CDF/DOC/COMP_UPG/PUBLIC/6092
M. Casarsa, A. Fella, Y. Gotra, R. Kennedy, T. Kim,
M. Neubauer, F. Semeria, I. Sfiligoi, A. Sidoti,
F. Würthwein
May 26, 2004
Version 1.0

CDF CAF User's Manual

Abstract

This document represents the User's Manual for the CDF Central Analysis Farm (CAF). The CAF is a large ($\sim$ 600 CPU) Linux-based PC cluster deployed at FNAL to serve the analysis computing needs of the collaboration. This manual is designed to both get new users started quickly as well as serve as a comprehensive reference and troubleshooting guide for the CAF. This is meant to be an evolving document, so please feel free to contact cdf_caf@fnal.gov for suggestions, errata, additions, etc.

Contents

1 Introduction

The CAF is a large (~ 600 CPU) farm of computers running Linux with access to the CDF data handling system and databases to allow CDF collaborators to run batch analysis jobs. Since standard Unix accounts are *not* created for users (i.e. you cannot "log into" the CAF), custom software provides remote job submission, control, monitoring, and output interface for the user after strongly authenticated via kerberos.

This manual is designed as both a reference and quick start guide for a typical CDF user that would like to use the CAF to execute `AC++` jobs. As such, a basic knowledge of `AC++` and the CDFII analysis software is assumed; for references on these items, please refer to the corresponding manuals[2, 3]. Additional information on the CAF may be found on the CAF Home Page at

http://cdfcaf.fnal.gov

1.1 The CAF in a Nutshell

The basic operational paradigm is for users to compile and link their analysis jobs on their desktop, execute their jobs via a distributed batch system - Farm Batch System Next Generation (FBSNG), and receive their job output back on their desktop or send it to special user scratch space on CAF FTP servers for later retrieval. This implies two fundamental assumptions. First, we assume that each user has access to a desktop with a cdfsoft installation. Two 8-way Linux SMP systems (fcdflnx2, fcdflnx3) are maintained by the CDF Task Force for users who do not have a Linux desktop, and as general reference platforms for users who have problems with their desktop installation.

Secondly, we assume that the ratio of amount of data read in divided by the amount of data written out of the user AC++ job is rather large, of $O(10^3)$, or more. "Standard usage" of the CAF at this point is thus to analyze official secondary datasets and produce small skims, histograms, and/or ntuples. Users with requirements significantly different from this may not find the current implementation of the CAF very useful. Future enhancements to the offline systems allowing write access to the data handling system as well as large volumes of general user disks should make the CAF more accommodating to these types of users. It should be noted, however, that since we provide few explicit restrictions on the types of batch jobs user can run, users have successfully used the CAF in creative ways, including large-scale Monte Carlo and tertiary data set production.

Figure 1 shows a schematic of the basic CAF architecture. Users typically submit jobs to CAF using a graphical interface (GUI). A feature of the CAF interface and FBSNG is the ability to submit and run many similar jobs in parallel. For example, a standard usage of this feature would be to analyze some dataset in its entirety by submitting O(10) jobs simultaneously that differ only in the specific data files input to a given job. These independent job pieces are referred to as job "segments" throughout this manual. Job segments are implemented in FBSNG constructs called "sections" and "processes", whichare described in more detail in Section 2.2.

The top-level shell script to be executed, binary executables, tcl scripts, and all other files needed to successfully run the user's job are archived into a temporary tar file that is copied from the user's desktop to the CAF head node, where the batch manager distributes the jobs to the available worker nodes. On the worker node, the tar file is untar'd and the top-level shell script is run. After user's script exits, the output is tar'd up and copied to the user's desired output location. At the end of the job, an email summarizing the status of the CAF job can be sent to the user, if requested.

1.2 Getting Access: Requesting a CAF account

If you are new to CDF and/or have never requested an account on CDF Run II analysis machines, you should fill out the web form found at

http://www-cdf.fnal.gov/computing/unixaccountrequest.html

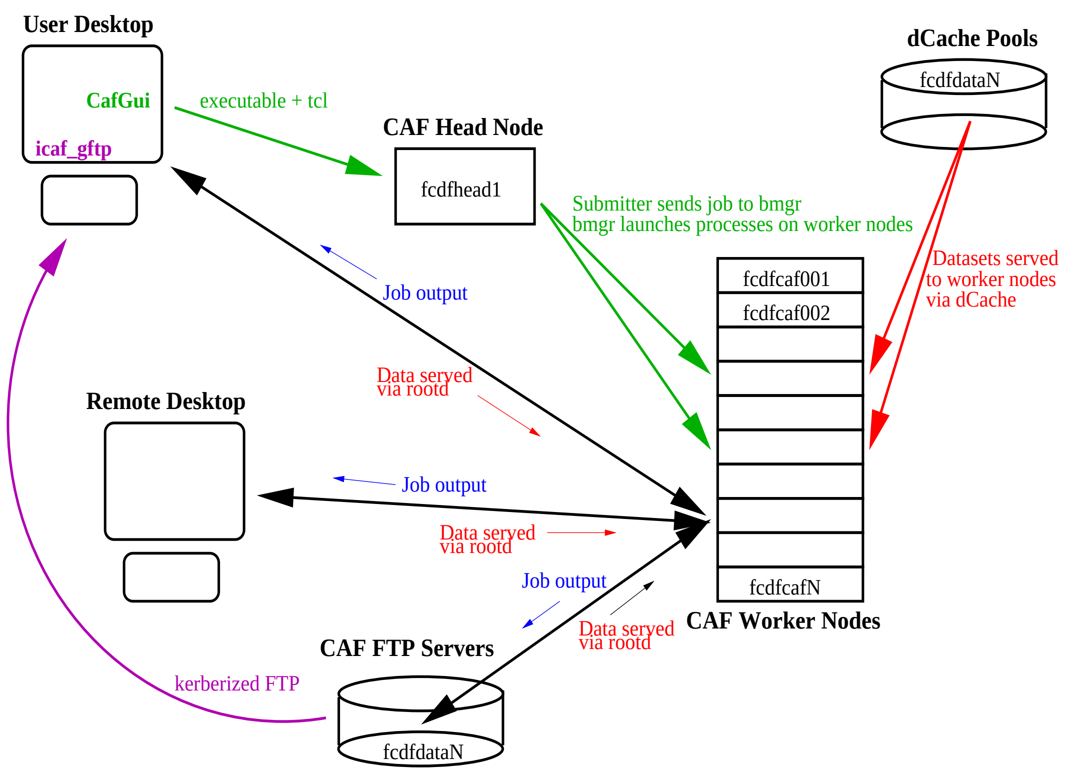

Figure 1: CAF flow chart architecture

Be sure to select fcdflnx2, fcdflnx3, and CAF as "machines" on which you would like an account.

If you have previously done the above (minus the CAF) and already have an account on fcdflnx2 and fcdflnx3, send an email request to Mark Neubauer (msn@fnal.gov) to receive CAF access. Please provide the following minimal set of information in your request:

- **Your full name and affiliation**
- **Your kerberos principal name** (e.g. "johndoe", if your full kerberos principal is "johndoe@FNAL.GOV"). After getting a kerberos ticket, your principal is displayed after "Default principal:" in the output of the "klist" command.
- **Your group's scratch FTP server name (if applicable)**: Some groups have purchased file servers for use as scratch disk in the CAF. If you are in one of these groups and you have been approved for use of this server with your group leader, indicate either the group label or explicit server name. If you are not a part of such a group you, can ignore this optional information and you will be assigned to one of the scratch FTP servers provided for general CAF users.

You will receive email with a CAF form letter as soon as your account has been created and will then be able to see your CAF queue via the web based monitoring described in Section 4.2. You will also receive scratch space on one of the CAF FTP servers for job output and retrieval using kerberized FTP or the available ICAF tools (described in Section 5.4).

1.3 Access to the CAF Client Software

The CAF user interface software - job submission interface, control and monitoring tools, etc. - are distribution through the development release of the CDF software. Since the development release is essentially never used for serious analysis, one might ask why the CAF software use development in this way.

The rationale behind this is that sometimes - although *very* infrequently - changes are made CAF server software which require simultaneous changes to the client interface software to ensure compatibility. The development release is updated nightly and and distributed to the central Linux platforms - fcdflnx2 and fcdflnx3. Also, the CAF client software is part of "development lite", which is a streamlined version of development that requires very minimal network and CPU resources to pull and build[1] onto remote machines.

Some key points regarding access to client software:

[1]In this context, "build" simply means that the CAF client shell scripts and copied into the proper location for use and made executable.

- To ensure that users are running the most up-to-date version of the CAF client software and that this software is compatible with the CAF server software in current use, the CAF client software is distributed through development.

- A streamlined version of development called "development lite" which requires minuscule network and CPU resources is available. If you plan to use the CAF, it is recommended that you use a platform in which development (or development lite) is updated regularly.

- To use the CAF, you *do not* have to do your analysis using the development software release. You will specify the software release used in your analysis in the executable shell script you send to the CAF.

- The development release is built nightly on the CAF standard Linux interactive platforms - fcdflnx2 and fcdflnx3.

- The development release is set up using:

  ```
  > source ~cdfsoft/cdf2.cshrc
  > setup cdfsoft2 development
  ```

  Throughout this document, successful set up of development using the above commands is assumed.

2 Preparing Your Job for the CAF

A key goal for CAF developers was to make the system as a flexible as possible with few limitations on what users are able to execute on the farm. Efficient use of finite computing resources is the responsibility of the collaboration as a whole, with guidelines provided by the system developers and expert users. In an effort to implement this generality, we provide an interface for users to execute arbitrary shell scripts they provide, support for job parallelism, and access to the official CDF software environment, data handling systems, and databases.

This section describes how to prepare a job for execution on the CAF and test that it first works outside of the CAF environment before large-scale submission.

2.1 Your Shell Script: What the CAF Will Run

As previously mentioned, the CAF will ultimately try execute on a CAF worker node whichever shell script you tell it to execute. In order for this to happen, you first need to prepare a shell script file and be sure that it has the proper attributes so that it is executable. Here you have a choice of shell - the standard UNIX shells: Borne Shells (sh/bash), C Shells (csh/tcsh), and Korn Shell (ksh) are supported. All of these shells are located in the /bin directory on CAF worker nodes. The Borne Shell (/bin/sh) is used in the examples throughout this manual[2].

2.1.1 Simple CAF Job (helloWorld.sh)

A simple example of a shell script (helloWorld.sh) one might submit for testing purposes is the following:

```
#!/bin/sh

# Get hostname of node script is running on
HOST=`/bin/hostname`

# Print message to stdout
echo Hello World! from ${HOST}

exit
```

When run on the CAF, this would return something like

```
Hello World! from fcdfcaf001.fnal.gov
```

in the stdout file of your job (details of how the CAF handles your job output will be discussed in Section 5).

Note that your shell script file must be executable for the CAF to successfully run it.

2.2 Parallelizing Your Job: Using CAF Job Segments

Analysis jobs within CDF can generally be broken up into independent, atomic pieces. For example, each piece might be searching for event candidates within a different part of a dataset. Obviously, running as many of these pieces at a given time as possible speeds up the overall job execution. The CAF provides a convenient interface for running parts of a job in parallel. We refer to each part of a CAF job as a *job segment*.

To be more precise, each job segment run on the CAF will start out *identical* in every way except for the value of a single integer referred to as a *segment iterator* passed as a parameter to your shell script. It is then up to the user to decide how they want to use the segment iterator to achieve the desired effect for their entire job. We provide the segment iterator merely as a way for users to distinguish each job segment from the others in their shell script.

[2]Under the Fermi Linux install on the CAF worker nodes, /bin/sh is a soft link to the Borne Again Shell, /bin/bash, therefore sh is equivalent to bash in this context

It should also be noted that the CAF will only transfer one copy of the user's input tarball to the head node, regardless of the number of job segments requested. The replication of their job segments to individual CAF worker nodes is handled internally within the system.

2.2.1 Simple CAF Job Using Segments (helloWorldSegments.sh)

As a simple example of a job that uses segments and the segment iterator, we return to the helloWorld script. Lets say that for some reason we want to run a job consisting of separate sub-jobs (i.e. segments), each of which prints out "Hello World! from <hostname>" like our previous example, except this time only after first waiting for some number of seconds. To be explicit, lets say that we want to run 20 such segments that wait 1, 2, 3, ... 20 seconds before printing out the Hello World! message. The following shell script (helloWorldSegments.sh) would achieve this:

```
#!/bin/sh

# First wait a number of seconds passed as
# the first parameter to the script
sleep $1

# Get hostname of node script is running on
HOST=`/bin/hostname`

# Print message to stdout
echo Hello World! from ${HOST}

exit
```

Aside from providing this shell script to the CAF, the user would input to the CAF interface the desire to run 20 job segments with the segment iterator range 1 to 20. The user also indicates which parameter in a set of possibly numerous parameters their script expects to place the segment iterator. In this simple example, our script expects a single parameter. The CAF would then run 20 separate instances of the helloWorldSegments.sh script, each with the segment iterator passed as a parameter to the script:

On the first CAF worker node:
./helloWorldSegments.sh 1

On the second CAF worker node:
./helloWorldSegments.sh 2

...

One the twentieth CAF worker node:
./helloWorldSegments.sh 20

The output of each job segment would then be sent to the user's desired output location.

The job submission interface including how to use job segments will be discussed in detail in Section 3.

2.2.2 Example Structure of a Parallel AC++ Job

Because of the flexibility of the CAF to execute whatever shell script the user provides, the helloWorld examples actually provide a reasonable starting point for users to get going on using the CAF for more useful purposes like data analysis. However, we provide a discussion of how one might want to organize oneself for a data analysis job using job segments to further facilitate using the CAF for an AC++ analysis job. Note that this is merely a suggestion - users are encouraged to organize themselves in whatever way they like.

