FERMILAB-CONF-04-365-CD

The CDF Central Analysis Farm

T.H.Kim*,M.Neubauer†, I.Sfiligoi‡,L.Weems§, F.Würthwein† *MIT, Cambridge, MA †UCSD, La Jolla, CA
‡Laboratori Nazionali di Frascati, INFN, Frascati, Italy §FNAL, Batavia, IL

***Abstract*— With Run II of the Fermilab Tevatron well underway, many computing challenges inherent to analyzing large volumes of data produced in particle physics research need to be met. We present the computing model within CDF designed to address the physics needs of the collaboration. Particular emphasis is placed on current development of a large O(1000) processor PC cluster at Fermilab serving as the Central Analysis Farm for CDF. Future plans leading toward distributed computing and GRID within CDF are also discussed.**

***Index Terms*— CDF, high energy physics, computing, batch, grid, physics analysis**

I. Introduction

RUN II at the Fermilab Tevatron began in March 2001 and will continue to probe the high energy frontier in particle physics until the start of the LHC at CERN. The accelerator facility underwent a major upgrade for increased energy ($\sim$10%) and instantaneous luminosity ($\times$10) over that attained in Run I. With a goal of attaining 9 fb^{-1} of integrated luminosity over Run II for each experiment, a very rich and exciting physics program[1] at Fermilab is expected over this decade.

In order to operate at the upgraded Tevatron and to exploit the physics potential of the new beam conditions, the CDF detector also underwent a major upgrade[2]. By the end of Run II, it is expected that the CDF collaboration will write up to 6 PB of data onto tape. Providing efficient access to such a large volume of data for analysis by hundreds of collaborators world-wide will require new ways of thinking about computing in particle physics research.

II. Computing requirements

It is important to understand the data and software characteristics involved before one sets out to solve an analysis computing problem.

The data produced by the CDF reconstruction is permanently stored in an STK tape robot. The Enstore software system developed at FNAL provides the interface layer to network-attached tape drives in the robot. In cases where fast, frequent access to large amounts of tape data is required, significant read and write cache disk is employed. The dCache[3] software system from DESY provides a scalable network-attached front-end disk cache to Enstore. In addition, an Oracle-based database system is used to store and provide access to metadata and calibrations.

In the context of CDF data analysis, we are trying to process a very large number (10^7 or more) of relatively small (tens to hundreds of kBytes) independent data elements. As such, we have the relative luxury of speeding up a typical analysis job through parallel processing of independent subsets of the job. Some additional characteristics of the CDF data are:

- Root I/O as the persistent data format
- Typical reconstructed (raw) data size of 30-150 (250) KB/event
- Typical Run dataset size of 10^7 events, with some of them going into the 10^8 range

TABLE I
CDF Computing needs for Run II.

Fiscal Year	CPU (THz[1])	Disk (TB)
2003	1.0	180
04	3.7	280
05	9.0	610
2006	13.9	840

The AC++ framework is used within CDF to develop the analysis software. Several standard modules are available to be used by the final user, while he has to write his own modules for his specific analysis. The typical analysis job is meant to select the interesting events and write out a standard ntuple ROOT file. Some additional characteristics of the CDF analysis software:

- Typical analysis jobs run at 5 Hz on 1 GHz P3 (or 11 Hz on a 2.66 GHz P4), corresponding to a few MB/sec input rate
- Analysis jobs are CPU rather than network I/O bound over Fast Ethernet

The CDF collaboration is composed of a large number of physicists, with several of them being active at the same time. Our current experience is showing us that roughly 100 of them are active on a typical day, each submitting 100-400 jobs a day. We expect to scale to roughly 200 concurrent users in the near future.

Our goal is to provide sufficient computing resources to allow each of these users to process a typical secondary dataset (e.g. produce standard ntuples) in one day.

The computing requirements to achieve this goal given our data and software characteristics are shown in Fig. I.

III. Central Analysis Farm (CAF)

The CAF grew out of the need to maximize the amount of computing we can provide for CDF at more or less fixed

[1]"THz" is relative to 1 GHz P3 performance

cost both in terms of hardware as well as human capital to operate the system. Fiscal pressures as well as the scale of the analysis computing challenge lead to a large batch based cluster of commodity PC hardware.

A. Computing model for CDF central analysis

A user develops and debugs his application on his desktop anywhere in the world. Real data are used to verify the correctness of the executables. To do so we provide low bandwidth access to all CDF data files from anywhere in the world interactively.

When the user produces the desired executable, he submits it to the CAF, splitting the dataset in several independent subsets. The CAF user interface forms a gzipped tar archive and sends it for execution to the CAF cluster.

