Framework for the Control and Monitoring of Grid based Data-Intensive Applications

Anand Balasubramanian

ABSTRACT

Data-intensive applications involve the processing of terabytes or petabytes of data that are distributed geographically among multiple data storage systems. Processing such large amounts of data places extreme demands on the computational resources available today. A 'computational grid' provides the processing capabilities required for data-intensive applications, by harnessing the compute resources provided by collections of workstations spread across multiple local and wide area networks. The three basic functionalities required for grid computation are: 1) the ability to move data across workstations, 2) the ability to launch processes on remote workstations, and 3) a security mechanism to guard the disparate resources. Several existing grid technologies provide these basic functionalities. A control program that makes of these basic functionalities is necessary to direct and monitor the applications running on the grid. This paper presents a modular framework for such a control program to efficiently manage any kind of data-intensive application. The system described herein also takes into account the heterogeneity of resources in the grid. The framework proposes the use of standard grid protocols for the basic functionalities of the grid. We conclude with a detailed description of the individual modules in the framework.

INTRODUCTION

Scientists in various fields of science and technology have made tremendous advancements in their research over the years. Many scientific research activities conducted by the researchers across the world are producing large volumes of data. The data being produced vary in magnitude ranging from several hundred megabytes to several hundred terabytes. Engineering applications that produce such large volumes of data also requires the analysis of such data. These kinds of applications require high data storage capacity and high computational resources. Such applications are categorized as data-intensive applications. Several scientists working around the globe will make use of the data produced by these applications. The data is analyzed for specific patterns and also to generate new datasets from the existing data. Several such applications exist and we will look at a few of them.

High Energy Physics is the study of the elementary particles and then forces that govern their interactions at the most fundamental level. The Tevatron Collider, the world's premier high-energy accelerator, is located at the Fermi National Accelerator Laboratory (Fermilab) in Batavia, Illinois, USA. D0 is one of the experiments done at Fermilab. The D0 experiment produces raw data from the accelerator in the order of a few megabytes per second. This experimental raw data is stored in tertiary storage devices and is also catalogued in the SAM (Sequential Access through Metadata) data management system. High Energy Physicist distributed all over the world will use this raw data to perform analysis work on them. This analysis work in turn will produce data that has to be stored back into the tertiary storage devices and also might be cached locally on the researcher's desktop for further use. This type of application is a clear example of a data-intensive application.

Climate Modeling is another example of data-intensive application. DOE's SciDAC program's climate modeling wing aims at creating an advanced climate simulation program that will accelerate the execution of climate models. Such simulation programs will produce large amounts of data. Researcher located at universities and laboratories across the country will use the data to perform analysis work. The results produced by the analysis work will in turn need to be stored and retrieved in an efficient manner. Besides the storage and retrieval of data, the analysis and simulation tasks will be demanding computationally as well.

From the above examples, it is very clear that such data-intensive applications place heavy demands on the data storage systems as well as the computational resources available today. Such demands on computational resources can be met by large scale resource sharing. Resources mainly refer to data and compute cycles. Very large data storage systems are required to store such huge volumes of data. The data generated by the various scientific applications described earlier are stored in tertiary storage devices capable of holding tera bytes of data. Different clusters will then access these storage devices to access the data. The location of the data plays a very important role on where the program that accesses that data is going to execute. The programs accessing the data stored in such data storage systems might be scheduled to execute in a site near the data storage location where the data can be moved easily or if only a small part of the data is required then that particular data set might be moved to the location where the program resides. So the storage of such large volumes of data and their replication is an important issue in such applications. Having a centralized location for such data might cause a bottleneck in terms of accessing them. If thousands of users are trying to access the data from this centralized storage system, it will severely reduce the access time for the data. Hence data storage and access is an important criterion for data intensive applications.

The programs that access this data will need computational resources where they can execute. These programs will run for hours together and will require tremendous amounts of cpu cycles for their execution. If a program runs on a single standalone computer, it will take hours or even days to complete. So other programs that are ready to execute will have to wait till this program completes. Hence the need for network of workstations came up. CPU cycles of workstations located in geographically distributed sites can be used for executing these programs. This can greatly increase the compute cycles required for executing such programs. Hence a gird which is a collection of workstations located across geographically distributed sites is the most viable solution for applications with such requirements. But since the workstations are distributed geographically, security concerns arise. Data has to be securely moved between sites. Another issue that arises with such kinds of resource sharing is the privileges that should be granted to the executing programs. A scientist sitting in one corner of the world might be using the resource in another corner. So trust has to be established before he is allowed to use the remote resource and also his access rights are fixed before hand. Data intensive applications will make use of geographically distributed compute resources as well data. Hence standard mechanisms for data transport across resources, secure execution environments for executing programs and authentication methods are essential for any data intensive application.