This example has the following characteristics:

- a top-level shell script called caf.sh which is what the CAF will execute.
- an executable to be run by caf.sh called ParticleFinder.exe which is the binary built within the standard AC++ software framework.
- a custom shared library called libMyModule.so containing containing user code which ParticleFinder.exe depends upon.
- a tcl file called run.tcl which represents the control input to ParticleFinder.exe.

Recall that the CAF will tar up and transfer the directory specified by the user and run the desired shell script with a segment iterator passed as a parameter if it is using multiple job segments. The structure of the working directory in this example is the following:

```
caf.sh                  - top-level shell script to be executed
exe                     - directory for executables
exe/ParticleFinder.exe  - AC++ executable to be run by caf.sh
results/                - directory for job output (ntuples, etc)
shlib                   - directory for shared libraries needed by exes
shlib/libMyModule.so    - shared library required by ParticleFinder.exe
tcl                     - directory for tcl files
tcl/run.tcl             - top-level tcl file input to ParticleFinder.exe
```

In this particular case, we are using CAF job segments to parallelize the total job. The segment iterator is the sole parameter expected by the shell script to be executed (caf.sh) shown below:

```
#!/bin/sh

# Setup Run II CDF software environment
export VERSION=4.8.4
echo "SETUP VERSION: $VERSION"
source ~cdfsoft/cdf2.shrc
setup cdfsoft2 $VERSION

# Specify local shared library directory
export LD_LIBRARY_PATH=.:./shlib:${LD_LIBRARY_PATH}

# Set enviroment variable to value of segment iterator
export SEGMENT_NUMBER=$1

# Analysis execution command
./exe/ParticleFinder.exe ./tcl/run.tcl

# Save return value of AC++ command
RETC=$?

# Job cleanup
/bin/rm -rf caf.sh exe shlib tcl

exit $RETC
```

There are a few important comments to be made about this shell script.

- The user must setup the CDF software version with which they built their AC++ binary (ParticleFinder.exe in this case).
- The local shared library path (shlib) must be added to the LD_LIBRARY_PATH environment variable.
- An environment variable (SEGMENT_NUMBER) is set to the segment iterator passed to the shell script. This variable can then be used elsewhere (e.g. run.tcl).
- The seemingly peculiar line "RETC=$?" stores the exit code of your AC++ executable in a temporary variable (RETC). The line "exit $RETC" then sets the exit code of your script to the exit code of your AC++ executable rather than later shell

commands. This exit code will help you as well as the CAF admins understand why your AC++ job failed when it does.

- All contents of the current directory except the results directory are removed as the last part of your shell script. Recall that the CAF will tar up everything in the working directory after the shell script exits and send it to the specified output location. To avoid receiving back what was put into the job rather than just the output, these job input directories are removed after AC++ binary (ParticleFinder.exe) completes.

Most of what is done in the run.tcl file will depend upon the user's particular job. Here we have shown just the portion of this file specific to the CAF:

```
set rootFile ./results/output$env(SEGMENT_NUMBER).root
if [file exists $rootFile] {file delete -force $rootFile}

module enable HepRootManager
module talk HepRootManager
  histfile set $rootFile
  createHistoFile set true
exit

# Other tcl statements specifying the job control...
```

When using CAF job segments, this entire working directory (in the form of a tarball) will be replicated to each worker node running a segment and the shell script will be run with the particular segment iterator (assuming the user specifies this to start at 1) corresponding to a given segment:

```
./caf.sh 1
./caf.sh 2
...
```

Note that in this example, the output ntuple generated by the analysis job is given a name specific to the job segment (i.e. segment 1 generates ./results/output1.root, segments 2 generates ./results/output2.root, etc.)

2.2.3 Relation between segments and FBSNG sections and processes

This section discusses the relationship between the CAF concept of *segments* for job parallelization and the FBSNG constructs of *sections* and *processes* which the batch system actually uses to implement parallelization. Understanding this relationship is generally required for making sense of the web-based FBS monitoring information.

When the user submits a job with some number of segments, the CAF will assign each segment a *section identifier* and a *process number*. A job segment is uniquely identified by its section identifier and process number. From the perspective of the batch system as configured in the CAF, sections and processes have the following characteristics:

- Sections are containers of processes. That is, each section contains some number of processes.
- The user's shell script with segment iterator passed as a parameter is launched as a process on a given worker node.
- Only one process is launched on a given worker node CPU.
- The batch system scheduler is based upon sections, not processes. That is, the scheduler will require enough CPU slots to become available to launch all of the processes within a section.
- Once a section is scheduled to run, the processes within the section are launched on the CAF worker nodes simultaneously.

Within the CAF, sections represent fixed size containers of 10 processes maximum. Therefore, the first 10 job segments are 10 separate processes in the first section, the next 10 segments are processes in the next section, and so forth. The logic of this scheme means that unless the number of job segments is divisible by 10, the last section will have fewer than 10 processes (i.e. the remainder). The section identifier naming scheme is based upon the user's kerberos principal, their job identifier (JID), and a number that uniquely identifies each section.

An example of how sections and processes are implemented is shown in Figure 2. In this example, a user with kerberos principal johndoe@FNAL.GOV wishes to run parallel instances of myscript.sh, so he submits a CAF job with 16 segments. His subsequent JID is 1234. The CAF will divide his job into two sections with the first comprised 10 processes and the second comprised of 6 processes. The naming scheme for each segment in terms of sections and processes is shown in Table 2.2.3.Notice that the segment number is always the sum of the last number in the segment identifier and the process number. For example, the segment number of 1234.johndoe_10.3 can be easily seen to be $10 + 3 = 13$. This is convenient for quick translation of the FBSNG web monitoring information to the job segments that the user submits.

2.3 Testing Your Script Locally

It is very important to first test your job locally before submitting to the CAF. Doing this will avoid wasting your time and, more importantly, wasting some of the finite computing resources used by the collaboration as a whole. You will also find it more difficult to debug

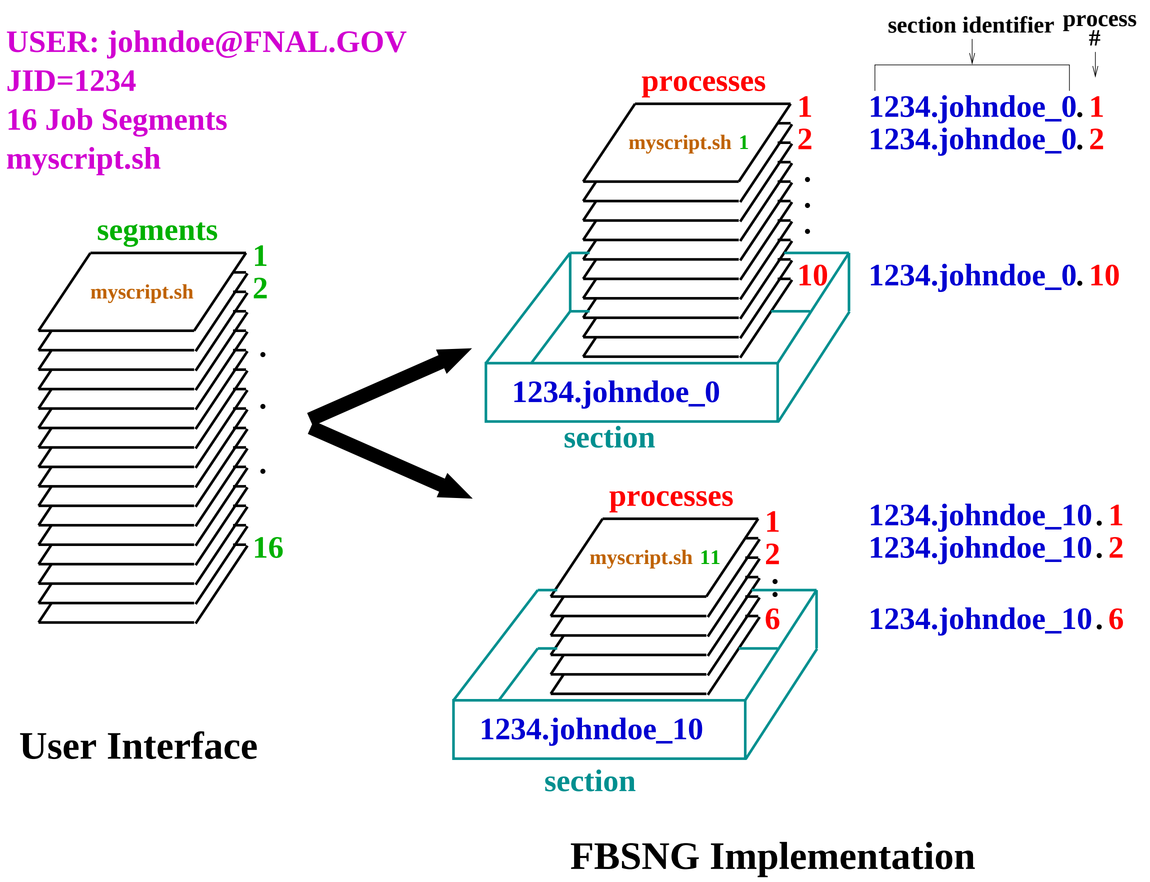

Figure 2: Relationship between job segments, sections, and processes for an example job of 16 segments.

Segment #	Section ID	Process #	Full segment name	execute
1	1234.johndoe_0	1	1234.johndoe_0.1	myscript.sh 1
2	1234.johndoe_0	2	1234.johndoe_0.2	myscript.sh 2
3	1234.johndoe_0	3	1234.johndoe_0.3	myscript.sh 3
4	1234.johndoe_0	4	1234.johndoe_0.4	myscript.sh 4
5	1234.johndoe_0	5	1234.johndoe_0.5	myscript.sh 5
6	1234.johndoe_0	6	1234.johndoe_0.6	myscript.sh 6
7	1234.johndoe_0	7	1234.johndoe_0.7	myscript.sh 7
8	1234.johndoe_0	8	1234.johndoe_0.8	myscript.sh 8
9	1234.johndoe_0	9	1234.johndoe_0.9	myscript.sh 9
10	1234.johndoe_0	10	1234.johndoe_0.10	myscript.sh 10
11	1234.johndoe_10	1	1234.johndoe_10.1	myscript.sh 11
12	1234.johndoe_10	2	1234.johndoe_10.2	myscript.sh 12
13	1234.johndoe_10	3	1234.johndoe_10.3	myscript.sh 13
14	1234.johndoe_10	4	1234.johndoe_10.4	myscript.sh 14
15	1234.johndoe_10	5	1234.johndoe_10.5	myscript.sh 15
16	1234.johndoe_10	6	1234.johndoe_10.6	myscript.sh 16

Table 1: Example naming scheme for a CAF job with 16 segments.

your job using the CAF since it is a remote batch farm basically without interactive login capability (by design).

Once your shell script is prepared, you should be able to successfully run it locally, with a test segment iterator as a parameter if your script requires one. If your desktop system is for some reason insufficient for testing out your script (e.g. no access to the data handling system), you should test your script by running it locally on fcdflnx2, currently our standard Linux platform for the CAF.

2.4 Kerberos Issues Related to Job Output

All CAF jobs are run as processes on CAF worker nodes under the Unix UID of cafuser. However, each user's unique identity needs to be established in the job to allow for authorized access to the user's output local (e.g. your desktop system). This is accomplished by generating a unique CAF kerberos ticket for each user of the form

```
caf/cdf/<user>@FNAL.GOV
```

So, our friend johndoe@FNAL.GOV would have his CAF jobs run using the kerberos principal caf/cdf/johndoe@FNAL.GOV.

A user's output location must allow incoming kerberized rcp connection for the CAF kerberos principal to receive output from the CAF. This is accomplished by having the CAF kerberos principal in the .k5login file in the user's home directory. For example, johndoe would have a minimal .k5login file like the following:

```
johndoe@FNAL.GOV
caf/cdf/johndoe@FNAL.GOV
```

Be careful to avoid extraneous spaces and other characters in your .k5login file! The CAF will check that the user's output location is available for `rsh/rcp` connection *at submission time* to avoid only finding out about a doomed job after the it has run.

3 CAF Job Submission

In this section, we present how to use the client interface software to submit jobs to the CAF.

3.1 Choosing a Process Type

This section discusses the use of process types within the CAF. Before we do this, its important to differentiate between our use of *queues* and *process types* within FBSNG.

- *queue*: Each user is assigned to a unique queue to which the FBSNG scheduler dynamically assigns a priority number to maintain fair share of CAF resources. A user's queue name is the same as their kerberos principal name. CAF jobs run as sections and processes within queues.

- *process type*: A process type specifies a collection of similar resources along with usage rules associated with those resources. For example, we have a process type called "long" which is allowed to have some maximum number of processes running on the CAF at a given time (e.g. 100), with each process consuming no more than 2 days of CPU and 4 days of real (i.e. wall clock) time.

Note that is our implementation of the CAF using FBSNG, process types are what are traditionally referred to as "queues" in other batch systems.

Each job is run under a single process type within a single queue. The user chooses their desired process type at submission time using the job submission interface. The current allowed process types for use are summarized in Table 3.1. There are three distinct categories of process types: general, group, and operations (cdfmc, cdfopr). Here we describe use of the general process types. Group queues and process types are discussed in Section 3.5.

