

# Use of SAM-SRM Interface for Movement of MC Data

Manoj Kumar Jha

*INFN-Bologna, Italy*

Gabriele Compostella, Antonio Cumo, Donatella Lucchesi, Simone Pagan

*INFN-Padova, Italy*

Doug Benjamin

*Duke University, Durham, USA*

Robert Illingworth

*Fermilab, Batavia, IL, USA*

## Abstract

The CDF experiment has generated more than  $2 fb^{-1}$  of raw data. As much as same amount of Monte Carlo(MC) data are needed for detector understanding and physics analysis. It is not feasible to produce these MC data on-site due to limitation on computing resources. The CDF remote sites or Grid Tier1 and Tier2 may be utilized for producing MC data. From our past experience, we learnt that one of the most important limitation in the heavy usage of off site resources is that the Worker Nodes(WN) were sitting idle just because another WN was transferring data to destination site at Fermilab. This situation leads to inefficient uses of computing resources and sometimes to the loss of the output. Hence, a framework is needed for transportation of MC data from remotes sites to Fermilab.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Prototype Model</b>	<b>3</b>
<b>3</b>	<b>Testing of Prototype Model</b>	<b>4</b>
<b>4</b>	<b>Data Movement from WN to SRM</b>	<b>4</b>
<b>5</b>	<b>Data Movement from SRM to SRM: Single File Transfer</b>	<b>4</b>
5.1	Composition of Files . . . . .	6
5.2	Performance Parameters . . . . .	7
5.2.1	Transfer Time . . . . .	7
5.2.2	Transfer Rate . . . . .	8
5.2.3	Latent time . . . . .	8
5.2.4	Cumulative Flow of Data . . . . .	9
<b>6</b>	<b>Implementation of the Prototype Model</b>	<b>11</b>
6.1	Authentication . . . . .	14
6.2	User Interface for Job Submission . . . . .	15
6.2.1	CAF Graphical User Interface(CafGui) . . . . .	15
6.2.2	CAF Command-line User Interface(CafSubmit) . . . . .	16
6.3	Monitoring and Control of CAF Jobs . . . . .	17
6.4	Job Output . . . . .	18
6.4.1	Accessing the CAF Scratch Space on SRM . . . . .	19
6.4.2	Identifying User's Scratch Space on SRM . . . . .	19
6.4.3	Accessing User's Data . . . . .	19
6.4.4	Accessing the other Scratch Areas . . . . .	21
6.4.5	Checking User's Quota . . . . .	21
6.4.6	Graphical User Interface . . . . .	21
<b>7</b>	<b>Future Plan</b>	<b>21</b>

# 1 Introduction

The CDF [1] experiment has generated more than  $2 fb^{-1}$  of raw data. As much as same amount of Monte Carlo(MC) data are needed for detector understanding and physics analysis. It is not feasible to produce these MC data on-site due to limitation on computing resources. The CDF remote sites or Grid Tier1 and Tier2 may be utilized for producing MC data. From our past experience, we learnt that one of the most important limitation in the heavy usage of off site resources is that the Worker Nodes(WN) were sitting idle just because another WN was transferring data to destination site at Fermilab. This situation leads to inefficient uses of computing resources and sometimes to the loss of the output. Hence, a framework is needed for transportation of MC data from remotes sites to Fermilab.

## 2 Prototype Model

Figure 1 shows a prototype model for movement of MC data from worker nodes to the destination Storage Resource Manager(SRM) [2]. The destination SRM in our case will be at Fermilab. Following are the components of the prototype model

- A SAM [3] station (not shown in the figure) and temporary space of around few Tera Bytes(TB) on SRM which are closer to each grid access sites. The grid access sites are LCGCAF, NAMCAF and PACCAF. More information on CAF portal can be found in [4]
- A SAM station and space on SRM near to the destination site. The destination site is at Fermilab.

This prototype model relies on integration of SAM with SRM. We used SAM because it is the default data handling framework of the CDF. The reasons for using SRM are to avoid unnecessary complications which may arise from different Storage Elements (SE) at different portal sites. The latest version of SRM in field is 2.0. The present version of SAM is compatible with SRM version 1.0 and it needs some development in order to be compatible with the SRM version 2.0.

The user jobs will run on the Worker Nodes(WN) and output data from these jobs will be transferred to SRM of portal site. A process at portal site will check the arrival of new file and its metadata in the SRM. The metadata is a file of the order of few bytes which contains all the information related to the MC output data. Using information from metadata, the same process will register these files into the SAM station of the grid sites. The process at destination site communicates with the SAM station of the portal site for its arrival of new file. The new file and its metadata will get transferred to the destination SRM through SAM station to station copy. Once the new files and its metadata arrives at destination SRM, its counterpart from the SRM at portal site will be erased. In this way, the used space in the SRM of portal site will be created again. The validation team will validate these new files and correct files will be moved

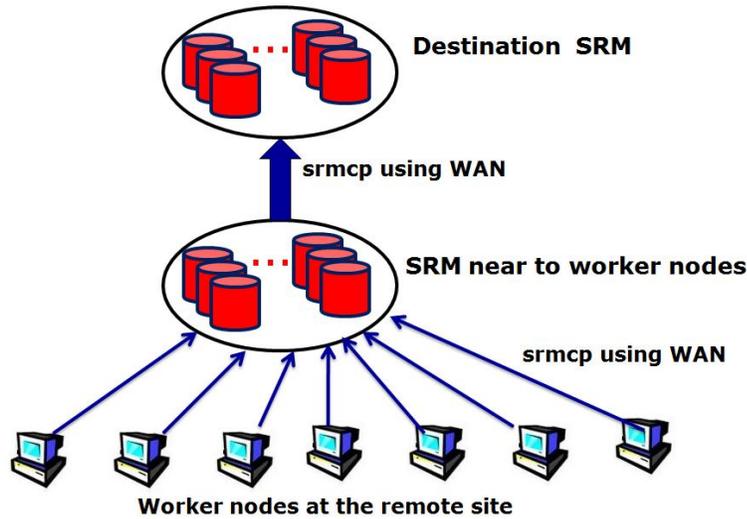


Figure 1: A prototype model for transfer of MC data from worker nodes to destination SRM.

to tape for storage. In the end, SAM database entries for arrival of new files in tape will be modified.

