CDF/DOC/COMP_UPG/PUBLIC/5961
The CAF development group
May 14, 2002

T. Kim, M. Neubauer, F. Würthwein
Massachusetts Institute of Technology
R. Colombo, G. Cooper, R. Harris, R. Jetton, A. Kreymer, I. Mandrichenko, L. Weems
Fermi National Laboratory
S. Belforte, M. Casarsa, S. Giagu, O. Pinazza, F. Semeria, I. Sfiligoi, A. Sidoti
INFN Italy
Y. Gotra, J. Boudreau
University of Pittsburgh
F. Ratnikov
Rutgers University
M. Paulini
Carnegie Mellon University

CDF CAF
Design Document — Stage1

Abstract

This document describes the Stage1 implementation of the CDF Central Analysis Farm (CAF). As such it focuses on the functionality and implementation of the CAF for Summer 2002 physics analysis. In addition, we point out the missing functionality that we intend to add throughout the summer and fall of 2002, and sketch an outlook where this project is going within the next couple of years. Please consult CafUtil/doc/UserGuide.ps in CVS if all you want to know is how to use this system to get your physics done.

Contents

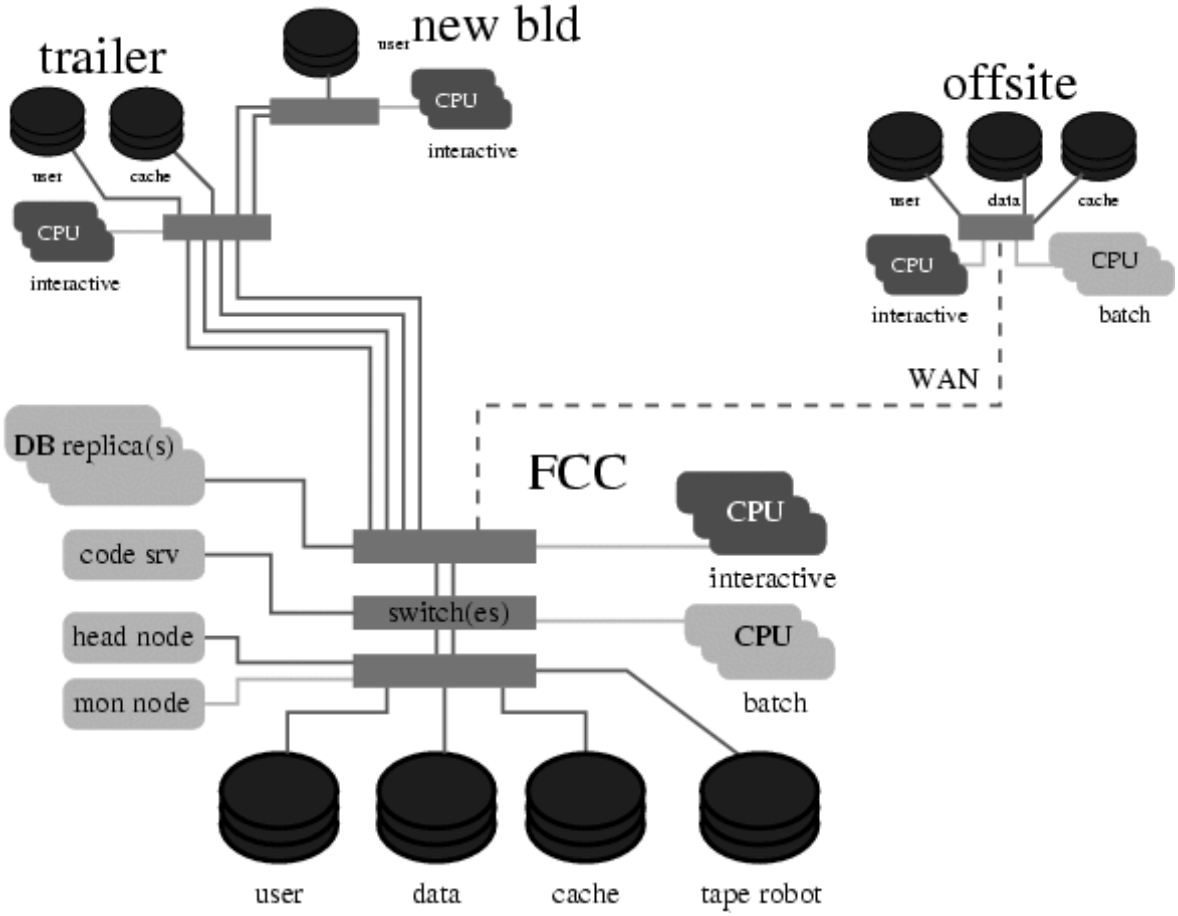

Figure 1: Schematic of the CDF Run II computing hardware arrangement. See text for a detailed description.

1 CDF Computing Model for Run II

The CDF computing model for Run II is to rely heavily on commodity computing equipment that is network attached, and may be accessed from anywhere in the world. The main restrictions to this general theme are imposed by the desire for simplicity and security.

Simplicity demands that we support only the Linux/Intel platform except for the legacy Irix/SGI which does not allow access to the main computing resources, and may be phased out over the next few years. Security constraints demand that we require the user to have at least a kerberos client, and preferably a kerberos server installed as well.

Figure 1 shows a schematic of the various hardware components in this model. There are essentially three types of facilities that we foresee: the central analysis farm (CAF) in FCC, the B0 trailer office computing, and offsite computing facilities which we will refer to

as "decentralized CAF" or DCAF.

Our initial focus (FY02) is on the CAF in FCC, as well as the mechanisms which allow users to access the CAF from anywhere in the world. In FY03 we then expect to expand the model to include DCAF and integrate the B0 trailer computing more tightly via the possible addition of a hierarchical data cache.

The remainder of this document focuses on the Stage1 implementation of the CAF which is being completed in May 2002. Let us start by describing some of the features of Figure 1. There are two types of computing resources, interactive (dark) and non-interactive (light). The bulk of the CPU power is in the form of non-interactive batch CPU. The bulk of the interactive CPU is on people's desks and laps in the form of desktop and laptop PC's. In addition, there is a small amount of interactive computing in FCC to establish an "authoritative reference platform" as well as a place for users without access to Linux/Intel. Data may be stored either on user, data, or cache disks, or may have to be staged from tape prior to accessing it.

For Stage1 we expect to have a production quality system that relies heavily on data that resides statically on user and/or data disks. The CDF DH group in collaboration with FNAL-ISD is in the process of developing a cache system based on dCache and Enstore. We expect this to become available for general users during Stage1. The data may be accessed from either interactive or batch CPU, independent of location but obviously limited in performance by the LAN and WAN bandwidth. At present the LAN connection between FCC and the B0 trailers is a single Gigabit Ethernet connection (dark links). We expect to increase this to up to 4 Gigabit Ethernet links dependent on needs over the next couple of years. Beyond that we expect to deploy up to four 10-Gigabit Ethernet links when those become available and are needed. While data disks are served by file servers via Gigabit Ethernet links, all CPU power is attached in the form of Fast Ethernet (light links) to the network.