At the CAF site as many instances of the user tar archive are submitted to the batch system as defined by the user at submission time. At execution time, the archive is unpacked, and the user's shell script is invoked with whatever input parameters declared at submission time. One of the input parameters is an integer to distinguish between different instances of the same archive. It is then up to the user to implement the details of the parallelization based on this integer.

After the user shell script terminates the CAF creates a tar archive of the user working directory on the local node in the cluster. An intelligent user will thus delete all temporary files from their working directory before exiting their shell script except. This tar file is then copied to a location defined by the user at submission time for job status verification or for input into a subsequent CAF analysis job. In principle, the output location may be anywhere in the world.

In practice most of the time the user selects the CAF local scratch space (50 GB allocated per user). This scratch space may be accessed transparently using a set of environment variables defined by the CAF for the user. The user may access their scratch space via ftp and rootd from outside the CAF, and via ftp, rsh, rcp, fcp, and rootd from inside the CAF. We refer to this as *icaf* to indicate that the intended use is as staging area for CAF output, sort of like imap for email.

The CAF is thus receiving one tar archive with the application, and sending out as many tar archives as there are instances of the user application requested at submission time.

Once all instances of a given submission have terminated, the CAF will parse a set of CAF logfiles created for this submission, and write a summary report to be emailed to the user. The objective with this email report is to provide the user with a quick overview of how well their submission completed. The body of the report provides sufficient information for the user to determine which instances have failed, as well as the reason for failure if known. It is thus very easy for a user to go back and debug individual instances by either inspecting the core and log files they received back with the output tar archive, or by running a specific instance interactively through a debugger.

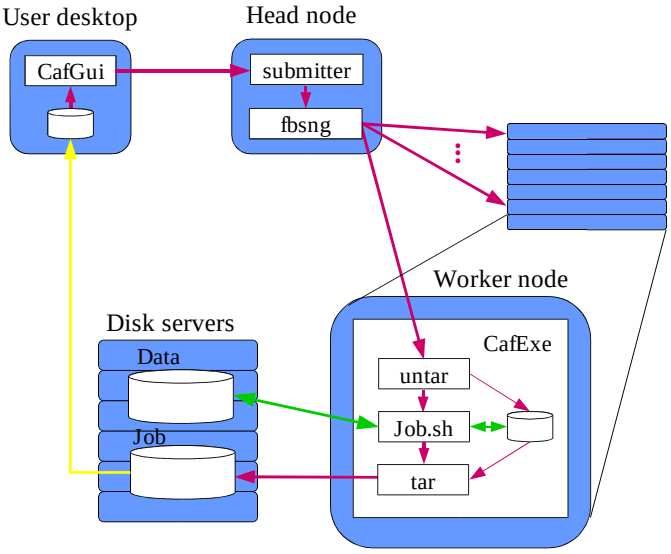

Fig. 1. CAF Overview

B. Hardware Details

The CAF is functionally comprised of three types of hardware :

- *worker nodes*, where users' jobs are executed,
- *storage nodes*, which host the data accessed by the analysis jobs running on worker nodes,
- *infrastructure nodes*, which provide important utilities for the CAF (head node, database nodes, code servers, etc.).

The CAF is currently composed of ∼300 worker nodes, with another ∼250 to be rolled in by the year end. The present worker nodes are a mixture of 1U and 2U dual CPU machines, each with a single Fast Ethernet interface and a reasonably sized scratch disk.

The mass storage is composed of ∼90 2 TB storage nodes, with another ∼20 5 TB storage nodes to be rolled in by the year end. The server nodes are selfcontained IDE RAID 4U and 5U server units (disk, CPU, network device, etc.), each with at least one Gigabit Ethernet interface and hot swap capabilities.

Of the currently installed storage nodes, ∼25 are used for the CAF scratch space, while the others are integrated into the CDF dCache pool.

The CAF scratch space can be accessed using either *kerberized rcp*, *kerberized FTP* or *kerberized rootd*. Files on the dCache servers are instead served via dCache's protocol which is similar to rfio. i.e. a remote file I/O that is POSIX like.

C. Software Details

The design goal for the CAF software is to provide users with secure access to CAF resources (batch CPU, scratch disk, data handling system) from their desktops anywhere in the world. To successfully implement such a system, we need to work within several somewhat contradictory design constraints and desirables:

- *FNAL security policy*: Kerberos authentication lab-wide

- *Job scheduling*: proven batch system, configurable, fair share capability, local support → FBSNG[4]
- *Administrative ease*: no user accounts, jobs run under single 'cafuser' UID
- *User identity*: Unique privileges for batch jobs and disk resources

Before submitting a job, the user groups all the necessary data under a subdirectory. The needed data include the executable, all the necessary shared libraries, the startup script and any other configuration file needed by the job.

