5-2009

PetarXiv: an Archive for Scientific Simulations in PetaShare

Tyler Barker

PetarXiv: an Archive for Scientific Simulations in PetaShare

by

Tyler Barker

Undergraduate honors thesis under the direction of

Dr. Gabrielle Allen

Department of Computer Science

Submitted to the LSU Honors College in partial fulfillment of
the Upper Division Honors Program.

May 2009

Louisiana State University
& Agricultural and Mechanical College
Baton Rouge, Louisiana

Acknowledgements

As with most things in life, this thesis would not have been possible without the help and guidance of many others. I would like to thank Dr. Gabrielle Allen for serving as my thesis adviser, Dr. Erik Schnetter for working with me throughout the project, and Dr. Tevfik Kosar for sitting on my defense committee along with Dr. Allen and Dr. Schnetter. Furthermore, I would like to thank the LSU Center for Computation & Technology for providing me an environment to perform this research. I would also like to thank the groups of PetaShare, Stork, and Condor and the support group of the LONI systems. Lastly, I would like to thank my parents, Gary and Carol Barker, for supporting me in every aspect of my life.

Table of Contents

Abstract

As the computational ability of computers increases, more and more branches of science are using computers to run simulations in their research. Since computers can perform more computations in less time, and more researchers are using computers, the amount of data created has grown and will continue to grow. It is important that work is done to more effectively handle this data.

PetarXiv allows scientists to archive data from simulations, making use of the extensive storage space in PetaShare. The data is archived onto the PetaShare sites in a way that allows for simple future access. Along with the output data, metadata is stored, which contains more information to uniquely identify the output. To retrieve past output data, the metadata attributes can be used as search queries to locate the data. These files can then be copied out of PetaShare and onto the user's computer for further analysis.

Currently, PetarXiv focuses on a small group of scientists studying numerical relativity and running simulations in the Cactus environment. Although the project currently only involves a small number of scientists, the same idea can be applied on a much larger scale in the future.

1. Introduction

The computational capabilities of the world's faster supercomputers continue to increase with time. The petaflop barrier was broken in 2008 by both the IBM Roadrunner of Los Alamos National Laboratory and the Cray XT5 Jaguar at Oak Ridge National Laboratory, and this advancement has not stopped [1]. Researchers have already begun anticipating the required design of an exascale machine. This has many important consequences for scientists in many areas, including climatology and astrophysics, that rely on computational simulations. Since more computations can be done in a smaller time period, simulations can be given a higher resolution to more accurately represent the world that is simulated. However, with an increased number of computations comes a larger size of output data. Data totaling over a petabyte, and eventually over an exabyte, are much more difficult to handle than smaller data sizes, and it is important that advances are made in data management to keep up with the increasing computational ability of supercomputers [2].

In most cases, data from scientific simulations need to be used by a group of researchers who may not all be in the same location. Therefore, it is not sufficient to just store the data; it is important that the data can be moved across a wide network. Managing a large amount of data should be treated as an application in itself, not as a subset of computational applications. However, data management needs to be treated differently from computational applications since they each have different needs [3]. Research has been done towards this end, including the Stork and PetaShare projects.

The goal of this project is to build on the current research to create an archive for scientific simulations. As mentioned before, computational simulations play a major role in the

research of many scientists. These simulations can take a long time to run and can produce a large amount of output. As more and more simulations are run, the vast amount of data produced can become unmanageable for a scientist. If storage space is limited, then output from past simulations may need to be deleted in order to run new simulations. Even if old output data remains, the structure of the data can become too unorganized to remain useful. The aim of this project is to allow scientists to store their output data in such a manner that it will be easy to retrieve the data in the future. This way, an archive of data can be kept and shared among groups of scientists. Therefore, scientists can spend less time worrying about data placement and can spend more time conducting their research.

2. Core Technologies

This work builds on several well established technologies. A brief overview of each is presented, with an emphasis on the aspects relevant to the project.

2.1 Condor

The Condor project began in the 1980s at the University of Wisconsin-Madison. Condor is a distributed batch system with a focus on high-throughput computing. Similar to other batch systems, Condor provides a job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management [4]. When a user submits a job, Condor, based on a certain policy, chooses where and when to run the job. The progress of the job can then be monitored, and the user is notified upon the job's completion. Condor achieves high-throughput computing by allowing an organization to use all of its computation power as a single resource.

```
universe = vanilla
executable = paddmeta
getenv = True
log = paddmeta.log
output = paddmeta.out
error = paddmeta.error
arguments = testsimulation
queue
```

Figure 2.1: An example Condor submit file.

If a workstation is not being used, Condor can automatically decide to use the workstation's CPU power to run a job that is in its queue.