Four types of computing equipment are needed to provide the CAF infrastructure. Those are shown in light grey as they are all non-interactive, of course. We expect to deploy as many database replicas as needed to guarantee that database access is not a performance limiting factor. While it is not drawn explicitly in Figure 1 we nevertheless expect that DCAF sites will want their own database replica(s). Two DB replica options are actively pursued within CDF. In the CAF we are likely to deploy Oracle based replicas on Linux SMP's while offsite DCAF sites may prefer MySQL based replicas. We expect both options to be available by the end of FY02. The CAF itself depends on a head node for its operations, and its performance is monitored from a mon node. A central code server provides a uniform CDF software base to interactive as well as batch CPU in the CAF. Towards the end of FY02 we expect to add a second code server in the B0 trailer to replace the aging and poorly performing ncdf09 code server. CDF software installations on these code servers are installed there from a central build platform (cdfpca) which is not shown. We intend to provide these code servers with sufficient bandwidth such that user builds on the interactive nodes are (almost) as fast as if the software was local.

The data path from online via tape robot to production farm and back into tape robot is also not shown. I.e., from the perspective of the CAF, data exists after it is run through standard reconstruction on the production farm. Re-reconstruction of raw data is in principle possible on the CAF but is expected to be done on the production farm. The CAF's purpose is primarily to provide compute power and data access to secondary datasets for physics analysis.

Additional details on the deployment plan and budget for this computing model may be found in Ref. [1]. The CAF deployment plan is described in detail in Section 3. Details on the data handling system for Run2 may be found in Ref. [2].

2 CAF Overview

CDF has in place a very powerful and easy to use code management and distribution system which allows users anywhere to maintain fully functional CDF software releases on their Linux/Intel computers. The basic idea behind the CAF is to extend this "compile and build anywhere" computing model with an "execute from anywhere" functionality, and at the same time provide sufficient CPU power and data access to satisfy the collaborations physics needs.

A model user is one who writes, compiles, and debugs his/her analysis code on a flight from Chicago to Boston on a laptop Friday evening, submits (e.g. via cable modem connection at home) a batch job to analyze a few TB worth of data, and comes back in to work on Monday morning to download the resulting histograms and/or core dumps.

Our model user would go through the following steps in this process:

1. Compile and debug the analysis executable on a desktop or laptop.
2. Write a little shell script that sets up any environment variables, including the CDF software version, and executes the executable. Or more likely, copy such a shell script from somebody else, and tailor it slightly to his/her needs.
3. Start up CafGui, fill the appropriate fields, and submit the job. The fields to fill are as follows:
   - The command to execute, and how many instances thereof.
   - The top level directory below which everything job specific (other than data) may be found.
   - Location where the job output should be copied to.
   - Type of batch process (e.g. short, medium, long). This is equivalent to the familiar concept of "short, medium, and long queue".

- Optional email address to notify upon job completion.

4. Upon submission the CafGui tar's and gzip's the specified directory, contacts the batch system at FNAL (Kerberos authentication required), and copies the tarball to FNAL. The user receives a job ID from the batch system. This job ID may be used to:

   - Follow the job status via a web or command line interface.
   - Kill the job.
   - Peek at any log files the job creates while running in batch @ FNAL.
   - Obtain a directory listing on the remote batch PC that executes the job, as well as any file server that may hold input data the job is using.

   The job ID may also be determined via the web interface. I.e., the user is not required to memorize job IDs of jobs she or he submits.

5. Job output may be handled in one of the following ways:

   - A gzip'ed tar archive of the user directory on the batch PC after completion of the job may be copied to one of the following locations:
     - The user desktop if that desktop allows incoming kerberized rcp.
     - A user scratch area that CDF maintains for all users, and from which it can be retrieved via ftp. A kerberized ftp GUI was developed to allow for convenient file manipulations.

     Output written out in this fashion requires user handling before it can be read back into a subsequent user job on the CAF.
   - Output files that are meant to be read in by subsequent analysis jobs on the batch system are best written directly by the user's executable to avoid unnecessary compression and de-compression. The following locations may be written to:
     - A user scratch area that CDF maintains for all users.
     - Disk space a University group may provide for its members, which is then maintained by CDF. (*available after August 1st 2002 only due to space constraints in FCC*)
     - The Data Handling system, and thus to tape. (*Available towards the end of FY03*).

     Output files written out in an appropriate format in this fashion may then be read back in into a subsequent user job on the CAF for further processing. There will be access to these files via rootd from the user's desktop in the B0 trailer, subject to the obvious LAN bandwidth limitations between trailers and FCC. Similarly, it is in principle possible to write from the CAF directly to a user's desktop if that desktop has a rootd daemon running that accepts kerberized write. This works except for the obvious bandwidth limitations.

The only requirements from this user should be:

1. Access to a CDF software installation on a Linux/Intel platform.
2. Kerberos client software and principal at FNAL.
3. Some minimal internet access.
4. Some minimal knowledge of unix. The CAF team provides example shell scripts that the user needs to be able to tailor to her/his needs.

The CDF CAF will be deployed crudely speaking in four stages over the course of the next two years. These stages are defined primarily by hardware deployment, and to a lesser extent by goals for CAF functionality. Their timing is given largely by the desire to deploy new hardware resources in time for the major summer conferences. The size of each deployment is driven by the Tevatron luminosity profile, as well as the observed CAF usage.

In Section 3 we define the timeline, as well as the hardware deployed in each period as foreseen today. The CAF as described here is intended to easily scale up to $O(1000)$ PC's and $O(100)$TB of disk space. The rational for this size was provided by the Fall 2001 review of the Central Analysis Computing, and is summarized in Section 3.

Section 6 provides the software implementation details for the Stage1 CAF as it exists today, while Section 4 discusses the hardware choices.

All of the CAF infrastructure software is maintained in the CDF CVS in the package CafUtil. The package and related issues are discussed further in Section 6.11.

3 CAF Sizing Estimate

CDF central analysis computing underwent a thorough review between mid August and November of 2001. This review benchmarked [4] a large variety of CDF applications, estimated the physics needs [5] based on input from the physics groups, and summarized their findings in a final report [6] that resulted in the birth of the present CAF project.

In this section we summarize the findings of this review, as well as the assumptions made in order to justify the scope of the CAF project. The single most important finding of the CAF computing review may be stated as follows:

There isn't a single imaginable CDF application that would be I/O limited given the presently foreseeable CPU/bandwidth ratio of commercial computing equipment.

	Lumi $[fb^{-1}]$	single CPU [GHz]	total CPU [THz]	Duals purchased	Duals total
FY02	0.3	1.3/1.8	0.18/0.58	69/160	229
FY03	1.2	2.5	0.96	192	421
FY04	2.5	3.5	1.34	192	613
FY05	4.1	5.0	2.29	+240-229	624

Table 1: Example purchasing profile to satisfy the CDF user analysis needs with the CAF. The "Duals purchased" column reflects the number of additional compute nodes needed in each year. The FY05 purchase numbers include replacement of the FY02 computing.

In particular, a Dual CPU PC is unlikely to have sufficient CPU power until after 2005 to saturate a single FastEthernet link. This statement is based on 100kB event sizes and 5GHz CPU's in 2005. In fact, given the respective growth in network bandwidth and CPU power it seems hard to imagine that network bandwidth to the CPU's will ever be a significant issue throughout Run II. More details on how this determines the CAF hardware choices may be found in Section 4.

