

3.11.2 High Availability of the Central Manager

The *condor_negotiator* and *condor_collector* daemons are the heart of the Condor matchmaking system. The availability of these daemons is critical to a Condor pool's functionality. Both daemons usually run on the same machine, most often known as the central manager. The failure of a central manager machine prevents Condor from matching new jobs and allocating new resources. High availability of the *condor_negotiator* and *condor_collector* daemons eliminates this problem.

Configuration allows one of multiple machines within the pool to function as the central manager. While there may be many active *condor_collector* daemons, only a single, active *condor_negotiator* daemon will be running. The machine with the *condor_negotiator* daemon running is the active central manager. The other potential central managers each have a *condor_collector* daemon running; these are the idle central managers.

All submit and execute machines are configured to report to all potential central manager machines.

Each potential central manager machine runs the high availability daemon, *condor_had*. These daemons communicate with each other, constantly monitoring the pool to ensure that one active central manager is available. If the active central manager machine crashes or is shut down, these daemons detect the failure, and they agree on which of the idle central managers is to become the active one. A protocol determines this.

In the case of a network partition, idle *condor_had* daemons within each partition detect (by the lack of communication) a partitioning, and then use the protocol to choose an active central manager. As long as the partition remains, and there exists an idle central manager within the partition, there will be one active central manager within each partition. When the network is repaired, the protocol returns to having one central manager.

Through configuration, a specific central manager machine may act as the primary central manager. While this machine is up and running, it functions as the central manager. After a failure of this primary central manager, another idle central manager becomes the active one. When the primary recovers, it again becomes the central manager. This is a recommended configuration, if one of the central managers is a reliable machine, which is expected to have very short periods of instability. An alternative configuration allows the promoted active central manager (in the case that the central manager fails) to stay active after the failed central manager machine returns.

This high availability mechanism operates by monitoring communication between machines. Note that there is a significant difference in communications between machines when

1. a machine is down
2. a specific daemon (the *condor_had* daemon in this case) is not running, yet the machine is functioning

The high availability mechanism distinguishes between these two, and it operates based only on first (when a central manager machine is down). A lack of executing daemons does *not* cause the protocol to choose or use a new active central manager.

CM is a stateful service. That's why apart from ensuring that there will always be a single matchmaker in the Condor pool, high availability mechanism should be able to protect the state in face of failures. A special daemon, called *condor_replication*, takes care of the replication and data consistency issues, ensuring that the CM has got an updated and consistent state.

Condor CM's state encapsulates the information about Condor pool resources sharing and users priorities and is stored in `Accountantnew.log` file. This file resides on the machine running the negotiator daemon, and is being updated once upon some period of time. Loss of this file after negotiator's machine failure will result in improper handling of pool's fairness policy.

condor_replication runs on each backup machine and on the machine, running the HAD leader. The replication daemons on different machines comprise a replication subsystem within the pool. The replication daemon, residing on the active HAD machine, is called the replication leader and CM's state file is transferred by it to other replication daemons, called replication backups once upon a period of time. The redundancy of this daemon enables us to have an up-to-date copy of the crucial file upon active CM machine failure on every backup machine.

There is a unidirectional communication between HAD leader and replication leader on the same machine. Each HAD leader sends corresponding messages to the local replication daemon, so that the replication daemon is aware, that its local HAD is the leader. Upon machine failure another HAD in the pool is elected as the HAD leader and after some time its local replication daemon becomes the replication leader as well.

3.11.2.1 Configuration

The high availability of central manager machines is enabled through configuration. It is disabled by default. All machines in a pool must be configured appropriately in order to make the high availability mechanism work. See section [3.3.22](#), for definitions of these configuration variables.

The stabilization period is the time it takes for the *condor_had* daemons to detect a change in the pool state such as a active central manager failure or network partition, and recover from this change. It may be computed using the following formula:

```
stabilization period = 12 * (number of central managers) * HAD_CONNECTION_TIMEOUT
```

To disable the high availability of central managers mechanism, it is sufficient to remove `HAD`, `REPLICATION` and `NEGOTIATOR` from the `DAEMON_LIST` configuration variable on all machines, leaving only one *condor_negotiator* in the pool.

To shut down a currently operating high availability mechanism, follow the given steps. All commands must be invoked from a host which has administrative permissions on all central managers. The first three commands kill all *condor_had*, *condor_replication* and all running *condor_negotiator* daemons. The last command is invoked on the host where the single *condor_negotiator* daemon is to run.