### 3 Testing of Prototype Model

Following steps are involved in evaluating the performance of the above proposed model.

1. Evaluation of performance parameters for data movement from WN to SRM near to portal site as mentioned in section 4.
2. Evaluation of performance parameters for data movement from SRM closer at portal site to destination SRM at Fermilab has been considered in section 5.

### 4 Data Movement from WN to SRM

### 5 Data Movement from SRM to SRM: Single File Transfer

A prototype model has been setup for movement of data between Storage Element(SE) at Fermilab and UCSD. Both the SE are being managed by dCache [5] based SRM. Figure 2 shows the implementation of the test model. The following SEs have been used for setting up the framework.

1. dCache managed SRM at Fermilab: “srm://cmsrm.fnal.gov:8443/srm/managerv2?SFN=/resilient/NONCMS\_GUEST\_30DAYLIFETIME/cdfguest/McData/”
2. dCache managed SRM at UCSD: “srm://t2data2.t2.ucsd.edu:8443/srm/managerv2?SFN=/pnfs/sdsc.edu/data2/cdf/McData/”

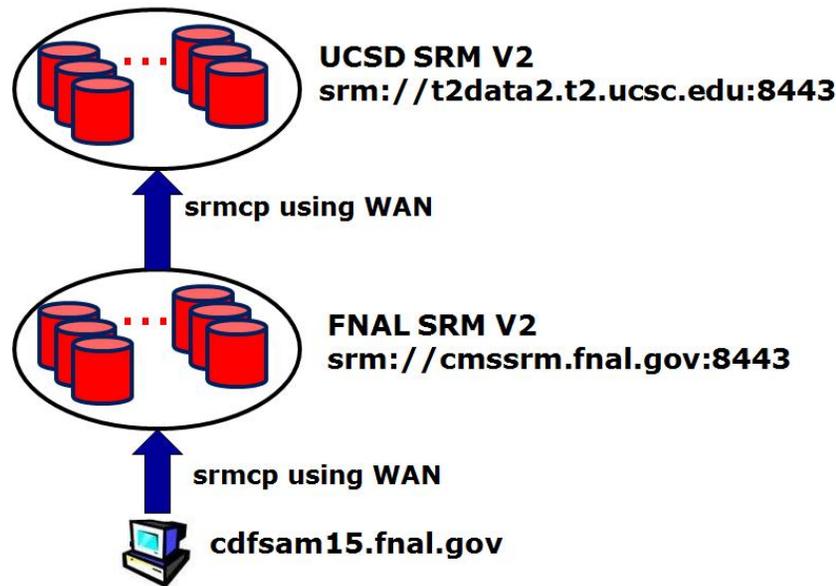


Figure 2: A test framework for movement of data between the SRMs.

A process is running on the `cdfsam15.fnal.gov` which create dummy files of random sizes and transfer the files to SRM at FNAL using `srmcp` [6]. This process generates 10 dummy files of random sizes. A cron job is being run at regular interval of time (15 minutes) for continuous creation of files. A different process is running on “`cdfsam15.fnal.gov`” for transfer of file from SRM at FNAL to SRM at UCSD. The following steps are involved between transfer of file between the two SRMs.

1. List the files available in the SRM folder at FNAL.
2. Pick a file from the list (say A).
3. Get a TURL(Transfer URL) corresponding to A . It is basically a `gsiftp` string.
4. Note the start time for copying of file A in the log file.
5. Copy the file A using `srmcp` between the SRM at FNAL and UCSD. We used all the default options of `srmcp`. At UCSD, the data is buffered through 3 heavy-duty grid ftp servers on WAN. Each grid ftp servers allows 50 streams per client.

6. When the file A is successfully transferred
  - Print the file name, size, start and end time for copying of file A in the log file.
  - Delete the file A from SRM at FNAL when it is successfully transferred in SRM at UCSD.
7. Go to the step 1 and repeat rest of the steps until all the files in the list is being transferred to SRM at UCSD.

A cron job is running which runs the copy process from FNAL to UCSD SRM at regular interval of time. In this document, we are summarizing the results for which the files transfer continuously takes place for a week between the two mentioned storage elements(SEs).

## 5.1 Composition of Files

We have used dummy files of random sizes. The random number generator has been biased intentionally for generating file sizes of larger values. The dummy files have been generated in size from few Mega bytes (MB) to 4 Giga bytes (GB). Figure 3 shows the generated dummy files in bins of 100 MB. For the comparison purpose, the dummy files have been divided on the basis of its sizes into different categories. Table 1 list the different categories of the dummy files based on its sizes. Figure 4 shows the composition of files used for testing the framework.

