mod_ha_cluster

About

The **High Availability Cluster Module** module is currently under development by Eliot Gable &lt;egable.at.gmail.com&gt;.

Please speed up creation by contributing to FreeSWITCH consulting earmarked mod_ha_cluster.

Contact Sharon White &lt;sharon@freeswitch.org&gt; to arrange contributions.

---

Why a new solution?

Why mod_ha_cluster instead of using corosync/pacemaker?

- It's N+x cluster, rather than active/passive. With pacemaker to have 16 nodes with a failover, you would need 32 total. With mod_ha_cluster, you could have 16 nodes and 4 failover slaves to handle the load, requiring only 20 machines.
- To recover calls, share registrations, or limits you would need a shared database backend. This will handle that between FreeSWITCH instances.
- Centralized configuration for all machines

Goal

The goal of this module is to provide carrier-grade high availability features to FreeSWITCH without the use of 3rd party software. This means all features would be supported directly inside FreeSWITCH without the use of Pacemaker + Corosync, a shared database, or any other external software which must be separately configured and maintained. Planned features include:

- Support for multiple master nodes
- Support for multiple slave nodes
- Slave nodes can take over for any master node
- IP-based fail over
- Support for optional "floating" maintenance IP
- Optional maintenance mode where a slave is brought online using the maintenance IP to replace a master while calls drain off the master
- Support for dynamic DNS updates so DNS can be used for load balancing across multiple master nodes on different IP addresses while the module maintains the list of available IP addresses (to support use of maintenance mode)
- Calls are recovered using "sofia recover" capabilities
- Very fast fail over (less than 1 second)
- Sharing of registration information between all cluster participants
- Sharing of session limiting and sessions per second limiting information between all cluster participants
- Real-time cluster node state monitoring through fs_cli
- Cluster events you can watch for over event socket
- Multicast auto-discovery of cluster nodes
- Automatic designation of master or slave state based on entire cluster state and rules configured in a shared config file
- Single config file which can be shared among all cluster nodes (no node-specific configuration)
- Online updating of configuration
- Online activation of configuration changes
- (eventually) STONITH / fencing support
- (possibly) Support for configuration of iptables firewall rules on node start-up

Development Stages and Funding Acknowledgments

In order to facilitate easier funding and greater accountability, the development for the initial release of the module and funding for it will be broken out into 4 stages. Each stage consists of primary tasks for that stage which will always remain in that stage and really define what that stage is about. The remaining tasks in that stage may be shuffled around to earlier or later stages, as needed, in order facilitate rapid development.

The development stages should be considered as a general guideline of how development will proceed. However, keep in mind that there are often many cases which arise where it makes more sense to make progress towards goals in multiple stages simultaneously. For that reason, some stages might take longer while others may be shorter. For example, while coding stage 1, I will likely have many opportunities to make easy progress on the goals of stage 2, and so stage 1 may take longer than stage 2 because some of the work for stage 2 is being completed in stage 1.

For accountability reasons, I will not be obtaining payment from FreeSWITCH Solutions for any given stage until all earlier stages are completed. Thus, even if I complete all primary tasks in stage 2 prior to completing all primary tasks in stage 1, I will not be obtaining payment for stage 2 until the final primary task in stage 1 is complete.

By agreeing to fund this project, you acknowledge that your payments to FreeSWITCH Solutions will be allocated to the stages in the order they are received. Thus, if stage 1 is not fully funded, you cannot contribute to stage 2. You also acknowledge that money collected for each stage will be paid to me upon my completion of all primary tasks in that stage, as I have outlined here. You also acknowledge that once funds are donated to this project, you may not obtain a refund for any reason, except in the case where development permanently ceases on this module AND your funds have not yet been utilized to pay what is due for the completion of the stages. You also acknowledge that development may not continue on future stages until they are fully funded, at my sole discretion.
Further, you acknowledge that any acceleration in the timeline due to unforeseen circumstances (like my having more time than expected to work on it, or there being less time required to write and debug code than initially anticipated, etc) will not reduce the amount due for any particular stage. I also acknowledge that any delays in the timeline caused by my having less time than anticipated or finding it more challenging than anticipated will not increase the amount due for any particular stage.