As the name implies, the `test` process type is use to run short test jobs on the CAF before large-scale job submission. CPU resources are set aside for exclusive use by the `test` process type to ensure quick turn-around of the test jobs. When jobs are submitted to the `test` process type, they will run the `test` queue rather than the normal queue tied to your kerberos principal. In this way, we can adjust the queue priority for test jobs so

Process Type	CPU/Real Time Limit	Description
test	5 mins/15 mins	test jobs
short	2 hours/4 hours	short length jobs
medium	6 hours/12 hours	medium length jobs
long	2 days/4 days	long length jobs
long_XYZ	2 days/4 days	for group queue XYZ
short_italy	6 hours/12 hours	for group queue
cdfopr	2 days/4 days	CDF Offline Operations
cdfmc	2 days/4 days	Production Monte Carlo

Table 2: Currently available process types within the CAF. The top four process types are for general use. The others have restricted access to assigned groups of users.

that they are immediately scheduled rather than having adhere to user's individual CAF share at a given time.

The main process types general users will submit to are the short, medium, and long process types. These process types differ in their time limits as well as the fraction of the CAF's CPU resources they are allowed to consume. Jobs that try to run longer than either the CPU or real time limit of the process type they are using will automatically killed by the FBSNG batch manager.

How does one choose which process type to use? Given the dynamical nature of the CAF utilization, there no one answer to guarantee maximal CPU resources under all circumstances. Resource allocation among the different process types is handled by the FBSNG scheduler based upon fair share between the queues over time scales that typical jobs take to execute. The FBSNG fair share is based on how long a user has waited and how much CPU they have used and not which process type they are using. It is generally preferred that user use the short and meduim process types. This assures that stuck jobs do not occupy a slot for two days. This also reduced the process's exposure to system failures. As an incentive, the long process type is restricted to use only a fraction of the CAF at any given time. This means that short and medium are in general a better deal. On the other hand jobs shorter than 20 minutes put stress on the system, and at no time should you submit jobs which have typical section lengths of less than 5 minutes.

3.2 Prerequisites for Any Large-scale Job Submission

The CAF is a finite computing resource that is also critical for the physics output of the CDF collaboration. Responsible use by individuals will help ensure that this resource is in the most efficient way possible for the benefit of everyone.

Here are some guidelines for responsible usage of the CAF, in particular for large-scale (i.e. many job substantial job segments) job submission:

- *Test your job by running it locally*
  As previously discussed in Section 2.3, it is important to first test a subset of your job (e.g. using your executable script to process the first 100 events of a representative data file) on your desktop before submitting it to the CAF. We reiterate this step here for its importance in avoiding waste of central computing resources.

- *Test your job by running a representative segment in the* `test` *queue*
  After verifying your job script runs successfully on your local platform, a subset of your total CAF job should be run in the test queue. Only after this step has been completed should you consider submitting all the segments in your job.

- *Avoid running lots of short job segments*
  Each segment run through the CAF consumes I/O and CPU resources, independent of the resources consumed by the user via their shell script and subsequent processes. To minimize this resource overhead, users should try to use as few segments as possible and utilize at least 50% of the time (CPU and/or real) limit in the process type in which they run. Lots of segments that each run 1 minute in the short process type which allows for 2 hours of CPU is a waste of CAF resources that adversely affects everyone.[3]

- *Avoid re-running successful job segments*
  If you find that part but not all of your job has failed for whatever reason, please take the time to only resubmit what has failed. You summary email will provide you with a list of failed job segments that you may need to re-submit. Do not just re-submit the entire job because it may be easier.

- *Think before you process!*
  This is probably the most important guideline of them all to help ensure successful analysis computing for CDF as a whole. Treat to the CAF and other CDF computing resources as precious shared resources that will never satisfy the needs of the collaboration if they are needlessly wasted. Do not use CAF CPU just because you can - only submit job N after thoughtful consideration of the output of job N-1, where at all possible. Before creating skims/tertiary datasets, refitting tracks or other such resource intensive tasks on large datasets, consider whether others in your physics groups may have already produced these data sets for general use.

3.2.1 Responsible Use of the Data Handling System (dCache)

This section describes the data access guidelines to help users optimize their use of the dCache service, to avoid problems not only for themselves but also for the rest of CDF.

[3]In the near future, the batch system will deduct a fixed queue priority for each segment run, a CAF shipping and handling fee in essence, to account for this overhead.

These guidelines assume you already know how to use dCache. If this is not the case, first consult [6] and Section 6.3.

- *Do not perform raw dataset skimming jobs on the CAF without detailed consultation with the associated Physics Group and the Data Handling Group before-hand.*

  There are many good reasons not to do this anyway, and consulting with experts should help determine a more efficient way to accomplish the same task. dCache was intended to serve production output and secondary data sets. There is only limited space in dCache, and limited bandwidth dedicated to serving raw datasets. Also, reading large amounts of raw data can be made more efficient with subtle tweaks to input module configuration, something that is not required for production output and secondary data sets. Raw data skimming jobs have effectively denied access a few times to analysis jobs reading secondary datasets in dCache. If it has to be done, let us work together to find a way to do so efficiently.

- *If you plan to access a larger dataset (>= 100 filesets) via dCache for the first time, please contact* cdfdh_oper@fnal.gov *first.*

  DCache treats all stage requests in a FIFO-like restore queue, whether a request is part of 2000 files requested by one user or part of 2 files requested by another. CDF DH would like to alleviate severe waiting by short jobs through intelligent pre-staging datasets for long jobs before the long jobs are run on the CAF. It is in your interest to warn cdfdh_oper rather than risk exceeding the time limits in the CAF, which would result in losing all your partially completed work. cdfdh_oper offers to pre-stage what you need so that you do not waste your time nor unnecessarily delay other jobs. This sort of tape-access resource management will be provided by SAM in the future, so there has been no effort to graft such onto the dCache service.

- *Please use the "log rule" when developing a physics analysis program. Be sure it runs correctly with a single input file first before running on larger chunks of data. Be sure it runs on a fileset correctly before running it on 10 filesets. And be very sure it works correctly -and- produces all the ntuples and plots you will need for a while before running it on larger chunks of data like 100 filesets and/or a full dataset.*

  The log rule was once an oft-quoted guideline to bringing up analyses to insure precious resources (tape access, CPU cycles) are not needlessly wasted on immature analyses. The economy of some of these resources may have changed nowadays but this still makes sense overall. As more and more physics analyses mature and machine luminosity increases, once plentiful resources will again be in short supply.

- *When moving a job to the CAF from another environment (fcdfsgi2, your desktop, old CAF to new CAF, etc.), always run it on a fileset first to be sure there are no environmental problems (bad link libraries, typo in tcl scripts) that will lead to widespread job crashes.*

Each environment has its own idiosyncrosies, and a little caution up front improves efficiency overall. Currently, there are at least 3 different versions of Linux and one of IRIX in the CDF computing environment, and even a "correct" program can crash badly if run against incompatible system libraries.

- *Try to break up large processing jobs reading whole datasets (or many tens of filesets) into separate jobs which together span the desired input sample, and submit these jobs separately. If there is a problem or error in one CAF job segment or chunk of the dataset processed, ONLY re-run on that portion of the data sample. How you divide up the dataset is up to you, but by large blocks of consecutive runs used to be a common practice.*

  Our current Framework-DH system has no automated means of identifying failed job segments for re-processing, nor for suspending segments across a computer system downtime. This is something that will be provided in part by SAM in the future. Users must do this manually for now, and it is preferred of course that this be done in a manner that does not re-run already successfully processed segments.

- *Be patient, but not too patient.*

  Your executable may stall while trying to open a file in dCache for a number of reasons, some of which are completely normal. If the file must be restored from tape, then tape contention (busy Enstore) may delay that restore for minutes or even hours. Nearly infinite stalls can occur when a user tries to restore a file from tape where that tape is marked NO ACCESS by Enstore (tape is being recovered by hand, can take days).

  If the Enstore queue is not too long and your job has been waiting on a file open for more than an hour, then it may be prudent to send e-mail to cdfdh_oper@fnal.gov with the job id and if possible the filename being opened. To help us understand the problem, one can set setenv DCACHE_DEBUG 2 for one's job to produce a paper trail of the dCache client-server messaging. And if it is possible to do so, one should leave a troublesome job segment running so that dCache operators and developers can investigate the precise state of the server when the client has data delivery problems.

3.2.2 Responsible use of databases

Guidelines to be provided by Alan Sill.

3.3 The CAF Graphical User Interface (CafGui)

Jobs are typically submitted to CAF by means of a Graphical User Interface (GUI) which is run from the user's desktop. An alternative command line interface presented in Section 3.4 is also provided.

The CAF GUI is distributed within the CafUtil package via the development release of the CDF Software as discussed in Section 1.3. The CAF GUI is started (in bash shell) by typing:

```
> source ~cdfsoft/cdf2.shrc
> setup cdfsoft2 development
> CafGui &
```

An example of the CAF job submission GUI which will pop up on your screen is shown in Figure 3.

3.3.1 Filling in the GUI Fields

The graphical submission interface (CafGui) is the way users specify their job to the CAF. As shown in Figure 3, various fields are to be completed by the user. The meaning of these fields are described below:

- *Analysis Farm:* The analysis farm to which the job will be submitted. For normal users, "caf" is the only option, although CafGui can be configured for other farms (discussed in Appendix B).
- *Process Type:* The process type to which the job will be submitted.
- *Initial Command/Segment Range:* The command to be executed on the CAF is specified in the Initial Command field. This is normally the user's top-level executable shell script. The two boxes to the right specify the segment iterator range (inclusive), used for job parallelization as discussed in Section 2.2. For each integer in this range, a job segment will be submitted with the $ replaced by this integer (i.e. the segment iterator).
- *Original Directory:* The directory containing the user's executable shell script plus other supporting files/directories required to successfully run the user's job. The submission interface will tar up everything in this directory (and any subdirectories) and send this tarball to the CAF head node for eventual use on the worker node where each segment is launched.
- *Output File Location:* The location where the output will be sent. This is in the rcp/scp format that includes the Unix ID, name of the remote machine and destination directory. A convenient interface to each user's CAF scratch disk (ICAF) is also provided (see Section 5.2.2).

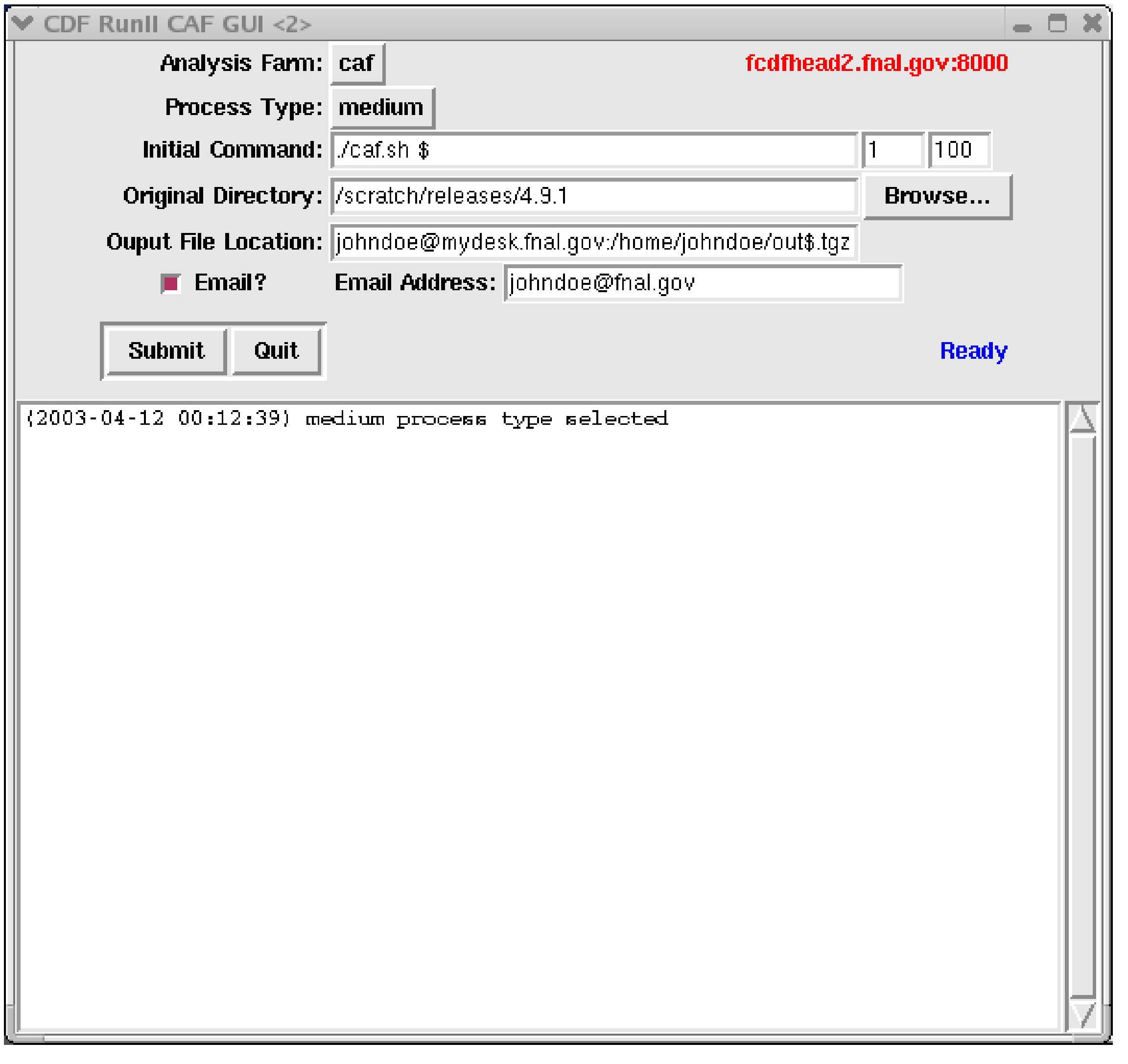

Figure 3: The CAF job submission GUI.

- *Email?/Email Address:* Here the user specifies whether they want a summary email after the job completes and, if so, to which email address this should be sent.

