THE CONDOR BASED CDF CAF

I. Sfiligoi, INFN LNF, Frascati, Italy
E. Lipeles, M. Neubauer, F. Würthwein, UCSD, La Jolla, CA, USA

Abstract

The CDF Analysis Facility (CAF) has been in use since April 2002 and has successfully served 100s of users on 1000s of CPUs. The original CAF used FBSNG as a batch manager.

In the current trend toward multisite deployment, FBSNG was found to be a limiting factor, so the CAF has been reimplemented to use Condor instead. Condor is a more widely used batch system and is well integrated with the emerging grid tools. One of the most useful being the ability to run seamlessly on top of other batch systems.

The transition has brought us a lot of additional benefits, such as ease of installation, fault tolerance and increased manageability of the cluster. The CAF infrastructure has also been simplified a lot since Condor implements a number of features we had to implement ourselves with FBSNG. In addition, our users have found that Condor's fair share mechanism provides a more equitable and predictable distribution of resources.

In this paper we present the new implementation of the CAF based on Condor as well a general description of the CAF principles and the Condor batch system itself.

THE CDF ANALYSIS FARM

The CDF Analysis Farm (CAF) [1] is a portal which allows a user to easily perform physics analysis on remote computing clusters. It was originally designed primarily for the CDF computing cluster at Fermilab but has since been deployed at many sites around the world. The basic functionality and user interaction is as follows:

1) Users develop and debug their applications on their desktops or laptops anywhere in the world.
2) When the user produces the desired executable, he submits it to the CAF, splitting the dataset in several independent subsets called sections. The user's working directory, containing the executable(s) and auxiliary files, is automatically archived and transferred by the submission tool.
3) At the execution site, the submitted directory is recreated and the user startup script is executed.
4) The user output is copied to a user specified location; usually to a central file system pool, but in principle it could be anywhere in the world, including the submission node.
5) After all the sections terminate, a comprehensive e-mail is sent to the user.

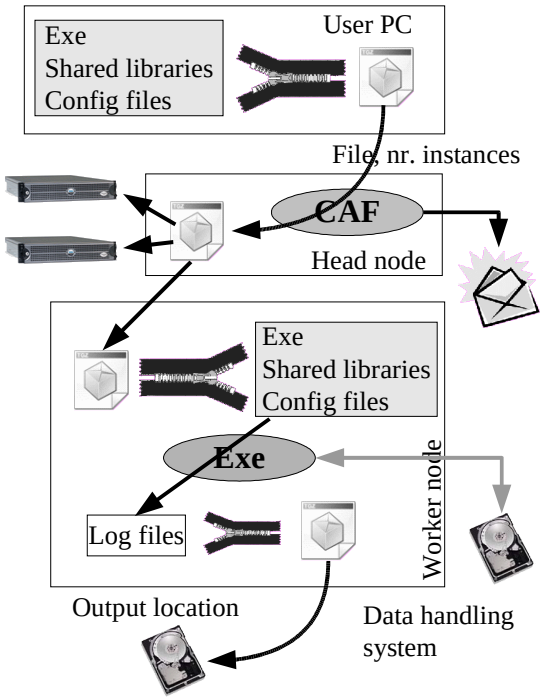

Figure 1: CAF job life

For user convenience, both a command line interface and a GUI interface are available for submission. A Web service portal is also envisioned but not yet implemented.

In addition to the simple submission interface, the system provides tools to easily monitor a section as if it was running on a local computer. The above is obtained by both a command line interface and via a set of Web pages. In particular, the CAF implements the following commands:

- jobs - Shows the list of your jobs/sections.
- top, ps, dir, and tail – Perform these standard commands on a worker node/working directory specified by a job and section id.
- debug – Runs a debugger (gdb) session on a running process of a specific section (see Fig. 2)

