DataMover: A Lightweight Extensible Data Movement Framework for Grid Environment

Wantao Liu[1,2], Rajkumar Kettimuthu[3,4], Ravi Madduri[3,4]
[1] School of Computer Science and Engineering, Beihang University, Beijing, China
[2] Department of Computer Science, The University of Chicago, Chicago, IL USA
[3] Argonne National Laboratory, Argonne, IL USA
[4] Computation Institute, The University of Chicago, Chicago, IL USA
liuwt@uchicago.edu, kettimut@mcs.anl.gov, madduri@mcs.anl.gov

Abstract

A common requirement within grid environments is the ability to transfer data from one location to another. The service and protocols required to support such data transfers will vary between grid deployments, the actor requesting a data transfer, and potentially the amount of data that is to be transferred. Due to the diversity of these factors, it's necessary to provide a middleware to hide the complexity and heterogeneity of such data transfer to upper-layer applications. In this paper, we describe the design and implementation of DataMover, which is a middleware for third-party data transfer in compliance with OGF DMI specification. DataMover is a lightweight extensible third-party data movement framework based on Globus Toolkit. It is designed as a set of WSRF-compliant web services and implemented using Java language. It offers a simple uniform interface for management of data movement task from a given source to a specified destination and supports multiple underlying transfer protocols. In addition, DataMover is able to find a route for data transfer task in which source site and destination site have different transfer protocols. Application developers can easily integrate their preferable data transfer protocol into DataMover. Finally, we evaluate the performance of DataMover in various scenarios.

1. Introduction

Data transfer is a common activity in distributed environments. It has been well studied for many years, and a handful of transfer protocols and mechanisms have been proposed and implemented, such as FTP [1], GridFTP [2], HTTP [22], and SCP [23].

Data transfer can be broadly classified into two categories: client-server transfer and third-party transfer. Client-server transfer is performed between a client and a server. Client initiates a transfer to read data from the server or write data into the server. FTP upload or download operation is an example of client-server transfer. In third-party transfer, there are three parties involved: one client and two servers. The client communicates with the two servers to coordinate the data transfer, and the actual data is moved between the two servers directly. Third-party transfer is the main focus of this paper.

To manage large data sets for distributed communities, authenticated third-party data transfers between storage servers are essential. A very good example for this is LCG [3], the grid computing infrastructure built for LHC, which produces several petabytes of data for 15 to 20 years. LCG is composed of 3 tiers of sites: tier-0 locates in CERN, below that is tier-1 sites in several participating countries; at the end of this hierarchy structure is tier-2 sites operated by universities or institutes. Since LHC experiments are a collaboration of scientists all over the world and the data amount produced is extremely huge, data produced by these experiments is not only stored at tier-0, but also replicated partly to tier-1 and tier-2. This type of data transfer activity is inherently suitable for third-party transfer, as it makes it easy to manage transfers between sites. The user does not have to log on to source or destination to do the transfer. Third-party is also useful at the context of data intensive jobs scheduling. When jobs are dispatched to remote sites for execution, data required by the job need to be first staged in from data storage site to execution site before job starts; Similarly, after job running, the result data need to be staged out to data storage site.

However, as stated above, there are a number of data transfer protocols; the way data transfers are performed utilizing different transfer protocols is also different in terms of programming interface for application developers. It's a huge challenge for them to integrate multiple transfer protocols into their applications for data transfers.

In order to hide the complexity and heterogeneity of underlying data transfer protocols, and mitigate burdens of application developers, Data Movement Interface (DMI) specification draft [16] has been developed jointly by EPCC, Fujitsu, IBM, Microsoft, and Argonne National Lab as part of open grid forum (OGF) [24] DMI working group. DMI specification draft defines a universal web service interface for application programmers to conduct third-party transfer without concerns of the underlying data transfer protocols.

Java language [20] has been widely used in web service programming due to its features like platform independent, object oriented, easy to study and use. Another reason why Java is so popular is that there are a lot of tools and middleware available for Java web service programming, a big community develops and maintains them.