In the example shown in Figure 3, johndoe is submitting a job to the medium process type from his desktop. The job will run 100 instances (segments) of the script caf.sh, each with a different integer from 1 to 100 passed as a single parameter to caf.sh. The caf.sh script and all other required files and subdirectories are located in the /scratch/releases/4.9.1 directory. This directory will be tar'd up, transferred to the CAF headnode, copied to each worker node running a job segment, and untar'd into a working directory within the segment will run. He has specified for the job output to get copied from the CAF to mydesk.fnal.gov under his account into his home directory /home/johndoe. The output tarballs will be called out$.tgz, with each $ replaced by the particular segment number. After all segments complete, he has requested to receive a summary email from the CAF, sent to johndoe@FNAL.GOV.

3.3.2 Customizing GUI Defaults

When the CAF GUI is started, choices that the user can make for entry fields, buttons, etc. are inserted by default. There is an attempt to make these sensible (e.g. email address is <uid>@fnal.gov), but the user will still likely want to customize the GUI defaults. To accommodate this, the GUI will look for certain environment variables to replace the defaults. If a given environment variable is not set, the normal GUI default is used. The environment variables which the CAF GUI looks for are the following:

- CAFGUI_ANALYSIS_FARM - analysis farm field
- CAFGUI_INIT_CMD - initial command field
- CAFGUI_PROC_TYPE - process type field
- CAFGUI_ORIG_DIR - original directory field
- CAFGUI_OUTFILE_LOC - output file location field
- CAFGUI_GET_EMAIL - get email after job completes? ('1' for yes, '0' for no)
- CAFGUI_EMAIL_ADDR - email address, if email is to be sent
- CAFTARFILEPATH - where CafGui generates temporary tarfile of original directory to be sent to the CAF (default is user's home directory)

These can either be set in the user's login scripts (e.g. .bashrc) or by simply uncommenting and modifying (e.g. replacing "user" with your user name) the following example lines in the CafGui executable script:

```
# export CAFGUI_ANALYSIS_FARM=caf
# export CAFGUI_INIT_CMD=./caf.sh
# export CAFGUI_PROC_TYPE=short
# export CAFGUI_ORIG_DIR=/home/user/myjobfiles
# export CAFGUI_OUTFILE_LOC=user@fcdflnx2.fnal.gov:/cdf/scratch/user/temp.tgz
# export CAFGUI_GET_EMAIL=1
# export CAFGUI_EMAIL_ADDR=user@fnal.gov
# export CAFTARFILEPATH=/cdf/scratch/user
```

3.4 The CAF Command-line User Interface (CafSubmit)

Although we recommend use of CafGui for CAF job submission whenever possible, a command line interface is also provided. This is designed for users with poor or slow network connection (e.g. modem) to the machine from which they wish to submit, making use of the graphical interface too painful.

After the development release is set up, the command line interface is used by typing CafSubmit with a set of parameters. The usage of CafSubmit is seen by simply typing CafSubmit at the command line:

```
> CafSubmit
usage: CafSubmit --tarFile=file --outLocation=file \
                --procType=ptype [--dataset=id] [--email=addr] \
                 --start=num --end=num command
```

The parameters are as follows:

`--tarFile=file`	Full path filename for tarball containing Original Directory, as defined for CafGui. Note that this is *not* the same as the Original Directory, but rather is the gzip'd tar file of this directory. (e.g. ./submitme.tar.gz)
`--outLocation=file`	Full path for output file (e.g. me@ncdfxx.fnal.gov:/home/me/out.tgz)
`--procType=ptype`	Desired process type (e.g. short)
`--dataset=id`	Optional SAM dataset ID, for use with SAM only (e.g. hbot0h)
`--email=addr`	Optional email address for summary output The default is *user-principal@fnal.gov* If no email is desired, enter santa@north.pole
`--start=num`	Beginning segment number (e.g. 1)
`--end=num`	Ending segment number (e.g. 100)
`command`	Command to be executed, normally a shell script (e.g. ./my.sh)

Note that CafSubmit requires that a tarball of your Original Directory exists before using it to submit to the CAF. To provide a concrete example, we show how johndoe would go about submitting the job shown in Figure 3 and discussed in Section 3.3 using the command line interface:

```
> cd /scratch/releases/4.9.1
> gtar zcvpf /tmp/mytar.tgz ./
> CafSubmit --tarFile=/tmp/mytar.tgz \
           --outLocation=johndoe@mydesk.fnal.gov:/home/johndoe/out\$.tgz \
           --procType=medium --email=johndoe@fnal.gov \
           --start=1 --end=100 ./caf.sh \$
```

Note the extra backslashes to indicate that the $ is literal and should not be interpreted by the local shell. A useful option with the tar command is `--dereference` which will include files and directories which are soft-linked.

3.5 Using the Group Queues

Numerous groups within CDF have contributed CPU (and disk) resources to the CAF. The arrangement[4] is that members of this group get near instantaneous access to these CPU resources when desired, but the pool of general CAF users within the collaboration can use some or all of these resources when not in use by the group. This sections describes how group members can use these resources. If you are not a member of one of the groups directly contributing CPU resources to the CAF, you may skip this section. Management of the group queue members by group administrators is discussed in Appendix A.

Using the group resources is quite easy - simply submit to the process type assigned to your group (recall Table 3.1). These process types each have a CPU quota equal to the amount of CPU contributed by the groups. The jobs submitted to a given group process type is run within the associated group queue rather than the user's own queues. Also, since some groups have few than 10 CPU's in their quota, sections in the group queue are run with a single process rather than the 10 processes per section in the general CAF process types.

For example, johndoe is a member of the MIT group and submits a 5 segment job to the long_mit process type. He is returned a JID of 1234. These segments will be run as 5 separate sections each with one process in the queue named mit. Since in general there are more than one member in a group, the user is associated with their job through use of their kerberos principal name in the section identifier. So johndoe's segments will be run under the mit queue and named as:

1234_johndoe_1.1
1234_johndoe_2.1
...
1234_johndoe_5.1

The scheduling between different users' sections with the group queues in *round robin* rather than FIFO (first in, first out). As an example, lets say that one of johndoe's MIT colleagues, janedoe (not related), submits a 9 segment job to the long_mit process type slightly after johndoe's job with JID=1234 was submitted. She's returned a JID 1235. The order that the segments will be *started* on the CAF will be the following (johndoe's segments in red, janedoe's segments in blue; first at top, last at bottom):

1234_johndoe_1.1
1235_janedoe_1.1
1234_johndoe_2.1
1235_janedoe_2.1
1234_johndoe_3.1
1235_janedoe_3.1
1234_johndoe_4.1
1235_janedoe_4.1
1234_johndoe_5.1
1235_janedoe_5.1
1235_janedoe_6.1
1235_janedoe_7.1
1235_janedoe_8.1
1235_janedoe_9.1

4 Monitoring and Control of CAF Jobs

Although the CAF is a non-interactive batch system in a user login sense, a key design goal of the CAF was to provide users authenticated control and monitoring capabilities very similar to what they would have if logged into a worker node running their job. This section describes the CAF client tools for job control and monitoring in Section 4.1 and the Web-based monitoring provided by FBSNG in Section 4.2.

4.1 CAF Client Tools for Control and Monitoring

In this section, we present the client tools used to attain information about and control CAF jobs. These are command line tools are available through the development release of the CDF software in the same way as the job submission interface.

4.1.1 cafjobs: Job Status and Information

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

```
> cafjobs
```

4.1.2 caflog: Checking Progress of a Job Segment

Displays the CAF system log to check the progress of your job segment

```
> caflog JID segmentNumber
```

4.1.3 cafkill: Kill a Job or Job Segment

Kill the job or specific range of job segments

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

4.1.4 caftail: Peeking at Files in Working Directory

Peek into the standard output files(logs):

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

4.1.5 cafdir: File Listing in a Job Segment's Relative Path

List a section's "relative" path:
cafdir JID segmentNumber [file/dir_name]
Example1: "cafdir 417 3 ."
will list all the files in the working directory of segment 3 of JID 417

4.1.6 cafhostdir: File List in the Absolute Path of a CAF Node

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 and directories 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 CAF. If there's something you need that isn't there send email to cdf_code_management@fnal.gov.

4.1.7 cafdataset: Full Path Listing of File in a Static CAF Dataset

Full NFS path file listing for a secondary dataset (by dataset ID):
cafdataset dataset
Example1: "cafdataset jbot0c" will show the NFS listing of all files on the CAF servers for dataset ID = jbot0c.
Note that full rootd string for each file can be inferred from this information (see the examples in Section 6).

4.1.8 caftop: Show resource utilization of CAF Job Segment

Show top on worker node running user's section to show resource utilization:
caftop JID segment

4.1.9 cafhold: Hold my queue or queue of a group I'm at part of

Hold my queue or queue of a group I'm at part of:
cafhold queueName

4.1.10 cafrelease: Release my queue or queue of a group I'm at part of

Release my queue or queue of a group I'm at part of:
cafrelease queueName

4.1.11 cafnode: Show node that job segment is running on

Show the CAF worker node that one of my job segments in running on:
cafnode JID segment

4.1.12 cafcprio: Change section priorities for jobs in my queue

Change section priorities for jobs in my queue:
cafcprio +/-PRIO JID

Examples:
¿ cafcprio +1 1234

increases the priority of all sections in JID=1234 by one unit

¿ cafcprio -4 1234

decreases the priority of all sections in JID=1234 by four units

Note: cafcprio does not operate on the test or group/special queues, only owners own queue.

4.2 CAF Web Monitoring - FBSWWW

The FBSNG batch system runs an HTTP server as part of its standard operation. This provides a monitoring system (FBSWWW) for users to gain much useful information about the state of their jobs and the batch status in general. The FBSWWW web page for the CAF can be access from the main CAF web page - http://cdfcaf.fnal.gov. Figure 4 shows an example of FBSNG web monitoring page.

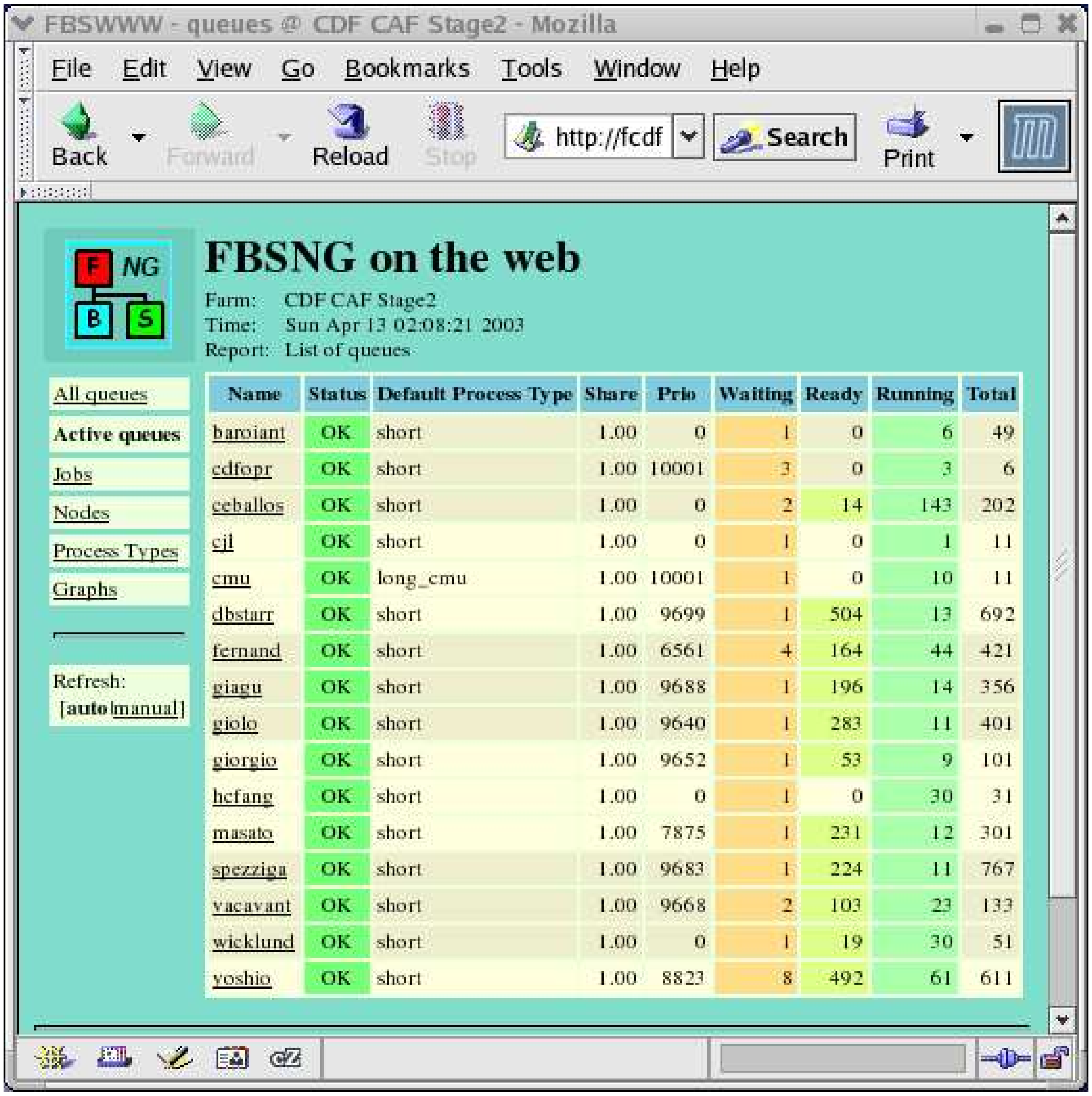

Figure 4: CAF monitoring web page

5 CAF Job Output

This section discusses the handling of user job output. For each job segment, the CAF system software will tar up and gzip your working directory after your shell script exists and copy it to your designated output location. Note that unless you remove files as part of your shell script, your output tarball will not only contain generated output but also the files in your Original Directory set to the CAF.