From the above discussion we can see that the data and programs accessing the data have to stored and distributed efficiently. Having many replicas of the same data set is also not a good solution as the data storage costs will go up tremendously. Hence an efficient means of transporting the required datasets, efficient scheduling decision with respect to allotting compute resources to a particular program is required. Several grid technologies are available that perform the basic functions required for the grid. Technologies like globus, condor etc. provide such functionalities required for grid processing of data-intensive application. In the next section this paper will discuss the related work that is being done.

Apart from the basic functionalities for the grid processing, there has to be a control program that makes use of these functionalities provided by existing technologies to effectively manage the resources in the grid. The control program also keeps track of the different data sets available for use and also the different data sets produced periodically in the grid. This paper proposes a framework for such a control program. The way this framework is employed in solving any grid based data-intensive application is described in the paper.

RELATED WORK

GLOBUS:

The primary aim of the Globus project is to provide the technologies that will harness the services provided by geographically distributed resources. Research in the Globus project concentrates on building computational grid infrastructures and also on the various issues encountered during the development of grid based applications. The Globus project has created the Globus toolkit. The Globus toolkit consists of the basic grid functions required for creating any grid based application. It is an open source architecture. Key grid computing components such as file transfer protocol, job launching mechanism and security is provided by globus. GridFTP is the reliable data transport protocol implemented in the globus project. It is based on FTP, but it adds more functionalities such as parallel transfer of data, partial transfer of data and a reliable mode for transferring data. It also provides the Globus Replica Catalogue which is used to maintain the replicas of files. The Monitoring and Discovery Service (MDS) is the resource information provider in the Globus project. It is based on the Light Weight Directory Access Protocol (LDAP). MDS can be queried to get the system information. The hardware profile of a workstation can be obtained by querying MDS. MDS can also be tuned to provide information about the applications executing on a resource. The Globus Resource Allocation Manager (GRAM) allocates the resources required by programs based on the requests it gets. It also manages the programs running on a particular resource. GRAM also updates the MDS with the availability of resources. Globus provides a security infrastructure sine security is an important feature of grid based applications. It uses the Grid Security Infrastructure (GSI) for providing secure authentication and communication. GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol. So these are the component technologies provided by the Globus project for grid computation.

CONDOR:

Condor is an ongoing research project at the University of Wisconsin at Madison. Condor's ideology is based on High Throughput Computing (HTC). Scientific applications requiring numerous floating point operations will soak up the CPU of any resource to a great extent. Hence a network of resources can be used to provide the compute resources required for such computations. Condor tries to utilize the CPU cycles of otherwise idle resources. It provides a job queuing mechanism. When jobs are submitted to the condor batch system, condor finds a resource that matches that particular job's needs and the job is sent to that resource for execution. Condor can be configured in a variety of ways so that it takes advantage of the wasted CPU cycles of geographically distributed heterogeneous workstations. Condor can be made to use the CPU cycles of desktops where the keyboard and mouse are idle. When condor detects that a resource is no longer available for use, it can do check pointing and migrate the job to another available resource. Resource requests are matched with resources by means of a mechanism called ClassAds. Jobs can make requests based on the hardware required for them to run and similarly machines can state their requirements about the jobs that they are willing to run. ClassAd mechanism is based on such specifications of the resources. Condor provides a module called DAGMan that takes care of dependencies between jobs.

DAGMan is a meta-scheduler for condor jobs that have inter-dependencies. A Directed Acyclic Graph(DAG) can represent a set of jobs which depend on each other. DAGMan will submit the jobs to Condor based on the parent child relationship. Children jobs are submitted only when then parent jobs complete. Hence by not submitting the jobs that depend on each other at the same time to condor, DAGMan reduces the workload on the Condor scheduler. Condor has various daemons that take care of running jobs on the resources. Since the basic functionalities required for grid processing is supported by Condor, it is used in many grid based applications.

CONDOR-G:

Condor–G system uses software from the Condor and the globus project. Condor-G uses the intra-domain features of Condor and the inter-domain features of Globus. Since Condor doesn't take into consideration inter-domain resource sharing, security was not incorporated to a great extent. But the Globus project provides the necessary security infrastructure required to securely use resources from different domains and also for secure transport of files. Hence Condor-G takes the best out of two different technologies and encapsulates into one technology that can used for grid computing.

GriPhyn (Grid Physics Network):

This project aims at creating a Petabyte-scale computational grid for data-intensive applications. Four physics experiments that generate large volumes of data is the reason for the creation of petabyte-scale computational grids. To meet the needs of thousands of scientists, GriPhyN will deploy computational environments called Petscale Virtual Data Grids (PVDGs). The philosophy behind petabyte-scale computational grids is that CPU resources and datasets should be harnessed and datasets can be derived from other datasets using specifications describing the datasets.