In this paper, we present design and implementation of DataMover, which is a lightweight extensible framework for performing third-party data transfer in grid environment based on DMI specification. DataMover not only provides a universal interface for application programmer, but also offers an adaptor interface for easy integration of data transfer protocols.

Specifically, DataMover is a set of WSRF-compliant web services implemented using Java language. It provides a universal programming interface for application programmers to conduct third-party transfers. While designing DataMover, the following principles are mostly valued:

- Comply with standard;
- Easy to use;
- Efficiency;
- Extensible.

The paper is organized as follows. In section 2, some previous work is reviewed; in section 3, design and implementation of DataMover is presented; experiment results are discussed in Section 4; future work and conclusion is in Section 5 and Section 6, respectively.

2. Related Work

SRB (Storage Resource Broker) [5] is an integrated data management system developed by SDSC, it concerns mainly on shared "collections", which refers to a logical name for a set of data objects. From user's perspective, SRB provides a single-image logical file system; they don't need to worry about where data objects are located. SRB employs a series of methods to optimize data transfer. Multiple streams are used for large file transfer, and for movement of a lot of small files, SRB first fill small files into a large buffer and then send the whole buffer in a single operation. Third-party transfer is also supported in SRB. Due to the extensibility of our framework, DataMover can simply integrate SRB as an underlying transfer mechanism. Now SRB is not supported in our current implementation.

CaGrid transfer [6] is a component of caGrid, and is implemented as an extension to Introduce, which is a graphical grid service design and generation tool developed by caGrid. It's capable of uploading data to service or downloading data to client. CaGrid Transfer is composed of four components: Transfer Service, Transfer Service Helper, Transfer WebApp and Transfer Client Helper. Among them, transfer service is the core component, which is a WSRF-based grid service. Only HTTP and GSI-based HTTPS are supported, they are encapsulated by Transfer Webapp, which is a Java servlet. When generating web service using Introduce, users can simply add CaGrid Transfer to their service through a graphical interface. As CaGrid Transfer holds data into a byte array, it is not suitable for big volume of data movement. In order to move big data, caGrid developed a component, named BulkData Transport [17], which is based on GridFTP.

GridFTP [2] protocol is an extension of plain FTP protocol. It defines general-purpose mechanism for secure, reliable, high-performance data movement. It has been widely used for efficiently transferring large volumes of data. Globus implementation of GridFTP [7] has a modular structure that supports multiple security options, multiple transport protocols, coordinated data transfer utilizing multiple computer nodes at the source and destination, and other desirable features.

RFT (Reliable Transfer Service) [8] is a component of Globus Toolkit [9]. It is implemented as a set of web services, and is able to perform third-party transfers using GridFTP with basic reliable mechanisms. Data transfer state is dumped into database, when a transfer fails, it can be restarted automatically from the broken point using the persisted data. It also supports adjustment of GridFTP parameters, like TCP buffer size, parallel streams, through web service interface. The major difference between our work and RFT is that RFT provides some mechanisms to guarantee reliable transfer. In addition, RFT is tightly coupled with GridFTP, However, DataMover is open to various transfer protocols.

In Stork [18], data placement activity is considered as the first class citizen in the Grid. It's capable of queuing, scheduling, monitoring and managing data placement activities. Stork accepts data placement jobs from DAGMan [19], and executes them according to given policy. Stork also support multiple data transfer protocols to conduct third-party transfer, and protocol adaptation can be finished at runtime. The difference between Stork and our work is that Stock is a data scheduler basically, it interacts with DAGMan to accept jobs, while DataMover presented in this paper interacts with application programmer directly, and it provides a layer of API to hide underlying complexity and heterogeneity of the transport mechanisms.

