful remove local hard disk Master 1 & \rno \\
 remove local hard disk Master 2 & \gcheck \\
 high load on Master1 & ? \\
 high load on Master2 & ? \\ly reconfigured IPMI interface(s)
```

CMDaemon status:

```
Node Status: running in active master mode

Failover status:
master1* -> master2
  mysql        [  OK  ]
  ping         [FAILED] (10)
  status       [FAILED] (12)
master2 -> [ERROR] Soap error(105)
```

CMDaemon state:

```
[root@master1 ~]# cat /var/spool/cmd/state
ACTIVE
```

### A.1.5 Disconnect network cable master1 (external network) ✓

*Expected result:* no failover is initiated
*Actual result:* no failover is initiated

On a "ifconfig down" on the external network nothing actually fails which is related to the failover mechanism. The problem is that the ntpd service cannot reach its servers anymore which can result in a time difference between the active and passive master node. There is by default no time synchronisation between the active master and the passive master.

Active master log:

```
ntpd[3051]: sendto(213.206.85.20) (fd=22): Network is unreachable
ntpd[3051]: sendto(213.239.154.12) (fd=22): Network is unreachable
ntpd[3051]: sendto(145.24.129.6) (fd=22): Network is unreachable
ntpd[3051]: sendto(131.211.84.189) (fd=22): Network is unreachable
```

### A.1.6 Disconnect network cable master2 (external network) ✓

*Expected result:* no failover is initiated
*Actual result:* no failover is initiated

When the network cable is unplugged on master2 ntpd cannot reach its timeserver thereby it is unable to update the local time. This could result into a problem when the time difference gets to big between the active master, the secondary master and the nodes.

We adivse that the ntpd configuration should include the other master server as a time source. The priority should be lower than the one of the regular time sources. In the event that the external network fails, and thus the external time sources can not be reached, the global time of the cluster nodes would still be consistent.

## A.2 Software

The following tests are done in a situation where master1 is the active master.

| action | rslt |
|---|---|
| kill CMDaemon on Master1 | × |
| kill CMDaemon on Master2 | × |
| kill MySQL on Master1 | × |
| kill MySQL on Master2 | × |
| kill NFS on Master1 | ✓ |
| kill NFS on Master2 | ✓ |
| kill NTPD on Master1 | ✓ |
| kill NTPD on Master2 | ✓ |
| kill LDAP on Master1 | × |
| kill LDAP on Master2 | × |
| kill NAMED on Master1 | ✓ |
| kill NAMED on Master2 | ✓ |
| kill DHCPD on Master1 | ✓ |
| kill DHCPD on Master2 | ✓ |
| remove local hard disk Master 1 | × |
| remove local hard disk Master 2 | ✓ |
| high load on Master1 | × |
| high load on Master2 | ✓ |

### A.2.1 kill CMDaemon on Master1 ×

*Expected result:* cmdaemon should be started (assumption)
*Actual result:* cmdaemon does not start

Because the cmdaemon does not restart a failover should be initiated. When the cmdaemon is not retarted or a failover is initiated the cluster will become

unavailable for the endusers.

Active master log:

```
CMDaemon: Info: Received SIGTERM. Shutting down.
CMDaemon: Info: Provisioning node master1 down, canceling running provisioners.
CMDaemon: Info: Provisioning node master2 down, canceling running provisioners.
CMDaemon: Info: Disconnect from database: cmdaemon
CMDaemon: Info: Disconnect from database: cmdaemon_mon
CMDaemon: Info: Disconnect from database: cmdaemon
```

Passive master log:

```
CMDaemon: Debug: Failover status, soap error: 105 [https://10.141.255.254:8081]
count = 56
CMDaemon: Debug: Failover: check if node is completely dead
CMDaemon: Debug: Not completely dead: ping failcount = 0
CMDaemon: Info: Master not completely dead, no quorum started
```

### A.2.2 kill CMDaemon on Master2 ×

*Expected result:* cmdaemon should be started (assumption)
*Actual result:* cmdaemon does not start

Active master log:

```
CMDaemon: Info: Failover: node slow
```

Passive master log:

```
CMDaemon: Info: Received SIGTERM. Shutting down.
CMDaemon: Info: Disconnect from database: cmdaemon
```

### A.2.3 kill MySQL on Master1 ×

*Expected result:* MySQL should be started
Actual result: MySQL does not restart
no failover was initiated

Active master log:

```
CMDaemon: Info: Unable to connect to monitoring MySql database:
ConnectionFailed, will try again (2) ...
```

Passive master log:

```
CMDaemon: Info: Starting mysql replication
CMDaemon: Info: Starting mysql replication
```

### A.2.4 kill MySQL on Master2 ×

*Expected result:* MySQL should be started
*Actual result:* When MySQL is killed on Master2 the cmdaemon on Master2 crashes.

```
CMDaemon: Debug: Reconnect to database
CMDaemon: Debug: Stop CMDaemonFailoverController
CMDaemon: Debug: End CMDaemonFailoverMysqlController::run 0
CMDaemon: Debug: Stop CMDaemonFailoverController
CMDaemon: Debug: End CMDaemonFailoverPingController::run
CMDaemon: Debug: Stop CMDaemonFailoverController
CMDaemon: Debug: End CMDaemonFailoverStatusController::run
CMDaemon: Debug: SysStateCollector: done
```

### A.2.5 kill NFS on Master1 ✓

*Expected result:* nfsd should be started
*Actual result:* nfsd starts

Active master log:

```
CMDaemon: Fatal: /etc/init.d/nfs status, exitcode = 3, signal = 0
CMDaemon: Info: Service nfs is not running, attempting to restart.
CMDaemon: Debug: Starting service nfs
CMDaemon: Debug: ProgramRunner: /etc/init.d/nfs start
CMDaemon: Fatal: /etc/init.d/nfs start, exitcode = 1, signal = 0
```

### A.2.6 kill NFS on Master2 ✓

*Expected result:* nfsd should be started
*Actual result:* nfsd starts

Passive master log:

```
CMDaemon: Fatal: /etc/init.d/nfs status, exitcode = 3, signal = 0
CMDaemon: Info: Service nfs is not running, attempting to restart.
CMDaemon: Fatal: /etc/init.d/nfs status, exitcode = 3, signal = 0
CMDaemon: Fatal: /etc/init.d/nfs start, exitcode = 1, signal = 0
```

### A.2.7 kill NTPD on Master1 ✓

*Expected result:* ntpd should be started
*Actual result:* ntpd starts

Active master log:

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/ntpd status
CMDaemon: Fatal: /etc/init.d/ntpd status, exitcode = 3, signal = 0
```

```
CMDaemon: Info: Service ntpd is not running, attempting to restart.
CMDaemon: Debug: Add event to database: 1246
CMDaemon: Debug: ProgramRunner: /etc/init.d/nfs status
CMDaemon: Debug: ProgramRunner: /etc/init.d/ntpd status
CMDaemon: Fatal: /etc/init.d/ntpd status, exitcode = 3, signal = 0
CMDaemon: Debug: Starting service ntpd
CMDaemon: Debug: ProgramRunner: /etc/init.d/ntpd start
```

### A.2.8 kill NTPD on Master2 ✓

*Expected result:* ntpd should be started
*Actual result:* ntpd starts

Passive master log:

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/ntpd status
CMDaemon: Fatal: /etc/init.d/ntpd status, exitcode = 3, signal = 0
CMDaemon: Info: Service ntpd is not running, attempting to restart.
CMDaemon: Debug: ProgramRunner: /etc/init.d/ntpd status
CMDaemon: Fatal: /etc/init.d/ntpd status, exitcode = 3, signal = 0
CMDaemon: Debug: Starting service ntpd
CMDaemon: Debug: ProgramRunner: /etc/init.d/ntpd start
```

### A.2.9 kill LDAP on Master1 ×

*Expected result:* ldapd should restart
*Actual result:* ldapd does not restart but it actually checks the status and does nothing with it.

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/ldap status
```

### A.2.10 kill LDAP on Master2 ×

*Expected result:* ldapd should restart
*Actual result:* ldapd does not restart but it actually checks the status and does nothing with it.

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/ldap status
```

### A.2.11 kill NAMED on Master1 ✓

*Expected result:* named should be started
*Actual result:* named starts

Active master log:

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/named status
CMDaemon: Fatal: /etc/init.d/named status, exitcode = 3, signal = 0
CMDaemon: Info: Service named is not running, attempting to restart.
CMDaemon: Debug: ProgramRunner: /etc/init.d/named status
CMDaemon: Fatal: /etc/init.d/named status, exitcode = 3, signal = 0
CMDaemon: Debug: Starting service named
CMDaemon: Debug: ProgramRunner: /etc/init.d/named start
```

### A.2.12 kill NAMED on Master2 ✓

*Expected result:* named should be started
*Actual result:* named starts

Passive master log:

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/named status
CMDaemon: Fatal: /etc/init.d/named status, exitcode = 3, signal = 0
CMDaemon: Info: Service named is not running, attempting to restart.
CMDaemon: Debug: ProgramRunner: /etc/init.d/named status
CMDaemon: Fatal: /etc/init.d/named status, exitcode = 3, signal = 0
CMDaemon: Debug: Starting service named
CMDaemon: Debug: ProgramRunner: /etc/init.d/named start
```

### A.2.13 kill DHCPD on Master1 ✓

*Expected result:* dhcpd should be started
*Actual result:* dhcpd starts

Active master log:

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/dhcpd status
CMDaemon: Fatal: /etc/init.d/dhcpd status, exitcode = 1, signal = 0
CMDaemon: Info: Service dhcpd is not running, attempting to restart.
CMDaemon: Debug: ProgramRunner: /etc/init.d/dhcpd status
CMDaemon: Fatal: /etc/init.d/dhcpd status, exitcode = 3, signal = 0
CMDaemon: Debug: Starting service dhcpd
CMDaemon: Debug: ProgramRunner: /etc/init.d/dhcpd start
```

### A.2.14 kill DHCPD on Master2 ✓

*Expected result:* dhcpd should be started
*Actual result:* dhcpd starts

Passive master log:

```
CMDaemon: Debug: ProgramRunner: /etc/init.d/dhcpd status
CMDaemon: Fatal: /etc/init.d/dhcpd status, exitcode = 3, signal = 0
CMDaemon: Info: Service dhcpd is not running, attempting to restart.
CMDaemon: Debug: ProgramRunner: /etc/init.d/dhcpd status
CMDaemon: Fatal: /etc/init.d/dhcpd status, exitcode = 3, signal = 0
```

```
CMDaemon: Debug: Starting service dhcpd
CMDaemon: Debug: ProgramRunner: /etc/init.d/dhcpd start
```

#### A.2.15 remove local hard disk Master 1 ×

*Expected result:* failover should be initiated
*Actual result:* no failover is initiated

#### A.2.16 remove local hard disk Master 2 ✓

*Expected result:* no failover should be initiated
*Actual result:* no failover was initiated

#### A.2.17 high load on Master1 ×

*Expected result:* failover should be initiated
*Actual result:* no failover was initiated

#### A.2.18 high load on Master2 ✓

*Expected result:* no failover should be initiated
*Actual result:* no failover was initiated