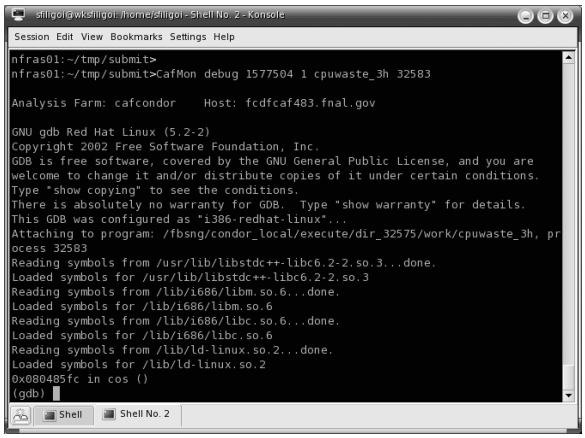

Figure 2: Debugging a CAF section

A set of administrative tools available to the user are also implemented. Due to authentication, the administrative actions are implemented only via command line tools, while the administrative monitoring is implemented also as a set of Web pages. The administrative tasks are:

- system status – Shows the load and memory consumption of a node or of the whole system
- slot occupancy – Shows which section is running on a given resource
- kill – Kills a specific job/section
- hold - Holds a job.
- release - Releases a job.
- chprio - Changes the relative priority of a users jobs.

THE CONDOR BATCH SYSTEM

Description

The Condor [2] batch system is a widely deployed system aimed at solving the High Throughput Computing [3] problem. It is a very modular and scalable system, able to perform well on both a dedicated cluster and on opportunistic resources. Condor is based on a marketplace paradigm; jobs ask for resources and the resources sell themselves.

A Condor pool consists of 3 logical node classes, as described in Fig. 3:

1) submission nodes,
2) worker nodes, and
3) a central manager node.

In CDF, we use a configuration with just one submission node and lots of worker nodes. The central manager must be unique by design and is the part that characterizes a Condor pool.

The jobs are submitted to the **schedd** process which stores them in a permanent storage and advertises their needs. On the other hand, the **startd** processes on worker nodes advertised their resources to the **collector** process.

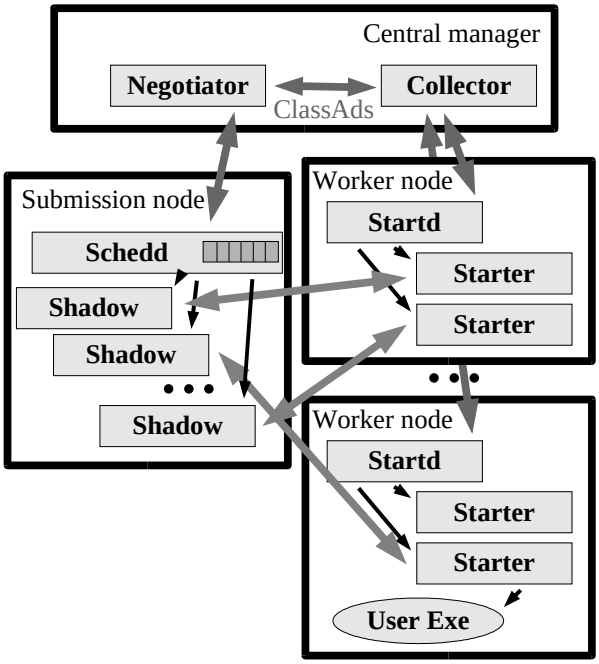

Figure 3: Condor overview

The negotiator process regularly fetches these advertisements (ClassAds) from the collector and from the schedd, and assigns jobs to virtual machines (VM). For every such association, a **shadow** and a **starter** process are created and all further communication is between these two entities.

Fair share is maintained based on recently consumed wall clock time. Condor keeps track of total wall clock time consumed by each user in a pool, including an exponential decay of past consumption. The user who has used the least gets the largest share of the available resources. For more details see [1].

In addition, the administrator can define a multiplicative factor which allows different (classes of) users to have different shares of the resources.

The CDF pool at Fermilab

The CDF pool at Fermilab is composed of about 340 worker nodes, each having two Xeon processors. Enabling the HyperThreading and using a little bit of overcommitment, we allocate six VMs for each node, creating a pool of over 2k VMs. Not all the VMs are allowed to be used all the time; one is reserved for test jobs allowing them immediate access to resources, while the others accept new jobs only if the load of the system is not too high and there is enough memory available for efficient running.

