

# Design Document for Weak Membership

## Colorado Phase I: Infrastructure

Sambit Nayak  
Ellard Roush  
Zoram Thanga  
Ganesh Ram Nagarajan  
Nandan Kumar  
Sun Microsystems, Inc.  
25 November 2008  
2008/1717

### Revision History

<i>Version</i>	<i>Comments</i>	<i>Date</i>	<i>Author</i>
1.0	Initial draft of CMM changes	25/11/08	Sambit Nayak
1.1	Minor changes to CMM sections	26/11/08	Ellard Roush
1.2	Some additions to CMM sections	06/01/09	Sambit Nayak
2.0	CCR section	06/01/09	Zoram Thanga
3.0	Added CLI design and User model	06/01/09	Ganesh Ram Nagarajan
4.0	Added weak membership scenarios section	06/01/09	Sambit Nayak
4.1	Review comments incorporated	09/01/09	Sambit Nayak
4.2	Review comments incorporated	09/01/09	Ganesh Ram Nandan Kumar

## 1. Introduction

Traditional behaviour of Sun Cluster software ensures that only one partition survives in a *split brain* scenario, when a cluster has been partitioned into disjoint sets of nodes (partitions) possibly due to network faults. We term this '*strong membership*' behaviour.

The Colorado Phase I project proposes a '*weak membership*' feature only for 2-node clusters with no shared storage. With this feature, when a 2-node cluster suffers a split brain, each partition (consisting of a single node each) can survive as an independent cluster. This feature does not replace *strong membership*, instead *weak membership* provides an alternative membership type for configurations that do not have sufficient resources to support *strong membership*.

## 2. *Installmode* reset

When a cluster is installed and configured for the first time, the *installmode* is set in CCR. This means that the first node of the cluster has a vote count of 1 and the other nodes have a vote count of 0. Once all nodes are in the cluster, the user resets *installmode* which will set the vote count of all nodes to their default values (1 vote for each node by default).

For a 2-node cluster, Sun Cluster software mandates that the user needs to configure a quorum device before resetting *installmode*. We keep this behaviour if the cluster is using *strong membership*.

If the user wants to use *weak membership* for a 2-node cluster, then Sun Cluster software should not necessitate the addition of a quorum device just to reset *installmode*. The user is allowed to switch the cluster to *weak membership* even when *installmode* is set. After switching the cluster to *weak membership*, the user can reset *installmode* without requiring a quorum device addition.

## 3. Changes in CMM

### 3.1 Minimum quorum votes for cluster

In a normal *weak membership* scenario, there are 2 votes - one each for a node. For the cluster to run under *weak membership*, CMM will define the required quorum votes to be 1 (out of a total of 2 votes) when running under *weak membership*. As a result, when a cluster is running under *weak membership*, CMM reconfigures fine even if only one node is up.

As explained previously, the user can switch the cluster to *weak membership* even when the cluster has *installmode* set. Since the *installmode* is not reset, the total vote count of the cluster nodes is 1 (1 for the first node). Once the cluster has been switched to *weak membership* in this situation, a CMM reconfiguration should allow the cluster to survive if the node having the sole vote is online.

To accommodate the above two facts, the minimum quorum votes required for a cluster to survive, when running under *weak membership*, will be defined as 1 if total configured vote count is 1, and as 1 if total configured vote count is 2.

### 3.2 Additional data stored in CMM

The CCR will store the membership type (*strong* or *weak*) to be used by the running cluster. Additionally, the CCR will also store any connectivity checks ('ping' targets) to be used by the CMM when reconfiguring under *weak membership*.

The CMM will store all this CCR data in its own local copy (in memory) for its own use.

When a node boots up, this data will be read from CCR along with other existing configuration parameters for CMM (such as boot delay timeout, failfast grace time, etc. ).

CMM will register with CCR to get callbacks when any of this membership data is changed on a live running cluster. In the callback, CMM will update its own local in-memory copy of this membership related data.

### 3.3 Connectivity check targets

During a CMM reconfiguration under weak membership, the CMM will do a set of configured connectivity checks to ensure that the node is 'healthy' enough to survive.

The CMM automaton already does a similar health check in begin state – to check if the node has enough memory or not, for example. This health check will also 'ping' the configured connectivity targets when reconfiguring under *weak membership*. This health check will halt the node if any one of the ping targets is not reachable, and on the other hand, if there are no 'ping targets' configured, then the health check is considered a success.