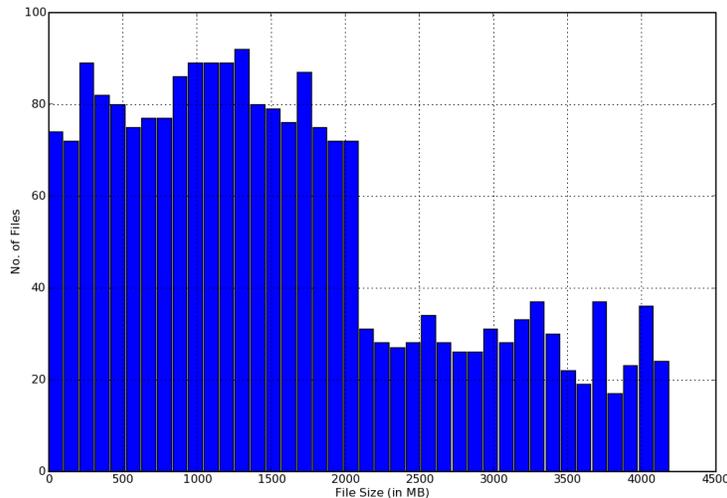


Figure 3: Distribution of file size.

Type	File Size (fsize)
A	$fsize \leq 10 \text{ MB}$
B	$10 \text{ MB} < fsize \leq 100 \text{ MB}$
C	$100 \text{ MB} < fsize \leq 1 \text{ GB}$
D	$1 \text{ GB} < fsize \leq 2 \text{ GB}$
E	$2 \text{ GB} < fsize \leq 3 \text{ GB}$
F	$fsize > 3 \text{ GB}$

Table 1: Different categories of files based on its size.

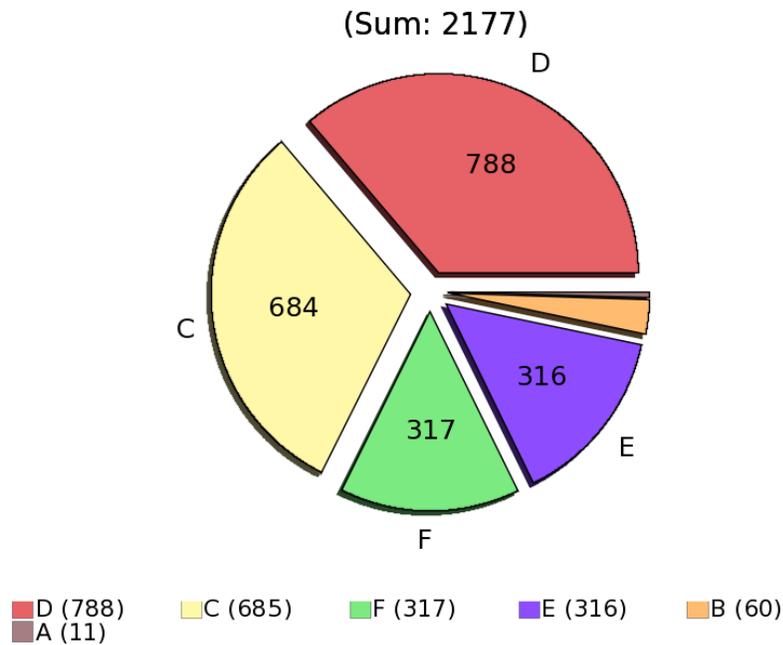


Figure 4: Composition of files which are used for transfer between two SRMs. The numbers in the pie chart indicate the number of files being generated in the particular category.

## 5.2 Performance Parameters

In the following sections, we are showing some parameters that will help us in evaluating the performance of the framework.

### 5.2.1 Transfer Time

Figure 5 shows the time needed for copying of files between the SRMs. SRM initialization time for dCache is of the order of around 10 seconds which can be seen from the transfer time needed for file types A and B in Figure 5. It is also clear from the figure

that transfer time increases proportionally with the file size. In file type C of Figure 5, there is a band of structure in transfer time and it may be due to fluctuations in the network performance. These fluctuations are not visible in other file types D and F of Figure 5 due to larger scales.

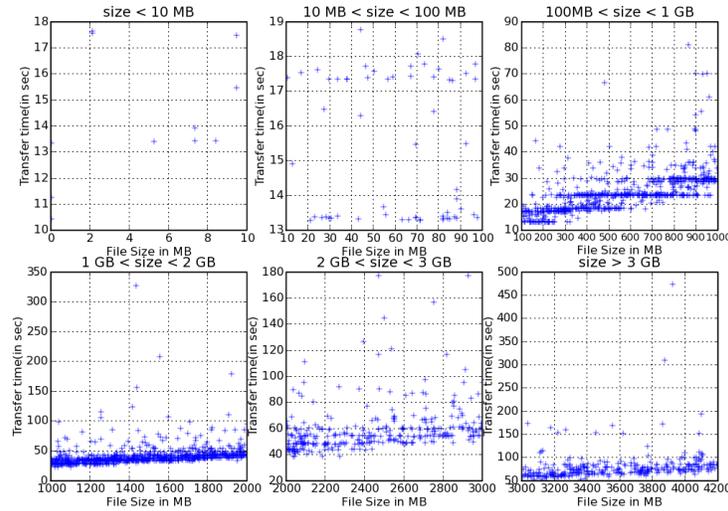


Figure 5: Time taken by different categories of files for copying between SRMs.

### 5.2.2 Transfer Rate

Figure 5.2.2 shows the transfer rates for different types of file between SRMs. It is evident from the figure that transfer rate increases with file size up to certain extent. In Figure 5.2.2, there also exist a band in transfer rate and it may be due to day to day fluctuations in network performance. The output of user analysis job will fall in the file type B while that of MC jobs lie in C and D. In these cases, the file transfer rate continuously increases with file size.

A qualitative comparison between transfer time and its rate can be seen from the Figures 7. The numbers in the left and right hand side of the pie-chart represent the average transfer time and rate for each category of the file types respectively. A quantitative comparison can be made between transfer time and rate for different file types. For example, the transfer time for file type A is larger in comparison to its transfer rate while for file types D and E, it is opposite.