1. `condor_off -all -neg`
2. `condor_off -all -subsystem had`
3. `condor_off -all -subsystem replication`
4. `condor_on -neg`

When configuring *condor_had* to control the *condor_negotiator*, the default backoff constant value is too small, and can result in a "churning" of the negotiator, especially in cases in which the primary negotiator is unable to run due to misconfiguration, etc. In these cases, the *condor_master* will kill the *condor_had* after the *condor_negotiator* exists, wait a short period, then restart *condor_had*. The *condor_had* will then "win" the election, so the secondary *condor_negotiator* will be killed, and the primary will be restarted, only to exit again. If this happens too quickly, neither negotiator will run long enough to complete a negotiation cycle, resulting in no jobs getting started. Increasing this value via `MASTER_HAD_BACKOFF_CONSTANT` to be larger than a typical negotiation cycle can help solve this problem.

To be able to run HAD-enabled pool without replication feature, one must perform the following operations:

- 1) Set the `HAD_USE_REPLICATION` configuration parameter to 'false' and thus disable the replication on configuration level
- 2) Remove `REPLICATION` entry from `DAEMON_LIST` and `DC_DAEMON_LIST` inside `$CONDOR_CONFIG` file

Alternatively, one may

- 1) Comment out the following line inside `$CONDOR_WORKSPACE/src/condor_had/StateMachine.h`:
`#define IS_REPLICATION_USED`
and add the following line
`#undef IS_REPLICATION_USED`
and thus disable the replication on compilation level
- 2) Compile the 'condor_had' binary from `$CONDOR_WORKSPACE/src/condor_had` directory:
`make release`
- 3) Remove `REPLICATION` entry from `DAEMON_LIST` and `DC_DAEMON_LIST` inside `$CONDOR_CONFIG` file

3.11.2.2 Sample Configuration

This section provides sample configurations for high availability. The two parts to this are the configuration for the potential central manager machines, and the configuration for the machines within the pool that will *not* be central managers.

This is a sample configuration relating to the high availability of central managers. This is for the potential central manager machines.

```
#####
# A sample configuration file for central managers, to enable the      #
# the high availability mechanism.                                     #
#####

# unset these two macros
NEGOTIATOR_HOST=
CONDOR_HOST=

#####
## THE FOLLOWING MUST BE IDENTICAL ON ALL POTENTIAL CENTRAL MANAGERS. #
#####
## For simplicity in writing other expressions, define a variable
## for each potential central manager in the pool.
## These are samples.
CENTRAL_MANAGER1 = cm1.cs.technion.ac.il
CENTRAL_MANAGER2 = cm2.cs.technion.ac.il
## A list of all potential central managers in the pool.
COLLECTOR_HOST = $(CENTRAL_MANAGER1),$(CENTRAL_MANAGER2)

## Define the port number on which the condor_had daemon will
## listen. The port must match the port number used
## for when defining HAD_LIST. This port number is
## arbitrary; make sure that there is no port number collision
## with other applications.
HAD_PORT = 51450
HAD_ARGS = -p $(HAD_PORT)

## The following macro defines the port number condor_replication will listen
## on on this machine. This port should match the port number specified
## for that replication daemon in the REPLICATION_LIST
## Port number is arbitrary (make sure no collision with other applications)
## This is a sample port number
REPLICATION_PORT = 41450
REPLICATION_ARGS = -p $(REPLICATION_PORT)

## The following list must contain the same addresses
## as HAD_LIST. In addition, for each hostname, it should specify
## the port number of condor_replication daemon running on that host.
## This parameter is mandatory and has no default value
REPLICATION_LIST = $(CENTRAL_MANAGER1):$(REPLICATION_PORT),$(CENTRAL_MANAGER2):$(
REPLICATION_PORT)

## The following list must contain the same addresses in the same order
## as COLLECTOR_HOST. In addition, for each hostname, it should specify
## the port number of condor_had daemon running on that host.
```

```

## The first machine in the list will be the PRIMARY central manager
## machine, in case HAD_USE_PRIMARY is set to true.
HAD_LIST = $(CENTRAL_MANAGER1):$(HAD_PORT),$(CENTRAL_MANAGER2):$(HAD_PORT)

## HAD connection time.
## Recommended value is 2 if the central managers are on the same subnet.
## Recommended value is 5 if Condor security is enabled.
## Recommended value is 10 if the network is very slow, or
## to reduce the sensitivity of HA daemons to network failures.
HAD_CONNECTION_TIMEOUT = 2

##If true, the first central manager in HAD_LIST is a primary.
HAD_USE_PRIMARY = true

##-----
## Host/IP access levels
##-----

## What machines have administrative rights for your pool? This
## defaults to your central manager. You should set it to the
## machine(s) where whoever is the condor administrator(s) works
## (assuming you trust all the users who log into that/those
## machine(s), since this is machine-wide access you're granting).
HOSTALLOW_ADMINISTRATOR = $(COLLECTOR_HOST)

## Negotiator access. Machines listed here are trusted central
## managers. You should normally not have to change this.
HOSTALLOW_NEGOTIATOR = $(COLLECTOR_HOST)

#####
## THE PARAMETERS BELOW ARE ALLOWED TO BE DIFFERENT ON EACH #
## CENTRAL MANAGERS #
## THESE ARE MASTER SPECIFIC PARAMETERS
#####

## The location of executable files
HAD = $(SBIN)/condor_had
REPLICATION = $(SBIN)/condor_replication

## the master should start at least these four daemons
DAEMON_LIST = MASTER, COLLECTOR, NEGOTIATOR, HAD
## DC_Daemon list should contain at least these four
DC_DAEMON_LIST = MASTER, COLLECTOR, NEGOTIATOR, HAD

## Enables/disables the replication feature of HAD daemon
## Default: no
HAD_USE_REPLICATION = true

## Name of the file from the SPOOL directory that will be replicated
## Default: $(SPOOL)/Accountantnew.log
STATE_FILE = $(SPOOL)/Accountantnew.log

## Period of time between two successive awakenings of the replication daemon
## Default: 300