In a distributed system, there must be a way to allocate the system's resources to run all of its users' jobs. Condor matches a user's job to a resource through classified advertisements (ClassAds). A ClassAd contains information about an object as well as requirements and preferences. For example, a ClassAd for a job can contain information such as the user's name and how much disk space is needed for the job. It can contain requirements such as the architecture and operating system that it needs to run on, and it can be set to prefer systems with a large amount of memory. A ClassAd for a machine should contain attributes like the architecture and operating system, and it can also specify requirements and preferences for what jobs can be run on the machine. A matchmaker searches the ClassAds of queued jobs and available machines in order to pair up a job and machine with compatible requirements. If there are more than one possible pairs, the preferences are used to make the match.

Besides job scheduling, Condor provides an extensive list of features. Priorities can be assigned to jobs or users to further control how jobs are scheduled. With certain jobs, Condor can create a checkpoint of the job; the job can then be run from that point on at a later time and

```
DATA INPUT      submit_file.stork
JOB PROCESS     process.condor
PARENT INPUT CHILD PROCESS
```

Figure 2.2: An example DAGMan submit file.

on a different machine. Condor can be configured to take periodic checkpoints of job. This way, if a job fails, maybe because of a system crash, the job can be rerun from the last checkpoint instead of from the beginning. Even though jobs in Condor are run on remote machines, Condor can preserve the local environment by remote system calls. Therefore, the job runs as if it were being run on the local machine [5].

Condor also supports running multiple jobs that have dependencies on each other. This is accomplished by the Directed Acyclic Graph Manager (DAGMan). In the directed acyclic graph (DAG), each node represents a program and the arrows represent dependencies. If a program is the child of another program, it usually requires the output of that program as its own input. Therefore, the program should not be run until its parent is done. In Condor, a DAG can be submitted as a single job. Each job in the DAG is then scheduled with the considerations of the DAG's dependencies.

2.2 Stork

Stork is a scheduler similar to Condor, but instead of dealing with computational jobs, Stork handles data placement [6]. Like Condor, Stork can interact with DAGMan; users can schedule computational and data placement jobs in the same DAG. Stork modularly designed so

```
[
     dap_type = transfer;
     src_url = "file:///home/tbarke5/perl/testsimulation/";
     dest_url =
"petashare://lsu/tempZone/home/numrel/simulations
     /simulation-qc0-output-i0000-qb4.loni.org-eschnett-
2008.11.23
     -10.42.46-15934";
     output = "/home/tbarke5/pcommands-1.0.1/bin/stork.out";
     err = "/home/tbarke5/pcommands-1.0.1/bin/stork.err";
     log = "/home/tbarke5/pcommands-1.0.1/bin/stork.log";
     recursive_copy = true;
]
```

Figure 2.3: An example Stork submit file.

that it can easily be extended to be used with any data transport protocol, storage system, or middleware. Currently, Stork can interact with data transport protocols such as FTP, GridFTP, and HTTP, storage systems like SRB, UniTree, and NeST, and data management middleware such as SRM [7]. Most importantly for this project, Stork is already set up to handle data movement into and out of PetaShare.

Stork is very useful in moving large files because of its ability to queue, manage, query, and restart data placement jobs. It can handle many errors that can occur during data movement without any human intervention. Like Condor, Stork uses ClassAds to match a data placement job to a certain machine. Before a data transfer, Stork determines which data transfer protocols are available between the source and destination hosts. If the user does not specify a protocol, Stork will decide which protocol to use. If a user does specify one, Stork will attempt to use this protocol first, but if this fails, it will automatically choose another protocol until it finds one that works.

A Stork user can specify the number of maximum retries and the maximum allowable run time. If a data placement job fails, Stork will retry the job without any prompt from the user as

long as it has not already retried the job the maximum number of times. Also, if a single job execution lasts for the maximum allowable run time, Stork will the kill the job and retry it. Therefore, a bug that causes a job execution to run forever without giving an error can be dealt with. All of these features of Stork work to minimize the effort that the user has to put into dealing with data transfer errors.

2.3 LONI

The Louisiana Optical Network Initiative (LONI) is a fiber-optics network that connects Louisiana's major research universities, including Louisiana State University (LSU), Louisiana Tech University, LSU Health Sciences Center in New Orleans, LSU Health Sciences Center in Shreveport, Southern University, Tulane University, University of Louisiana at Lafayette and University of New Orleans [8]. The network allows scientists across the state to collaborate, utilizing over 85 teraflops of computational capacity. LONI connects and provides support for many of the state's supercomputers, with Queen Bee at the center.

Queen Bee is located in the Information Systems Building in Baton Rouge. It began running in 2007 and was considered the 23rd fastest supercomputer in the world in the June 2007 Top500 listing [9]. More recently, it was listed as the 76th fastest supercomputer in the November 2008 Top500 listing [10]. Queen Bee sports a 50.7 TFlops eak performance, 668 compute node cluster running Red Hat Enterprise Linux version 4 operating system, with each node containing a dual Quad Core Xeon 64-bit processor operating at a core frequency of 2.33 GHz. Its name comes from the nickname of former Louisiana governor Kathleen Babineaux Blanco, honoring her commitment to LONI during her administration.