5.1 Copying Output Directly to a Remote Machine

There are several reasons why a user might want to copy files generated in their CAF job directly to a remote machine rather transferring them in the output job tarball. It could be that the output is large or that the user wants the output before the job completes.

Simple rcp -N can be used to copy output from a CAF worker node to the remote machine. Recall from Section 2.4 that the destination account on the remote machine must accept incoming rcp connection from the CAF kerberos principal of the form

```
<user>/cdf/h2caf@FNAL.GOV
```

to receive output from the CAF. Note the "-N" flag here. The kerberos tickets in use on the CAF can not be forwarded. The "-N" flag is needed to communicate this fact to rcp.

5.1.1 Using the Farms Remote Copy Utility (fcp)

The recommended way of copying job output from within a CAF job to a remote machine is to use fcp when at all possible. The fcp protocol[10] provides a much more robust remote file transfer mechanism than rcp for large farms where many simultaneous transfers to a single machine are common. In fact, fcp is used to transfer files internal to the CAF, including the job output tarball to users' scratch disk (described in Section 5.2).

To use fcp, one must first ensure that the destination machine is running the fcp server daemon, fcpd. Installation of fcpd is discussed at [10]. To use fcp within your CAF shell script, you need to include the following lines:

```
source ~cdfsoft/cdf2.shrc
setup fcp
```

The format for fcp is in principle exactly the same as rcp. However, we find that fcp is not installed identical at all CDF sites throughout the world. We thus suggest not to take chances, and instead use the "-c" flag to guarantee uniform behaviour as follows:

```
fcp -c /usr/krb5/bin/rcp -N ./myFile johndoe@mydesk.fnal.gov:/home/johndoe/
```

Another example of when it is necessary to use the "-c" flag is recursive copy of files using fcp:

```
fcp -c /usr/krb5/bin/rcp -N -r ./skim_files johndoe@mydesk.fnal.gov:/home/johndoe/
```

This would copy all of the files in ./skim_files and all subdirectories to johndoe's desktop. Note the -N flag is required to use the non-forwardable CAF user kerberos ticket generated during the job.

5.2 CAF FTP Servers for Use as Scratch Disk

Each CAF user is assigned scratch disk space on a file server local to the CAF. Users nominally are constrained to a soft quota of 25 GB and a hard quota of 50 GB[4]. As with CAF CPU, some groups have also contributed scratch disk to the CAF. Users in these groups are subject to different quota restrictions, as per the policy in [4].

5.2.1 General Rules on Usage

There are several important things users should be aware of regarding the use of their CAF scratch disk. These are summarized below:

- *Access policy*: User's can manage and interact with their CAF scratch disk in the following ways:
  - Users may copy files to and from their scratch disk within their CAF jobs. The fcp protocol rather than rcp should be used in *all* circumstances within CAF jobs.
  - Users may use kerberized FTP to access their scratch disk from outside the CAF (e.g. from their desktop). A custom GUI interface called icaf_gftp using kerberized FTP is particularly convenient for this and is described in Section 5.4.
- *Scratch means scratch!*: The CAF scratch area is meant to provide temporary buffer space for CAF job output until it can be moved to a permanent location outside the CAF. *The CAF scratch disk is not backed up!*. While the scratch disk is under RAID configuration and therefore quite robust, you should operate as if the contents of your scratch space could disappear at any moment. We assume no responsibility for loss of data in the CAF scratch area.
- *Honor thy neighbor*: The CAF scratch disk is a shared resource. Many users are on the same physical hardware, so each needs to act responsibly. In particular, only transfer files from within CAF jobs using the fcp protocol. Linux systems cannot handle large numbers of simultaneous rcp connections, which is why fcp is (in part)

[4]In Unix, these differ in the following way. Users can exceed their soft quota for some time period (1 week in the CAF), after which they will be disallowed to write new files until they are under this quota. Hard quotas are strictly enforced by the OS to never be exceeded.

a queuing mechanism for rcp "under the hood". If you hit the scratch FTP servers with too many simultaneous rcp requests, not only will many of these transfers fail, but you also hose your colleagues transfers as well.

- *Remote access to root files*: Each scratch FTP server runs the rootd service, allowing remote read access to root files in your scratch area (and possibly others) within CAF jobs. Using rootd for data access is described in Section 6.1.

5.2.2 Using Your Scratch Disk

In this section, we discuss use of CAF file servers configured as FTP servers of user scratch disk space.

Each user is allocated scratch space one of the CAF ftp servers. Two subdirectories (icaf/, scratch/) are created in each user's root path (/cdf/scratch/<user>) on the FTP servers. This directories are designed to serve two different purposes for the user:

icaf/ - *for typical job output (log files, ntuples, etc) to be retrieved after the user's job completes.* To send job output to your icaf/ directory, simply specify

icaf:temp$.tgz

for the Output File Location field in the CAF job submission interface. For example, a segment iterator of 600 (recall the meaning of $ in CafGui). This will send the job output to your ICAF space under the file name

temp600-JID.tgz

where JID is your job ID number.

scratch/ - *for data (e.g. small skim output) to be accessed by a subsequent CAF job via rootd.* As previously mentioned, the FTP servers are configured for rootd service, so a user can access data in the scratch area using rootd protocol. Two environment variables are set during each CAF job to aid in user access to the scratch area on the FTP servers.

- CAFOUT_NODE - the FTP server name for your assigned scratch space (e.g. fcdf-data007.fnal.gov)
- CAFOUT_SCRATCHPATH - the absolute path to your scratch space (e.g. /cdf/scratch/johndoe/scratch)

As an example, lets say that johndoe would like to write out a root file called skim.root on one CAF job and access this file on a subsequent CAF job. The last part of his first job's script would be to fcp skim.root to his scratch area:

```
fcp ./skim.root johndoe@${CAFOUT_NODE}:${CAFOUT_SCRATCHPATH}/skim.root
```

In his next job where he wants to access this file using rootd, he would include

```
talk DHInput
   include file root://$env(CAFOUT_NODE)/$env(CAFOUT_SCRATCHPATH)/skim.root
exit
```

into the tcl file for this subsequent job.

5.3 FTP Access to CAF Scratch Disk

Users can interact with their scratch disk using kerberized FTP. The standard command line **ftp** gives all of the basic functionality one requires of a disk repository, allowing one to download/upload files, get file and directory listings, create directories, etc; the user must just know the node on which his data are stationed. See the ICAF section (Section 5.4.1) for more information on how to find out the correct node.

However, an often more convenient custom graphical interface to FTP we call "ICAF" is also provided. This is discussed in Section 5.4.

5.3.1 Using Kerberized FTP to Manage Your Scratch Disk

To connect via FTP to a CAF file server that you have access to (e.g. your CAF scratch disk or a group account like "cdfdata" on a particular FTP server), you simply type

```
> ftp <file server name>
```

You should be using kerberized FTP by default. You can check this by typing

```
> which ftp
```

which should return something like /usr/krb5/bin/ftp. If not, use this full file path for ftp.

As an example, johndoe wants to access his CAF scratch disk using FTP. Let's say that he knows from previous experience that his scratch server is fcdfdata006:

```
> ftp fcdfdata006
Connected to fcdfdata006.fnal.gov.
220-                             NOTICE TO USERS
220-
220-       This  is a Federal computer (and/or it is directly connected to a
220-       Fermilab local network system) that is the property of the United
blah blah blah

220 fcdfdata006.fnal.gov FTP server (Version 5.60) ready.
334 Using authentication type GSSAPI; ADAT must follow
```

```
GSSAPI accepted as authentication type
GSSAPI authentication succeeded
Name (fcdfdata006:johndoe):
232 GSSAPI user johndoe@FNAL.GOV is authorized as johndoe
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Note that johndoe could have entered another account he has access to (e.g. cdfdata) at the Name (fcdfdata006:johndoe): prompt rather than just hitting return.

User johndoe can see his scratch and icaf:

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 24
drwxr-xr-x   37 johndoe      cdfcaf       8192 Jun 11 00:09 icaf
-rw-r--r--    1 johndoe      cdfcaf         83 Apr 29 10:14 .k5login
drwxr-xr-x    4 johndoe      cdfcaf        106 May 15 16:14 scratch
226 Transfer complete.
ftp>
```

He then has the full capabilities of FTP - get, put, mkdir, etc.

5.3.2 Anonymous FTP Access

In Section 5.3.1, we discussed how users can manage their scratch disk using kerberized FTP. However, access required the user to have an account on this server or else full kerberize access through the .k5login file. In many cases, users want to share the contents of their scratch disk with others, but do not want to give full access to their account. For this purpose, we allow anonymous FTP connections to each of the CAF FTP servers.

Using anonymous FTP is exactly the same as described in Section 5.3.1 except that one types either "anonymous" or "ftp" at the Name (¡server name¿:¡user¿): prompt.

Lets say that johndoe wants to access a file called Higgs.root to verify than his colleague janedoe has indeed discovered the Higgs. She tells him that the file can be found the scratch subdirectory of her scratch disk on fcdfdata007. A session in which he downloads Higgs.root into his local directory for inspection is shown below:

```
> ftp fcdfdata007
Connected to fcdfdata007.fnal.gov.
220-                              NOTICE TO USERS
220-
220-          This  is a Federal computer (and/or it is directly connected to a
```

```
220-       Fermilab local network system) that is the property of the United
blah blah blah

220 fcdfdata007.fnal.gov FTP server (Version 5.60) ready.
334 Using authentication type GSSAPI; ADAT must follow
GSSAPI accepted as authentication type
GSSAPI authentication succeeded
Name (fcdfdata007:msn): anonymous
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd janedoe/scratch
ftp> get Higgs.root
local: Higgs.root remote: Higgs.root
200 PORT command successful.
150 Opening BINARY mode data connection for Higgs.root (21933952 bytes).
226 Transfer complete.
21933952 bytes received in 2 seconds (1.1e+04 Kbytes/s)
ftp>
```

5.4 Accessing the CAF Scratch Disk using the ICAF Tools

The easiest way to interact with the scratch disks is by means of the ICAF tools. The ICAF Tools are a set of command line and graphical user interfaces that hide most of the complexity of the command line **ftp**.

5.4.1 Identifying your Scratch Disk Areas

The first information every user needs, is the list of Scatch Disk Areas he is allowed to access. To obtain this information, the

```
> icaf_info
```

command is available. This command prints out all the needed information in a human-readable form, like in the following example:

```
Node    : fcdfdata014.fnal.gov
Out dir : /cdf/scratch/sfiligoi/icaf
Scratch : /cdf/scratch/sfiligoi/scratch
Groups:
  Italy2
```

```
    Node    : fcdfdata026.fnal.gov
    Account : cdfdata
    Scratch : /cdf/scratch/cdfdata
  Italy3
    Node    : fcdfdata013.fnal.gov
    Account : cdfdata
    Scratch : /cdf/scratch/cdfdata
  Italy4
    Node    : fcdfdata015.fnal.gov
    Account : cdfdata
    Scratch : /cdf/scratch/cdfdata
```

As can easely be seen, the above information gives the user the node (fcdfdata014.fnal.gov here) on which the CAF Scratch Disk is located, together with the absolute paths of the Job Output and Scratch areas. (respectively /cdf/scratch/sfiligoi/icaf and /cdf/scratch/sfiligoi/scratch here)

In addition to these two standard areas, several users have also access to the CAF Servers bought by their groups. In the above case, the user has access to three areas, codenamed Italy2, Italy3 and Italy4. For each of them the progam has given him the node name, the login account (since these servers does not have a different account for every user) and the absolute path.

The above information, while complete, is not very suittable for use in scripts. For that use the following two commands are available:

```
> icaf_node
```

returns the name of the node on which the CAF Scratch Disk is located.

```
> icaf_groups
```

returns the list of codenames of the group areas.

5.4.2 Accessing your data

While it is usefull to have the complete information about the scratch area, most of the time users just want to access their data and in the most simple way too.

- To have the listing of files in the Job Output area, just use:

  ```
  > icaf_ls
  ```

  or use:

  ```
  > icaf_ls -f short
  ```

if you are not interested about the details of your files.

Like with ordinary **ls** the selection can be restricted to a subset of files, like in the following example:

```
> icaf_ls "*.tgz"
```

Notice the double quotes ("); they are needed to prevent the shell to interpret the wildcards (***** here).

- The most used action is getting data from the remote node. The command to use is:

  ```
  > icaf_get <filenames>
  ```

  It will copy the specified file(s) to the local directory. You can also use wildcards, just remember to quote the expression.

  If the current directory is not the right place to put your files, you can specify another directoy where to put them:

  ```
  > icaf_get <filenames> <dir>
  ```

  If you instead want to get a file and save it with another name, use the following syntax:

  ```
  > icaf_get <remote_filename> <local_filename>
  ```

  Wildcards cannot be used in this case.

  Or you can print the content of a file to the standard output:

  ```
  > icaf_get <remote_filename> -
  ```

  Usually **icaf_get** prints out a character for every 32K of data transferred; this is usefull on slow connections, but can be annoying when you transfer large files. To disable it, use the **-h** option; pass **0** to completelly disable it or a large number to get the progrees char only every `x` Kbytes:

  ```
  > icaf_get -h <Kbytes> <filenames>
  ```

- The command:

  ```
  > icaf_rm <filenames>
  ```

can be used to delete one or more files.

Alternatively, **icaf_get** can be invoked with the **-d** flag:

```
> icaf_get -d <filenames>
```

Used this way, the file(s) will be removed from the remote node once (and only if) the file has been copied.