The basic assumptions that guide the sizing of the CAF are as follows:

- Secondary datasets will be produced in an organized and centralized fashion by only a handful of users.
- (Re)Tracking of large volumes of data will be done in an organized fashion by only a handful of users.
- The purpose of the CAF is to provide general user computing to analyze (and re-analyze) secondary datasets.
- The CAF should enable 200 simultaneous users to analyze roughly 5nb cross section each within a matter of days.
- The CAF purchasing plan is based on the present luminosity profile for the Tevatron, as well as a doubling of CPU power every 18 months.

Based on the measured performance of the CDF software, and the above guidelines we arrive at a compute power need of roughly 1THz/fb^{-1}. We then add 10% for R&D, assuming that we will want to be able to set aside 10% of the CAF at all times for development purposes. In FY02 we have an additional pedestal due to CAF contributions from individual institutions as well as increased computing needs for commissioning the detector and reconstruction software and data format.

In addition to CPU power we also need large amounts of disk space. Table 2 shows the requests by the physics groups at the February 15th 2002 joint physics group meeting.

	Data	MC	user/other	Total
Top/EWK	11	18	18	47
Exotics	13	9	18	40
QCD	41	10	20	71
Bottom	63	20	40	123
Total	128	57	96	281

Table 2: Example disk space needs projection. The units here are TB of disk space needed per $2fb^{-1}$ of data taken. We use the total in the lower right hand corner to arrive at the requirements profile versus time listed below.

	Disk [TB]	Servers	LAN BW/srv [MB/sec]	LAN NIC	source/ sink
FY02	97	43	70	1	11
FY03	209	75	110	1	7.5
FY04	385	107	175	2	7.0
FY05	558	96	280	2	6.5

Table 3: Example file server specs versus time. "Servers" is the total number of file servers in the CAF each year. Next comes the expected bandwidth out of a newly purchased server each year, the number of Gigabit Ethernet ports per newly purchased server, and finally the source/sink ratio.

At that meeting each of the four physics groups presented their estimates for the amount of static disk space they need based on their Run I experience, as well as Run II aspirations. For the purpose of this document we have taken the liberty of rescaling the numbers presented at that meeting from an assumed 150kB per event to the desired 100kB per event. To be conservative, this was done only for the data column.

We then combine the physics groups request from Table 2 with the luminosity profile from Table 1 to arrive at the disk space needs profile versus time as presented in Table 3.

Table 3 also shows the aggregate file server I/O assuming the technology as described in Section 4. For present technology the bandwidth I/O out of a file server onto the network is CPU limited. The change in I/O performance over time takes Moore's Law scaling of the CPU's into account. Whenever Moore's Law scaling warrants we increase the number of GigE ports per file server.

As a sanity check we also include the bandwidth ratio $\frac{\text{source}}{\text{sink}} = \frac{\text{I/O from disk}}{\text{I/O into CPU}}$. The latter number needs to be significantly larger than one such that statistical fluctuations in the data access patterns do not lead to I/O bandwidth bottlenecks at the data servers. We intend to use the Stage 1 implementation of the CAF to gain sufficient operational

	Racks	LAN ports [Gig/FE]	switch modules	switches
FY02	17	50/230	9	2
FY03	27	85/422	15	3
FY04	38	230/614	28	4
FY05	36	210/625	28	4

Table 4: Space and LAN needs based on the worker nodes and file servers. The LAN port count includes infrastructure nodes of various types. We assume 16 GigE and 48 FE per module and Cisco 6513 switches.

experience to understand to what extend the desired factor of $\frac{\text{source}}{\text{sink}} > 5$ is sufficient for smooth operations of the CAF.

Last but not least, Table 4 shows the resulting space and LAN requirements based on the assumption of 4U per file server and 1U per Dual PC worker node. The port count includes fcdfhead1, fcdfmon1, fcdfcode1, as well as some DB replicas. The switch module count includes ports on the switch for inter-switch connections. For the total switch count we assume that a switch allows 11 modules (Cisco 6513) or equivalent, and we deliberately do not fill all the switches with modules as we expect to be aggregating ports into switches based on physics groups or datasets. We do not expect to allow all ports to be connecting to each other in a random fashion. Instead we expect to be using some sort of partitioning to keep traffic largely within rather than across switches.

4 CAF Hardware Choices

The hardware choices for the CAF are driven by the following three simple facts:

1. The total CPU power needed is large: $O(1000)$ CPUs.
2. The total data volume is large: $O(100)$ TB of PAD data.
3. The ratio of I/O bandwidth per CPU clock cycle is very small: $O(1)$ MB/(second$\times$GHz)

The first of these requires us to look for the most clock cycles per \$\$, thus leading to the same worker nodes as the CDF production farms. The only difference is that we decided to opt for 2GB RAM per worker node. This is twice the amount presently used on the production farm, and eight times that of Level3. We decided on this generous amount of RAM because it is still not a major cost factor, and we have no control over the executables that will be used on the CAF.

The second fact requires us to look for the most cost effective disk space. Cost effective may not be equated with "cheap" in this case. In particular, the mean time between failures for 100TB's worth of 100GB disks is 10years/10^3 = 3days. I.e. we ought to expect roughly two disk failures per week. In order to minimize operational hazzle we require all disk systems to be RAID5, hot-swappable, and equipped with some sort of hardware monitoring system that sends out emails to a specified operator whenever a disk fails. The operator will then walk through the isles once a week and replace the failed disks together with their hot-swappable canister.

The third fact implies that we may use network-attached disk in conjunction with worker nodes connected via FastEthernet links. Based on the benchmarking results we estimate a Dual P3 1GHz worker node to analyze 10Hz of 100kB events, or 1MB/sec of data. Detailed benchmarking of file servers [7] indicates that a Dual 1GHz file server equipped with a single Gigabit Ethernet link may source up to 50MB/sec of data, depending on the number of simultaneous clients. The limiting factor appears to be the CPU power in the server system rather than the disk subsystem. For local read in a RAID50 arrangement we have achieved as much as 190MB/sec for two threads, and 120MB/sec for 32 threads. Beyond 32 threads the performance was limited by the available 1GB of RAM in the test system.

Based on these measurements we expect that a file server is capable of serving 30-50 worker nodes simultaneously. This has to be compared to the ratio of worker nodes/file servers of 69/16 = 4.3 for Stage1. From this we conclude that we have built in a comfortable order of magnitude "dynamic range" to allow for fluctuations in the way the worker nodes access the filer servers. This was referred to as source/sink ratio in Section 3. Using the protoCAF of one file server and 16 worker nodes we presently monitor the file server and worker node network I/O, as well as their CPU utilization to gain operational experience. In addition, we are in the process of implementing a control mechanism that would manage the worker node access to the file servers and eliminate excessive overloading. This is discussed in more detail in Section 7.

The file servers we are buying for Stage1 are configured as follows:

- Dual 1.4GHz PIII with 512kB L2 cache.
- Intel SDS2 motherboard.
- Dual 3ware Escalade 7850 RAID-IDE controllers with 8 Maxtor 160GB drives each.
- 2GB of registered ECC RAM.
- SysKonnect 9843 GigE fiber NIC.

The worker nodes are mostly Dual 1.26GHz PIII nodes with 2GB SDR SDRAM, except for the 16 Dual Athlons used in the protoCAF which have only 512MB DDR SDRAM at this point. We expect to upgrade the memory on these systems.