2.4 PetaShare

PetaShare is an NSF-funded project that aims to ease the management of large-scale data produced by scientists. PetaShare can potentially allow scientists across the world to collaborate on a project a share their data seamlessly. Technologies currently being developed in the project include data-aware storage systems, data-aware schedulers, and cross-domain metadata scheme which take the responsibility of managing data resources and scheduling data tasks from the user and perform these tasks transparently [11]. A prototype of PetaShare is currently being deployed across many Louisiana universities: Louisiana State University, Tulane University, Louisiana Tech, the University of New Orleans, and the University of Louisiana at Lafayette. The prototype will manage 250 terabytes of disk storage and 400 terabytes of tape storage, using the 40 gigabit per second Louisiana Optical Network Initiative (LONI) infrastructure to make its interconnections.

There are three client tools that can be used to access and manage data stored in PetaShare sites. petaFs allows a user to access the data with standard UNIX commands. petashell allows users to attach programs on local machines to data in PetaShare sites. The programs can view the PetaShare resources as a file system. Finally, pcommands is a set of specialized commands for interacting with PetaShare resources. They are mostly similar to UNIX commands and are named the same with the letter ‘p’ in front (pls instead of ls). These commands are based on the icommands of the Integrated Rule-Oriented Data System (iRODS) [12]. Important for this project is the pmeta command, based on the imeta command of iRODS. This command enables the addition and removal of metadata and can be used to search for data using the associated metadata.

3. System Design

In creating an effective archive for simulation data, many issues must be considered. The most obvious issue is finding a place to store the archive. Depending on the size of the user group, finding enough space to sustain an archive over a lengthy period of time can pose a problem. Data from a single scientific simulation can come in a variety of sizes and can reach over one hundred gigabytes. Therefore, a simulation archive will likely need many terabytes of storage space, and quite possibly, over a petabyte of space could be needed. Because of the large file sizes, it would also be helpful to have a fast connection to the storage space to minimize file transfer times. The longer it takes to transfer a file, the higher the chance that an error can occur during the transfer. Finally, the location of the storage should be accessible by anyone who will use it, and preferably, it should be accessible from the users' local machines.

As mentioned before, moving such large files can become problematic. A file transfer can go on for hours only to fail towards the end. The user may not know that the transfer failed until trying to access the file. Also, even if an error does not appear, it may not be clear that the file was transferred completely and correctly. Therefore, software specialized in data placement can become very helpful in the creation of an archive. Such a program could automatically restart a file transfer if it fails and could check file sizes and checksums after a transfer to verify that the correct information was transferred. This would require less human surveillance during the movement of files and could save a lot of time.

Once there is a place and method to store the data, a file structure needs to be created that allows a user to search and retrieve data from the archive. First, the directory containing the individual simulation data needs to be named. If the user is allowed to specify this name, safeguards need to be taken to ensure that duplicating a filename does not overwrite data, unless

1. Find a place to store the data that is large enough, reliable, and easily accessible.
2. Find a method to move large data securely, ensuring that the correct data is moved and handling errors with relatively little user intervention.
3. The archive must have a file structure, and it is important to safeguard against duplicate filenames.
4. Implement a method to search through the archive for a specific set of data.
5. Enable users to archive and retrieve data as simply as possible.

Figure 3.1: A list of design requirements that any simulation archive needs.

this is what the user intends. Alternatively, a unique name can be given to each simulation added to the archive to alleviate this problem.

Making the archive simple to search through can prove to be one of the most challenging aspects of creating such an archive. One method to do this is to organize the file structure in a meaningful way that simplifies searching. For example, all simulations from a single user or a single project could be placed in a single directory. Therefore, a user could easily narrow down the search for a simulation by user and project name. However, this method limits the number of ways to search by the complexity of the directory structure. Trying to add directories to represent more than just user and project names could severely obfuscate the file structure and make it very difficult to use.

Another method to allow for easy searching is to associate a set of metadata with each simulation in the archive. This is very similar to tags that identify videos on YouTube, but instead of words or phrases, attribute-value pairs are used to identify the simulations. If the user

name is stored with the simulation, then a user could easily obtain a list of all simulations with a certain user name. This method is not as limited as the aforementioned method. The ways of searching are limited only by the amount of metadata that is associated with a simulation. However, this method is also more complicated to implement. The metadata has to be generated somehow, and the storage space has to provide a way to use this metadata to search its files.

4. Case Study

PetarXiv is a prototypical simulation archive that addresses many of the issues enumerated in the previous section. It is limited in scope, restricted to a group of scientists researching numerical relativity and running simulations in the Cactus framework.