Once everything is in place, the user can invoke a kerberos client interface, specifying the required information about the job (local path to the subdirectory, startup script to run, number of desired parallel sections, output URL). The interface authenticates the user (i.e. presents a valid kerberos ticket) to a *submitter* server daemon running on the *head node* which receives the job information and a tarball containing the specified subdirectory. The communication is achieved by means of a proprietary protocol via a kerberized TCP/IP socket. Once all the information has been received, the server daemon performs the actual job submission to the FBSNG Batch Manager (bmgr), one FBSNG job for every user requested section, plus a mailer job that will send the final e-mail.

Once a user's job is scheduled and sufficient CPU resources become available, a standard executable (CafExe) common to every worker node is launched with

- a common 'cafuser' UNIX user ID
- appropriate parameters to completely specify the user's job
- a kerberos principal unique to the user, generated from a single "service" principal on the farm

CafExe copies over and unpacks the user's tarball from the head node, sets up the proper environment for CDF analysis software, and runs the user's shell script which in turn executes whatever the user has specified.

After the user shell script terminates, CafExe tars up the user's working directory and sends the tar archive to the location requested by the user. The user-specific kerberos principal is used for unique access privilege to the output location(s).

Have also a look at Fig. 1.

D. Batch manager policy

Fair share is implemented using the concepts of queues and process types (ptypes). FBSNG allows us to specify parameters that govern the fair share algorithm between queues and allows to set quotas and time limits for process types.

We use these to create *small*, *medium*, and *large* ptypes with time limits ranging from 2 to 24 hours of CPU and form 4 to 48 hours of wall clock time. To guarantee sufficient response time for *small* we restrict *medium* and *large* to a total quota of less than the total number of CPU's available in the farm. Fair share between users is established by having a queue per user.

While most of the CPU and disk resources are owned by the collaboration as a whole and commonly funded, we also allow institutions to contribute funds to the CAF. We then guarantee that their users receive first pick of idle resources up to the level of their contribution. As a result, roughly 1/3 of the CAF was funded from sources outside the common fund.

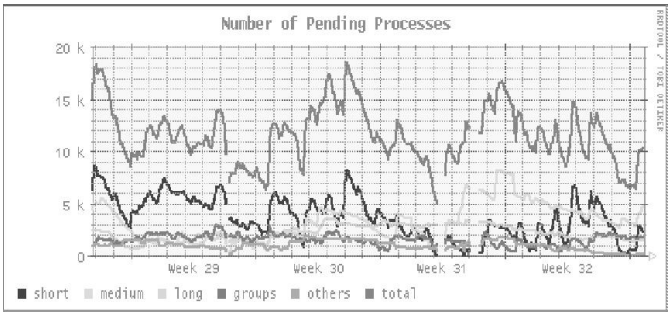

Fig. 2. Monitoring example - Number of job sections waiting to be run

We implement this by requiring the user to select the proper group queue. The drawback of this method is the arising of anomalous situations where, due to high group activity, users sometimes get less CPU if submitting to the group queue instead of the general one.

IV. The icaf service

The CAF scratch space is spread over several physical storage nodes, but the user should not be aware of this. Moreover, when a user needs to be moved from a storage node to another due to administrative needs, this should be completely transparent to the user and as simple as possible for the system administrator.

The *icaf* service is our solution to the problem. Running on the head node, the *icaf* server daemon works as a directory service. After identifying the requester (by means of the presented kerberos ticket), information about the user's physical storage node and remote directory name is sent back.

The information about users and storage nodes is kept in a plain text file, one line for every registered user. This format is the most readable one both for humans and programs for this type of information.

To make the service more useful, a full set of tools has been developed:

- *icaf_info* simply prints out the information obtained by the *icaf* server,
- *icaf_get*, *icaf_put*, and *icaf_rm* implement the most used operations of data movement,
- *icaf_quota* shows the users quota,
- while *icaf_gftp* implements a full featured X-based kerberized ftp client with a one-click pointer to the icaf area

V. CAF Monitoring

While the CAF is fundamentally a batch based system, we were unwilling to sacrifice the core functionality provided by an interactive system. We thus implemented not only the usual batch monitoring functionalities but also a core set of services that allow a user to watch their jobs as if they were running on their local desktop instead of a remote cluster.