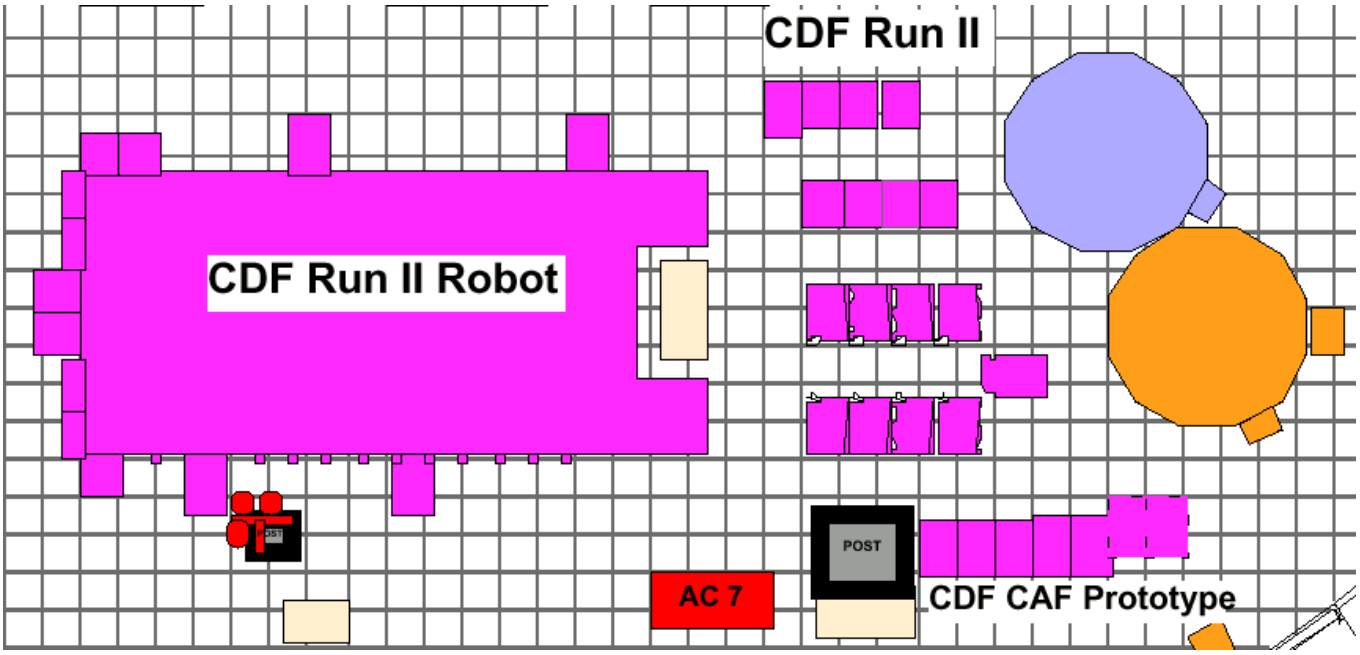

Figure 2: Layout of the physical space in FCC 2nd floor. The Stage1 CAF occupies the seven racks referred to here as "CDF CAF Prototype".

5 CAF Physical Layout

Figure 3 shows the layout of the physical space in FCC 2nd floor. There is no more space in FCC2 for the CAF to expand beyond Stage1. Stage2 and beyond will thus be housed on the 1st floor of FCC.

Figure 3 shows the layout of the physical space in FCC 1st floor. There is space for 37 racks for the CDF CAF in the space that was formerly occupied by ACPMAPS, the decommissioned Lattice QCD cluster. A rack holds 32 1U worker nodes, or 8 4U file servers. 600 1U worker nodes and 60 4U file server would thus use up roughly 26 of the 37 racks. This makes it clear that we have some amount of flexibility with respect to space but can not afford for all worker nodes to be housed in 2U chassis.

6 CAF Implementation Details — Stage1

In this section we describe the existing Stage1 implementation of the CDF CAF in some detail. All CDF specific software referred to here is maintained in CVS in the package CafUtil. A guide to the package as well as products the CAF depends upon is provided in Section 6.11. The first frozen release for which this package exists is 4.5.0.

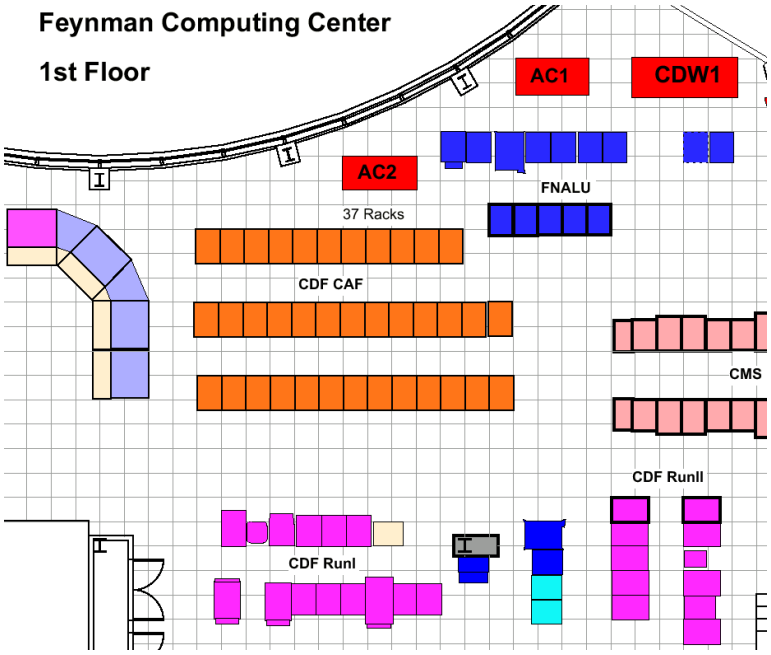

Figure 3: Layout of the physical space in FCC 1st floor. There is space for 37 racks for the CDF CAF in the space that was formerly occupied by ACPMAPS, the decommissioned Lattice QCD cluster. A rack holds 32 1U worker nodes, or 8 4U file servers. 600 1U worker nodes and 60 4U file server would thus use up roughly 26 of the 37 racks.

6.1 Overview of Infrastructure Software

User interactions with the CAF are built based on client-server pairs that use Kerberos to authenticate the user.

All the CDF specific components are discussed in detail below. For discussion of fbsng, the batch system used on the CAF, we refer to Ref. [3]. The configuration of fbsng as used in the CAF is discussed in Section 6.6.

Figure 4 depicts the various components, and how they interact for the purpose of job submission (solid line), retrieval of analysis results (dashed line), and command line driven user monitoring and control (dotted line).

In the present section we provide a brief overview of how the components work together to provide the desired functionality. The following sections then describe the implementation of the various components in some detail.

The basic client-server pairs are listed below. In each case the client is listed first.

- CafGui $\leftrightarrow$ "submitter" for job submission from user desktop to CAF. Submitter runs on fcdfhead1. CafGui is started by user on the user desktop.
- "cafkill/cafjobs/cafdir/cafhostdir/caftail" $\leftrightarrow$ "monitor" for user interaction with their job on the CAF (e.g. kill job, peek at log files, directory listing). Monitor runs on fcdfhead1 node. The tools cafXXX are command line tools started by user on the user desktop. Monitor uses rsh to execute commands on remote nodes within the CAF.
- ICAF tools for retrieval of the gzipped tar archive of the working directory after the user application exits. This is discussed in detail in Section 6.10.

