Cacti High-availability symmetrical cluster/grid (revision 3).

Tested Versions: cacti 8.6j, cactid, Mysql 5.1, Apache 2.2, PHP 5.1, DRBD

Tested Plug-ins: Plugin architecture, Thold, Haloe, settings
Other plugins may work

Other modifications: 1 minute polling intervals, 60 days of 1-minute data

Tested OS: CentOS 5

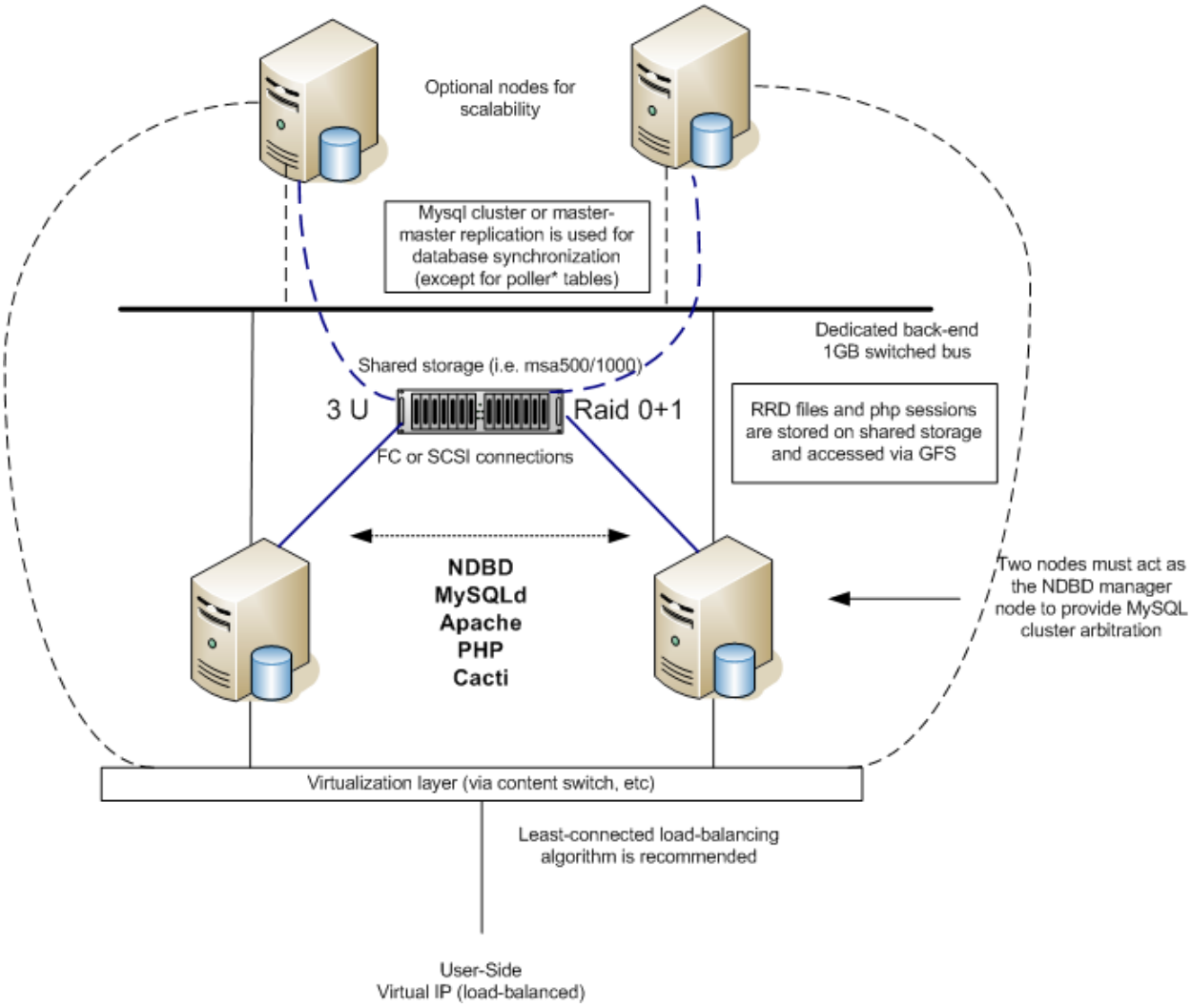
Purpose

To provide a scalable, redundant, modular architecture for large-scale cacti polling. This cluster is meant to provide continuous polling of all elements during a partial cluster/grid outage. It is also meant to provide a linear measurement of elements per node scalability to guarantee all elements are polled in less than 60 seconds. This architecture is not for the faint of heart to install. This guide also assumes that you are fluent with unix; it is not meant to hold your hand through every step.