Putting this health check in the *begin state* of the CMM automaton means that this health

check will be done in every CMM reconfiguration – when the node boots up first, and also when the node loses connectivity with its peer node resulting in a possible *split-brain* scenario.

To 'ping' a '*ping target*', the CMM does a door upcall to userland *qd\_userd* daemon in order to execute the 'ping'. The *qd\_userd* daemon, that is launched by *bootcluster* script before CMM starts, will execute the 'ping' command for the '*ping target*' and the result of the ping command is passed back to CMM. If the door upcall cannot be performed or if the ping fails, then the CMM considers the check to have failed and the automaton will panic the node.

### **3.4 Change in membership type and properties**

The user can change the membership type being used by the cluster and/or one or more membership type properties.

The CLI that changes membership type will first commit the change to CCR. As a part of the commit operation, CCR will deliver callbacks to CMM and CMM will update its own local in-memory copy of such data for its own use.

After the commit to CCR, the CLI will trigger a CMM reconfiguration if the membership type has changed; a change in membership type is a major change for the cluster and an immediate reconfiguration will ensure that the cluster is running fine under the new membership type. The CLI will use the CMM control interface to trigger a reconfiguration.

If only the connectivity check targets information changes, the CLI will not trigger a CMM reconfiguration.

## **4. Cluster rejoin**

If a cluster running with Weak Membership goes into a split-brain scenario and then the nodes are able to communicate again later, the nodes cannot automatically join back to form a single cluster, without one of them being rebooted. Today, once a cluster node departs the cluster, that node cannot rejoin under the same node incarnation number. This check exists in `path_manager::update_node_incarnation()`.

The `path_manager::update_node_incarnation()` function will be enhanced to also check for the presence of information in the CCR indicating that CCR data changes occurred during a *split-brain* under *weak membership*. If a node finds that there are such unresolved data changes in its own local copy of CCR, this path manager software will not allow the formation of a path with the peer node, thus preventing the nodes to form a single cluster. The CCR will have marked any post *split-brain* CCR change that is unresolved, and the path manager software will check for such 'marks'.

## **5. CCR changes**

### **5.1 Flag To Indicate CCR Change During Split-Brain**

CCR changes can be done when the cluster is using *weak membership*. If the change is done on a node that thinks it is the sole member in the cluster membership, then the situation is similar to a *split-brain* scenario for that node. We mark any CCR change done in such a scenario, as two nodes could have different CCR changes done in such a situation, and they need to be resolved before the two nodes can form a single cluster in future. Existence of a file `"/etc/cluster/.split_brain_ccr_change"` on a cluster node serves as an indicator that CCR changes were done during *split-brain* on that node.

The file is created (if it doesn't already exist) upon successful completion of a CCR transaction during *split-brain*. It is created with 600 permission bits, owned by root.

If all the steps of the commit are successful in *updatable\_copy\_impl::commit\_transaction()*, the *split-brain* change file will be created at the end of that function.

## 5.2 New convenience functions in 'os' class

A new method *os::file\_create()* is introduced to create files. The creation mode will be *O\_EXCL | O\_CREAT*.

A new method *os::file\_exists()* will be used to query whether or not there is CCR change during *split-brain* on this node.

## 5.3 How To Indicate A Table Was Changed During Split-Brain

No special indication is present in the changed CCR table itself.

Let the generation number increase normally whenever the table is changed. The presence of the *"/etc/cluster/.split\_brain\_ccr\_change"* will indicate that some change exists.

## 5.4 How To Select The Winning CCR Copy

The administrator will run a CLI utility to mark the CCR on a cluster node as the truth copy. The CLI uses a *ccrlib* interface - *int ccrlib::mark\_ccr\_as\_winner()* - to mark the winning/truth CCR copy. This *ccrlib* interface will delete the *"/etc/cluster/.split\_brain\_ccr\_change"*. The CCR tables themselves are not modified.

## 5.5 How To Select The Loser CCR Copy

The administrator will run a CLI utility to mark the CCR on a cluster node as the losing copy. The CLI uses a *ccrlib* interface - *int ccrlib::mark\_ccr\_as\_loser()* - to mark the loser CCR copy. This *ccrlib* interface will delete the *"/etc/cluster/.split\_brain\_ccr\_change"*, and then every table on the losing node is marked as invalid (generation number *-1*). Marking every table with this generation number *-1* will ensure that when this node joins the winner node to form cluster, it will update its CCR copy using the winner node's CCR. The loser node cannot form a cluster of its own, as well.