In the following we briefly describe the general flow of a job from user submission to user notification about the results. A schematics of this is depicted in Figure 4.

A user submits a job using the CafGui. The GUI is a client contacting the submitter service to transfer the necessary information to the submitter for submitting the job in the appropriate fbsng queue.

Once the job is submitted, the fbsng batch manager (bmgr) takes over. When appropriate resources are available, bmgr launches the CafExe on a worker node using the fbsng launcher.

The CafExe receives all the information it needs to know for starting up the user application via command line arguments that are specified when the submitter submits the job to fbsng. Details on the CafExe may be found in Section 6.5.

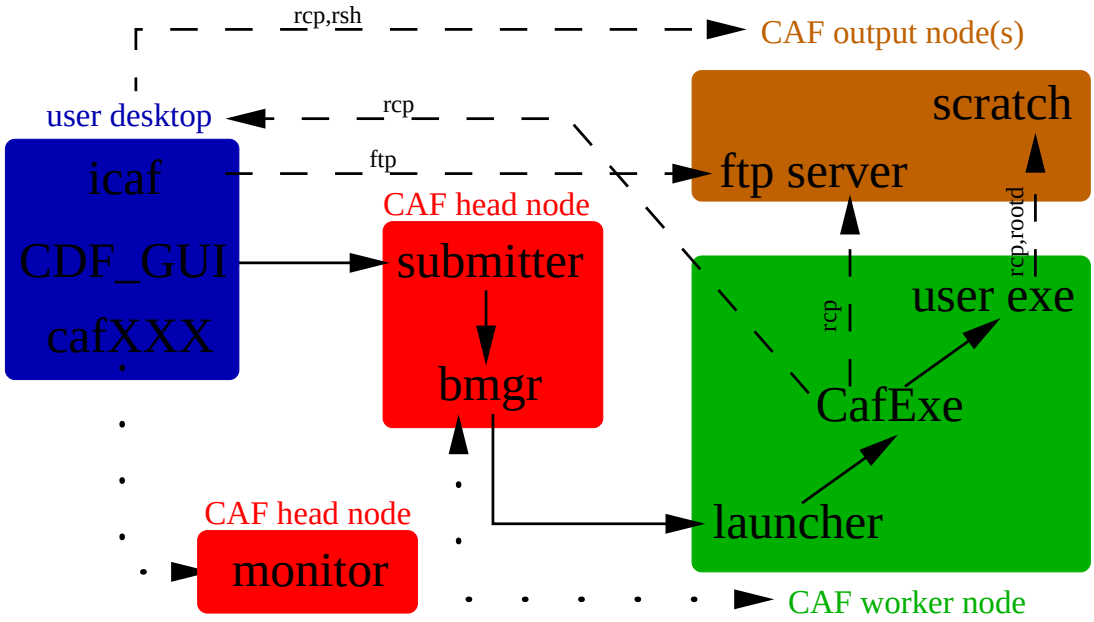

Figure 4: Schematics depicting the various CAF software components, and where they run. Solid/dashed/dotted lines depict connections for submission/output/monitoring. The arrow points away form the object that initiates the communication or data transfer. The available protocols for output data transfer are also indicated. For additional details see text.

The CafExe produces a gzipped tar archive of the user working directory on the worker node after the user application exits. This archive is then copied to a location specified by the user at time of submission.

The submitter daemon on the head node forks a process for every client. This process exits after the application tarball is transferred, and the job is submitted to fbsng. The execution time for a submitter process is dominated by the time it takes to scp the user application tarball from the user desktop. We expect this to be on the order of minutes. The CAF design assumes 200 job submissions within a couple of days. We expect that one submitter will suffice even if this was off by an order of magnitude or two.

We expect that the head node is not fully loaded from the submitter alone. Assuming this is correct we will also run fbsng bmgr on fcdfhead1, the CAF head node.

All components other than submitter and bmgr easily scale by implementing multiple copies of each object. E.g. we intend to have a fixed number of output nodes that are shared among users. A given user will have their icaf as well as scratch space located on the same file server. Quotas will be enforced. In the icaf area a cron job will delete all files older than one month automatically.

As an aside, we note that attaching the CAF to the emerging Grid infrastructure for job submission [9] as a future development direction is by no means ruled out. We comment on this in some more detail in Section 8.3.

6.2 Unix Accounts on the CAF

All actions on the CAF are performed from within a very limited number of user accounts as follows:

- **cdfcaf**: Account from within which the CAF "submitter" service is run as discussed below. A Kerberos service principal exists for this account on fcdfhead1 to allow remote access to all worker nodes in the CAF. We have equipped one of the worker nodes with a GigE NIC for use as fcdfhead1.
- **cafuser**: System accounts on worker nodes from within which the user application is executed. This account is strictly local to each worker node. Special Kerberos arrangements are made to identify the user whose application is running at any given moment. This is discussed in detail in Section 6.9.
- **cafmon**: Account from within which the CAF monitoring tools are run as described below. This account exists on fcdfmon1 as well as each worker node. A Kerberos service principal exists for this account on fcdfmon1 to allow remote access to all worker nodes in the CAF.

6.3 User interface for job submission

See the CDF code repository [10] CafUtil/doc/UserGuide.ps for details on how to use the CafGui user interface.

The intended use is for a user to submit a single "job" with many parallel "sections" to run through a dataset in an efficient manner. Each section is executed separately on a worker node. A job is defined by the executable tarball. I.e. each time a user successfully pushes the submit button on CafGui this creates a new job. For Stage1 each section results in a separate results tarball. Part of every submitted job is a special section that is run after all other sections have completed. This "clean-up" section parses the log files from all sections, and generates a summary email message that is sent to the user. No attempt at concatenating the analysis output from all sections is made for Stage1. We consider this a possible future extension of the CAF.

We deliberately do not support a command line interface to the CAF to provide an incentive not to submit gazillion jobs but to use a few jobs with gazillion sections instead. This is to avoid multiple remote copy of the same executable tarball.

This won't stop power users as they will easily turn the python that makes up the GUI into one that uses the submitter API from the command line. On the other hand, power users are hopefully aware of the copy overheads, and thus won't do something foolish. We monitor the number of jobs submitted by each user and will take appropriate measures if we find a user abusing the system.

The CafGui is started by simply typing CafGui after setting up the CDF software. It is supported from 4.5.0 onwards. The "binary" CafGui is actually a shell script that sets up the development version of the CDF software, and executes the development version of CafUtil/cdf_gui/CafGui.py, a python script. This was done such that users benefit from potential future upgrades of the user interface independent of the frozen release they happen to be using to build their application. As a side effect, a user needs to have access to a development release of the cdf software. We intend to include CafUtil into development-lite but have not done so yet.

6.4 Submitter service

Submitter sits between user and the fbsng batch manager (bmgr). It is a service that waits for clients to attach. The port number is specified in CafUtil/submitter/setting.py and hardcoded into CafUtil/cdf_gui/CafGui.py .