These are the three major researches going on in the field of grid computing. The need for large scale resource sharing has prompted computer scientists all over the world to create computational grids that can meet the requirements of data-intensive applications. A framework for making this large scale resource sharing a reality is required. Several designs exist that try to achieve this. But each one has its own drawbacks. The most important criterion is to handle heterogeneity of the resources and to assume as little as possible about the hardware and software of a particular resource. A framework based on such as assumption will be able to adapt itself to different kinds of computing environments with the minimum changes.

Design Criteria

The primary focus of this design is to build a framework that can support any kind of data-intensive application. Heterogeneity of the resources is also taken into account. This framework can support any type of application because it is going to use intelligent wrappers for a particular type of application. The core components of the framework will be the same for different kinds of application. The difference lies in the stage at which the particular application gets executed. The reason this kind of an approach is taken is because, for a particular type of application, there maybe special kinds of files that have to be created before the application can run. This is specific to that particular application and need not be part of the core components in the framework. Hence the application is enclosed by an intelligent wrapper script written specifically for that particular application and then it is executed. This is the key feature that enables the framework to support any kind of application. The other important design criteria are as follows:

- Keep processors busy
- Minimize bandwidth requirements, avoiding strategies where requirements scale with the number of researchers times the number of input files
- Optimize cache resources to facilitate future processing needs
- Use scalable control techniques
- Applicable to multiple experiments
- Extend easily to future platforms and grid technologies
- Handle all known types of jobs
- Provide for both sharable and private resources
- Provide guidelines for job design that promote efficient use of a distributed environment, but accept legacy code that does not yet meet those guidelines

Design Approach

- The core scheduling algorithm is centralized, other processes are distributed. Scalability is achieved by supporting this core logic with multiple discovery and disposition processes as needed.
- A single control cluster and set of databases (per experiment) is used for the bulk of the control processes. The databases will mirror the status of the grid itself, with some latency.
- Part of the control software is custom-written for each experiment, and part would be the same for all experiments that use this approach. The intent is to confine job-dependent processing to custom written modules, and allow the main control software to be unfettered by such considerations.
- Processing daemons play a significant part for tasks such as monitoring, discovery, and mirroring
- A custom package wrapper is used to incorporate intelligence at the running job level

- Grid processing is most efficient when binaries can be run in parallel with small output that can be forwarded, but can still be done even if that is not possible.
- Any routine forwarding of large output files is discouraged in order to conserve bandwidth. At the least, a private cache should be the target of the forwarding so that it happens infrequently.
- Actual grid protocols (such as Globus) are wrapped to provide a measure of independence and flexibility for the control software itself, and only basic services are needed so that there is a high probability that any future grid tool can be used in the grid fabric.
- Software versioning is assumed, to provide unambiguous specification of what the job is
- A typical node in the grid requires communication mainly with the control cluster, which in turn communicates with all other nodes and the researcher's desktop. Occasional access to other nodes is required to pull files for replication (and then the other node's URL is known so there is no discovery process needed).
  - Put an intelligent wrapper around each running job segment:

  Monitors the binary itself. Responds to control commands and inquiries. Performs pre- and post-job processing (pulling software, forwarding output).
  - Use custom-written versions of specific components:

  Specifically, job creation and job wrapper tasks each have built-in experiment knowledge. Isolating job vagarities will reduce the complexity of the overall system and thus coding time. Custom code allows end users to control many aspects of the job, especially job creation.

  - Design to require only the minimum set of grid protocols and tools:

  Namely, file transport and process launching are all that is asked of the grid primitives, and proposed protocols for these functions will be embraced. Thus, heterogeneous grid tools can be simultaneously implemented (e.g., Globus and SAM).

- Separate the functions of file selection and file location:

  File selection is expected to be experiment-dependent and use existing catalogues, accessed by custom-written code. File location is handled by a new database, accessed by the new control software.

  - Simplify assumptions about job characteristics:

  Use cases so far encountered appear to need only linear chaining, mainline splitting, and mainline merging .

Modules in the Proposed Framework:

The framework consists of five core modules which will give the functionality to support any kind of data-intensive application. The five modules in the framework are as follows:

- Job Creation
- Pre-Scheduler
- Resource Monitoring Service (RMS)
- Scheduler
- Dispatcher

The various modules in the framework communicate with each other. They make use of a a set of databases for accessing necessary information that will enable in the successful performance of the framework. The databases are as follows:

1. Researcher
2. Cluster
3. Node
4. File Location
5. Job
6. RMSInfo
7. Thread

The overall operation of the modules in the framework is shown along with the databases in fig 1. in the following page.