- Although the scratch area should be used just for storing CAF outputs, sometimes it is usefull also to copy some files from your local machine to the scratch area. The command to use is:

  ```
  > icaf_put <filenames>
  ```

  It will copy the specified file(s) from the local directory to the scratch area.

  There are several parameters to configure it:

  ```
  > icaf_put [-d] [-h <nr_kbytes>] filename+ [remotename|remotedir]
  ```

  See the descriptions of the other commands above for more details.

5.4.3 Accessing the other scratch areas

The commands as presented above in Section 5.4.2 seems to be able to access only the data in the Job Output area. Instead, all of them can be configured to access also the other areas using the **-g** option.

```
> icaf_... [-g <group>] ...
```

where `group` is either **scratch** or the codename of one of the group areas.

Here is a real-live example:

```
> icaf_ls -f short -g scratch
icaf_gftp.py
GNUmakefile
GORETEX1.mpeg
icafcli.py
icaf_gftp.py~
temp_1-241.tgz
quota.pyc
quota.pyc1
quota.pyc2
```

```
quota.pyc4

> icaf_get -g scratch icaf_gftp.py - | grep icafcli
import icafcli
icafdict = icafcli.fetch()

> icaf_rm -g scratch quota.pyc4

> icaf_ls -g Italy4 -f short "top/jetsamples/*98-45622*"
top/jetsamples/jetsamples_beamlinetest_198-45622.tgz
top/jetsamples/jetsamples_beamlinetest_298-45622.tgz
top/jetsamples/jetsamples_beamlinetest_398-45622.tgz
top/jetsamples/jetsamples_beamlinetest_98-45622.tgz

> icaf_get -g Italy4 top/jetsamples/jetsamples_beamlinetest_98-45622.tgz - |\
  tar -tzf -
errorcaf-98.log
outputcaf-98.log
outputlocation.txt
run_secvtx_jetsamples.caf
secvtx_gjet09_98.log
secvtx_gjet09_98.lum
```

5.4.4 Checking your quota

Use:

```
> icaf_quota
```

to check the quota on the central scratch server.

If needed,

```
> icaf_quota <group>
```

can be used to check the quota on any of the group scratch servers.

Remember that to check the quotas on the local machine you can use

```
> quota
```

If quotas are not implemented, use:

```
> df -k .
```

to check the available space.

5.4.5 Using the GUI

While the command line tools allows you to do most of the work, it is often a bit tedious to use them when we are in a nice X environment. For this reason, a graphical interface to the kerberized ftp has been developed.

To start the GUI, use:

```
> icaf_gftp
```

An image of the ICAF GUI is shown in Fig. 5.4.5. The content of the local directory is displayed on the left side, whereas the Job Output area is shown on the right side. The URL fields on top of the window contain the paths of the visualized directories.

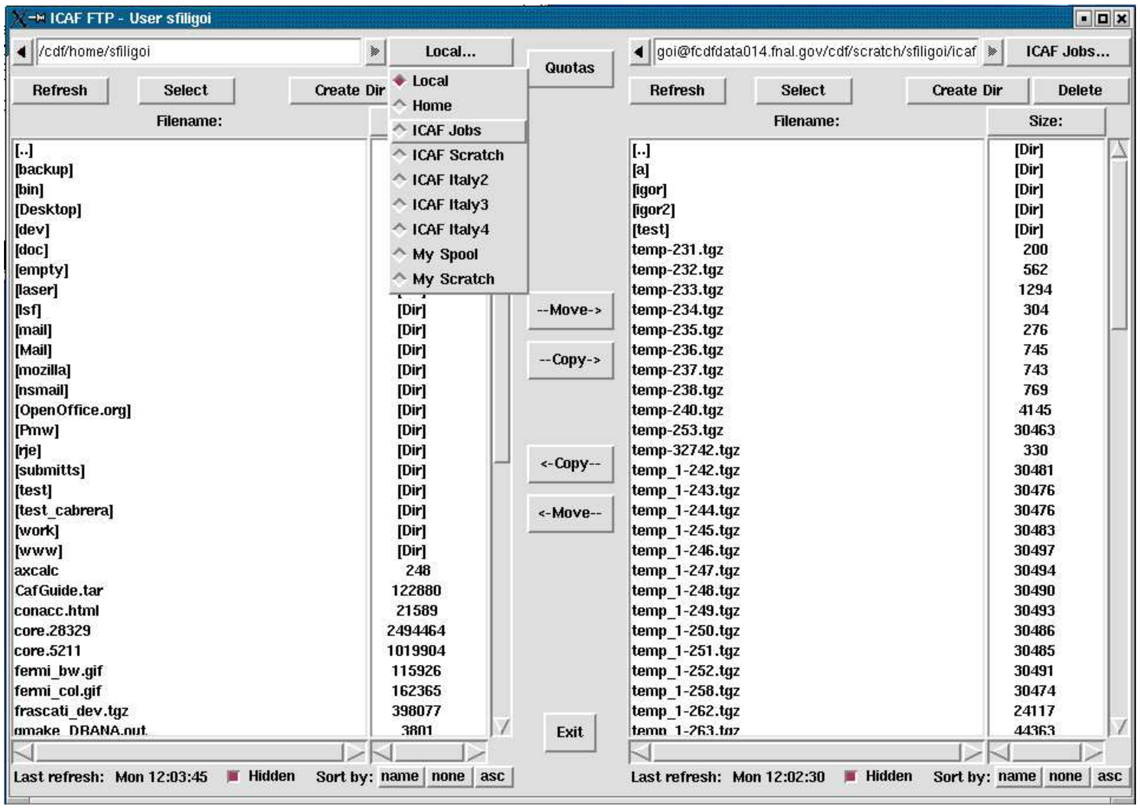

Figure 5: ICAF GUI

The two windows represent a point-in-time snapshot of the relative directories; at start, the file name and the file size listboxes are shown. When needed, the file size listbox can be changed with a listbox showing the file attributes or the creation time, using the button of the top of the listbox.

By deafult, hidden files (files starting with a dot) are not displayed; uncheck the **Hidden** checkbox if you want to show them.

The order of files in the window is determined by the sort options displayed in the bottom right corner. The files can be sorted by:

- name - file name
- ext - file extension (part after the first dot)
- size - file size
- time - creation time
- jobid - job ID, extracted from the file name
- none - unsorted

User can specify two sort options; the second being used when the first one finds out two equal results. The user can also decide if it wants the sorts to be ascending od descending.

Users can navigate through the local and remote filesystems by editing the URL field or simply by clicking on the directories (the names between square brackets) in the frames below. Another very usefull possibility is to use the shortcut button, placed next to the URL field. It allow the user to jump directly to:

- Local - current local working directory
- Home - user's local home directory
- ICAF Jobs - the Job Output directory
- ICAF Scratch - the ICAF scratch area
- ICAF ... - directories on the group accounts (like Italy2, Totonto, etc.)
- `<user-defined-URLs>` - user defined URLs, defined in the user's .icafrc (have a look at the .icafrc in the CafUtil/icaf directory for an example)

Each time you change the displayed directory, it is put in an internal history list. Using the buttons located near the USR field or by using Backspace, Left and Right keys, you can return to any previously visited directory and also come back to where you started. (like the Back and Forward buttons in a web browser)

Multiple files can be selected and copied (or moved) between the local and remote node or between two remote nodes by pushing the buttons in the central region. The selected files can also be removed by pressing the Delete key or by pressing the Delete button.

Please note that since the content of the directories can change and the windows show only a point-in-time representation, users are strongly encouraged to refresh the two windows (pressing the Refresh button) before performing any action, if a sensitive amount of time has passed since the last refresh (see the two labels on the bottom left of the window).

To select multiple files, the Control or Alt key must be held down while selecting the files (the usual GUI combination). In addition, the + (Plus) key can be used to select a subset of files using a pattern, while the * (Asterix) key can be used to invert the current selection. The later two does not apply to directories.

To create a new directory, press the Insert key or press the Create Dir button.

The same functionalities as described above, can also be obtained by means of a popup menu (using the mouse right button). Moreover, the popup menu can also be used to rename a file or to change the attributes of the selected files (see Fig. 5.4.5).

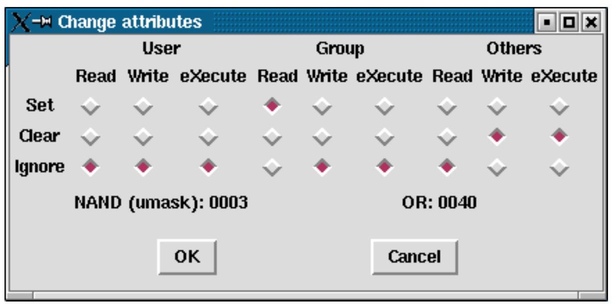

Figure 6: Attibutes changing

In addition, the user can also open any file just by double-clicking on it. The result is a popup window where the user has to select the action to take. Depending on the type of the file and its location, the possible choices may vary a bit; in Fig. 5.4.5 you have an example for a remote tar file, which has all the available options.

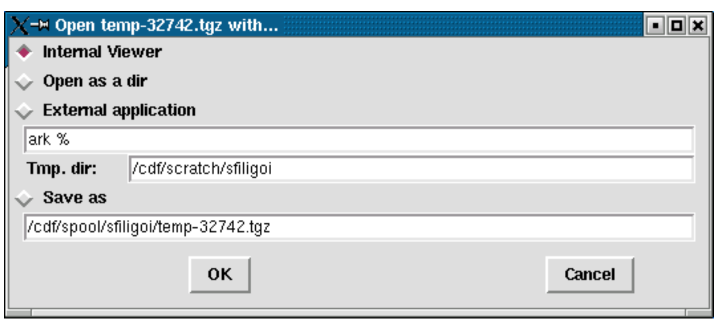

Figure 7: Open a file

The possible choices are:

- Internal viewer - Open the file using the internal viewer; for most files this means a simple text viewer
- Open as a dir - Available only for tar files, this option is equivalent to a virtual Change Directory.
- External application - Open the file using an external application; the application itself can be specified by the user, although a reasonable default is always present. The perchent (%) will be substituted with the file name.

  If the file is not local, it will be first copied to a local directory; the user can change the default temporary directory.
- Save As - The file will be saved with the specified name.

Last but not least, the user can also check its quota, both local and the remote ones, by pressing the Quota button. An image of the window that pops-up is shown in Fig. 5.4.5.

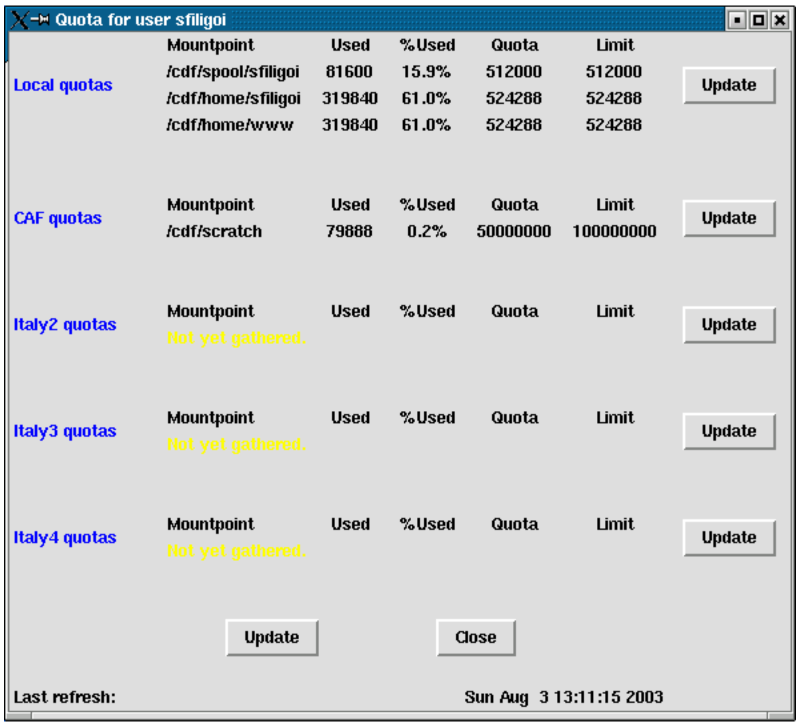

Figure 8: Quota window

5.5 Job summary email

The mailer is run as the last section of each job. It has an own high priority queue and its duties are:

```
1) Collect information by all the ExitLogFile written by CafExe process
for each segment in each node
2) Calculate aggregate and summary information
3) Archive the information in DHLoggerArchive.log file for farther analisys
4) Format and send email to the user
```

In each node, the CafExe process writes a log file and sends it to the head node at the end of job segment execution. The file contains information about the state of the job segment (success, failing, timing, error type, etc..), regarding input file, output file and dataset processing. To obtain I/O information, the AC++ executable creates a log file named logIOFile in which the information for each file processed by DHInput Module and FileOutput modules is stored. The file name is stored in an environment variable called IO_MONITORING_LOG_FILE. This environment variable should not be modified or unset.