When a client attaches submitter forks a child process to deal with that client. Each child is independent of all other children, as well as of the parent.

Note: The fact that this forks a new process for every client means that we may be running many python programs in parallel (one for each child). This may be significant overhead at some point. We may want to rewrite the submitter in C when and if this becomes an issue.

Here's what happens in a child process:

1. authenticates user
   - after authentication submitter can access principal like fkw@FNAL.GOV . We cut off whatever is in front of @FNAL.GOV and use that as fbsng queue name to submit to.
2. receives basic info:
   - initial command to execute
   - process type
   - output location
   - email address
   - number of sections to execute
3. receives tar file
   - write tar file to submitter's disk at: cafIn/userName_pid.tgz (pid is the child's PID)
4. contact bmgr to submit all sections of the job using FBSNG python API. This is equivalent to:
   fbs submit blabla.jdf
   Using the API means that creation of blabla.jdf is unnecessary.
5. if submit successful return JID to user, and rename tar file to jid.tgz .
6. child exits.

Note: If a job started execution before the user tarball is renamed to jid.tgz then the CafExe will not be able to find the tarball, and thus fail. We implement retry after sleep in CafExe to guarantee that this does not happen.

bmgr is in charge of section from the time the submitter is finished to the time the section finishes executing.

bmgr starts CafExe with all the information it needs to retrieve the user tar file, start the user application, and rcp the gzipped results tar archive to the user specified output location. All of this information is specified in the command line. The latter is specified by the submitter at the time of submission of the job and its sections to fbsng.

6.5 CafExe

The CafExe is located in CafUtil/CafExe/CafExe.cpp . It's function is to control the user application and provide a job control interface for fbsng. This includes the following functionality:

- Handling of exceeded time limit for batch. CafExe catches SIGINT from launcher and does as follows:
  - if in the middle of user application exe, send SIGKILL to the user app.
  - if before user app. execution stop, and write error message into logfile.
  - if in the middle of tarring results up, or copying, write warning into logfile that tar/copy may be incomplete as launcher sends SIGKILL next and cleans up the whole directory !!!
- CafExe is launched with root privilege, learns the user identity as a command line argument, obtains a special CAF kerberos ticket for that user as discussed in Section 6.9, setuid into cafuser account, executes the user specified command, tars up the user directory on the worker node after completion of the user application, and uses the user specific CAF Kerberos ticket to copy the gzipped tar file to the user specified output location.

6.6 fbsng configuration used in CAF

The batch system chosen for use on the CAF is fbsng [3], the "Fermilab Batch System Next Generation". This choice was made for the following reasons:

1. fbsng satisfies our immediate needs for Stage1.
2. We have full access to the source code which is written in python.
3. We have received excellent support from one of the fbsng authors, Igor Mandrichenko (fnal-ISD). ISD is committed to support fbsng as a batch system for the computing facilities/farms that make up the "Grid". It is thus quite likely that only little effort on our part will be required to integrate into the emerging Grid infrastructure if and when we decide to do so. See Section 8.3 for additional discussion of this.

4. FBSNG comes with a suite of monitoring tools, including web based, read only monitoring . We thus received a first prototype user level monitoring tool for free. We have extended this tool as discussed in Section 6.7.1.

The remainder of this section briefly describes the Stage1 configuration of fbsng. For additional details consult [3] as well as CafUtil/fbsng. The latter directory contains the scripts used for configuring the batch system as is.

In addition to the concept of "job" and "section" already introduced above, we use two important fbsng concepts: "*Queue*" and "*Process type*".

Process type is used in fbsng to define resources a section needs to execute on a worker node. We use this in Stage1 to guarantee that there is one and only one section per CPU, i.e. two sections per Dual CPU worker node. In addition, we define short, medium, and long *Process type*. They differ by the maximum amount of CPU time a user application is allowed for execution on a worker node. The initial times are 2,6, and 48 hours. In addition to CPU time we also implement a wall clock time limit to kill sections that get "lost in cyberspace" and consume zero CPU time but also never finish. This happens with the CDF software at a rate of roughly 0.1-1% of all sections we have seen so far. However, we expect to be tuning these time limits throughout Stage1 based on the usage patterns we observe. In addition, we may specify the fraction of the CAF that each of these *Process type* is allowed to consume, as well as which worker nodes are allowed to run which *Process type*. The former is used to guarantee that the CAF is not filled up with long sections, thus blocking recently submitted short sections until the first of the long sections completes. In other words, a fraction of the CAF may be set aside for sections of *Process type* short.

Queue is used in fbsng to manage fair share policies between different users. In other words each user has their personal *Queue*. The batch system then guarantees that each user submitting jobs gets a fair share of the CAF resources. For details on the fbsng scheduling algorithm see Ref. [3].

In addition to these general *Process type* and *Queue* we also have defined a set of special *Queue* & *Process type* pairs. These were set up for two purposes:

- University groups who buy CPU power to be added to the CAF receive full use of these resources as elaborated below.
- A fixed number of CPU's may be singled out for official physics group usage like secondary/tertiary/... dataset creation.

In both of these two cases the intention is that designated users get to jump the backlog of sections waiting for execution whenever designated nodes become free. This way all users may use those nodes whenever they are not used by any of the designated users.

This is implemented by assigning a minimum priority to the special *Queue* that exceeds the maximum priority of the general *Queue*. The special *Process type* is used to designate the nodes on which the special *Queue* is allowed to execute its sections. We have verified that this setup has the desired effect on a small test system.

Finally, a special test queue was created that has higher priority, only a 10min time limit, and may be submitted to by all users. This is meant to provide users with a quick turn around queue to test their shell scripts and executables.

6.7 User interfaces for job monitoring

6.7.1 Web based tools

Two types of web based tools are available to monitor what's going on on the CAF:

- User level monitoring
- Admin level monitoring

Both share a common web entry point (http://cdfcaf.fnal.gov/) and are "read only". They differ mostly in the kind of information that is being presented. In the present document we discuss only the user level monitoring.

Using the FBSNG API (Application Programmer's Interface) a web based monitor has been implemented to allow users to check the status of submitted jobs and the CAF load.

The data are sampled and collected using APIs and RRDTool [8], so that a historical but compact database can be maintained. The update frequency for the various plots and tables range from once to twice every 10 minutes. Depending on browser, reloading the web page may be required to see the latest plots.

The introductory page (http://cdfcaf.fnal.gov/cgi-bin/caf/users/monitor) presents an overview of the whole system with the most relevant current and statistical information. This allows users to quickly reckon available resources:

- the load for the three different process types (short, medium, long)
- the number of running and pending sections
- an estimate of the waiting time.

Users can observe and print daily, weekly and monthly graphs showing the CPUs and CPU-time utilization for one or more queues.

Furthermore, links to the tabular version of the current status are provided. These values are the web version (fbswww) of the FBSNG command-line interface.

In the following we list a few questions and how a user might answer them using these tools:

- *How many sections do I have in the CAF?*
  Click on "Queues" near the top of main "User Monitoring" page. Columns "Ready" and "Running" display the number of pending and running sections. The column "waiting" is meant to indicate the number of sections that can not be executed because they depend on prior execution of some other section. Users do not have access to this feature of fbsng.