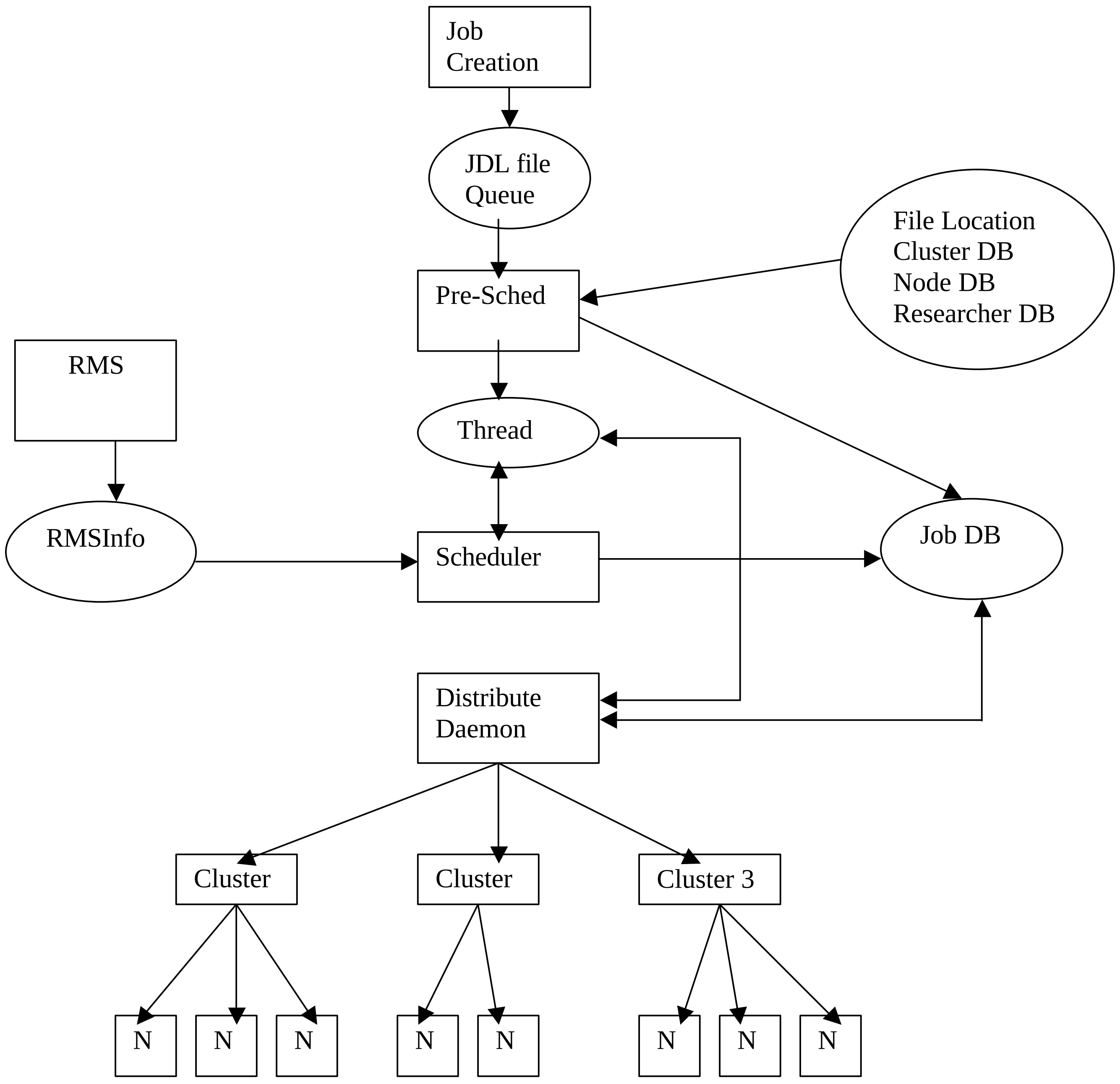
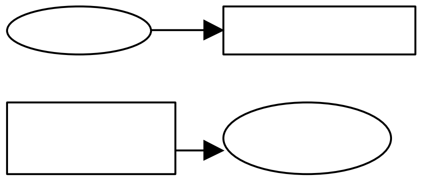