5.5.1 Interpretation of email content

A typical mail can appear like this:

```
Date: Sat, 23 Aug 2003 20:34:53 -0500
From: cdfcaf@fcdfhead.fnal.gov
To: fella@fnal.gov
Subject: JID 829 summary

JID is 829.
Segment number from 1 to 5
Output location: icaf:temp_$.tgz
Submitted: Sat Aug 23 19:10:33 2003
Ended    : Sat Aug 23 20:34:53 2003
Job duration:   1:24:20

Segment times:    Mean       RMS
         Real:   1:01:49   0:16:00
          CPU:   0:43:05   0:01:24
         Wait:   0:00:21   0:00:01
                                                #Segments
                                                ---------
```

```
Completed with exit status = 0 (OK):                5
Completed with exit status != 0:                    0
Canceled:                                           0
Timeout or possibly canceled while running:         0
User oriented error:                                0
Caf system error:                                   0
Unknown error:                                      0

Total segments:                                     5

Segment   Node      Exit               Comment
      1 fcdfcaf010     0  Real: 1:12:10 CPU: 0:44:10 Wait: 0:00:21
      2 fcdfcaf009     0  Real: 1:12:26 CPU: 0:44:09 Wait: 0:00:21
      3 fcdfcaf018     0  Real: 0:44:17 CPU: 0:41:29 Wait: 0:00:23
      4 fcdfcaf019     0  Real: 0:44:25 CPU: 0:41:37 Wait: 0:00:22
      5 fcdfcaf015     0  Real: 1:15:48 CPU: 0:44:04 Wait: 0:00:21


Data access summary

Datasets: aexp08,hbot0h


INPUT data summary:

           RecRead EvtRead RO(sec) OC(sec) Size(MB) KbPerRec KbPerEvt FailOpen
Aggregate  7.8e+04 7.8e+04      26   18405  8.3e+03       --       --        0
Average    1.6e+04 1.6e+04     5.2  3681.0  1.7e+03    108.7    108.7        0


OUTPUT data summary:

           RecWrote EvtWrote OC(sec) Size(MB) KbPerRec KbPerEvt
Aggregate   2.3e+05  2.3e+05   55308  2.5e+04       --       --
Average     4.7e+04  4.7e+04 11061.6  5.1e+03    111.4    111.5


INPUT specific data:

Segment RecRead EvtRead RO(sec) OC(sec) Size(MB) KbPerRec KbPerEvt FailOpen
      1   15605   15598       6    4299  1.7e+03    108.7    108.7        0
```

```
      2   15605   15598       6    4314  1.7e+03    108.7    108.7         0
      3   15605   15598       4    2640  1.7e+03    108.7    108.7         0
      4   15605   15598       4    2645  1.7e+03    108.7    108.7         0
      5   15605   15598       6    4507  1.7e+03    108.7    108.7         0



OUTPUT specific data:

Segment RecWrote EvtWrote OC(sec) Size(MB) KbPerRec KbPerEvt
      1    46818    46794   12916  5.1e+03    111.4    111.5
      2    46818    46794   12961  5.1e+03    111.4    111.5
      3    46818    46794    7938  5.1e+03    111.4    111.5
      4    46818    46794    7954  5.1e+03    111.4    111.5
      5    46818    46794   13539  5.1e+03    111.5    111.5
```

The first part describes general job information, time table and error analisys. The second part entitled "Data access summary" describes information retreived by DHInput and FileOutput modules in AC++ system:

The list of datasets accessed:

```
Datasets: aexp08,hbot0h
```

and four tables : I/O Data Summary, Input per segment data, Output per segment data.

The first table shows aggregate and average for input data access. The fields are the following:

```
RecRead: total number of records read in segments
EvtRead: total number of events read in segments
RO(sec): total delay time between Request and Opening the input files
OC(sec): total delay time between Opening and Close of the input files
Size(MB): total number of MByte read
KbPerRec: kb read per record
KbPerEvt: kb read per event
FailOpen: number of failure in opening files
```

In second table shows aggregate and average for output data. The fields are the following:

```
RecWrote: total number of records written in segments
EvtWrote: total number of events written in segments
OC(sec):  total delay time between Opening and Close of the output files
Size(MB): total number of MByte written
KbPerRec: kb written per record
KbPerEvt: kb written per event
```

The last two tables report specific per segment information about input files accessed and output files generated.

6 Issues regarding Data Access and Management

In this section, we discus how to access data from jobs on the CAF. Unless data files are included in the user's input tarball so that they are local to the worker node running the job (not recommended!), remote means of data access are required. Currently, data files in root I/O format are accessed from the CAF using either dCache, rootd, or SAM[8]. SAM is in beta release and as a distributed data handling system will become increasingly important in CDF when we begin trying to use offline computing resources across the globe in a coherent fashion. It also has many new features to allow browsing of datasets, creation of private datasets, easier storage of files from any site in the world, and strict cataloging of all jobs that you have run.

This section is not meant to represent documentation for data handling, as much documentation on the subject exists elsewhere. We will primarily reference this documentation and focus on parallelization and other issues related directly to CAF-type jobs.

6.1 Using rootd for Remote Data Access

ROOT files can be read over the network using the TNetFile class in conjunction with the rootd server[9]. Data delivered via does not involve any local directory structure like NFS, but rather represents an 'on demand' data service. The remote machine you are trying to read data from using rootd must be running the rootd server *and* be configured to allow access to the particular ROOT files you are trying to access.

The standard AC++ input module DHInput provides rootd interface within the CDF Analysis Software Framework. This module is described in detail in [5]. The format for the include statement within DHinput using rootd is:

```
include file root://<rootd server name>//<directory name>.<file name>
```

Wildcards (*) are also supported in the above file name.

As an example of rootd access, lets say that johndoe wants to read skim.root in his scratch area on fcdfsgi2. In this case, his tcl file would contain:

```
talk DHInput
  include file root://fcdfsgi2.fnal.gov//cdf/scratch/johndoe/skim.root
  show include
exit
```

Of course, fcdfsgi2 must be running the rootd server and export users' scratch area, which is currently the case.

Since rootd is a file-based service, parallelization is essentially done "by hand". A simple way to run a rootd caf job with many segments would be break the total set of include statements into several tcl file fragments, each representing a different set of include statements sourced in a particular job segment. Each tcl file fragment would look something like:

```
include file root://<rootd server name>//<directory name>.<file 1>
include file root://<rootd server name>//<directory name>.<file 2>
include file root://<rootd server name>//<directory name>.<file 3>
...
```

The tcl file fragments would be named with a sequential set of integers, input_N.tcl with N = 1..# of files, one for each job segment to be run. The user's shell script would set an environment variable to the current segment number:

```
...
# Set enviroment variable to value of segment iterator
export SEGMENT_NUMBER=$1
...
```

which would be used in the main tcl file to source the corresponding tcl file fragment like the following:

```
...
talk DHInput
  source input_$env(SEGMENT_NUMBER).tcl
  show include
exit
```

6.2 Accessing Data on CAF Scratch Disk

This section follows up on discussion in Section 5.2.2 on using your CAF scratch disk as it pertains to rootd file access. Each FTP server serving the CAF scratch disk runs the rood server with read access to the directory /cdf/scratch which is top-level scratch area. So, let say johndoe this time wants to access a file skim.root in his CAF scratch area. His tcl file in this case would contain:

```
talk DHInput
  include file root://$env(CAFOUT_NODE)//$env(CAFOUT_SCRATCHPATH)/skim.root
  show include
exit
```

Using rootd with the scratch area provides a convenient way for users to run in their CAF job over data produced in a job. It also allows users to share ROOT data files with others in a read-only manner.

6.3 dCache

The primary way of accessing data from the CAF is with dCache. The dCache product provides a front-end disk cache to Enstore and the tape storage system. Its application to CDF data handling is described extensively in [6].

The dCache client interface in AC++ is also contained with DHInput. DHInput as it pertains to datasets, filesets, and dCache is described in both [5] and [6], so we just provide an example here.

Lets say that we want to access the entire hbot0h dataset, which resides in the book with name cdfpbot. Our tcl file would contain:

```
talk DHInput
  include dataset hbot0h book=cdfpbot
  cache set DCACHE
  show include
exit
```

Note the additional cache set DCACHE indicating that we want dCache to manage access to this particular dataset.

6.3.1 Parallel Job Segments and dCache

Given the volume of data in a typical dataset, it is very unlikely that user would run a single CAF job segment for access to the entire dataset using dCache. Here we present the suggested way of parallelizing dCache jobs within the CAF.

The DHInput module provides a splitInput command that can be used in parallelization. The format of splitInput is shown below:

```
splitInput slots=<# of dataset pieces> this=<piece # for this job>
```

Note: The this parameter starts at **0!** (i.e. the first piece is this=0).

The relation between slots and this should be quite obvious:

- slots $\leftrightarrow$ # of job segments

- this ↔ (segment iterator - 1)

Note that the above relationship between this and the segment iterator assumes that the segment iterator starts at one. In the our suggestion for how to organize a parallel dCache CAF job, it is not necessary for the segment iterator to start at one.

The shell script (caf_dcache.sh) one would use to run a dCache job using parallel job segments could look something like this:

```
#!/bin/sh

# Normal cdfsoft setup
...

# dCache related environment variables
#    First parameter is the segment iterator, denoted by $ in submission
#    Second parameter is the lower segment range
#    Third parameter is the upper segment range
export NUM_SEGMENTS=‘expr $3 - $2 + 1‘
export DCACHE_ITER=‘expr $1 - $2‘

# Analysis execution command, clean up, etc
...
```

The DHInput part of the tcl file would be:

```
talk DHInput
  include dataset hbot0h book=cdfpbot
  splitInput slots=$env(NUM_SEGMENTS) this=$env(DCACHE_ITER)
  cache set DCACHE
  show include
exit
```

If the user wants to run a dCache job with 100 segments, they would send to the CAF an Initial Command of

```
./caf_dcache.sh $ 1 100
```

and a segment range from 1 to 100.

6.4 Accessing Data with SAM

An alternative way of accessing data from the CAF is with SAM. This has been released for open-beta testing by users who are willing to help with debugging the system.

The SAM product provides connection between CAF Jobs and their data consumption, taking the role of the current Data Catalog but with more functionality. When installed on the resources of remote institutes, it is the data handling software for the CDF DataGrid. On the CAF, it keeps track of the files in Enstore which in turn interfaces to the tape storage system. Documentation for SAM may be found in [8].

The SAM client interface in AC++ is also contained with DHInput. DHInput as it pertains to datasets and filesets operates in the same way as with DCache.

We use the same example as in [?] for hbot0h. The syntax is nearly the same, but books are not needed to specify the dataset and you give the dataset in the CAF GUI, then pick it up in an environment variable. The reason for this is that by telling the CAF your dataset, the CAF can more efficiently schedule the matching of cpu to your files.

The SAM tcl file would be:

```
module disable ConfigManager
talk DHInput
# Adjust the max files to ensure you dont time out!
  maxFiles set 40
  include dataset $env(SAM_DATASET)
  cache set SAM
  show include
exit
```

There is an command analogous to DCache cache set SAM indicating that we want SAM to manage access to this particular dataset. The disabling of the ConfigManager is needed to ensure the program does not crash after waiting for a tape and loosing its database connection. Since files will automatically be delivered to each process, and since it can take time for processes to get started, it is necessary to ensure that the maximum number of files any process gets can be processed within the time allowed for the batch queue. For example, if 15 files take 30 min (typical AC++Dump time), then 60 files is the maximum that the short queue (2 hours) can handle. You would want to make sure you set a maxFiles to 50 or 55 to be on the safe side. If you have 690 files in your dataset (as jbot0h does) then you need 13 processes and would want to submit "from 1 to 13". Since CAF puts 10 procesees in a section, you may want to submit 1 to 20.

The shell script (caf_sam.sh) one would use to run a SAM job would look like:

```
#!/bin/sh

# Normal cdfsoft setup
...
#SAM setup
setup sam -q prd
export CDF_USER_NAME=stdenis
```

```
# For Beta testing, please use this!
export SAM_INPUT_DEBUG=1
# Analysis execution command, clean up, etc
...
```

The only difference is that you have to setup sam explicitly.

6.4.1 Mini Tutorial on how to create datasets using SAM

To try to start using sam you need to at the very least know how to create yourself a sam dataset, and how to verify what files are in the dataset you have defined.

This can be done using 3 simple commands:

- sam translate constraints
- sam create dataset definition
- sam create snapshot

The first allows you to querry the dataset for all files that pass certain criteria. E.g., to get all files in xbhd0d within the run range]162900,162948] would be:

```
setup sam -q prd
sam translate constraints --dim="cdf.dataset xbhd0d and RUN_NUMBER > 162900 and RUN_NUMBER <=162948"
```

To then turn this specification into a dataset definition you would do:

```
setup sam -q prd
sam create dataset definition --defname="janedoe-dataset-No1" \
    --group=test --defdesc="for a how to example" \
    --dim="cdf.dataset xbhd0d and RUN_NUMBER > 162900 and RUN_NUMBER <= 162948"
```

At this point you have created the dataset definition “janedoe-dataset-No1” but have not yet frozen its content. If new files were added to xbhd0d within your run range than your dataset definition will pick those up until you freeze it using the following commands:

```
setup sam -q prd
sam create snapshot --defname="janedoe-dataset-No1"
```

If you from now onwards do:

```
sam translate constraints --dim="dataset_def_name janedoe-dataset-No1’’
```

than this will always give the same list of files, even if new files for xbhd0d in your run range are added in the future. You have thus frozen your dataset into a "snapshot" which you can use for analyzing data without fear of the underlying files ever changing.

Let's assume at some later point you want to check if somebody did indeed add more files to xbhd0d into your run range. You can do so as follows:

```
setup sam -q prd
sam translate constraints --dim="cdf.dataset xbhd0d and \
         RUN_NUMBER > 162900 and RUN_NUMBER <=162948 \
 minus dataset_def_name janedoe-dataset-No1 "
```

The keywords "minus" and "plus" allow you to add and subtract datasets (i.e. previously frozen snapshots of dataset definitions) from each other to your hearts content.

A host of other useful information may be found at [7].

6.4.2 Testing code on lnx2 using SAM

If you want to use SAM on lnx2 for testing, you have to set some setup sam and environment variables:

```
setup sam -q prd
export CDF_USER_NAME=myname
export SAM_STATION=cdf-sam
export SAM_GROUP=test
SAM_PROJECT=$CDF_USER_NAME$SAM_STATION'date | sed s/'[ :]*'//g'; export SAM_PROJECT
```

For the sam project name you can choose anything you wish, but it must be unique. Hence putting your date and user name is fairly safe.