Abstract

The theory behind the cacti cluster/grid is that the total number of hosts polled would be divided up equally amongst the active nodes within the cluster/grid, therefore allowing modular scalability of elements.

The database replication / redundancy would have to be a multi-way, real-time replication scheme – this will be provided using either MySQL cluster package or master-master MySql replications. The core files (round-robin database, etc) will have to remain in sync at all times - in order to keep them synchronized real-time a shared storage solution will be required. Ideally, a shared storage (HP MSA500 or MSA1000) or SAN using GFS would provide this functionality. For a more cost-effective solution, DRBD can be used to replicate this data over IP in real-time – but this limits storage access to two nodes, until shared via GNBD. Disk I/O will become one of the most critical parts of the cluster's success. IP-based storage will typically not yield the best performance. A load-balancer would be used in front of the cluster nodes to provide a VIP between all the nodes (and handle client requests). Cisco CSS, SLB, and other open-source solutions are available for this functionality.



Core/Minimum components:

Formula for nodes required for cluster/grid:

S = sustainable number of hosts per polling cycle, per node, with an average of 30 elements per host.
This is dependent on the speed/type of processor, number of processors, number of threads, etc. You should allow for space within the polling cycle for timeouts, etc. Here are some guidelines per processor:

3.0ghz = 50 hosts

Nodes = 1 + (S / total number of elements)

Note: The above formula factors in an additional node to ensure all the elements can be polled during a failure of one node.

The cluster/grid is dependent upon shared storage that will store all the round-robin databases and session pointers (for apache/php). The storage should be on RAID0 + 1 (striping + mirroring) array – our system ran on 14 disks on a HP MSA1000 array with 2Gb fiber-channel connections. Our previous system ran on a HP MSA500 with SCSI connections.

The nodes will run all the normal components of a cacti system – Apache, PHP, Mysql. These nodes will run either MySql MAX with clustering instead of normal MySql or Mysql master-master replication. Also, they will run the `cluster-patched' version of cacti, which includes polling/keepalive scripts.

All the nodes will have to run snmpd – to provide accurate real-time status to the cluster-status web page. The apache server will be used to exchange keepalives between the nodes and determine states of various processes. They update the 'cluster_nodes' table which determines which nodes will do polling for what hosts.

There would be a dedicated 1GB/s switched network bus between all the nodes. In larger deployments, other options can be used for MySQL replication i.e. SCI or infiniband. On a side note, two of the nodes will have to run nbd_mgmd, which is the management process for the NDBD storage engine on the MySQL cluster.

Recommended hardware list:

- 2 x Application nodes
  - 2 x 3.0ghz+ CPUs, 4GB RAM, 2 x 1000baseT NICs, hardware RAID 0 + 1 recommended.
  - 64-bit CPU(s) even better, and/or a dual-core.

- 1 x shared storage array with 2+ connections (I.e. MSA500 or MSA1000)

- Load-balancing switch / device
  - These document uses Cisco SLB with redundancy. There are other alternatives (including some open-source ones), but VIP redundancy is required.

  - Dedicated 1GB/s switch is recommended (low-latency i.e. cisco 4948 is recommended); if there are two nodes, a crossover cable will work - otherwise a VLAN will suffice. 1GB/s is a must.

Theory of operation:

Nodes that are attached to shared storage will need linux cluster package that include CMAN, GFS, etc. CentOS offerers these packages in their "plus" section.

On bootup, the cluster nodes auto-mount /mnt/(storage)/ and directories to the shared storage (this is typically accomplished via GFS service). Two directories are symbolically linked to this storage – (cacti_dir)/rra and /var/lib/php/session (php session dir). Then the httpd process starts and it will use the files located on these mounted/linked partitions.

Every 60 seconds, all nodes in the cluster will run a `keepalive' script which will query the `cluster_nodes' table in the cacti database on the local box. The table will contain all the information for each node in the cluster including name, ip addresss, uptime, node order, status, etc. For each node, this script will then do a ping and http get to each other's cluster status page. Based on positive outcomes of all the tests, it will declare each node alive and then insert the status and uptime into the database. Once the poling is complete, the script will then query the table for alive nodes, and then order them based on name.