4.1 Cactus

Cactus is a framework for parallel high performance computing, used by scientists from many fields, including numerical relativity and computational fluid dynamics [13]. Cactus gets its name from its design. The central core is known as the flesh, and modules connecting to the flesh are known as thorns. The flesh does not do much outside of parsing parameters, managing memory, and activating the required thorns. The rest of the work is done by the individual thorns. This design makes the Cactus framework very modular. Thorns are mostly independent of one another, and a user can always extend Cactus by developing his or her own thorns [14].

The most important of such thorns for this project is named Formaline. When a simulation is run with the Cactus framework, and the Formaline thorn is active, Formaline collects build and run time information about the simulation. Example information includes the run user of the simulations and parameter values that were used. It stores this information and

1. Using the expansive storage space of PetaShare to store the simulation archive.
2. Stork is specialized in data placement and secures data movement by enabling automatic retries in the case of an error and uses checksums to ensure the correct data is moved.
3. All simulations of the scientific group are placed in a single directory, and duplicate filenames are avoided by using the unique simulation ids given by Cactus.
4. The metadata structure built into PetaShare is used. The metadata comes from the simulation information provided by the Formaline thorn of Cactus.
5. Only one command, parchive or pretrieve, is needed to archive data or retrieve data from the archive.

Figure 4.1: A list describing how PetarXiv met the design requirements in the previous list.

transmits it in various ways. One method of transmission is to output formaline-jar.txt files, which contain a list of attribute-value pairs. An example formaline-jar.txt file can be found in the appendix of this report.

4.2 PetarXiv

PetarXiv is an integration of the aforementioned core technologies with the specific goal of archiving data from scientific simulations. In PetarXiv, all simulation data are placed into a single directory on PetaShare. Since it was designed for a small group of scientists working in the same scientific area, having one directory creates a community of simulation data that can be accessed by anyone in the group. This way, scientists who are geographically distant from each other could share their data as easily as scientists working next to each other.

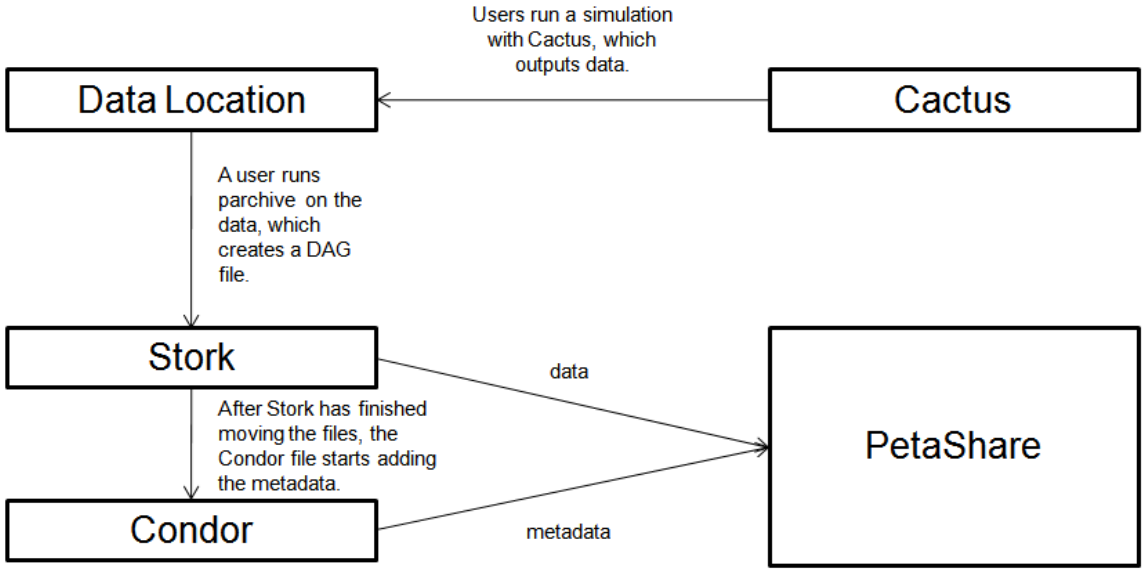

Figure 4.2: A diagram representing the flow of execution in the archival of data.

Using PetarXiv only requires the use of two Perl scripts, called parchive and pretrieve, that move data into and out of the archive. When parchive is used with data from a Cactus simulation, a Stork submit file is automatically generated and submitted to move the data onto PetaShare. Cactus gives a unique simulation id to any simulation run within its framework. Therefore, PetarXiv uses this id to name the directory containing all of the data on PetaShare. This prevents the possibility of having duplicate filenames, so data will not be overwritten. After the data is moved onto PetaShare, a Condor job is run to associate metadata with it, using data provided by the Formaline thorn of Cactus. The parchive script searches the data for a formaline-jar.txt file. This file is then parsed into attribute-value pairs, and these pairs are sent to PetaShare as metadata for the simulation.

Command		Description
parchive	Created for PetarXiv	Copies simulation data onto the archive and associates metadata with the simulation.
pmeta	Distributed with PetaShare	Used to search through the archive using the attribute-value pairs of the metadata.
pretrieve	Created for PetarXiv	Copies simulation data from the archive onto the user's machine.