If you wish to donate $10,000 total towards development efforts, but you want to hedge against possible future issues with the project, I recommend you break your donation amount down according to the four primary stages listed. In that case, you would donate $2,500 initially, and upon completion of stage 1, you would donate another $2,500, etc. Or, you may optionally donate $2,500 and when stage 1 is fully funded, donate another $2,500 towards stage 2, prior to stage 1 being completed. If you wish to break your donations down according to stages, please express your interest in doing so when you make your donations, and I will keep track of who to contact based on my progress on the module.

All estimated delivery dates are assuming the stage is funded before I finish the prior stage, with the exception of the first stage, which I am working on right now under the assumption it will be fully funded based on current interest levels.

The stages are as follows:

**Stage 1**

Current funding goal: $0 out of $12,500 required

Estimated Delivery: None

- (DONE) Designing a configuration file
- (DONE) Parsing the configuration file
- (DONE) Bootstrapping a node into the STARTING (discovery) state for Timer-A seconds before moving into normal operational mode
- (DONE) Sending heartbeats to all configured multicast groups out all configured interfaces
- (DONE) Receiving heartbeats from other nodes
- (IN PROGRESS) Moving multicast event reception into its own thread and pushing all events through a FIFO queue
  - (IN PROGRESS, initial commit 2/17/13) Add generic dispatcher system to FreeSWITCH Core
    - Add mcast_receiver to FreeSWITCH Core
    - Add mcast_sender to FreeSWITCH Core
  - Move multicast reception / sending into mcast_receiver and mcast_sender implementation hooked to dispatchers
- (PRIMARY TASK) Processing heartbeats to detect state of other nodes
- (PRIMARY TASK) Deciding what state the current node will be in, including its position in the slave list if it is a slave
- (PRIMARY TASK) Launching the node into Master state where it takes calls and broadcasts call states to other nodes

**Stage 2**

Current funding goal: $0 out of $12,500 required

Estimated Delivery: None

- Sending of heartbeats on a separate thread for the purpose of pushing them internally through the FS core event dispatching thread so heartbeat failure will occur if the core event thread can no longer dispatch events (for example, in case of a deadlock or some bit of code hooked into an event spinning the CPU)
- (PRIMARY TASK) Failure detection based on loss of heartbeat
- (PRIMARY TASK) Sending of registration events to other nodes
- (PRIMARY TASK) Reception and tracking of registration events from remote nodes
- (PRIMARY TASK) Reception and tracking of call state information from other nodes

**Stage 3**

Current funding goal: $0 out of $12,500 required

Estimated Delivery: None

- (PRIMARY TASK) Making fail over function so the primary designated slave can recover calls
- (PRIMARY TASK) Set up tracking of call states in a way that it supports multiple masters and can recover calls for an individual failed master
- (PRIMARY TASK) Synchronization of existing call states from other nodes if it is in the slave state
- Write real-time status commands for fs_cli cluster state monitoring
- Sending of session limits to other nodes
- Reception and processing of session limits from other nodes
- (PRIMARY TASK) Write maintenance mode command to put a master into maintenance mode and bring up the primary designated slave on the maintenance IP
- (PRIMARY TASK) Add dynamic DNS support so maintenance mode can update DNS records to remove the master's IP and add the maintenance IP (meaning the master's IP becomes the new maintenance IP)

**Stage 4**

Current funding goal: $0 out of $12,500 required

Estimated Delivery: None

- (PRIMARY TASK) Lots of testing and bug fixes
- (PRIMARY TASK) Release
Stage 5

Current funding goal: $0 out of $12,500 required

Estimated Delivery: None

In addition to the 4 stages listed above, this 5th stage is considered a necessary piece of the solution. However, the solution will be usable by many people with only the first 4 stages completed. Thus, this 5th stage will be completed after initial release and is more for polishing the solution and providing coverage for additional, more rare failure scenarios.

- Add STONITH / fencing support
- Add firewall rule support

Stage 6

Current funding goal: $0 out of $12,500 required

If enough funding is provided and people have interest in additional features and functionality aside from what is listed so far, those features will be developed in stage 6, and additional stages will be added as needed to cover continued feature requests and development. No features / functionality for stage 6 will be developed until all 5 prior stages are fully funded and complete.

Timeline

From initial conception and design to the current state of code development, I have spent about 220 hours (as of 2/17/2013) on this module. I am estimating another 400 - 800 hours of development remaining before release. Right now, there are no financial backers for this module. This means I am only able to spend 5 - 10 hours per week sporadically on developing the module. Some weeks, I am not able to spend any time on it. If you would like to see this module completed more quickly, money can make it happen.