### 5.2.3 Latent time

In Figure 5.2.3, we tried to show the amount of time being not used(latent) in transfer of the files in the framework. In transferring file from SRM at Fermilab to UCSD, the

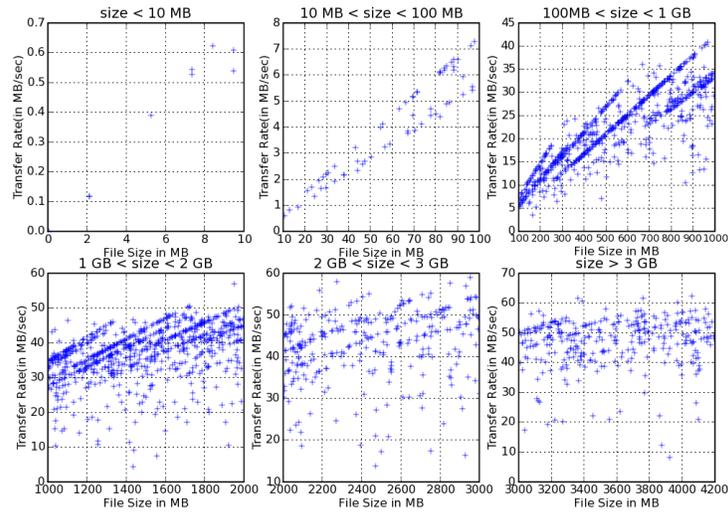


Figure 6: Transfer rate for different categories of files.

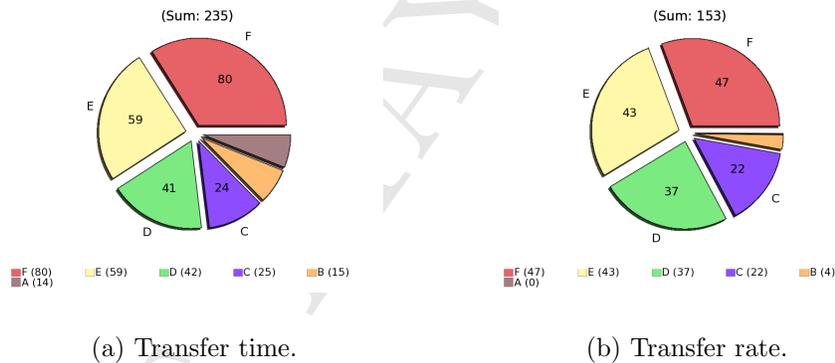


Figure 7: A qualitative comparison between transfer time and rate.

major latency include deletion of files from SRM at Fermilab (after successful copy of the previous file in the list at UCSD) and getting transfer URL(TURL) for the next file. Figure 5.2.3 estimates the average latency involved in each consecutive transfer for different file types.

### 5.2.4 Cumulative Flow of Data

Figures 5.2.4 and 11 shows the amount of data which can be transported between SRMs per hour and day respectively. Around 25GB/hr and 600 GB/day amount of data can be transported using wide area network(WAN) between SRMs at FNAL and UCSD.

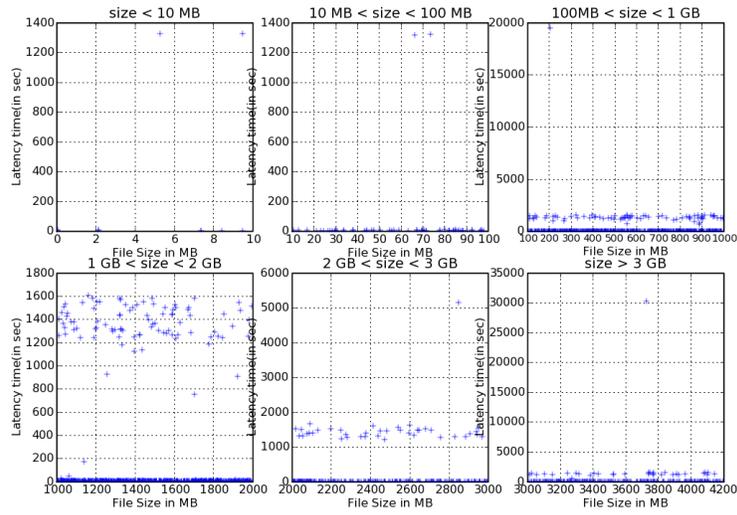


Figure 8: Latent time involved in each consecutive transfer between SRMs.

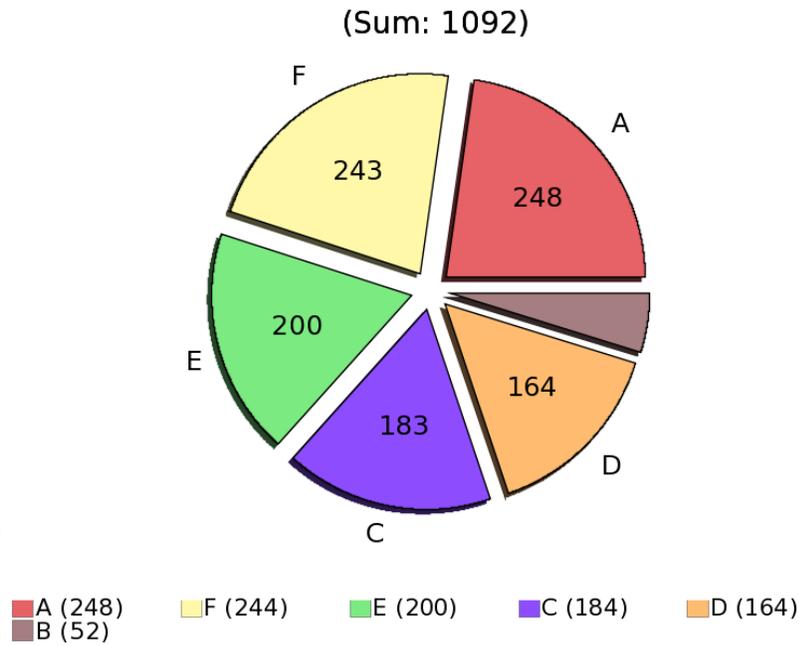


Figure 9: Latent time involved in each consecutive transfer between SRMs.