Figure 4.3: A table giving brief descriptions of the three commands needed to use PetarXiv.

To search for a specific simulation, the pmeta command is used. Users can find simulations by specifying the desired attribute-value pairs such as the user name and the date that simulation was run. Once a simulation is found, the user can use the pretrieve command to get the data from PetaShare. When pretrieve is called, another stork submit file is generated to move the simulation data from PetaShare into the user's current directory.

5. Results

This project was conducted over the course of a single school year. The first semester consisted of the planning stages and gaining access to the computers and software required for the project. Then, during the second semester, PetarXiv was developed and tested. The found results are presented in this section.

5.1 Functionality

PetarXiv is designed to be very simple to use, requiring only one command to archive and retrieve data. An example session using PetarXiv is as follows:

Assuming the user already has data from a Cactus simulation in a directory named output, only the parchive command is needed to archive the data.

```
$ parchive output
================
Sending request:
    [
        dest_url =
"petashare://lsu/tempZone/home/numrel/simulations/tbarker-simulation-
qc0-output-i0000-qb4.loni.org-2008.11.23";
        src_url = "file:///work/tbarke5/simulations/output;
        err = "/home/tbarke5/pcommands-1.0.1/bin/stork.err";
        output = "/home/tbarke5/pcommands-1.0.1/bin/stork.out";
        dap_type = transfer;
        recursive_copy = true;
        log = "/home/tbarke5/pcommands-1.0.1/bin/stork.log"
    ]
================

Request assigned id: 150
Adding metadata: jobid = "run-qc0-output-qb602-tbarker-2008.11.23-
16.43.44-1834"
Adding metadata: jobtype = "default"
Adding metadata: app_title = "QC-0"
Adding metadata: start_time = "Nov 23 2008 -0600 10:43:48-0600"
Adding metadata: output_files = "qc0-output"
Adding metadata: host = "qb602"
Adding metadata: nprocs = 64
Adding metadata: local_username = "tbarker"
…
```

Once data is on the archive, the user can search for it using the metadata, such as the local user name. The pmeta command provided by PetaShare can be used to search the metadata.

```
$ pmeta qu -C local_username = tbarker
collection: /tempZone/home/numrel/simulations/tbarker-simulation-qc0-
mclachlan-i0006-qb4.loni.org-2009.03.04
----
collection: /tempZone/home/numrel/simulations/tbarker-simulation-qc0-
output-i0000-qb4.loni.org-2008.11.23
```

Finally, to retrieve the data from PetaShare, the only the pretrieve command needs to be used.

```
$ pretrieve tbarker-simulation-qc0-output-i0000-qb4.loni.org-2008.11.23
================
Sending request:
    [
        dest_url = "file:///work/tbarke5/simulations/tbarker-
simulation-qc0-output-i0000-qb4.loni.org-2008.11.23";
        src_url =
"petashare://lsu/tempZone/home/numrel/simulations/tbarker-simulation-
qc0-output-i0000-qb4.loni.org-2008.11.23";
        err = "/home/tbarke5/pcommands-1.0.1/bin/stork.err";
        output = "/home/tbarke5/pcommands-1.0.1/bin/stork.out";
        dap_type = transfer;
        recursive_copy = true;
        log = "/home/tbarke5/pcommands-1.0.1/bin/stork.log"
    ]
================

Request assigned id: 151
```

The structure of PetarXiv was only functional for about four weeks and used by three users, including two scientists in the numerical relativity group and the author of this paper. During that time period, data from nine simulations were moved onto the archive, totaling over two hundred gigabytes in disk space. Metadata was also added for these simulations, and tests were done to ensure the functionality of searches using the metadata. These simulation data were also retrieved from the archive and placed back on Queen Bee.

5.2 Problems

As mentioned in the previous section, PetarXiv was only used over a span of four weeks. This was not nearly enough time to effectively test such a system. There were not enough test users, and not enough simulations were placed on the archive. Thus, while the mechanics of the system could be tested, the utility of it could not be tested. In order to test the system's utility, it would have to be used over a lengthy period of time so that numerous simulations could be