Bistro [10][11] is designed as an internet scale application for upload data, scalability is the key issue addressed by that work. Upload refers to the communication mode in which many clients want to transfer data to a server. The basic approach in bistro is to introduce intermediate sites, termed "bistros". Clients first push data into a bistro, and then the data is pulled by destination server from the bistro. In that work, data transfer is divided into four categories in terms of communication model: one-to-one, one-to-many, many-to-

many, and many-to-one. According to this classification, bistro belongs to "many-to-one", and our work falls into the "one-to-one" category.

3. Design and Implementation of DataMover

3.1. Design principles

As stated above, the following principles are used to guide design of DataMover:

- Comply with standard: DataMover implements DMI specification, so that it has standard web service interface and can seamlessly interact with clients which are developed for other DMI implementation. This makes life of application programmers easier. Furthermore, data transfer in grid environment is error-prone; it can be interrupted due to server crashes, network outages etc. Also, users want to check the status of their transfers periodically or receive notifications about the state of the data transfers. This requires data transfer state to be maintained by DataMover. Due to the stateless of plain web service, we use WSRF [12] and WS-N [13] specifications to maintain state and notify users, which is a standard way to support state in web service.
- Easy to use: DataMover is a layer of middleware between underlying data transfer protocols and upper layer applications. We do not intend to impose a steep learning curve to application programmers. The complexity and heterogeneity of different underlying data transfer protocols should be hidden by DataMover, meanwhile, the programming burden should be mitigated. So the application developers only see a simple interface, enabling them to concentrate on their application logic, without being troubled by the details. To achieve this goal, a simple API is designed to expose to application programmers, and there is no complex configuration in DataMover.
- Efficiency: DataMover provides a universal interface for third-party data transfer; it's a layer of middleware and definitely imposes some overhead. One of our objectives is to minimize the overhead. Our performance experiments result show that the overhead is very limited, that's why we call DataMover "lightweight".
- Extensible: Due to the diversity of underlying data transfer protocols, it's not practical for DataMover to incorporate all protocols. In this sense, extensibility is a key requirement for DataMover. Application programmers should be able to easily integrate new transfer protocols into DataMover to satisfy their own need.

3.2. Architecture of DataMover

Figure 1 shows structure of DataMover from application programmer's perspective. This figure is from DMI specification. DataMover exposes two web services to application programmers: FactoryService and InstanceService. FactoryService is a plain web service, and is responsible for initiating InstanceService. Since a site may support multiple underlying transfer protocols, the FactoryService need to negotiate with source site and destination site to select a transfer protocol to undertake the