```

```

REPLICATION_INTERVAL = 300

## Period of time, in which transferer daemons have to accomplish the
## downloading/uploading process
## Default: 300
MAX_TRANSFERER_LIFETIME = 300

## Period of time, which the newly joined machine waits from requesting pool
## versions to selecting the best of them
## Default: 2 * (HAD_CONNECTION_TIMEOUT + 1)
NEWLY_JOINED_WAITING_VERSION_INTERVAL= 5

## Period of time between two successive sends of classads to the collector by HAD
## Default: 300
HAD_UPDATE_INTERVAL = 300

## The HAD controls the negotiator, and should have a larger
## backoff constant
MASTER_NEGOTIATOR_CONTROLLER = HAD
MASTER_HAD_BACKOFF_CONSTANT = 360

## The size of the log file
MAX_HAD_LOG = 640000
## debug level
HAD_DEBUG = D_COMMAND
## location of the condor_had log file
HAD_LOG = $(LOG)/HADLog

## The size of replication log file
MAX_REPLICATION_LOG = 640000
## Replication debug level
REPLICATION_DEBUG = D_COMMAND
## Replication log file
REPLICATION_LOG = $(LOG)/ReplicationLog

```

Machines that are not potential central managers also require configuration. The following is a sample configuration relating to high availability for machines that will *not* be central managers.

```

#####
# Sample configuration relating to high availability for machines      #
# that DO NOT run the condor_had daemon.                               #
#####

#unset these variables
NEGOTIATOR_HOST =
CONDOR_HOST =

## For simplicity define a variable for each potential central manager
## in the pool.
CENTRAL_MANAGER1 = cm1.cs.technion.ac.il
CENTRAL_MANAGER2 = cm2.cs.technion.ac.il
## List of all potential central managers in the pool
COLLECTOR_HOST = $(CENTRAL_MANAGER1),$(CENTRAL_MANAGER2)

```

```

##-----
## Host/IP access levels
##-----

## Negotiator access.  Machines listed here are trusted central
## managers.  You should normally not need to change this.
HOSTALLOW_NEGOTIATOR = $(COLLECTOR_HOST)

## Now, with flocking (and HA) we need to let the SCHEDD trust the other
## negotiators we are flocking with as well.  You should normally
## not need to change this.
HOSTALLOW_NEGOTIATOR_SCHEDD = $(COLLECTOR_HOST)

```

3.3.22 Configuration File Entries Relating to High Availability

These macros affect the high availability operation of Condor.

MASTER_HA_LIST

Similar to `DAEMON_LIST`, this macro defines a list of daemons that the *condor_master* starts and keeps its watchful eyes on. However, the `MASTER_HA_LIST` daemons are run in a *High Availability* mode. The list is a comma or space separated list of subsystem names (as listed in section [3.3.1](#)). For example,

```
MASTER_HA_LIST = SCHEDD
```

The *High Availability* feature allows for several *condor_master* daemons (most likely on separate machines) to work together to insure that a particular service stays available. These *condor_master* daemons ensure that one and only one of them will have the listed daemons running.