fig.1



- Left hand side is input to the right hand side

- Right hand side is the output of left hand side

N = Node in a cluster

The modular design with respect to the scheduler will eliminate the overhead on the scheduler. Since the pre-scheduler does many tasks before the scheduler executes, the workload on the scheduler is significantly reduced. The pre-scheduler also takes care of splitting the job request into different threads of execution. It updates the thread information in the thread database and stores the job request in the job database using a unique i.d. Only the scheduler has an overall view of the system with the aid of the RMSInfo database. Therefore, by moving many of the scheduling decisions that do not require an overall view to the pre-scheduler, we reduce the workload for the scheduler. The RMS will query the resources for system information and will store it in the RMSInfo database. The RMS will do this task at regular intervals. Therefore the scheduler will have the latest information about the system. The scheduler scans the thread database to make the scheduling decisions. Once it decides the target node for a thread, it will change the state of the thread to *to-be-executed*. The distribute daemon will scan the thread database to check for threads that are ready to be executed. Once it finds a thread that is ready to be executed, it will move the files required for execution to the target node using a wrapper script. It also has a wrapper around the thread to execute it and a wrapper to collect the results and send it back to the researcher or to a desired location. So the wrapper moves the files to the target node and executes the thread. The reason for letting the job itself gather the results is because if we have a separate daemon to do the task of gathering the results, it will create a bottleneck.

Communication Methods

For the grid to function, processes must be able to send and receive files between nodes (FTP-like capability) and launch tasks on other nodes (RPC-like capability). All control and application functions that are about to be described can be implemented with just these two capabilities. In order to meet the stated criteria of extensiblity to future grid platforms, this design requires that there be a software wrapper around each *grid primitive* that may be actually used to provide the FTP and RPC services. For example, if Globus is used for the frabric on a set of nodes on the grid, then instead of invoking Globus tools and functions directly, all control software described in this design will instead invoke a *wrapper* function that in turn implements a *standard protocol* to perform FTP and RPC tasks. There are several implications of this approach:

- We do not require that the entire grid be a Globus grid, or any other specific fabric: The grid may now, or in the future, have clusters that use different fundamental protocols for transport and security. This is nothing more than a good OO practice: implementing a new grid protocol would simply require writing and testing the proper wrapper functions. (If the grid were to consist of mixed protocols, it would likely be possible to implement gateway machines in a manner that is transparent to the control software).
- It is in the design stage (now) that this decision must be made, so that control software can be written to use *just* an FTP and an RPC capability:
  This design essentially requires a *lowest common denominator* approach to actual grid access, so that the large amount of control software now being written can be easily extended in the future even if a different grid protocol is to be incorporated.

Any reliance at this point on a “special feature” of any particular grid tool would reduce the chance of easy future extension.

- This will make it easier to interface with existing data transport functions such as SAM:
  SAM would be invoked whenever a master copy of a file must be extracted from the tape robot (but only to place the copy onto a local disk cache). This is simply a case where an FTP-like service is needed, but the fundamental protocol is not Globus. The control software would simply request the file transport, and wrapper functions would handle the details. To implement, one would place Globus on the SAM clusters to forward the disk file. Although difficult in implementation, it appears to the control software to be a simple FTP service.

Job Creation

The job creation program will be a web-interface. When a researcher wants to submit a job to the grid, he will go to the web-interface to submit the job. The web-interface will provide the researcher with all the necessary information to construct the job. It will display a catalogue of files to the researcher and he can choose the type of application he wants to run. The researcher can also specify the parameters if any and also if the application has a special parameter file, he will be allowed to specify it. The researcher will also be given the option of choosing the cluster in which he would like his application to run. But the researcher can choose only the cluster to which he is authorized to use. Otherwise he will be denied access. The web-interface is the job creation program. It will take all the information provided by the researcher and will create the job. The job will be described by a Job Definition Language (JDL). The JDL is the standard that is used to describe the different components in the job. The different components are the different binary files that have to executed, the parameters for the job, the different parameter files if any, input files and the output files, the credentials of the researcher who submitted the job and so on. The job creation program will generate the JDL file for a particular job and it will place it in the JDL file queue that is visible to the Pre-scheduler. Hence the output of the job creation program is the JDL file.

The job creation program will run on the researcher desktop and will access the different databases in the system. A custom job creation program will be written for each application. So this custom job creation program will be able to identify the necessary files that are required for that application and will provide it to the researcher as a catalogue from which he can choose. Some job creation programs will focus mainly on listing the input files that a researcher needs to select from so that he can create a new job. Other job creation programs might concentrate on displaying the cluster information along with the files present in the cluster, so that the researcher can efficiently select the files and the target cluster. The job creation program should provide template for the researcher to create new jobs. It should also be able to clone new jobs by looking up jobs that are archived in the job database that are stored permanently. These are the primary tasks of the job creation program and its primary output is the JDL file describing a particular job.