- *Which of the sections of my job is currently running?*
  Click on "Queues" near the top of main "User Monitoring" page. Click on your user name. A page appears that has a table with one row for each of your sections. The status of each section is visible. Clicking on the section name provides detail info on this section. This includes information on what node the section was/is being executed.

- *What sections are being executed where?*
  Click on "Nodes" near the top of main "User Monitoring" page.

- *Why aren't my sections executed despite the fact that there are idle CPUs?*
  Click on "Process Types" near the top of main "User Monitoring" page. You can see there the quota for each process type. Your sections may belong to a process type that has exhausted its quota.

- *How might I figure out if my sections are CPU or I/O limited?*
  First you figure out which node the section runs on that you want to look at more closely (see above). Then you go to the admin monitoring (http://cdfcaf.fnal.gov/cgi-bin/caf/admin/cafmon) and select: "View by node", "day", select the appropriate node, and click on submit. This provides you with detailed statistics on the node. Next do the same thing for the node you read data from and the node you write data to.

6.7.2 Command line tools

A number of kerberized clients form the user interface to the monitor service described in Section 6.8. These are command line tools as follows:

- To get the brief list/status of jobs in my queue:

```
> cafjobs
```

- To check the progress of a specific section:

```
> caflog JID sectionNumber
```

- Kill the job
  - Kill the specific ranges of sections:

```
> cafkill JID Section_Range_List(e.g. 1-5 8 20- )
```

  - Kill the whole job:

```
> cafkill JID
```

- Peek into the standard output files(logs):

```
> caftail JID sectionNumber filename [LINE]
```

In addition to these "job control" functions we also support remote directory listings as follows:

- List a section's "relative" path:
  cafdir JID sectionNumber [file/dir_name]
  Example1: "cafdir 417 3 ."
  will list all the files in the working directory of section thkim_3 of jobID 417
  Example2: "cafdir 417 3 ../../cdf/data"
  will list the NFS mounted data area as seen by your job executing in batch.

- List the file of specific node with absolute path:
  cafhostdir hostname file/dir_name
  Example1: "cafhostdir fcdfdata001.fnal.gov /cdf/data"
  will list all the files in the top level data area of fcdfdata001.fnal.gov .
  Example2: "cafhostdir fcdfdata001.fnal.gov cdfsoft/dist/releases"
  will tell you which releases are currently installed on the protoCAF. If there's something you need that isn't there send email to cdf_code_management@fnal.gov (cc to fkw@fnal.gov).

6.8 Monitor service

The monitor service is located in CafUtil/monitor/monitor.py .

Monitor contacts bmgr to determine status of JID, kill it, or peek at the log files(s) the user application may have created. The latter includes the stdout and stderr of the part of the user application that is started by CafExe. However, the interface is more general as is discussed in Section 6.7.2. The service it implements is contacted from remote clients as described in Section 6.7.2.

6.9 Security Issues

Apart from the usual concerns about the security of high performance computing systems this systems has some that are specific to the fact that we do not have any user accounts established on the worker nodes. In this section we discuss how we have addressed these issues.

Let us start with spelling out the requirements:

- There are no individual user accounts on a worker node. Each and every user's application is executed from within the same unix account on the worker node.
- A user application needs to be able to write to and read from some scratch disk space that is owned by this user and nobody else.
- The user is allowed to execute a regular shell script on the worker node. Users thus have all the possibilities available to them that unix supports. No attempt shall be made to restrict this.
- NFS mounting of disks shall be avoided if at all possible, especially for data read and write operations.
- Users shall not be allowed to execute any programs on the nodes that serve the scratch disks except rm,mv,cp,rmdir,mkdir in their directory.
- The total number of service principals shall be kept to a minimum.

We have satisfied these requirements as follows:

- FNAL agreed to add a new type of Kerberos principal to its KDC. A special CAF service principal, cdfcaf/fcdfhead1@FNAL.GOV, shall be allowed to create special caf/cdf/<user>@FNAL.GOV principals for each user. The tickets these principals receive are non-forwardable. This is to avoid "roaming users".

- We will use the cdfcaf service principal at fcdfhead1 to create a special caf/cdf/<user>@FNAL.GOV principal every time we establish an fbsng queue for a new user.
- The keytab file with all the caf/cdf/<user>@FNAL.GOV entries will be installed on all worker nodes such that it is root accessible only.
- CafExe, which is launched as root can thus use this keytab file to obtain a Kerberos ticket from the FNAL KDC for a given section and user on a given worker node. To do so it first sets the KRB5CCNAME environment variable to "FILE:/tmp/krb5cc_<user>_<pid>" (credential file), run kinit to create the credential file for this session, and change the owner of this credential file to the cafuser. The cafuser account can thus access this credential file. Once the child process is spawned, it will inherit the KRB5CCNAME environment variable and can access network resources using this credential file. A user has no access to the keytab file and can thus not obtain appropriate tickets. A user may add their caf/cdf/<user>@FNAL.GOV principal into their .k5login on their desktop if they wish to access resources on their desktop from the CAF. The CafExe renews the ticket every 12 hours to allow user applications to exceed the maximum ticket lifetime at FNAL. The CafExe destroys the ticket it created before exiting.
- Once every few months a CAF maintenance operation renews all principals with the KDC.
- The node that has the scratch area also runs an ftp server. A user can access their scratch and icaf areas on the output file server assigned to them using ftp. We provide a kerberized ftp GUI for this purpose.

We will then assign a fixed number of users to each file server for their scratch space, and enforce quotas on this space. The user has read/write access from their sections being executed on CAF worker nodes as well as from their desktop. However, no user can use up precious file server resources by executing anything on those nodes.

6.10 ICAF output model

The protoCAF required the results tar file to go straight back to the user desktop. This requires a Kerberos server to be run on the desktop in order to allow kerberized access into the desktop. Not all sites will want to support such a service.

Furthermore, the user may want their output ultimately on a node that is neither reliably nor permanently attached to the network. The obvious example being a laptop. Instead, the user may want to deal with their histograms in a similar fashion as they do with their email:

- receive all output tar files on a central server.

- connect to this server whenever convenient to check for newly arrived files.
- select some or all of the new files for download.
- optionally delete files at the server without downloading.
- automatically delete files on the server after successful download.

In addition, a cron job will delete all files older than one month automatically.

An example user may have compiled/debugged/submitted a job with 100 sections just before going home to eat. After dinner the user notices that the first section finished and checks the size of the output tar file. She then notices that the output is way bigger than expected. I.e. something went obviously wrong. To avoid exceeding her disk space quotas she then quickly kills the job using "cafkill", and leaves the output tar file from the one completed section to be downloaded from her desktop for detailed study the next day. Off she goes to drown her sorrow in the local pub.

The obvious solution for such a system is based on FTP; the FTP protocol itself provides directory listing, download and file removal. Using a kerberized FTP daemon/client pair, widely used inside the CDF collaboration, also the security concerns are solved. Unfortunatelly, all kerberized ftp clients widely available at the time of writing are command line oriented, so we implemented also a simple kerberized ftp GUI.

FTP itself is enough to move files around, but by itself still requires the user to know exactly where his files are located. As we intend to grow the disk space available to users over time we will want to be able to re-assign users from one file server to another without the user noticing. For this purpose an ICAF name server was also implemented. The only service the ICAF name server provides is to return, given the user kerberos ticket, a tuple containing the name of the ICAF file server, the tar.gz output directory and the user scratch directory.