Note that this command to mark the loser has to be run in non-cluster mode. The node must then be rebooted into cluster mode.

## 5.6 How To Check from CCR for existence of split-brain changes

*ccrlib* will provide an internal interface - *bool ccrlib::split\_brain\_ccr\_change\_exists()* - that other subsystems (such as CLI and *path\_manager*) can use to query if this node has *split-brain* CCR change(s). This interface will exist both in kernel and userland.

## 5.7 Changes to 'ccradm'

The internal utility - *ccradm* - is also modified to provide the following features, similar to the CLI.

- *ccradm -q* : check whether *split-brain* CCR change exist

- `ccradm -m <"winner"/"loser">` : mark a node as the "winner" or "loser"

## 6. CLI changes

To support different membership models, a new Quorum Object is introduced by name “**membership**”. This object will be managed completely using the `clquorum` (1CL) command. This object can neither be created nor be destroyed. It is associated with a special quorum type named “system”. This quorum object has two properties namely, “ping\_targets” and “multiple\_partitions”. Modifying these properties will allow the cluster to move from one membership model to another.

**Usage:**

**`clquorum set -p ping_targets=IPAddress1[,IPAddress2] -p multiple_partitions={ true | false } membership`**

The **ping\_targets property** is of type `STRING_ARRAY`. It can take multiple string values. Each of these string values must be valid Host IP addresses. The default value for this property is `NULL` i.e., an empty string. This property is stored in CCR in the “infrastructure” table under the key named “**cluster.properties.ping\_targets**”. This property can use the “+=” and “-=” operators to add and remove ping\_targets from/to the existing set of values.

**`clquorum set -p ping_targets+=IPAddress3 membership`**

**`clquorum set -p ping_targets-=IPAddress3 membership`**

When cluster is configured to run with the ‘**weak**’ membership model having `multiple_partitions=true` in CCR, cluster nodes ping all the configured ping\_targets during a cluster reconfiguration. This is a health check for the cluster nodes. If any of these ping\_targets is unreachable from a node, the cluster node will force itself to panic.

The **multiple\_partitions property** is of type `BOOLEAN`. It can take either take a `TRUE` or a `FALSE`. The default value for this property is `FALSE`. This property is stored in CCR in the “infrastructure” table under the key named “**cluster.properties.multiple\_partitions**”.

### 6.1 To setup WEAK Membership model operational

0. The administrator is installing the cluster and has finished with “scintall”. By default the cluster will run with the Strong membership model. The administrator has to take action to make the cluster use the Weak membership model. These steps are described in this section.

1. Set the weak membership model with the health checks (ping\_targets). The ping targets is a generic health check done when cluster is running with weak membership. It is done in every CMM reconfiguration, not necessarily only after a node cannot communicate with its peer. The motivation is that if a node cannot talk to external IP address, then it is likely that it will not be of much use in hosting applications/services. Currently, “ping\_targets” is available only for weak membership model

**`clq set -p multiple_partitions=true -p ping_targets=IPAddress1,IPAddress2 membership`**

This action might result in data corruption or loss.

Are you sure you want to enable multiple partitions in the cluster to be operational (y/n)

[n] ?

One can use the optional "-y" option to the above mentioned command to bypass the confirmation

#### **NOTE**

There will be no form of validations that will be performed to check if there are indeed any shared devices connected to the cluster node. It is a mandatory pre-condition for the administrator to make sure there are no such shared devices between the cluster nodes. There will be a check to make sure there are no quorum devices configured on the cluster before switching to weak membership model.

3. Now reset install-mode by running reset command.

**clq reset**

### **6.2 To setup STRONG Membership model operational**

0. The administrator is installing the cluster and has finished with "scintall". By default the cluster will run with the Strong membership model. This section describes the remaining steps.

1. Add a quorum device

**clq add d3**

2. Now you can optionally add a ping\_target health check, ping\_targets which can be used optionally. "ping\_targets" is not "required" in Colorado-I for strong membership and will not be used by the strong membership infrastructure. The default value for multiple\_partitions is set to "false". This property can also be optionally set to "false" in the command below.