Pre-Scheduler

The task of the pre-scheduler is to parse the Job Description Language (JDL) file and make as many scheduling decisions as possible that do not require a global view. This will reduce the amount of work the scheduler has to do. The JDL file is the output of the job-creation program that runs at the researcher's desktop. The JDL file is written to a JDL file queue that is periodically scanned by the pre-scheduler. The pre-scheduler will perform its functions on the JDL file and then send a result back either accepting or rejecting the job request. The pre-scheduler will also generate a unique job i.d. and return it along with accept or reject reply. This number will be used by the researcher for making queries about the job. The pre-scheduler will maintain the last unique i.d. it generated. The pre-scheduler will also store the JDL file in a job database. It is also responsible for breaking down the job request into multiple phases and threads. It updates the thread database with this info.

Tasks of the Pre-Scheduler:

- Parse the JDL file
- Make a list of input files
- Make a list of output files
- Make a list of binaries to be run including the version
- Contact the Location database and verify the specified files are present
- Contact the Cluster database to verify the clusters specified in the JDL file
- Contact the Node database to verify the nodes
- Contact the Researcher database to authenticate the identity of the researcher
- Once the verifications are done, either accept or reject the job request
- From the JDL file determine the DAG that might exist in the job request, or
- If the DAG is already specified in the JDL file, then use it
- Add the job in the Job database or the distribute queue and mark it *to be scheduled*
- Generate a unique i.d. for the job
- Scan the directory where the JDL file is placed at specified intervals, or
- Provide an API, which the Job-submit program can call so that it can invoke the pre-scheduler every time a request arrive
- Update the job database.
- Break the job request into phases and threads.
- Update the thread database
-

Jobs that are not input-dependent would already know the target nodes because they were determined during job creation. For input-dependent jobs, pre-scheduling includes a discovery of every location of every needed input file (accessing the Location Catalogue) which provides a list of clusters that already have the input file present. The pre-scheduler attempts to apply as many scheduling rules as it can without having to know anything about the present global state of the grid. For example, a job might have been forced by policy to run on a specific private cluster, hence the pre-scheduler will have to put all packages onto the designated cluster. Certainly, only those clusters with the proper OS profile (Platform ID) are to be considered. Basically, the pre-scheduler attempts to eliminate as many cluster options as it can, so that whatever is left is essentially an optimization decision that is best left to a single process with a global view. Thus, it will prepare a list of *candidate* packages that could be sent out to accomplish the processing of a given package, and assign a cost to each. *Cost* might consist of a certain amount of data transport, a certain amount of re-processing to create the needed input file at some cluster, and so forth. The pre-scheduler passes the job onto the scheduler by changing the job's status to "to be scheduled".

Resource Monitoring Service (RMS)

The Resource Monitoring Service is used to keep track of the system information of each of the resource in the grid. System information refers to the number of jobs running on a resource, the CPU speed, the main memory available, the cache available etc. These system level information is required for the scheduler to make decisions regarding assigning the jobs to a particular resource. This has to be made available to the scheduler because it has to know the global state of the grid before it makes any scheduling decisions. If the scheduler were to check the state of each resource before it made a scheduling decision, it will be a very time consuming task and will be practically infeasible. Hence we have a module called the Resource Monitoring Service (RMS) which does the task of keeping track of the system level information. This module will periodically scan the grid to check for system changes. This module will also be notified by certain other components in the framework about system changes.

The RMS will provide the RMSInfo database to the scheduler. This database will contain the system level information for the resources in the gird at a given point in time. The RMS will update this database with all the changes in the system state from time to time. The reason for having the RMSInfo database is because of efficiency reasons. If the scheduler were to query the RMS for system information for every scheduling decision it has to make, it will be a very slow process and the scheduler cannot make that many scheduling decisions. Since the RMS is going to constantly update the RMSInfo database with the system level information for each of the resource in the grid, the scheduler will make use of this database to get information for its scheduling decisions. Since the information contained in the RMSInfo database gives a global view of the grid, it will help the scheduler in making its decisions. This is the primary function of the RMS module. The RMS will not update the RMSInfo database for every single change in state of the system. It will update the RMSInfo database only when there is a significant change in the state of the system. The RMS module greatly reduces the overhead on the scheduler and lets the scheduler concentrate on finding the best resource for a particular application and not worry about finding the state of the whole system.

Tasks of RMS:

- Maintain system information
- Update RMSInfo database if there is significant change in system information

Scheduler

There is a single core job scheduling process for the entire experiment. Having a global view, it examines the jobs needing scheduling in priority sequence and makes optimization decisions from the candidates that the pre-scheduler processes provided. Then the actual launch of each package is left to the job distribution processes. The design calls for the load on the core scheduler itself to be light enough that it will not become a bottleneck.