The Condor pool is used mainly for user analysis, and presently has 785 registred users, out of which 40 to 100 are active at any given moment in time.

All authentication is based on Kerberos. On the head node, an automated submitter daemon runs under one account and uses kerberos credentials for authentication. On the worker nodes, jobs run under nobody-like accounts, one account for every VM of the node. This prevents different jobs from interfering with each other. In particular, this mechanism protects both a single user, running on two different VMs on the same node, from himself, and a user from another, since the two sections are always running under two different UIDs.

We also use kerberos for authentication between Condor daemons, making possible secure data transfers between them.

THE CONDOR BASED CAF

Job submission

The CAF infrastructure essentially serves as a portal to the Condor batch system. The **submitter** process listens on a well known port and establishes kerberized python sockets. Once the user is authenticated:

1) A staging directory is created.
2) The user tar ball is uploaded.
3) Condor submit files are created.
4) A DAGMan job is submitted.

When the Condor DAGMan starts, it submits the user sections as Condor jobs. We use a flat DAG, with just the SAM start section as a dependecy. For more details about DAGMan see [1].

We also have a **mailer** process that looks after the DAGMan jobs and acts after any of them terminate; the main tasks are cleanup and sending a mail to the user.

The reason to have a separate mailer process instead of doing this as part of the DAG, is because we want to run those tasks after every DAGMan job. If you kill a DAGMan job, no more nodes are executed in the DAG.

Section execution

The assignment of sections to virtual machines is handled directly by Condor. We also rely on Condor to transfer all the necessary data, including the CAF wrapper, to the worker nodes and to transfer the log files back to the head node. Some files, like the kerberos keytab file, are encrypted during the transfer due to security issues, while others are transferred in clear to keep the system load low.

Once the files are on the worker node:

1) The user kerberos ticket is extracted from the keytab file.
2) The keytab file is **deleted.**
3) The user tar ball is unpacked into the local directory.
4) The tar ball is deleted (to save space).
5) The user provided startup script is executed.
6) The exit code is recorded in a log file.
7) The local directory is tarred and sent to the user specified output location (via kerberized rcp), logging the success or failure in the log files.
8) All the local files, apart from the log files are removed.

Condor transfers back any remaining files.

Moreover, our wrapper does also a little bit of monitoring. In particular, it monitors the processes that a user runs during the lifetime of his job and log them in a local log file to preserve this information.

System monitoring

The CondorCAF has two processes to do the system monitoring: the **monitor** process and the **xml_monitor** process. Both of them use the same monitoring libraries, but talk different protocols; the first communicates over kerberized python sockets while the later uses XML over the telnet protocol.

These monitoring processes allow access to information about jobs, VMs, and user priorities. This information is gathered both by parsing log files and by querying the Condor batch system:

1) The user priorities are gathered by calling **condor_userprio.**
2) The information about the VMs is obtained by calling **condor_status.**
3) The information about the jobs is gathered by parsing log files. We cannot use **condor_q** since this easily overloads Condor's schedd. Unfortunately, the log files lack information about which VM a job is running, so we use the information gathered by **condor_status** and write this information into yet another log file.

   In addition, we also parse the IO logs, provided by some applications, and the process logs, provided by the CAF wrapper.

Interactive monitoring

One of the most outstanding services of the CAF is the interactive monitoring which is implemented by means of handshaking between the CAF wrapper and a CAF router. The whole process looks like this (see also Fig. 4):

1) The user contacts the monitor process via a kerberized python socket.
2) The monitor process contacts the local **cafrout** process and obtains an ID.
3) The monitor process starts a Condor CoD (Computing on Demand) session to the section's worker node. The CoD session writes the ID and the desired command to a local unix pipe on the worker node and exits.
4) The unix pipe is read by the CAF wrapper which calls back the cafrout process on the head node (the head node name and cafrout port are given at job submission time).