**clq set -p ping\_targets=IPAddress1 membership**

3. You can also remove the ping\_targets by specifying the following command

**clq set -p ping\_targets= membership**

4. Now reset install-mode by running reset command.

**clq reset**

### **6.3 To Switch from Strong to Weak membership model**

1. In order to switch from Strong to Weak membership model, the cluster must be fully formed. In other words, both nodes must be up and both nodes must be in the same cluster partition. The transition cannot be done while the cluster is in a split-brain condition.

2. Remove the last available quorum device

**clq remove -F +**

3. Now set switch to weak membership model by setting the multiple\_partitions property to true. If the ping\_targets property was not set earlier, this property is mandatory with this command.

**clq set -p multiple\_partitions=true -p ping\_targets=IPAddress1[,IPAddress2] membership**

This action might result in data corruption or loss.

Are you sure you want to enable multiple partitions in the cluster to be operational (y/n) [n] ?

#### **NOTE**

There will be no form of validations to check if there are indeed any shared devices connected to the cluster node. It is a mandatory pre-condition for the administrator to make sure there are no such shared devices between the cluster nodes. There will be a check to make sure there are no quorum devices configured on the cluster before switching to weak membership model. There will be a check to ensure that there are exactly 2 nodes in the cluster.

### **6.4 To Switch from Weak to Strong membership model**

1. In order to switch from Weak to Strong membership model, the cluster must be fully formed. In other words, both nodes must be up and both nodes must be in the same cluster partition. The transition cannot be done while the cluster is in a split-brain condition.

2. Set the `multiple_partitions` property to `false` to switch to Strong membership.

```
clq set -p multiple_partitions=false membership
```

3. Now add a quorum device (quorum server or a shared disk) to provide high availability. Without a quorum device, any disconnect between the cluster nodes or single node failure will result in complete cluster panic.

```
clq add d3
```

### **6.5 Validations**

1. The values specified for `ping_targets` must be a valid IPAddress. Each of the IPAddresses added to the `ping_targets` property will be validated using the `gethostbyname()`. Invalid IPAddress will throw a usage error.
2. Only one IPAddress value can be added or removed using the `+=` and `-=` operators. The `=` operator can be using to set more than one IPAddress values in one go.

The values specified for `multiple_partitions` must be either “true” or “false”. Any other values will throw a usage error.

### **6.6 Resolving outstanding/pending CCR updates on cluster nodes after split-brain**

All changes are restricted to only the `clnode` command.

Usage: **clnode resolve-config -n <winner-node> [local-node]**

Weak membership

OPTIONS:

- ? Display this help text.
- F Force.
- n Specify the winner node.
- v Verbose output.

This subcommand is interactive but with -F option the command will proceed without the user interaction.

This command must be run on both nodes of the cluster after a split-brain. This command will be used for a 2 node cluster in the case of split brain. If a split brain occurs in a cluster, this command will validate the ccr of one node(say, node1) and invalidate the ccr of other node(say, node 2). When the node 2 is rebooted, it rejoins the cluster with the ccr copy of node1. A node is called "winner node" if its ccr is validated and called "loser node" if its ccr copy is invalidated. This command needs to be run on both the nodes. The local-node argument is optional. If this argument is not specified with the command, the current node i.e. the node on which the command is being executed will be taken as "local-node".

For marking the CCR on a winner node to be the truth copy (copy that should be used by the other node joining the cluster).

Mark the CCR data as VALID copy.

Can be run from the winner node either in cluster or non-cluster mode

**Example 1:** Marking a node as winner node.

**clnode resolve-config -n phys-winner phys-winner**

*This command will accept cluster configuration changes made on "phys-winner".*

*Proceed (y/n)? [n]*

The -F option can be specified in order to bypass the interactive confirmation question.

For marking the CCR on a loser node to be an invalid copy (copy which should be used only until this node joins the cluster with the other node which should by now that the truth copy of CCR).

Mark the CCR data as INVALID copy.

Must be run from the loser node only from non-cluster mode

**Example 2:** Marking a node as loser node.

**clnode resolve-config -n phys-winner phys-loser**

*This command will discard cluster configuration changes made on "phys-loser". Accept configuration from "phys-winner". Proceed (y/n)? [n]*

## 6.7 Query for unresolved CCR changes

Administrator can query from either nodes of the cluster in both cluster and non-cluster mode using the 'clnode status' command. The clnode status output will show the administrator whether there are any pending unresolved CCR updates in the cluster nodes configured.