The order of the nodes comes into play when the poller.php script needs to determine what elements are polled by each node. Each node will run the poller.php script (cluster-modified) every minute (as a normal cacti box will run). The poller script then checks the cluster_nodes tables to see what order it is in, and use this number as a multiple. For example, if a node is number three out of four alive nodes, and there are 200 hosts that need to be polled by cacti, this node will poll hosts 101-150. If the current cluster status is down (or it cannot query the mysql database) it will not poll anything. The poller script requires use of cactid, as it is the fastest polling method available and natively multi-threaded. It is recommended that the cactid is configured with multiple processes (up to 100 if the box can handle it), even if there is only one processor in the cluster node (this is configured via the cacti gui console-> settings -> poller -> Max threads per process).

Installation

You should determine your storage architecture and install appropriate components. DRBD can be found at http://www.drbd.org or installed with many package managers (yum on centos if Plus repository is enabled, apt-get on ubuntu, etc). GFS and clustering can be installed with centos during normal install operation.

You can test by mounting your shared storage and make sure it is writable on all nodes – and newly created files are visible on all nodes.

Large amounts of RAM is a MUST. The MySQL cluster stores all data in memory for fast replication and access, and Linux caches various data it uses (i.e. rrd data) so you must have at least 2GB of RAM in each node. Note that it is possible to create a highly distributed architecture utilizing separate machines for the NDBD storage engine, MySQL services, HTTP service, etc; but this installation guide will not cover that architecture.

The recommended installation procedure for the nodes will be to build one box and mirror the hard drive. These nodes should have a version of linux (CentOS 5 recommended) that includes snmpd, apache, PHP, and php-mysql libraries (you can install these separately if you wish.) There should also be a complier installed (for cactid, etc). MySQL should be downloaded from the MySQL website since you will need the MAX version for clustering (we tested with pre-complied RPMS, but you can compile if you like). You may have to do a `forced' RPM installation/upgrade because of PHP dependencies. You should do preliminary cluster, gfs, and mysql configuration now. Then you will have to mirror your drive at this point and start configuring the nodes seperatly.

Mysql – option 1 - mysql cluster

Pros: synchronous replication, N-way scalability

Cons: slow startup, slower queries, recovery can be difficult, static limits on table rows, etc

Example: http://dev.mysql.com/tech-resources/articles/mysql-cluster-for-two-servers.html

Mysql NDBD nodes should be configure with 1 replica per node, so the mysql cluster can survive with one node. You will also need to run ndb_mgmd for the MySQL NDBD storage engine arbitration on two nodes (one primary, one backup) – this is described in the mysql cluster installation guide. It should run the ndb_mgmd process as a service.

When using mysql clustering, the table type that will be replicated will have an engine type of NDBCLUSTER. The poller* tables will remain as type myisam as this data must remain local to each node.

Going forward, you should make sure your mysql cluster is working (as described in the cluster install doc – create a table on one node and make sure it is visible on the other). You should also create a startup script for the NDBD process and make sure it starts up before MySQL; or incorporate it into the MySQL startup script (as described in the mysql cluster install doc).

Please note that if you plan on upgrading an existing installation of cacti, please see **appendix A**. You will first have to create the database on all nodes (`mysqladmin create cacti'). Then you need to use the SQL script `cacti.cluster.sql' included in the distribution to create the tables. I.E. `mysql cacti < cacti.cluster.sql'. You will have to run this script only on one box if you have installed mysql cluster. Then you will have to run the per-node SQL script `cacti_cluster_node.sql' to create the temporary poller tables on each node.

Mysql – option 2 - master-master cluster

Pros: very fast startup and fast queries

Cons: scalability becomes difficult past two nodes

Example: http://www.howtoforge.com/mysql_master_master_replication

Master-master replication will require that you filter/block replication for all the poller* tables, as this data must be per-node. There are various master-master monitoring packages available to monitor the replication. If you want to scale past two nodes, you will need to setup table types of 'proxy' (except for poller* tables) and point them to a VIP between your two mysql master-master nodes.

You can create the database & tables on each node using the normal cacti.sql script. If you are trying to do this with an existing installation, you should replicate tables as described in the above master-master replication guide.

Cacti installation

Make sure apache and PHP and working, etc. Then you will need to symbolically link (cacti_dir)/rra and /var/lib/php/session to your shared storage. You can rename the original directories or empty them altogether.