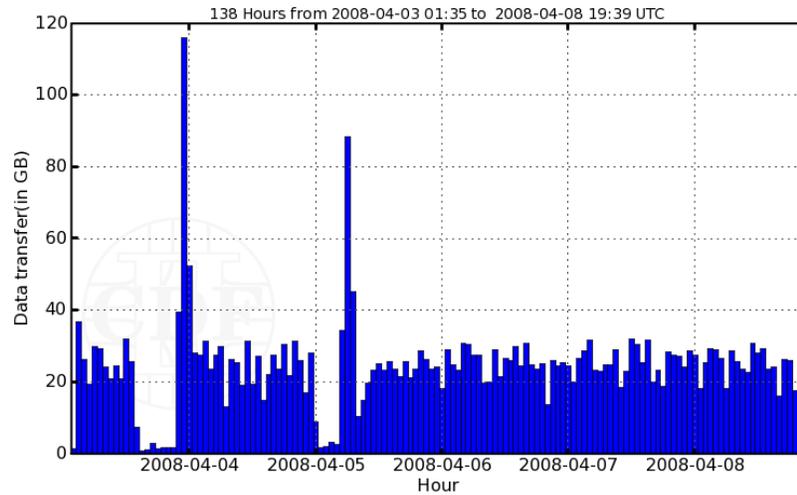


Figure 10: Amount of data which can be transported per hour between SRMs.

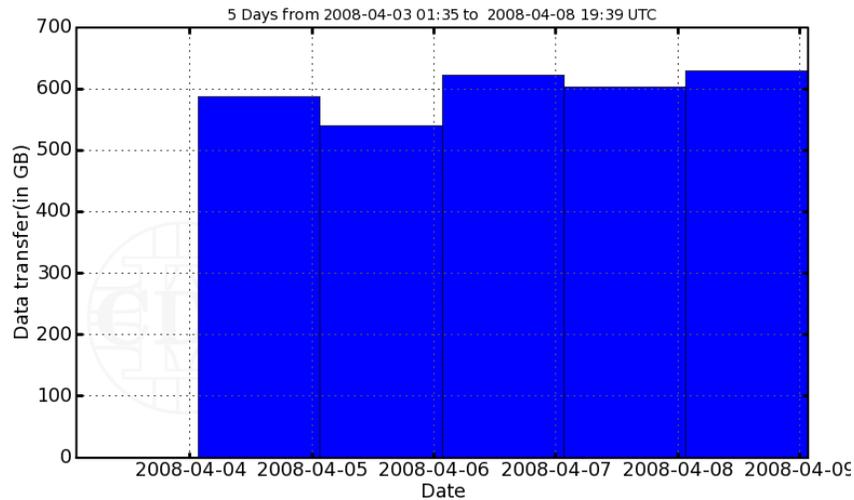
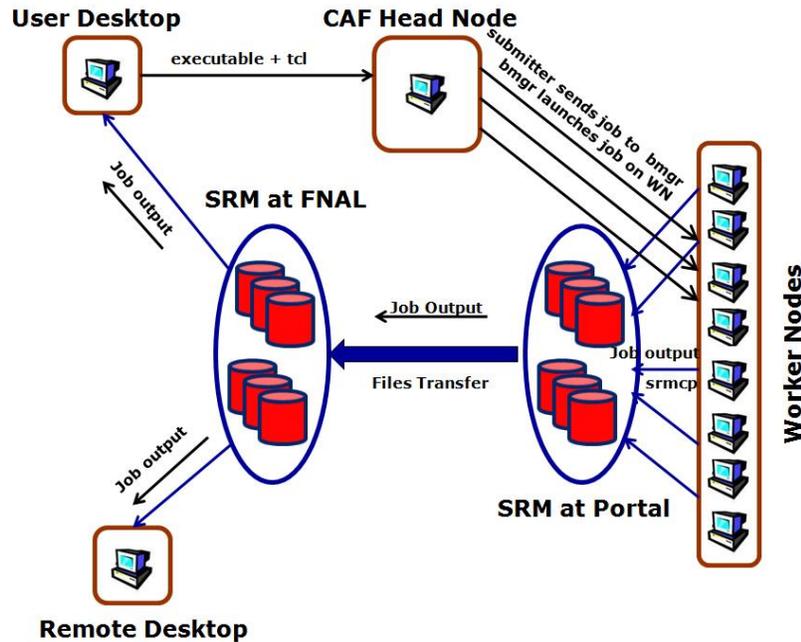


Figure 11: Amount of data which can be transported per day between SRMs.

## 6 Implementation of the Prototype Model

In this section, we describe the implementation of SAM-SRM interface in the CDF Central Analysis Framework(CAF). User interactions with the CAF are built based on client-server pairs that use Kerberos to authenticate the user. More information CAF can be obtained from [4]. Figure 12 depicts the integration of prototype model in the CAF. The basic client-server pairs are listed below.



15

Figure 12: Schematics depicting the various CAF software components, and where they run. Solid/dashed/dotted lines depict connections for submission/output/monitoring. The arrow points way from the object that initiates the communication or data transfer. The available protocols for output data transfer are also indicated.

- “CafGui or CafSubmit” ↔ “submitter” for job submission from user desktop to CAF. Submitter runs on CAF headnode. This is mentioned in Section 6.2.
- “cafkill/cafcjobs/cafcdir/cafcdir/cafcdir/caftail” ↔ “monitor” for user interactions with their job on the CAF (e.g. kill job, peek at log files, directory listing) described in Section 6.3. Monitors run on CAF headnode.
- ICAF tools for retrieval of the gzipped tar archive from the destination SRM. This is discussed in detail in Section 6.4.1.