**clnode status**

--- Node Status ---

Node Name	Status
-----------	--------

-----

pcool2	Online - Unresolved Split-Brain CCR Data Changes - present
--------	--

pcool1	Offline
--------	---------

## 6.8 Restrictions

Changes for supporting different membership models and resolving pending/unresolved CCR updates will be only provided using the New Command line (clquorum & clnode) commands. The respective Old CLI will not support these new quorum type objects and the respective properties.

## 6.9 Future Considerations

The new clquorum object “membership” will support only “ping\_targets” & “multiple\_partitions” for Phase I. The future phases **may** support the following properties:

```
amnesia_protection={ true | false }
connectivity={ direct | indirect | star }
health_cmd={ custom_command }
```

## 6.10 Sample Output

```
bash-3.00# clq list
membership
pcool1
pcool2
```

```
bash-3.00# clq status
```

```
==== Cluster Quorum ====
```

```
--- Quorum Votes Summary ---
```

Needed	Present	Possible
1	1	2

```
--- Quorum Votes by Node ---
```

Node Name	Present	Possible	Status
pcool1	1	1	Online
pcool2	0	1	Offline

```
--- Membership Health Check (current status) ---
```

Node Name	Health Check Type	Entities	Status
pchips1	Ping Targets	IPAddress1	Ok
pchips1	Ping Targets	IPAddress2	Fail
pchips2	Ping Targets	IPAddress1	Unknown
pchips2	Ping Targets	IPAddress2	Unknown

```
bash-3.00# clq show
```

```
==== Cluster Nodes ====
```

```
Node Name:          pchips1
Node ID:            1
Quorum Vote Count:  1
Reservation Key:    0x4912D1DB00000001
```

Node Name: pchips2  
Node ID: 2  
Quorum Vote Count: 1  
Reservation Key: 0x4912D1DB00000002

=== Membership ===

Name: membership  
Type: system  
Multiple Partitions: yes  
Ping Targets: IPAddress1, IPAddress2

## Normal Output during split brain

```
bash-3.00# cnode status
```

```
=== Cluster Nodes ===
```

```
--- Node Status ---
```

Node Name	Status
-----	-----
pchips1	Online
pchips2	Online

## When there are unresolved CCR updates during split brain

```
bash-3.00# cnode status
```

```
=== Cluster Nodes ===
```

```
--- Node Status ---
```

Node Name	Status
-----	-----
pchips1	Online - Unresolved CCR updates pending
pchips2	Offline

## 7. Weak Membership Scenarios

Here is a list of scenarios that could occur when the cluster is using weak membership, and how the proposed design will play out in such scenarios. Note that we are only discussing a 2-node cluster scenario for weak membership.

1. Both nodes have formed a 2-node cluster together, and the cluster is running with weak membership.
  - If one node dies, the other node reconfigures; if the latter node is able to ping all configured ping targets successfully, then it survives as a 1-node cluster.
  - If the communication path between the two nodes breaks down, each node starts reconfiguring. If a node is able to ping all configured ping targets successfully, then it survives as a 1-node cluster. So, either one or two 1-node clusters survive, or no clusters survive.
2. A node is running as a 1-node cluster with weak membership. The other node tries to join the first node. Neither node has unresolved split-brain CCR changes. Both nodes form a 2-node cluster after reconfiguration.

3. Each node - Node 1 and Node 2 - is running as a single node cluster, since the communication path between them is not functional. Each node has unresolved split-brain CCR changes marked on it. The administrator decides that Node 1 is the CCR winner and Node 2 is the CCR loser. The administrator reboots Node 2 into non-cluster mode, and marks it as loser. The administrator marks Node 1 as winner. The administrator then reboots Node 2 into cluster mode. Both nodes join to form a 2-node cluster with no CCR differences; Node 1's CCR is propagated to Node 2.
4. Node 1 is running as a 1-node cluster with weak membership. A CCR change is done on it; this is marked as a split-brain CCR change. Node 2 boots up and tries to join Node 1. Node 1 does not allow the path formation with Node 2 since Node 1 has unresolved CCR changes; regardless of whether Node 2 has unresolved CCR changes as well or not. Node 2 goes on to form its own 1-node cluster, independent of Node 1's single node cluster.
5. Each node - Node 1 and Node 2 - is running as a single node cluster, since the communication path between them is not functional. Each node has unresolved split-brain CCR changes marked on it. The communication path comes up. The nodes do not join each other and continue running as single node clusters.