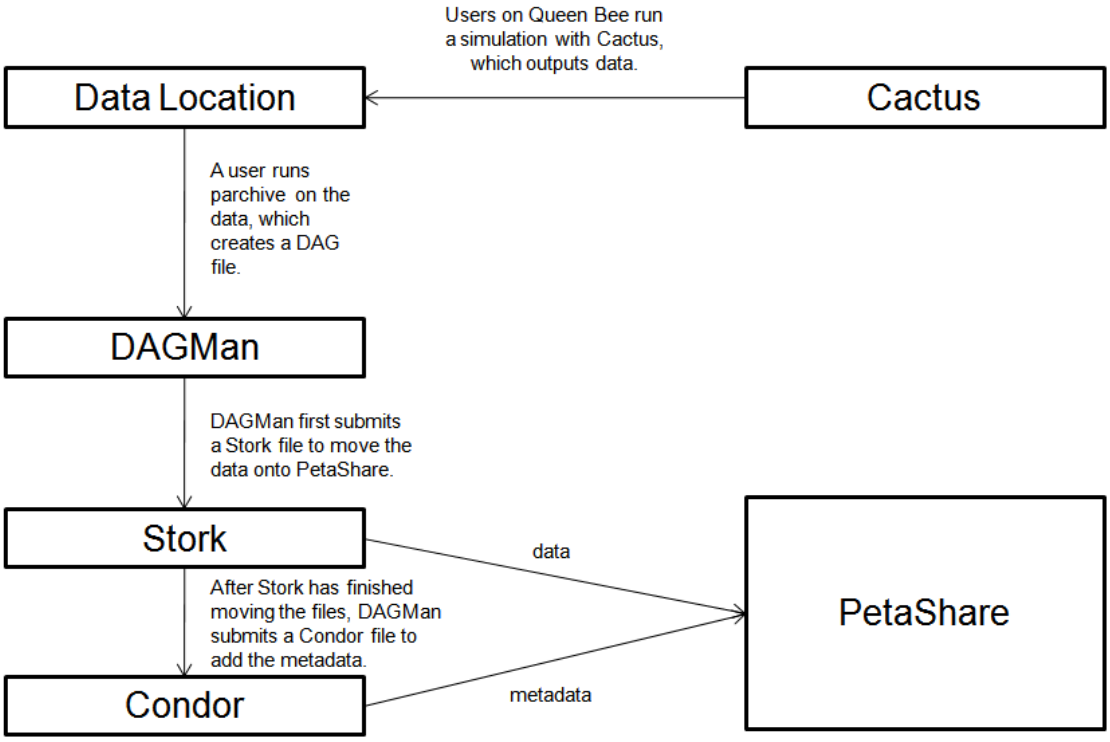

Figure 5.1: A diagram showing the desired flow of execution in the archival of data.

archived. With enough simulations on the archive, it would become necessary to use the metadata to search for a specific simulation. Since there were so few simulations on PetarXiv, searches using the metadata were only done for testing purposes, not practical purposes.

There were also some functional issues with PetarXiv that dealt directly with the computers and software used in the system. During the time that PetarXiv was used, it was common for either Queen Bee or the PetaShare sites to be down for maintenance. If either of these were down, the archive could not be used at all. This caused many delays in the development and testing of PetarXiv and could pose a problem for any future simulation

archives. If a scientists is relying on the archive to store simulation data, such problems would most likely delay his or her research.

Furthermore, the addition of a simulation's metadata was not as efficiently implemented as planned. The planned method to add metadata was to create a DAG to submit to DAGMan. The parent node would include the Stork submit file to move the data. The child node would contain a Condor submit file that added the metadata. If this worked, DAGMan would wait until the Stork job was finished to call the Condor job. However, DAGMan was never set up correctly. Therefore, another method was needed to ensure that the Condor job waited for the simulation data to be copied before associating the metadata. The implemented Condor job first checks to see if the directory of the data exists before trying to associate metadata with it. If the directory does not exist, it periodically loops until the directory is added, at which point it adds the metadata. However, this is not as efficient as using DAGMan since the Condor job can be forced to wait in a loop while the Stork job is waiting to be submitted. If DAGMan were used correctly, the Condor job would not be called until the Stork job is finished. If DAGMan were installed and configured properly, adding its functionality would be trivial.

6. Conclusion

Increases in computational ability have allowed scientists to more accurately represent the world in simulations. However, these computations would be meaningless without the output they create. Therefore, it is important that output is stored in a reliable location that can be readily accessed in a secure manner. This project serves as a starting point for this larger mission. A discussion has been presented on the generic aspects that such an archive should have, and a prototype has been implemented. Although PetarXiv was not tested properly to

assess its effectiveness, it can serve as a framework for future research. It is important to note that the methods used in PetarXiv only represent one solution. The overall structure of a simulation archive will most likely be similar to that of PetarXiv, but the individual pieces can be altered. PetarXiv focuses only on simulations run in the Cactus framework, but there should be archives for any type of simulation. Also, Stork and PetaShare do an efficient job of moving and storing data, but other applications can be substituted to perform the same tasks.

A larger group of people will be needed to develop and maintain a functional simulation archive, but it is a worthwhile task. Hopefully, future researchers will further this project to enable scientists to manage their data efficiently, lest data management becomes a hindrance in our progress towards an exascale computing world.

Works Cited

[1] Betsy Mason, “Supercomputers Break Petaflop Barrier, Transforming Science”, *Wired.com*, Nov. 18, 2008. http://blog.wired.com/wiredscience/2008/11/supercomputers.html

[2] Tevfik Kosar and Miron Livny, “A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems”, *In Journal of Parallel and Distributed Computing*, Vol.65 No.10 (2005) pp.1146-1157.