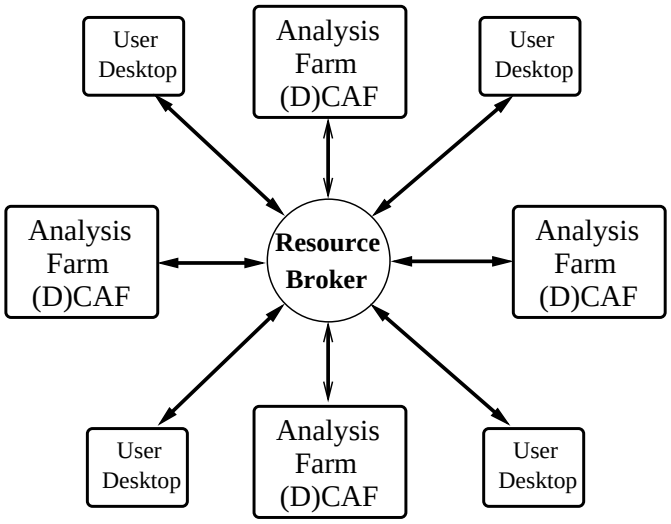

Fig. 3. A distributed CAF

The standard batch functionalities are Web based. Using his favorite browser, a user can monitor both the status of his jobs and the status of the system as a whole. Among other details, the web based monitoring provides the number of jobs waiting to be executed, as can be seen in Fig. 2.

The interactive like services are *ls*, *tail*, *top*, and *debug* equivalents. The first three allow the user to obtain information about the local environment a given instance of their job is executing on without the need to know where that environment is located. All the user needs to specify is the instance and submission ID. The debug service allows the user to attach a gdb session to their running executable. To do this the user needs to specify the UNIX PID in addition to section and job id. The user may look up the latter on the CAF monitoring pages.

VI. Toward the grid

SAM[5] is another product under intensive evaluation for use in CDF data handling. SAM provides distributed data access, flexible dataset history and management, and optimizations for limited fabric resources (e.g. network or tape bandwidth), and has been demonstrated to work with Enstore and the CDF analysis software framework.

The distributed nature of the collaboration and the ever increasing demand of CPU power is pushing the CDF collaboration to distribute its CPU and disk resources all over the world. Especially institutions outside the USA want to keep their resources inside the funding country.

To address these needs, we are looking toward a more distributed structure, like the one described in the Fig. 3.

We have started by installing the current software in several locations and were running real analysis jobs on them for a while. Although these sites are working, they have some drawbacks:

- The sites are independent and they do not communicate with each other.
- The user must explicitly select the site to run on.
- Each site has its own (sub)set of available data.
- The use of FBSNG outside FNAL is very limited, so it was difficult to find local expertise at the remote sites.

For all of the above reasons, we are now preparing to move our CAF software to a Condor batch manager. This will solve all our problems except the data problem; Condor is able to move jobs between sites and, due to much wider establised user base, expertise is much easier to find around the globe.

To solve also the data problem, we are integrating SAM in our CAF infrastructure and into the CDF analysis framework. We envision to use SAM as the resource broker inside the distributed CAF and as the input interface for the CDF analysis jobs.

VII. Conclusions

The Central Analysis Farm has proved to be a very good solution for the CDF computing needs. Since its introduction, the resources available to the user has been increased by 2 order of magnitude over the SMP model used in Run 1, with the same amount of money spent. Users are also pleased by the system and find it very easy to use and very helpful in error diagnosis.

As more and more users are using the system, the system is proving to be stable and scalable, and we are able to add new resources keeping the needed manpower essentially constant.

We feel very satisfied with the current implementation inside FNAL, but the need of a distributed model is pushing us for some serious rework of the software. We are though looking forward to the substitution of FBSNG with Condor and to the introduction of SAM into the core CAF services.

References

[1] D. Acosta et al., The CDF Collaboration, *The CDF Experiment at the Tevatron - The First Two Years of Run II*, FERMILAB-PUB-03/162-E, 2003.

[2] CDF Collaboration, *The CDF II Technical Design Report*, FERMILAB-Pub-96/390-E, 1996.

[3] M. Ernst et. al., *dCache, a distributed storage data caching system*, Proceedings of CHEP 2001, Science Press, 2001.

[4] J. Fromm, K. Genser, T. Levshina, I. Mandrichenko, *FBSNG - Batch System for Farm Architecture*, Proceedings of CHEP 2001, Science Press, 2001.

[5] A. Baranovski et. al. , *SAM Managed cache and processing for clusters in a worldwide grid-enabled system*, FERMILAB-TM-2175, 2002.