A user submits a job using the “CafGui or CafSubmit”. Once the job is submitted, the Condor [7] batch manager(bmgr) takes over. When appropriate resources are available, bmgr launches the CafExe on worker node using the Condor launcher. The CafExe receives all the information it needs to know for starting up the user application via command line arguments that are specified when the submitter submits the job to Condor. The CafExe produces a gzipped tar archive of the user working directory on the worker node and copy the output tar archive and its metadata to the user folder on SRM at portal sites. The metadata contain the output location of the file specified by the user at the time of job submission. Proxy certificate of the user on worker node will be used for copying output files from worker node to SRM at portal site using the

command:

```
srmcp -2 -retry_num = <retryNumber> -retry_timeout = <retryTimeout>
-copyjobfile = <file> -report = <tmp>
```

where,

*retryNumber*: Number of retries before client gives up.

*retryTimeout*: Number of milliseconds to sleep after a failure before the client tries to retry.

*file*: Path to the text file containing the list of sources and destination. Each line has a format: <source-url> <destination-url>

*tmp*: Path to the report file which contain the results of the execution of srmcp. The each line in the file have the following format:

```
<src url> <dst url> <return code> [<error>]
```

The following return codes are supported:

0 - success

1 - general error

2 - file exists, can not overwrite

3 - user permission error

A cron job will check the arrival of new file in each user's folder on SRM at portal site.

There are two ways in which files in SRM at portal site can be copied to destination SRM at Fermilab or the user's specified location. These are:

- SAM station to station copy:

SAM station to station copy method can be used for transferring files between the two SRMs. A SAM station attached to SRM will be needed at portal, destination site or at the user's preferred location. A process at portal site uses metadata for registering new MC files in SAM database. The same process initiates the copy process between the two SRMs using SAM station to station copy method. After successful transfer of file, the SAM station to station copy method deletes the file from the SRM at the portal site. Around 10 to 12 copy process can be initiated at a time for a better transfer rate between the SRMs. The merit of SAM station to station copy method rely on the fact that file management happens on demand. Job monitoring for user's can also be implemented in clear and transparent way using the SAM interface.

- Process using *srmcp*:

Following command in a process can be used for transferring files between the two SRMs.

```
srmcp -2 -retry_num = <retryNumber> -retry_timeout = <<retryTimeout>
-copyjobfile = <file> -report = <tmp>
```

The options are same as explained above. In order to achieve a better transfer rate between the SRMs, the value of options are: *retryNumber* = 10

*retryTimeout* = 60000

*file*: Contain a list of 12 files.

A separate process will verify the files in SRM at portal and destination site. At-

tributes like size and checksum of file can be used for file verification. If the file found to be correct at destination SRM, then its counterpart from the SRM at portal site will be erased. Otherwise, the process will again copy the file into the destination SRM. Job monitoring can be implemented on the basis of existence of file in SRMs.

## 6.1 Authentication

The security context on the CAF is based on krb5. This is implemented such that each user has a unique krb5 principle at each site. The keytab for this principal is accessible only by the CAF infrastructure, and never by the user. The infrastructure thus obtains a krb5 ticket at runtime on the compute node, before starting the user application. The compute node will use this ticket for transferring the file in SRM at PORTAL site. Since the user cannot get the ticket outside from the CAF framework, a mechanism is needed for authentication with the storage element. Following are the two ways in which it can be achieved:

- CAF headnode authenticates the user:
 

A request either from user or service will first talk to CAF headnode for authentication. After authentication, the CAF headnode will instruct the grid entities to initiate the request on behalf of user or service. For example, following steps should be needed for a user who wants to copy a file from SRM at Fermilab to his local desktop. These are:

  1. Using the Fermilab kerberos principal of the user, the copy process contacts the CAF headnode.
  2. CAF headnode generates the user proxy certificate from the keytab file on headnode.
  3. CAF headnode contacts the SRM and initiates the request for copy from SRM to user's desktop. In order to do that, a gridftp server should be running on user's desktop.
- Add the Fermilab kerberos principal of each user in the CDF VOMS:
 

Another simple way is to just add the user's Fermilab kerberos principal in the CDF VOMS. It will be an extra line in the grid map file of CDF VOMS for each user. In this case, following steps will be needed for copying file from SRM to user's desktop.

  1. Using the Fermilab kerberos principal of the user, the copy process creates the ticket using the command *kx509*.
  2. Using this ticket, copy process authenticates the user with the SRM and initiates the copy request from SRM to user's desktop. One doesn't need the gridftp server at the user's end in this case.

The second step seems to be simple in first instance. Since one would haven't to worry about the problem related to the gridftp servers at the user's end.

## 6.2 User Interface for Job Submission

A user can submit his job by means of a Graphical User Interface(GUI) which is run from the user's desktop. An alternate command line interface presented in Section 6.2.2.

### 6.2.1 CAF Graphical User Interface(CafGui)

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

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

The CAF GUI contain various fields to be completed by the user. The meaning of these fields are described below:

- *Analysis Farm:* The analysis farm on which the user want to submit his job. All the farms that appear by default in the GUI are available to all users, but they vary in strength. A descriptions of the various farms is available at

<http://cdfcaf.fnal.gov/>

- *Data Access:* This defines the dataset to be accessed and the method used to access it. The data access method is "GenMC" which represents the Monte Carlo generation.
- *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 the command user want to run. Usually it's a shell script in the local directory. Note that it must be executable. The two boxes to the right specify the segment iterator range(inclusive), used for job parallelization as mentioned in [4].
- *Original Directory:* This is the directory that will be tarred up to send along with the command. This allows user to send configuration files, and scripts.
- *Output File Location:* This is the location where it will send the output tarball. Its format will be similar to scp format that includes the Unix ID, a string which corresponds to the URL of destination SRM, and the path of the folder relative to the user's home area on destination SRM. For example, the output location for a user *jhondoe* who wants to copy his output tarball in the folder "*~/MCGen*" of the SRM at Fermilab will be:

```
jhondoe@FNAL_SRM: ~/McGen/out$.tgz
```

The CAF framework will store the information of destination SRM URL and its corresponding string in a python dictionary object. The string represents the 'key' and the SRM URL its value. The string "FNAL\_SRM" can also be obtained from the command "icaf\_node".