[3] T. Kosar, S. Son, G. Kola, and M. Livny, “Data Placement in Widely Distributed Environments”, *In Grid Computing: The New Frontier of High Performance Computing*, Editor: L.Grandinetti, Elsevier Press, November 2005.

[4] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny, "Condor - A Distributed Job Scheduler", in Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*, The MIT Press, 2002.

[5] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[6] http://storkproject.org/

[7] Tevfik Kosar and Miron Livny, “Stork: Making Data Placement a First Class Citizen in the Grid”, *In Proceedings of 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004), Tokyo, Japan, March 2004.*

[8] http://www.loni.org/about/

[9] http://www.loni.org/systems/system.php?system=QueenBee

[10] http://top500.org/list/2008/11/100

[11] http://petashare.org/

[12] https://www.irods.org/index.php/icommands

[13] http://www.cactuscode.org/About/

[14] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus Framework and Toolkit: Design and Applications. In *Vector and Parallel Processing - VECPAR '2002, 5th International Conference*. Springer, 2003.

Appendix A

PetarXiv is a prototype of an archive for scientific simulation data that uses the expansive storage space of PetaShare. If you will, please help me test PetarXiv by following these instructions.

Note: Currently, you can only use PetarXiv on QueenBee and with Cactus simulations that have formaline-jar.txt files associated with them.

Setting up pcommands:

1. Go into the pcommands directory
   ```
   $ cd /home/tbarke5/pcommands-1.0.1
   ```
2. Run the setup script
   ```
   $ ./setup
   ```
3. Log in as numrel user
   ```
   $ pchangeuser
   ```

   User name: numrel
   Password: [enter password]

Set Stork environment variables:

1. `$ export PATH=$PATH:/home/tbarke5/stork/bin:/home/tbarke5/stork/sbin`
2. `$ export STORK_CONFIG=/home/tbarke5/stork/etc/stork_config`

Test to make sure software is working:

1. `$ condor_q`

   ```
   -- Submitter: qb3.loni.org : <208.100.92.21:50608> : qb3.loni.org
    ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD

   0 jobs; 0 idle, 0 running, 0 held
   ```

2. `$ stork_q`

   ```
   ===============
   job queue:
   ===============
   ===============
   ```

If the stork_q command gives an error, try starting the stork server with the command stork_server.

Archiving data of a simulation:

1. Go into the directory that contains the simulation directory.
2. `$ parchive [simulation directory]`

Searching the archived data:

```
pmeta qu –C [Attribute Name] = [Attribute Value]
```

You can go to https://www.irods.org/index.php/imeta for more information about the pmeta command. The pmeta command is analogous to the iRODS imeta command.

The attribute names and values come from the formaline-jar.txt files of the simulation data. An example formaline-jar.txt file is shown on the next page.

For example, to search for simulations by a specific user, type:

```
pmeta qu –C local_username = [user name]
```

Retrieving data for the archive:

1. Go into the directory that the simulation data should be placed.
2. `$ pretrieve [simulation directory on PetaShare]`

Do not use the entire file path on PetaShare – just the name of the directory.

Example formaline-jar.txt file:

```
jobid="run-qc0-output-qb602-eschnett-2008.11.23-16.43.44-1834"
jobtype="default"
app_title="QC-0"
start_time="Nov 23 2008 -0600 10:43:48-0600"
project_name=""
output_files="qc0-output"
host="qb602"
nprocs=64
portal_username=""
local_username="eschnett"
parameter_filename="qc0-output.par"
executable="./cactus_einstein-queenbee-mvapich2-optimise"
data_directory="qc0-output"
app_visibility="public"
notification_reports=""
notification_methods=""
Cactus_version="4.0.b16"
config_id="config-einstein-queenbee-mvapich2-optimise-qb4.loni.org-work-eschnett
-Calpha"
build_id="build-einstein-queenbee-mvapich2-optimise-qb4.loni.org-eschnett-2008.1
1.14-22.30.21-774"
compile_date="Nov 14 2008"
compile_time="16:35:18"
simulation_id="simulation-qc0-output-i0000-qb4.loni.org-eschnett-2008.11.23-10.4
2.46-15934"
run_id="run-qc0-output-qb602-eschnett-2008.11.23-16.43.44-1834"
run_user="eschnett"
run_date="Nov 23 2008 -0600"
run_time="10:43:48-0600"
run_host="qb602"
run_title="QC-0"
argc=4
argv[0]="./cactus_einstein-queenbee-mvapich2-optimise"
argv[1]="-L"
argv[2]="3"
argv[3]="qc0-output.par"
parameter_filename="qc0-output.par"
current_dir="/work/eschnett/simulations/qc0-output-i0000/output-0000-active"
out_dir="qc0-output"
run_date="Nov 23 2008 -0600"
run_time="10:47:18-0600"
cctk_iteration=0
cctk_time=0
simulation="done"
```