It is also useful to know that for debugging problems,

```
export SAM_INPUT_DEBUG=1
```

can be helpful.

6.4.3 Parallel Job Segments and SAM

One of the features of SAM is automatic parallelization of files. However, as with any automatic feature, it is easy to shoot yourself in your foot. In this case, you should be sure to try to keep the number of sections you request about a factor of two or three smaller than the number of files you have. If you don't, you will get back some core dumps for the jobs that did not get any files. All your data will be processed though.

7 I've Run Into Some Sort of Trouble - Now What do I Do?

Successful submission and execution of your CAF jobs involves many intermediate steps, any of which could fail. Problems you encounter in using the CAF may be due system problems or could be due to some oversight or misunderstand on the part of the user. This section provides suggestions on how to debug your problem and instructions on how to seek help from experts.

7.1 Proper Steps to Take in Resolving Your Problem

Please take the following steps in resolving your problem:

1. *Check for email from CDF_CAF_USER*
   Each user of the CAF should be a member of the CDF_CAF_USER mailing list, as all important system related notifications are sent to CAF users via this list. If you are not currently a member of this list, join at

   ```
   http://listserv.fnal.gov/archives/cdf_caf_user.html
   ```

   Please read any email you've recently received from this list before reporting any problems, as they may be due to CAF down time or other known and reported problems.

2. *Reproduce problem on the standard linux platform - fcdflnx2*
   All CAF users also have an account on fcdflnx2, the standard linux platform for the CAF. This machine has the same Linux OS distribution and CDF software environment as the CAF worker nodes. Problems with either job submission or execution should always be checked on fcdflnx2 before asking for help. This is not to say that definite problems do not exist if they are not reproduced on fcdflnx2, just that this information will be used to diagnose the problem further.

3. *Ask yourself critically - "Could my problem be unrelated to the CAF?*
   This part relates to failed job execution. The CAF is a vehicle to run CDF analysis jobs. If those analysis job are fundamentally flawed (e.g. some serious code bug), they will fail on the CAF as they will on an interactive desktop or *fcdflnx2*. Also, the problem could be related to some combination of Code Management, Database or Data Handling systems, which needs to be addressed by a different set of experts from those within the CAF Group. Jobs should always be tested as thoroughly as possible on a local linux machine before submitted to the CAF.

4. *Consult the documentation*
   Look for a discussion related to your problem area in this document. In particular, see if you problem has already been addressed in the Troubleshooting and/or Frequently Asked Questions (FAQ) sections. An up-to-date User's Guide in both postscript and HTML format is available from the CAF Home Page - http://cdfcaf.fnal.gov.

5. *Search the cdf_caf archives*
   Since the CAF has been in production operation since May 2002, most of the normal problems users encounter have already been raised and solved in discussions on the cdf_caf mailing list. You should thoroughly search this list before asking for help from the CAF experts. The archives of this list are available at:

   `http://listserv.fnal.gov/archives/cdf_caf.html`

6. *Ask your favorite fellow CAF user*
   When we refer to "CAF experts", we are not just referring to those who designed or operate the CAF. The CAF in its current state is the result of excellent feedback over time from those who actually use it to get their work done. You are encouraged to tap into the expertise and experience of your collaborators that use the system.

7. *Send email to cdf_caf@fnal.gov*
   If your specific problem has not been resolved after the previous steps, send email to the cdf_caf mailing list with a reasonably detailed description of your problem. "Why can't I submit?" or "Why did my job fail?" is not sufficient amount of information. Please provide the submission error you are getting (from both your normal submission platform *and* fcdflnx2, if these are different) or the job ID (JID) and segment number of the failed job or job segment.

A Maintaining Your Caf Group

Recall from the discussion in Section3.5 that some groups in CDF have contributed CPU resources to the CAF. These groups get preferential but not exclusive use of these resources. Each CAF group has a list of administrators that are given the authority add and delete members from their group. The people allowed to control the roster of members in each group is maintained by the CAF administrators.

A.1 Adding or Deleting Members

We provide a interface which will authenticate group administrators via kerberos and allow them to view and modify their group roster. This interface is called `updategroup` and is part of the development release of the CDF software in the same way as the other CAF tools. The usage of `updategroup` is the following:

```
usage:  updategroup [-u filename|-r groupname]
        -u: Update to server
        -r: Retrieve from server
```

As the above usage implies, the -u option is used to upload a roster file and the -r option is display the group roster.

As an example, lets say that `johndoe` is the administrator of the MIT group which currently has three members: himself, `janedoe`, and `msn`. He wants to remove `msn` from the MIT group. He would do the following:

- *Write the current list of MIT members to a file*: The output of

  ```
  > updategroup -r mit
  ```

  is the following:

  ```
  Analysis Farm: caf    Host: fcdfhead1.fnal.gov

  mit
  johndoe
  janedoe
  msn
  ```

  The CAF administrator (`johndoe`) should redirect this output to a file:

  ```
  > updategroup -r mit > mit_roster.txt
  ```

- *Modify the roster file*: The mit_roster.txt file must be edited to include only the group name as the first line and the group members (minus msn) as the subsequent lines. The contents of the file mit_roster.txt is now:

```
mit
johndoe
janedoe
```

- *Upload the file*: The CAF administrator now modifies the MIT group roster by uploading the file mit_roster.txt:

```
> updategroup -u mit_roster.txt
```

B CAF Client Software Interface to Other Farms

The CAF client software, including the job submission interface, control/monitoring tools, and ICAF tools, are completely configurable for other analysis farms.

B.1 The .cafrc file

The information on each analysis farm is contain within a single configuration file called .cafrc. This file has a very simple format:

```
[global]
af_list=<farm 1>,<farm 2>,...
default_af=<farm 1>

[<farm 1>]
<information about farm 1>

[<farm 2>]
<information about farm 2>

...
```

As you can see, the .cafrc file has a global section and a section for each analysis farm. Within the global section, there is a list of analysis farms which are available to the user and a default choice. Each allowed analysis farm in af_list must have an associated section with the same name that contains information about the farm. Note that there can be more farm sections than farms listed in af_list. This allows for one to add information about other analysis farms without actually making them directly available for use - a placeholder, in essence.

Each farm section contains specific required fields. The current fields are listed below:

- host: Hostname of machine running the submitter, monitor, and groupupdate daemons. This is typically referred to as the *headnode* of the farm.
- submitter_port: Port number at which the submitter daemon listens
- monitor_port: Port number at which the monitor daemon listens
- groupupdate_port: Port number at which the groupupdate daemon listens
- svc: Principal used on the headnode to authenticate users. The nominal CAF installation uses the host principal of the headnode, so this is usually set to "host"

As an example, lets say that johndoe wants to be able to interface to both the CAF at FNAL and his own farm at MIT on which there exists a CAF installation. His .cafrc file might look something like:

```
[global]
; Global definitions
;
; List of available analysis farms and default choice
af_list=caf,mitcaf
default_af=caf


[caf]
; CDF Central Analysis Farm
;
; Headnode name, submitter port, and service/host principal
host=fcdfhead2.fnal.gov
submitter_port=8000
monitor_port=8020
groupupdate_port=8040
svc=host
;
; List of available process types and default choice
pt_list=short,medium,long,test,cdfmc,cdfopr,long_mit
default_pt=short


[mitcaf]
; johndoe's farm at MIT
;
; Headnode name, submitter port, and service/host principal
host=cdfhead.mit.edu
submitter_port=8000
monitor_port=8020
groupupdate_port=8040
svc=host
;
; List of available process types and default choice
pt_list=short,medium,long,test
default_pt=short
```

One might wonder how the CAF client software knows where to look for this file. The search order is the following:

- first look in user's home directory
- then look in user's current directory
- then look in $PROJECT_DIR/CafUtil/cdf_gui/
- then give up and use hard-coded defaults (defaults to CAF at FNAL)

B.2 Selecting the Current Working Farm

CafGui will try to read and use the information contained in the .cafrc file using the search order described in Section B.1. In this way, the Analysis Farm field in CafGui is a pull down menu of available analysis farms (those listed in af_list) with the other fields Process Types dynamically adjusted to reflect the information in the .cafrc file.

For command line tools, an environment variable named CAF_CURRENT is used to select the desired analysis farm to which to apply a particular command. In our example, johndoe could select to inquire about his jobs running his MIT CAF by typing the following (under bash):

```
> export CAF_CURRENT=mitcaf
> cafjobs

Analysis Farm: mitcaf    Host: cdfhead.mit.edu



 JID        Queue        From   To    Status
No jobs related to your queue
```

Note that the export command renders the environment variable being set as persistent within the shell. Therefore, setting CAF_CURRENT in this way can be thought of as setting the current analysis farm to be used in all CAF command line tools until it is changed.

C Troubleshooting

C.1 Problems starting up the GUI

Problem: You type:
`CafGui`
And all you see is some gibberish on the command line but no GUI on your screen.
Possible Solutions:
You may have forgotten to set your `DISPLAY` environment variable.
The tcl library may not exist on your system. To check this type: "setup tcl v8_3_1". If you get an error than your cdfsoft products installation is incomplete.
Your kerberos installation may be non-standard, or in a non-standard place. It should be in /usr/krb5/lib .
You may be missing python in your PATH. To check this try: "which python".
You need to execute CafGui script in the same directory that also contains CafGui.py and krb5module.so for now. E.g. if you copied the script into your bin area then it wouldn't work any more. This feature will get fixed shortly.

C.2 Problems with kerberos

Problem: You see an error message involving: "ImportError: libkrb5.so:" on your screen.
Solution: Verify if your installation of kerberos is either in the wrong place, or doesn't include the right stuff. It's supposed to be in /usr/krb5/lib . If it's somewhere else then change the appropriate line in the shell script CafGui appropriately. Then try again. If you still have trouble try "locate libkrb5.so". If this doesn't exist then your kerberos installation is sufficiently non-standard that you need to ask your system administrator to contact us at cdf_caf@fnal.gov to straighten things out.
Problem: You submit a job, it runs, but the output never comes back to the place you designated. The email you get back says something about "/usr/krb5/bin/rcp ... Permission denied".
Solution: There are a number of different causes for this:

1. Your .k5login file isn't set up correctly. Make sure there are no spaces after "cdfcaf/fcdfhead001.fnal.gov@FNAL.GOV".

2. Your .cshrc does things that are impossible in a non-interactive shell. Check out fcdflnx2:~fkw/.cshrc to see how you can safeguard against this. Then verify that it works by doing an rcp of some file into the designated location from a remote PC. E.g.

```
ssh fcdflnx3
rcp .cshrc fcdflnx2:~/test
```

3. The desktop you want to send your output to doesn't have a kerberos server running. This may be the case if you can't rcp into your desktop even after you take care of the .cshrc issues mentioned above. When in doubt, try it out on fcdflnx2 first to make sure your .cshrc issues are resolved and then on your desktop. If it works on fcdflnx2 but not on your desktop then you need help from your system administrator.

C.3 Is the submitter working?

If the job doesn't run and the message box says:
`ERR:Connection refused. Submitter information might be changed or submitter is down` means that the submitter is down. Send email to cdf_caf@fnal.gov and complain.

One of us will then check if it is working by logon the caf server and:

```
cdfcaf@fcdfhead001 submitter]> ps auxw|grep submit
 cdfcaf    4741  0.0  0.2  4580 2936 pts/2    S    13:53   0:00
 /local/ups/prd/python/v2_1/Linux-2-4/bin/python ./submitter.py
```

If you cannot see that line, it is not running. So you can try to restart it by:
`/etc/init.d/submitter start`

C.4 Incompatibility from RH 6.1 to RH7.1?

Problem: The job didn't run. After untarring the received file the ___error.txt file contains the line:
`Cannot find libreadline.so.3`. **Solution:** Add the following lines to your bash shell script, just before the command for the `AC++` job.

```
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
ln -s /usr/lib/libreadline.so.4 libreadline.so.3
ln -s /usr/lib/libhistory.so.4 libhistory.so.3
ln -s /usr/lib/libncurses.so.5 libncurses.so.4
```

If you are using csh/tcsh then replace the `export ...` above by:
`setenv LD_LIBRARY_PATH .:$LD_LIBRARY_PATH`

Explanation: For reasons unknown RH6.1 and RH7.1 use some binary compatible libraries but have them named differently. If you compile on a RH6.1 system and try to run your job on the CAF which is all RH7.1 then you are stuck with unresolved symbols, and your exe crashes. Adding the softlinks resolves the symbols.

D Frequently Asked Questions

References

[1] *FBSNG - Next Generation of FBS* `http://www-isd.fnal.gov/fbsng/`

[2] R. St. Denis: CDF Note 5543 *Offline User's Guide*

[3] K. Bloom: CDF Note 5294 *Getting Started with the CDf Run II Offline*

[4] M. Neubauer, *et. al.*: CDF Note 6001 *Recommendations for University-owned Computing for CDF Run2*

[5] F. Ratnikov: CDF Note 5336 *Input and Output Modules user guide*

[6] R. Kennedy CDF Note 5981 *Introduction to dCache Use at CDF*

[7] `http://projects.fnal.gov/samgrid/cdf/cdf.html`

[8] A. Boehnlein, *et. al.* CDF Note 6096 *Proposal for the SAM and SAM-Grid Joint Project Organization*

[9] *Accessing ROOT Files Remotely Via a rootd Server* `http://root.cern.ch/root/NetFile.html`

[10] *FCP - Farm Remote Copy Utility* `http://www-isd.fnal.gov/fcp/`

[11] With many thanks to I. Mandrichenko for providing us useful scripts and examples.