- *Email Address*: When the job finishes, it will send an email to the address specified here.

### 6.2.2 CAF Command-line User Interface(CafSubmit)

CafSubmit is the command line version of CafGui. It is easier if user wish to submit several different jobs in a row. In order to run CafSubmit, user have to do things slightly differently from CafGui. The CafGui tars up the directory specified by the user and sends it to the headnode automatically. For CafSubmit, user have to create a tarball first. Inside it, the tarball must contain the script user want to run, and any local files that the script requires. This tarball will be identified with the `-tarFile` option. Following options are recognised by CafSubmit:

- *tarFile*: This is the location of the tarred up executable command and associated files.
- *outLocation*: This is where the output of the job will be sent. It will be similar to the *Output File Location* as mentioned in Section 6.2.1.
- *procType*: This is the process type for user job and it will run as (test, short, medium, or long being the usual options).
- *start*: This is the number that will denote the first section.
- *end*: This is the number that denotes the last section.
- *email*: Address where email will be sent upon job completion.
- *dhaccess*: This defines the dataset to be accessed and the method used to access it. The data access method is "GenMC" which represents the Monte Carlo generation.
- *farm*: CAF on which the user wish to send his/her job. It should be specified in the user's .cafrc file.
- *group*: Group queue on which user wish to submit. The user should be member of the concerned group.
- *command*: User's executable command. Usually a shell script.

## 6.3 Monitoring and Control of CAF Jobs

The user can monitor the progress of his/her job using the tool CafMon. It can be accessed using the development release of the cdfsoft2 distribution. The normal usage is:

```
CafMon [--farm <farm>] <command> <command-specific-params>
```

where,

- *farm*: CAF on which the user submitted the job.
- *command*: a function that user want CafMon to perform. A list of those functions is provided below.
- *command-specific-params*: option for different commands.

Following functions will be supported for monitoring and control of user's jobs.

- *jobs*: This gives user a list of the jobs and sections that are currently running. If the transfer of [JID] [section] output file is going on or in queue, then it should print the list of jobs and sections till the job [JID] [section] output transfer process ends from SRM at portal site to destination SRM at Fermilab.
- *top [JID] [section]*: It is equivalent to running the top command on the workernode running the section by the user. If the transfer of [JID] [section] output file is going on or in queue, then the CPU utilization from transfer process should be added in it.
- *ps [JID] [section]*: It is equivalent to running the ps command on the workernode running the section by the user. If the transfer of [JID] [section] output file is going on or in queue, then a snapshot of the current process should be added in it.
- *dir [JID] [section]*: Shows the working directory contents of the user's specified section. If the transfer of [JID] [section] output file is going on or in queue, then it should list the directory contents of the [JID] [section] on the SRM\_PORTAL.
- *tail [JID] [section]<JID> [file]*: Executes the tail command on <file>. If the transfer of [JID] [section] output file is going on or in queue,, then it should prints the message "[JID] [section]<JID> [file]: File is on SRM\_PORTAL".
- *head [JID] [section] [file]*: Executes the head command on <file>. If the transfer of [JID] [section] output file is going on or in queue, then it should prints the message "[JID] [section]<JID> [file]: File is on SRM\_PORTAL".
- *cat [JID] [section] [file]*: Writes the contents of <file> to the standard output. If the transfer of [JID] [section] output file is going on or in queue, then it should prints the message "[JID] [section]<JID> [file]: File is on SRM\_PORTAL".

- *node [JID] [section]*: Returns the node the section is running on, as well as the VM it is assigned to. If the transfer of [JID] [section] output file is going on, then it should return the full path of the [JID] [section] output file on the SRM\_PORTAL and of destination SRM. If the [JID] [section] output corresponding to JID is waiting for transfer to destination SRM, then it should return the full path of the [JID] [section] output file on the SRM\_PORTAL.
- *log [JID] [section]*: Writes job.log to standard output. If the transfer of [JID] [section] output file is going on, then it should print the message “[JID] [section]<JID> [file]: File is on SRM\_PORTAL”.
- *chprio [JID] [section]*: If the transfer of [JID] [section] output file is in queue, then it should affect the priority for transferring files from SRM\_PORTAL to destination SRM.
- *chgroup [JID] [section]*: If the transfer of [JID] [section] output file is going on or in queue, then it depends on the configuration of the SRM\_PORTAL. If it allows changing the group of [JID] [section] output file on SRM\_PORTAL, then change it. Otherwise, print the message ”[JID] [section] file is on SRM\_PORTAL and chgroup operation not supported.”
- *kill [JID] [section]*: Self explanatory. Kills that section. If just JID is entered, kills the entire job. If the transfer of [JID] [section] output file is going on or in queue, then it should kill the transfer process and remove all the files corresponding to [JID] [section] from the SRM\_PORTAL.
- *hold [JID] [section]*: Causes job to go into hold mode. Stays there until released. If the transfer of [JID] [section] output file is going on or in queue, then hold the operation.
- *release [JID] [section]*: Releases held job.
- *debug [JID] [section]*: If the transfer of [JID] [section] output file is going on or in queue, then print its file size and full path in SRM\_PORTAL. If the transfer of [JID] [section] output file is going on, then also print the full path of [JID] [section] output file in the destination SRM.
- *slots*: Checks for free and used slots on a given CAF.