To use this feature, the lock URL must be set with `HA_LOCK_URL`.

Currently, only file URLs are supported (those with `file:...`). The default value for `MASTER_HA_LIST` is the empty string, which disables the feature.

HA_LOCK_URL

This macro specifies the URL that the *condor_master* processes use to synchronize for the *High Availability* service. Currently, only file URLs are supported; for example, `file:/share/spool`. Note that this URL must be identical for all *condor_master* processes sharing this resource. For *condor_schedd* sharing, we recommend setting up `SPOOL` on an NFS share and having all *High Availability condor_schedd* processes sharing it, and setting the `HA_LOCK_URL` to point at this directory as well. For example:

```

MASTER_HA_LIST = SCHEDD
SPOOL = /share/spool
HA_LOCK_URL = file:/share/spool

```

A separate lock is created for each *High Availability* daemon.

There is no default value for `HA_LOCK_URL`.

HA_daemon_LOCK_URL

This macro controls the *High Availability* lock URL for a specific daemon as specified in the configuration variable name, and it overrides the system-wide lock URL specified by `HA_LOCK_URL`. If not defined for each daemon, `HA_daemon_LOCK_URL` is ignored, and the value of `HA_LOCK_URL` is used.

HA_LOCK_HOLD_TIME

This macro specifies the number of seconds that the *condor_master* will hold the lock for each *High Availability* daemon. Upon gaining the shared lock, the *condor_master* will hold the lock for this number of seconds. Additionally, the *condor_master* will periodically renew each lock as long as the *condor_master* and the daemon is running. When the daemon dies, or the *condor_master* exists, the *condor_master* will immediately release the lock(s) it holds.

`HA_LOCK_HOLD_TIME` defaults to 3600 seconds (one hour).

HA_daemon_LOCK_HOLD_TIME

This macro controls the *High Availability* lock hold time for a specific daemon as specified in the configuration variable name, and it overrides the system wide poll period specified by `HA_LOCK_HOLD_TIME`. If not defined for each daemon, `HA_daemon_LOCK_HOLD_TIME` is ignored, and the value of `HA_LOCK_HOLD_TIME` is used.

HA_POLL_PERIOD

This macro specifies how often the *condor_master* polls the *High Availability* locks to see if any locks are either stale (meaning not updated for `HA_LOCK_HOLD_TIME` seconds), or have been released by the owning *condor_master*. Additionally, the *condor_master* renews any locks that it holds during these polls.

`HA_POLL_PERIOD` defaults to 300 seconds (five minutes).

HA_daemon_POLL_PERIOD

This macro controls the *High Availability* poll period for a specific daemon as specified in the configuration variable name, and it overrides the system wide poll period specified by `HA_POLL_PERIOD`. If not defined for each daemon, `HA_daemon_POLL_PERIOD` is ignored, and the value of `HA_POLL_PERIOD` is used.

MASTER_<name>_CONTROLLER

Used only in HA configurations involving the *condor_had*.

The master now has the concept of a "controller" daemon, typically with the *condor_had* daemon serving as the controller process. With this setup, all "daemon on" & "daemon off" commands to the controlled daemon are directed to the controller daemon, which then handles the command, and, when required, sends appropriate commands to the *condor_master* to do the actual work. This allows the controlling daemon to know the state of the controlled daemon.

As of 6.7.14, this should be specified for all configurations using *condor_had*. To configure the *condor_negotiator* to controlled by *condor_had*:

MASTER_NEGOTIATOR_CONTROLLER = HAD

HAD_LIST

A comma-separated list of all *condor_had* daemons in the form IP:port or hostname:port. Each central manager machine that runs the *condor_had* daemon should appear in this list. If HAD_USE_PRIMARY is set to True, then the first machine in this list is the primary central manager, and all others in the list are backups.

All central manager machines must be configured with an identical HAD_LIST. The machine addresses are identical to the addresses defined in COLLECTOR_HOST.

HAD_USE_PRIMARY

Boolean value to determine if the first machine in the HAD_LIST configuration variable is a primary central manager. Defaults to False.