Once the pre-scheduling stage has completed, the threads are placed in the distribute queue or the thread database. At this point it is the duty of the scheduler to assign it to the appropriate cluster using its global view. The scheduler will read RMSinfo to get system information and then make the decisions. Once the scheduler has arrived at the decisions based on the system information and various priorities, it will place the thread in a state where it has to be distributed to the target cluster. The distribute daemon performs this task.

Tasks of the Scheduler:

- Process one of the thread that have been marked *to-be-scheduled*
- Read RMSInfo to get the system information about the various nodes in the different clusters
- No CPU should be neglected
- Cache purging should be minimized, because purged files might be used again
- Commonly used files should be replicated in many locations so that the options for processing are increased
- Mark the thread as *to-be-distributed* once a target resource has been selected
- Will schedule a thread with a parent only when the parent has completed successfully.
- If the parent thread has failed, then the scheduler will remove all the threads down the hierarchy.

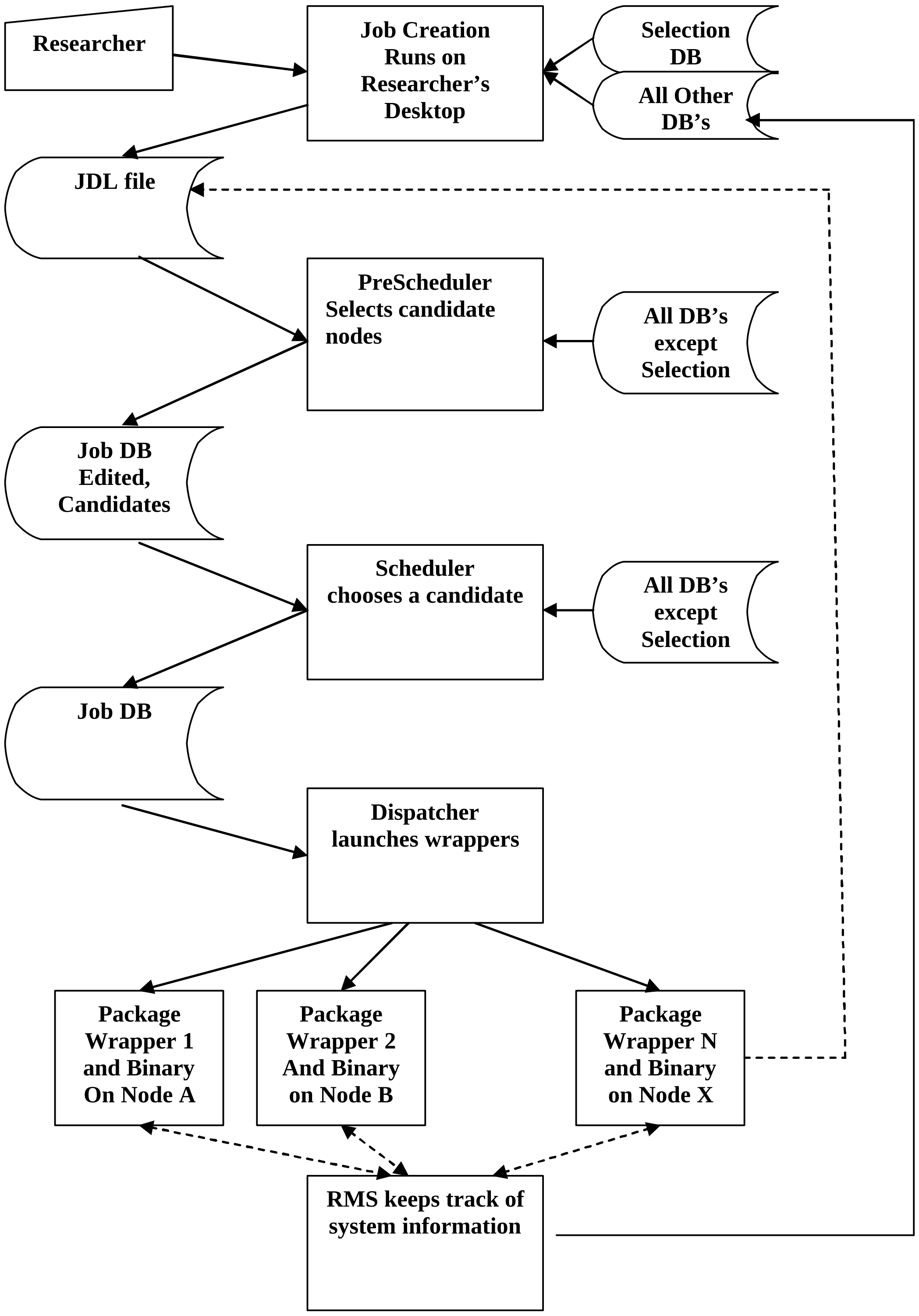
Distribute Daemon

The distribute daemon will scan the thread database and will distribute the thread marked as *to-be-distributed* to the target node once the target node becomes available. The distribute daemon will also handle the job of moving the files across from the central repository, including the intermediate input files. In practice this task will be taken care of by a wrapper script for that particular thread. There is also a wrapper script present to execute the thread. Once the thread completes its execution, the results are gathered by another wrapper script. The threads are not removed from the area of execution unless the researcher makes a request to purge them. But if the memory available to hold that thread becomes less, in that case it might be purged or moved over to another location and archived. When the job creation program created the job description, it would have asked the researcher about the method that has to be used to transfer the results. Based on that, the job creation program would have added a phase to the job stating what to do after the completion of all the binaries. Based on that, the results of the different phases will be operated upon.

Tasks of the Distribute Daemon:

- Scan the thread database for thread that are marked as *to-be -distributed*
- Move the required binaries to their target nodes
- Move the input files to the target node of the binary
- Once the node becomes available, launch the thread
- Update the status of the thread in the thread database.

In the following page, fig 2 will give the job flow in the proposed framework.


Researcher
Job Creation Runs on Researcher's Desktop
Selection DB
All Other DB's
JDL file
PreScheduler Selects candidate nodes
All DB's except Selection
Job DB Edited, Candidates
Scheduler chooses a candidate
All DB's except Selection
Job DB
Dispatcher launches wrappers
Package Wrapper 1 and Binary On Node A
Package Wrapper 2 And Binary on Node B
Package Wrapper N and Binary on Node X
RMS keeps track of system information

Features

Scalability:

The most important feature of any grid based solution is scalability. Since the grid is expanding continuously, the ability of the framework to handle more resources that are geographically distributed is very vital for the success of the framework. The proposed framework is highly scalable. The two different modules that make this possible are the pre-scheduler and the dispatcher. There can be multiple pre-schedulers and dispatchers through out the grid. The pre-schedulers themselves can be geographically distributed and one pre-scheduler can parse the JDL files in the JDL queue of one particular region. The same holds good for the dispatcher too. If more dispatchers are present, then they will distribute more threads at any given time. This feature makes it easy to increase the load on the grid and still produce a good throughput. The scheduler will not be a bottleneck because the workload on the scheduler is greatly reduced by the presence on pre-schedulers.

Heterogeneity:

In a grid environment, the resources that are present will be of different kinds. Each resource might have a different hardware, different operating system running on them. So to assume that all the resources in the grid would have the same configuration is a grave mistake. This framework assumes that the resources have the minimum of configurations available. It doesn't assume that any custom built software to be present on the resources. The use of wrappers in the framework makes this possible. For example if a resource has gridftp, in that case, the wrapper function to copy files will make use of gridftp. If the resource has only ftp running on it, then in that case, the wrapper will make use of ftp. The wrapper scripts are intelligent enough to figure that out. Hence heterogeneity of resources can be handled easily using this framework.

Reliability:

Reliability is very important in any distributed computing environment. The proposed framework makes sure that the system is reliable. The researcher who submits a job gets an i.d. for the job that he submitted to the system. The researcher can make use of this i.d. to track the status of his jobs. The wrapper scripts that execute a particular job are self-contained units. They will execute the binary and will update the thread database with the status information. Even if the job fails, the scripts will make sure that the proper status information is propagated to the scheduler so that it can take the necessary action. Once a binary fails, the wrapper script will catch the result and this is made visible to the scheduler. The scheduler will then either reschedule the job, or will discard the job, based on the criteria specified while creating the job.

Security:

Since the grid involves different organizations who share their resources, it is important to establish a certain level of security mechanism. Only authorized people are allowed to access the grid. This has to be implemented strictly. Otherwise, malicious intruders can cause serious harm to the resources. The framework uses a researcher database to keep track of the researchers who are entitled to use the grid. Apart from this, the framework calls for the use of GSI for authentication. This will make the system very secure.

Conclusion:

In this paper we have proposed a framework for the control and monitoring of grid based data-intensive applications. The paper describes the different areas of scientific research that fall under the category of data-intensive application. We have briefly looked at the current technologies in grid computing. The paper investigates the issues that arise in dealing with data-intensive applications. A detailed description on how the proposed framework can solve the problems encountered in managing data-intensive application is given. The individual modules in the framework are described in great detail. We are currently involved in the development of a prototype for the proposed framework for dealing with data-intensive applications in High Energy Physics.