Appendix B

parchive:

```
#!/usr/bin/perl
use strict;
use warnings;

my($simulation) = $ARGV[0];
my($a) = ` find $simulation -name "SIMULATION_ID" -print0 `;
my($b) = ` more $a `;
print $b;
chomp($b);

my($d) = ` pwd `;
chomp($d);
my($filepath) =  $d."/".$simulation;

my($size) = ` du -sk $simulation `;

open (MYFILE, ">>/home/tbarke5/pcommands-1.0.1/bin/sizes.txt");
print MYFILE $size;
close (MYFILE);

open (MYFILE, ">/home/tbarke5/pcommands-1.0.1/bin/submit_file.stork");
print MYFILE "[\n";
print MYFILE "dap_type = transfer;\n";
print MYFILE "src_url = \"file://".$filepath."\";\n";
print MYFILE "dest_url =
\"petashare://lsu/tempZone/home/numrel/simulations/".$b."\";\n";
print MYFILE "output = \"/home/tbarke5/pcommands-1.0.1/bin/stork.out\";\n";
print MYFILE "err = \"/home/tbarke5/pcommands-1.0.1/bin/stork.err\";\n";
print MYFILE "log = \"/work/tbarke5/stork.log\";\n";
print MYFILE "recursive_copy = true;\n";
print MYFILE "]";
close (MYFILE);

open (MYFILE, ">/home/tbarke5/pcommands-1.0.1/bin/process.condor");
print MYFILE "universe = vanilla\n";
print MYFILE "executable = /home/tbarke5/pcommands-1.0.1/bin/paddmeta2\n";
print MYFILE "getenv = True\n";
print MYFILE "log = /work/tbarke5/paddmeta.log\n";
print MYFILE "output = /home/tbarke5/pcommands-1.0.1/bin/paddmeta.out\n";
print MYFILE "error = /home/tbarke5/pcommands-1.0.1/bin/paddmeta.error\n";
print MYFILE "arguments = ".$filepath."\n";
print MYFILE "queue";
close (MYFILE);

system ('stork_submit /home/tbarke5/pcommands-1.0.1/bin/submit_file.stork');
system ('condor_submit /home/tbarke5/pcommands-1.0.1/bin/process.condor');
```

paddmeta (used in Condor job):

```
#!/usr/bin/perl
use strict;
use warnings;

my($simulation) = $ARGV[0];

my($a) = ` find $simulation -name "SIMULATION_ID" -print0 `;
print $a;
open FILE, $a or die "Can't open file: $!";
my($b) = " ";
while (<FILE>){
   $b = $_;
}
close FILE or die "FILE: $!";

print "\n";
print "Directory name is: \n";
print $b;
chomp($b);

my($c) = ` find $simulation -name "formaline-jar.txt" -print0 `;
my($index) = index($c, "formaline-jar.txt");
my($formaline) = substr($c,0,$index+17);

system ('pcd /tempZone/home/numrel/simulations');

my($num) = 0;
while($num==0) {
   my($test) = ` pls $b 2>&1`;

   if (substr($test, 0, 5) ne "ERROR") {
      print substr($test, 0, 5);
      $num = 1;
   }
}

open(my $in, "<", $formaline) or die "Can't open file: $!";
while (<$in>){
   if (/([^=]+)[=]([^\n]+)$/) {
      if (/[\\][=]/) {}
      elsif (substr($_,0,6) ne "thorns" && substr($_,0,11) ne "parameters/")
{
        my($out) = ` pmeta add -C /tempZone/home/numrel/simulations/$b $1 $2
2>&1 `;
        print "Adding metadata: ".$1." = ".$2."\n";
      }
   }
}

close $in or die "$in: $!";
```

pretrieve:

```
#!/usr/bin/perl
use strict;
use warnings;

my($simulation) = $ARGV[0];

my($d) = ` pwd `;
chomp($d);

open (MYFILE, ">/home/tbarke5/pcommands-1.0.1/bin/submitfile");
print MYFILE "[\n";
print MYFILE "dap_type = transfer;\n";
print MYFILE "dest_url = \"file://".$d."/".$simulation."\";\n";
print MYFILE "src_url =
   \"petashare://lsu/tempZone/home/numrel/simulations/".$simulation."\";\n";
print MYFILE "output = \"/home/tbarke5/pcommands-1.0.1/bin/stork.out\";\n";
print MYFILE "err = \"/home/tbarke5/pcommands-1.0.1/bin/stork.err\";\n";
print MYFILE "log = \"/home/tbarke5/pcommands-1.0.1/bin/stork.log\";\n";
print MYFILE "recursive_copy = true;\n";
print MYFILE "]";
close (MYFILE);

system ('stork_submit /home/tbarke5/pcommands-1.0.1/bin/submitfile');
```