Now you can proceed to install the cluster-patched version of cacti. Once the database had been loaded, you can bring up the web interface using one box, and configure the nodes. Go to console -> cluster and add the nodes. Note that the `hostname' of the nodes should match the output of the command `hostname' in order to work properly. This page does a real-time poll of the processes via snmp every time you refresh it, so you will know immediately if there is a problem. If you are not getting a response from a specific node, make sure snmpd is running properly on that node and configured properly. At this point you should configure cron with the keep-alive script for the cluster (cacti_root)/cluster/check_cluster.http.php which will run every 60 seconds on every node – this script will determine the cluster status, order, etc by parsing the 'clustat.php' file from each node.

Once all the nodes show a `green' status, you are ready to start the polling process. It is recommended that you set logging level to `debug' on the cacti GUI, and run the poller script manually for the first time (`php (cacti_dir)/poller.php') and watch the output. Then you should look over the log file to see if there were any issues. You should do this on every node. When you run the poller script manually, you should see the current node's cluster_id. After you are happy with the results, you can configure cron in the normal fashion and have it do the poll every minute. If you did an upgrade, you should copy your old rrd files now.

The final step is to configure your front-end load-balancer which will present the cacti nodes to the client with one IP address. The load-balancer will almost double your client-side performance because it should direct each http request (even each image request) to a separate node, providing a noticeable performance improvement on the client-side. We have tested this with a Cisco IOS SLB – here is a snapshot of the configuration.:

```
ip slb natpool CACTINAT 10.1.2.251 10.1.2.251 netmask 255.255.255.0
!
ip slb probe CACTI http
 port 80
 interval 30
!
ip slb serverfarm CACTI-SRV
 nat server
 predictor leastconns
 nat client CACTINAT
 failaction purge
 probe CACTI
 !
 real 10.1.2.71
  faildetect numconns 3
  inservice
 !
 real 10.1.2.73
  faildetect numconns 3
  inservice
!
ip slb vserver CACTI-VIP
 virtual 10.250.1.2 tcp www
 serverfarm CACTI-SRV
 inservice
!
```

Once you have successfully configured your load-balancer, you should be all done!

Monitoring

The cluster-patch version of cacti has a 'cluster' tab on the console page which provides a simple interface to view the real-time status of each cacti cluster node. It also provides an interface to configure the cluster nodes. Email alerts are to be added to the various scripts. Log file is under (cacti_root)/log/check_cluster.log

For MySQL cluster monitoring, you should use 'ndb_mgm' on the arbitrator node to view the status of the NDBD client engines and MySQL clients. You may need to purge sessions if a node disconnects uncleanly.

Caveats

In theory, this setup would allow for a partial failure within the cacti monitoring architecture, and allow for a consistent collection of data. In some certain circumstances there may be unrecoverable dataloss. For example, since the keepalive script only runs every 60 seconds, it may be possible to lose a portion of the data since there may be an unknowingly dead node that is supposed to poll a set of data, but is offline so it cannot. This would mean that in theory only one sample of data would be lost for a specific set. Also, it may be possible for further complications to arise if there are not enough nodes to poll all the data with the polling period.

Dataloss could also occur if there are issues with your shared storage, so you should test various failure scenarios and their impact on recovery and performance.

Other issues may come up when clusters are brought down unexpectedly and may require manual recovery of mysql clustering / replication.

Bottlenecks will be either in the form of Disk I/O or CPU utilization – so hardware may be the only solution in some circumstances.

With this architecture and 2 x nodes, we were able to poll 200 hosts, 6000 elements in 30 seconds with both nodes active. Linear scaling is would theorize double scalability with twice the amount of nodes. This architecture could theoretically scale to 1600 hosts, 36000 elements if the interval is left at 5 minutes with two nodes.

Minor changes were made to Haloe plugin to ensure only one node did haloe post-processing (last node).

You must also remember to manually synchronize new/custom files between nodes (I.e. xml files, scripts etc)

Other performance suggestions are to increase firefox settings (using about:config) such as network.http.max-connections-per-server, etc.

Other things to do: alerting on failures

Appendix A: Upgrading previous install to MySQL cluster-based

WARNING: please do not attempt to use a production box for this upgrade. You should always setup the cluster apart from what you have in production so you have a roll-back plan. This is only for MySQL clustering, not master-master replication.

This first step is to dump your sql tables from your existing database. You can do this with `mysqldump –opt cacti > cacti_orig.sql' . Then you will need to do a replace on the database type: `replace ENGINE=MYSAM ENGINE=NDBCLUSTER -- cacti_orig.sql'. Then use this SQL script to build the database on the cluster (only on one of the nodes).
You will also have to copy the RRD files over from the original box, but it is recommended that you do this once the cluster is fully operational. Then you will have to run the per-node SQL script `cacti_cluster_node.sql' to create the temporary poller tables on each node.