## 6.4 Job Output

This section discusses the handling of user job output. For each job segment, the CAF system software will tar up and gzip user’s working directory after the shell script exists and transfer it to the designated output location in destination SRM at Fermilab. Each CAF user will be assigned scratch space on SRM at Fermilab and in future at local SRM to the CAF. Users nominally will be constrained to some quota on SRM.

### 6.4.1 Accessing the CAF Scratch Space on SRM

The easiest way to interact with the scratch space on SRM will be by means of the ICAF tools. The user already have experience of using the ICAF tools. Hence, existing ICAF tools will be modified in such a way the transition to SRM will be smooth to user. The modified ICAF tools will print the existing as well as the relevant information needed after adopting the proposed model in the CAF. The modified ICAF tools will be a set of command line tools that hide most of the complexity of the grid authentication, proper protocol and environment for interacting with the SRM.

### 6.4.2 Identifying User's Scratch Space on SRM

The first information every user needs will be the list of Scratch space on SRM he/she will be allowed to access. To obtain this information, the existing command

```
>icaf_info
```

should be modified in such a manner it prints out the Scratch space and output directories of the user on destination SRM at Fermilab. The command

```
>icaf_node
```

will return the name of unique string which corresponds to the destination SRM URL. For simplicity, it should also return the full path of the SRM URL. If some group has their own area on SRM, the unique string may be get by the command

```
>icaf_groups
```

which returns the list of codenames of the group areas.

### 6.4.3 Accessing User's Data

Most of the time users just want to access their data and in the most simple way too. The following ICAF tools should be modified such that it should produce the desire output.

- To have the listing of files in the job output area on SRM, user will use:

```
>icaf_ls
```

or use

```
>icaf_ls -f short
```

for presenting the list of files in condensed format. Like with ordinary `ls`, the selection should be restricted to a subset of files, like in the following example:

```
>icaf_ls '*.tgz'
```

- The most used action is getting data from the Scratch space on SRM. The command to use is:

```
> icaf_get <filenames>
```

The command `icaf_get` will fist check whether the file resides on the icaf disk-server or on the scratch space in SRM at Fermilab. Accordingly, it will copy the specified file(s) to the local directory. One can also use wildcards, just remember to quote the expression. If the user want to copy the files from his/her scratch

space on SRM to local folder, use the following syntax:

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

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

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

The command *icaf\_get* should prints out a character for every fixed size of data transferred; this is usefull on slow connections, but can be annoying when user transfer large files. To disable it, the user should -h option; pass 0 to completely disable it or a large number to get the progress char only every x Kbytes. The command *icaf\_get* should also support all the options of the protocol *srmcp*.

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

- The command:

```
> icaf_rm <filenames>
```

can be used to delete one or more files on SRM. Alternatively, if *icaf\_get* will be invoked with the -d flag:

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

the file(s) will be removed from the Scratch space on SRM (and only if) the file has been copied.

- Command for renaming file on SRM is:

```
> icaf_mv <filename> <newname>
```

where,

filename and newname are the path of the old and new file on SRM.

- Empty folder on SRM will be deleted using:

```
> icaf_rmdir <foldername>
```

If the folder is not empty, the proper warning will be issued.

- For creating a folder on SRM, following command will be used:

```
> icaf_mkdir <foldername>
```

- Although the scratch area should be used just for storing CAF outputs, sometimes it will be usefull also to copy some files from user's local machine or from SRM to the Scratch area on SRM. The command to use will be:

```
> icaf_put <filenames>
```

It will copy the specified file(s) from the local directory to the Scratch area on SRM. There will be several parameters to configure it:

```
> icaf_put [-d] [-h <nr.kbytes>] filename+ [remotename|remotedir]
```

The above command will also be used for copying files between two SRMs. The user would have to specify the URL of source and destination SRMs. The SRM URLs consist of unique string and path relative to user home area or the URL of SRMs. This command should support all the options of *srmcp* which user can provide it from the command line.

- The space reservation command on SRM will be for administrative purpose only and it may be obtained using the following syntax:
 

```
> icaf_reservespace -desired_size <sizeinGB> -guaranteed_size
<sizeinGB> -retention_policy <type> -access_latency <type> -lifetime
<duration> -space_desc <folderName> SRM_URL
```

#### 6.4.4 Accessing the other Scratch Areas

The commands as presented above in Section 6.4.3 seems to be able to access only the data in the job output area. All of the above command should 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.

#### 6.4.5 Checking User's Quota

Following command will be supported for checking quota on Scratch space on SRM:

```
> icaf_quota
```

If needed,

```
> icaf_quota <group>
```

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

#### 6.4.6 Graphical User Interface

A GUI similar to icaf\_gftp for managing files on SRM will be greatly appreciated by the users.

## 7 Future Plan

There are also some other spin off from this study. One can use this model for transporting real data from productions site to the WN at remote site for further processing of real data.

## References

- [1] CDF Homepage, <http://www-cdf.fnal.gov>
- [2] SRM Working Group Homepage, <http://sdm.lbl.gov/srm-wg/>
- [3] SAM Homepage, <http://d0ora1.fnal.gov/sam/>
- [4] CAF Homepage, <http://cdfcaf.fnal.gov>
- [5] dCache Homepage, <http://www.dcache.org/>

- [6] srmcp Homepage, <https://srm.fnal.gov/twiki/bin/view/SrmProject/SrmcpClient>
- [7] Condor Homepage, <http://www.cs.wisc.edu/condor/>

DRAFT MAY 16, 2008