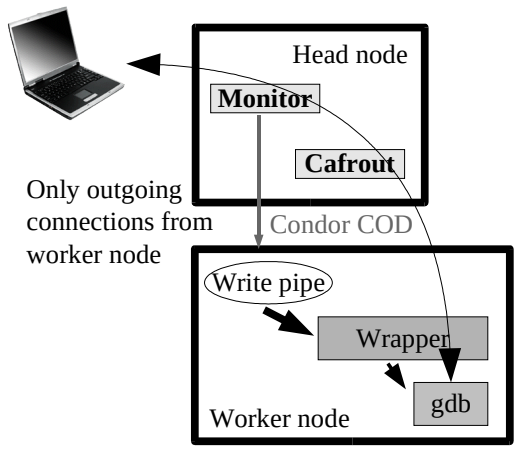

Figure 4: Interactive monitoring

5) After handshaking, the user command is executed and both input and output are linked to the open socket.
6) On the head node, the cafrout process routes the data between the CAF wrapper socket and the monitor socket. On the other hand, the monitor process does the routing between the cafrout socket and the user kerberized python socket.

Web monitoring

Apart from the command line monitoring, the CondorCAF comes with a ready to use Web site software.

The Web software works in polling mode. Every few minutes (configurable) a snapshot of the system is taken, by talking to the xml_monitor process, and stored on the Web server. Any request from a Web client is thereafter served by analysing the data stored locally.

In addition, the full information about any finished job is also stored on the web server. So there is really never a query to the head node due to a HTTP query, insulating the head node from the Web activity.

CONCLUSIONS

The new Condor based CAF was developed 10 months ago and has been in production now for more than 6 months. After the initial problems with the Condor batch system due to the size of our pool and the number of queued jobs, solved by successive versions of the package, Condor has been working very smoothly. Most of our users are very happy with the transition from the FBSNG-based CAF to the Condor-based CAF, especially due to the much more “fair” negotiation policy.

The Condor system has also significantly simplified the CAF software; most of the file transfers now performed by Condor were previously the task of the CAF infrastructure and in particular the kerberos keytab file transfer was a problem and required suid executables on the worker nodes.

However, we do have still some problems with Condor. The fist problem is the difficulty to optimally configure the pool. Condor has only a few levers to limit the load on the submitter node. Since we use a single submitting node, it is relatively easy to overload it if not properly configured.

The second, larger, problem, is the lack of group priority policies. In the Condor system, every user is like any other user and all the negotiation is based on the priority numbers of the individual users; the user with the best priority number wins. There is no way to impose quotas or other group policies.

We are however working closely with the Condor group to solve all the problems, including the inclusion of a Hierarchical Fair Share (HFS) mechanism, and we are very satisfied by the collaboration. Although sometime the desired extensions do not come as fast as we would like, they are doing a terrific job considering the number of customers they are serving.

FUTURE DIRECTIONS

At Fermilab we still have in production the old FBSNG-based CAF. This was done for two reasons:

1) To have a ready-to-use backup solution in case the Condor-based CAF were fail during its first months of life
2) Condor is not able of managing groups, while in our FBSNG-based CAF we have implemented a mechanism to handle that

We expect to have a first version of HFS available in a few weeks time frame. As soon as we have it, we will implement the groups in the Condor-based CAF. After we are confident it is working for us, all the nodes of the old CAF will be transferred to the new one.

The next obvious step is to run user analysis on resources distributed all over the world. We are taking an incremental approach to this. At this point in time, we have CAFs replicated in several sites; refer to [4] for more details. Moreover, we are trying to run Condor-based CAF on a generic Grid site by using Condor Glide-Ins submitted via a Global Gatekeeper. We, then, want to link all this sites together using Condor-C. The exact details about job brokering have not yet been defined, but we hope to have a working system by next summer.

BIBLIOGRAPHY

[1] T.H.Kim et.al., The CDF Analysis Farm, IEEE NSS 2003 proceedings.

[2] The Condor Project Homepage, http://www.cs.wisc.edu/condor/

[3] High Throughput Computing, http://www.cs.wisc.edu/condor/htc.html

[4] A.Sill et.al., Globally Distributed User Analysis Computing at CDF, CHEP2004 proceedings.