The ICAF name server runs on the head node and its port number is specified in CafUtil/icaf/icafcli.py and CafUtil/icaf/icaf.py. The first file is a python module that incapsulates all the necessary steps to connect to the server while the other is the server itself.

The information of the ICAF name server are used by the command line tools (icaf_ls, icaf_get, icaf_rm, icaf_node and icaf_info) and by the kerberized ftp GUI (icaf_gftp).

6.11 CafUtil package and required products

A package called "CafUtil" was created to hold the various pieces of CAF software, documentation, and example shell scripts for users. The first frozen release for which this

package exists is 4.5.0.

The package itself depends on the development release because of the CafGui. In addition, CafGui depends on python, python_krb5, and an appropriate version of tcl/tk and icaf depends additionally on an appropriate version of kftp. All of these products are distributed with cdfsoft. Python is needed for CafGui, icaf, and the cafXXX suite of command line tools. Python_krb5 contains the krb5module.so for python, while kftp contains the gsscmodule.so for python. The version of this shared library does depend of course on the version of python. These python modules are not part of the default python installation in Linux, thus the need to provide them via cdfsoft. The CafGui and icaf_gftp require tcl v8_3_1 while v8_0_2 is still the default in CDF. While the default tcl installation in RH6.2 and higher is sufficient for the CafGui and icaf_gftp we are nevertheless distributing v8_3_1 with cdfsoft to be certain that an appropriate version is available in a well defined location on the user desktop.

The only remaining piece of software that the user interface to the CAF depends on is a krb5 client. This is not distributed with the CDF software. The user is expected to have an appropriate installation as part of their Linux OS in /usr/krb5 .

7 Managing Data Access

We expect to use rootd for data access for stage1. Rootd is a server daemon that starts a seperate server for each client. The server then transfers data to the client via a network socket. No mounting of the remote file system on the client node is necessary.

We prefer this over NFS for two reasons. First it allows for location independent access to the data. And second it is more stable than cross mounting all file server disks to all worker nodes. Rootd thus provides us additional confidence that the CAF can scale up to its full size. As an aside, the technologies pursued by the data handling group utilize protocolls very similar to rootd.

7.1 Data Management

The focus of the CAF is on providing resources to the collaboration rather than policing these resources, especially when it concerns disk space. We do not believe that such policy decisions should be made by the technical folks. Instead, we expect to rely heavily on the physics groups to decide what data should be on the CAF disks during Stage1. It remains to be seen to what extend this is a viable operational model.

In addition to data disks, we provide user disks for general use. We expect to enforce

quotas on these disks, and possibly install automatic clean up of at least part of the general user disks. See Section 6.10 for additional detail.

In the longer term we may assign the static disks as static dCache pools, and thus have the data managed and maintained via the data handling system.

7.2 Limitations of stage1 Data Access Management

It is quite obvious that we cannot allow all worker nodes to simultaneously contact one and the same file server. Fundamentally, every file server is a limited resource and shall support only a finite number of simultaneous connections. It is thus possible for jobs to be file server I/O limited if too many sections require files from the same file server.

There are in principle two ways of managing the potential resource congestion:

- implement an intelligent replica mechanism.
- implement a queuing mechanism that manages the number of simultaneous connections using techniques as used in fipc and fcp [11]. Combine this with a large source/sink ratio as discussed in Section 3.

While we believe that the former is the more efficient way of dealing with this in the long term, the latter is easier to implement. We will thus monitor file server resource usage during stage1 and implement a solution as needed.

8 Future Areas of Development

The present Section briefly touches upon some of the likely future enhancements.

8.1 Control and Monitoring

The most important future enhancement is in the area of monitoring and control. In order to be able to manage a computing farm with ~600 worker nodes, ~80 file servers, and a variety of infrastructure nodes like code server, database servers, head node, etc. we need to streamline the management and operations of the farm.

We do not discuss any details of this "streamlined management and control" simply because we consider this an active area of development throughout Stage1. Describing something today that changes tomorrow is not a very satisfying exercise.

Having said this, we envision a combination of NGOP and the kind of time series bookkeeping we do with the RRD tools to be the likely backbone of the evolving control and monitoring tools for the CAF.

8.2 Data Handling

We explicitly do not expect to provide any data handling functionality in Stage1. For Stage1 we will simply put all the data on disk, and provide the user with the means to obtain directory listings of the various file servers to figure out pathnames.

We intend to coordinate the population of the file servers with the group that organizes secondary dataset stripping for the physics groups.

In addition, we intend to set aside some of the available disk space for integration of the data handling system, in particular dCache. As a first step we will assign static dCache pools. Once that is working we will move on to dynamic pools that allow staging of files from the tape robot.

The key design choice for Stage1 is that DH functionality is strictly optional, depending on progress with dCache and Enstore.

8.3 Grid

The Stage1 CAF concentrates on job submission from anywhere to a central analysis farm. It does not address data movement, nor does it address any issues with respect to brokering of resources as all resources are in one place.

We expect that CDF will gradually move towards a computing model that includes DCAF (Decentralized CAF) in Europe, and possibly Korea and Japan. To fully exploit a situation with multiple sites that have significant computing resources CDF will have to tackle both of the issues mentioned above. We expect to do so by interfacing to some of the emerging Grid tools.

We expect this interface to be implemented in parallel to the existing CafGui $\rightarrow$ submitter $\rightarrow$ bmgr structure. We thus expect to continue operating the CAF while we commission a structure more like: GridUI $\rightarrow$ GridBroker $\rightarrow$ GridInterface $\rightarrow$ bmgr .

Viewed from a Grid perspective, the CAF is just another computing resource. The Grid does not own this resource. The local batch system, in our case fbsng, negotiates how the CAF resources are allocated, just like it does now.

We expect the first DCAF's to emerge during FY2003. We expect to start paying attention to the global Grid bonanza during Stage2 of the CAF.

9 Acknowledgements

We gratefully acknowledge the invaluable help and support from ISD, OSS, DCN, and FSS. Most notably, we want to thank Steve Timm, David Tang, Phil Lutz and Matt Crawford. In addition, we'd like to thank the Fall 2001 computing review, especially the outside reviewers Tom Davis (NERSC), Charly Young (SLAC), and Iain Bird (JLAB). The CAF as described here was born over beer and pizza in the evening hours of the October workshop/review.

References

[1] CDF Plan and Budget for Computing in Run 2, CDF 5914.

[2] A Tape Handling System for CDF in Run2, CDF 5822.

[3] http://www-isd.fnal.gov/fbsng

[4] CDF CAF Review Report on Benchmarking, CDF 5743.

[5] Physics Analysis Computing Needs Assessment, CDF 5787.

[6] Final Report CDF CAF Review Fall 2001, CDF 5802.

[7] http://mit.fnal.gov/~msn/cdf/caf/server_evaluation.html ,
CDF 5962, in preparation.

[8] http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/

[9] See for example the "Workload Management Work Package" (WP1) of the European DataGrid project: http://server11.infn.it/workload-grid/

[10] http://cdfcodebrowser.fnal.gov/CdfCode/

[11] http://www-isd.fnal.gov/fcp/ , http://www-isd.fnal.gov/fipc/