HAD_CONNECTION_TIMEOUT

The time (in seconds) that the *condor_had* daemon waits before giving up on the establishment of a TCP connection. The failure of the communication connection is the detection mechanism for the failure of a central manager machine. For a LAN, a recommended value is 2 seconds. The use of authentication (by Condor) increases the connection time. The default value is 5 seconds. If this value is set too low, *condor_had* daemons will incorrectly assume the failure of other machines.

HAD_ARGS

Command line arguments passed by the *condor_master* daemon as it invokes the *condor_had* daemon. To make high availability work, the *condor_had* daemon requires the port number it is to use. This argument is of the form

`-p $(HAD_PORT_NUMBER)`

where HAD_PORT_NUMBER is defined with the desired port number. Note that this port number must be the same value here as used in HAD_LIST. There is no default value.

HAD

The path to the *condor_had* executable. Normally it is defined relative to \$(SBIN). This configuration variable has no default value.

MAX_HAD_LOG

Controls the maximum length in bytes to which the *condor_had* daemon log will be allowed to grow. It will grow to the specified length, then be saved to a file with the suffix .old. The .old file is overwritten each time the log is saved, thus the maximum space devoted to logging is twice the maximum length of this log file. A value of 0 specifies that this file may grow without bounds. The default is 1 Mbyte.

HAD_DEBUG

Logging level for the *condor_had* daemon. See SUBSYS_DEBUG for values.

HAD_LOG

Path to the log file.

REPLICATION_LIST

The following list contains entries for all known replication daemons in the pool for the configuration machine, including the port numbers for each one of them. Normally each machine's replication list must contain the same entries, necessarily in the same order.

STATE_FILE

This file is protected by the replication mechanism. It is replicated between all the replication daemons listed in `REPLICATION_LIST` configuration parameter. The default is `$(SPOOL)/Accountantnew.log`

REPLICATION_INTERVAL

This parameter determines how frequently the replication daemon wakes up to do its periodic activities: probing for update of the state file, broadcasting the update to backups, monitoring and managing the downloading/uploading process by transferer processes etc. Since the accounting information file normally changes, as negotiator daemon wakes up, then `REPLICATION_INTERVAL` value must be like `NEGOTIATOR_INTERVAL`. That is why default `REPLICATION_INTERVAL` equals to default `NEGOTIATOR_INTERVAL`. The default is 300.

MAX_TRANSFER_LIFETIME

This parameter is intended to handle the case of stuck transferer processes: be it a network interruptions or huge state file or any other reason. During this period of time the transferers must download the file from leader to backup machine. We advise this parameter be equal to $2 * \text{Average size of state file} / \text{network rate}$. Therefore the default is 300.

NEWLY_JOINED_WAITING_VERSION_INTERVAL

Before entering the pool, newly joining machine must download the best replica of the state file from the pool. In order to do that, it requests for the state file versions from all the pool machines and waits certain amount of time to let the machines react and send their versions to it. Its value depends on the effective network rate, set it to higher value than default, if the networking is really slow or if the security is enabled. The default is $2 * (\text{HAD_CONNECTION_TIMEOUT} + 1)$

HAD_UPDATE_INTERVAL

HAD sends classads to the collector once in a period of time, defined by this configuration parameter in order to let the collector know the most updated information about the state of HADs in the pool. Upon each change in the HAD state (like turning from active to passive or vice versa), the collector is being updated immediately. The default is 300.

HAD_USE_REPLICATION

This configuration parameter enables administrator of the machine disable/enable the replication feature on Condor machine configuration level. The default is no.

REPLICATION_ARGS

Command line arguments passed by the *condor_master* daemon as it invokes the *condor_replication* daemon. To make replication work, the *condor_replication* daemon requires the port number to use. This argument is of the form

`-p $(REPLICATION_PORT_NUMBER)`

where `REPLICATION_PORT_NUMBER` is defined with the desired port number. Note that this port number must be the same value here as used in `REPLICATION_LIST`. The default: is no.

REPLICATION

The path to the *condor_replication* executable. Normally it is defined relative to \$(SBIN). The default is no.

MAX_REPLICATION_LOG

Controls the maximum length in bytes to which the *condor_replication* daemon log will be allowed to grow. It will grow to the specified length, then be saved to a file with the suffix .old. The .old file is overwritten each time the log is saved, thus the maximum space devoted to logging is twice the maximum length of this log file. A value of 0 specifies that this file may grow without bounds. The default is 1 MB.

REPLICATION_DEBUG

Logging level for the *condor_replication* daemon. See SUBSYS_DEBUG for values.

REPLICATION_LOG

Path to the log file. The default is no.