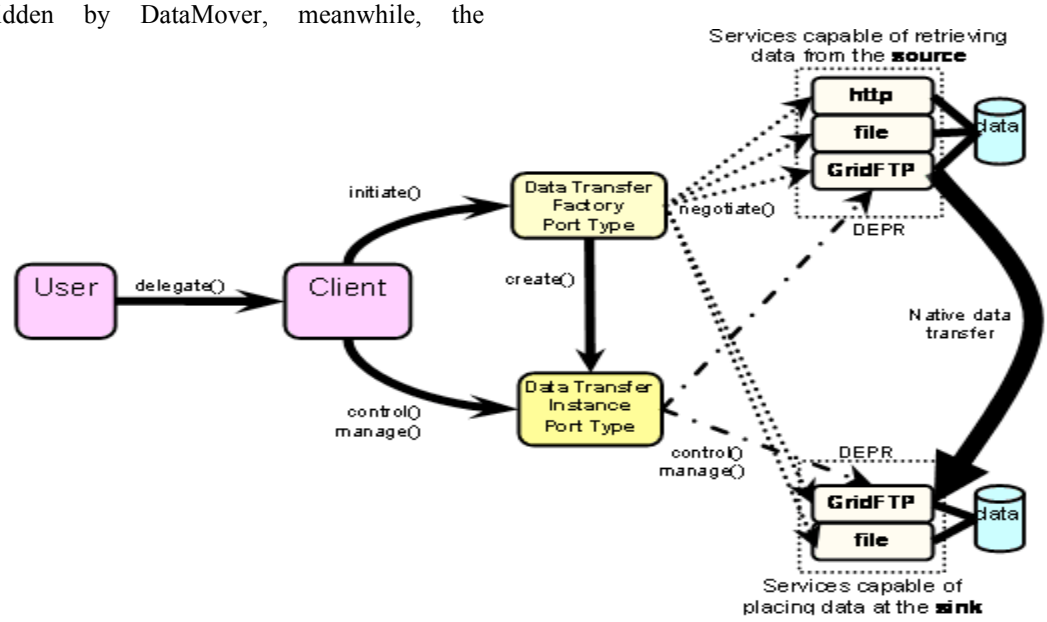

Figure 1 Structure of DataMover from application programmer's perspective, from DMI specification

transfer according to parameters provided by client. DataMover is also able to conduct a transfer between two sites which don't have a common transfer protocol; this feature will be elaborated later. After initiation, client talks with InstanceService to perform the actual data transfer and manage the whole transfer process. The data flows from source site to destination site directly, without involving of InstanceService. InstanceService is a WSRF-compliant web service. A resource is created for each transfer and notification mechanism is employed to report state to user. Access information of a site is encapsulated into a Data End Point Reference (DEPR), which is a specialized form of EPR from WS-Addressing [14]. Each site has a unique DEPR to identify itself. It also contains information about which transfer protocols are supported by the site, and FactoryService uses this information to negotiate a third-party transfer between source site and destination site.

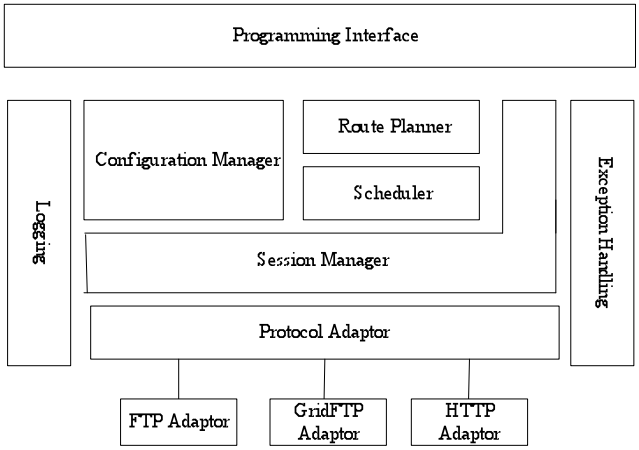

Figure 2 DataMover's architecture

DataMover is designed as modular. Figure 2 demonstrates its architecture. We will examine this architecture from bottom up. Due to the complexity and heterogeneity of different underlying data transfer protocols, a layer is required to hide details of these concrete protocols to upper layers, we call this layer protocol adaptor. It provides a flexible mechanism to add support for new transfer protocols. Application programmers only need to implement a specific interface of protocol adaptor to plug a new protocol into the whole architecture.

In DataMover, an end-to-end transfer is called a "session". A session consists of one or more point-to-point transfers; it can be regarded as a "logical transfer". The concept of session makes it easier to implement some complicated features over basic data transfer function, and the extensibility of the whole architecture is also enhanced, which is one of our design principles. A session is managed and monitored by session manager, through which the upper layer of DataMover manipulates a session. Application programmers are also able to pass command to session manager through web service interface.

Scheduler is responsible for making scheduling decisions of data transfer. It maintains a queue, where each data transfer will be assigned a place using a scheduling policy. Currently, DataMover only supports "First Come First Serve" policy.

Route planner accepts source and destination DEPR, and makes a plan about how to execute the transfer based on selected policy. This module is basically designed for those transfers in which source site and destination site don't have a common transfer protocol, in this case, one or more intermediate sites is required to carry out the transfer successfully. Currently, only "FirstFit" policy is implemented. This feature will be explained at next section detailed.

Configuration manager is the component in charge of setting configuration parameters of DataMover, such as transfer retry times, logging setting etc.

The programming interface exposed by DataMover is in the form of WSRF-compliant web services. We will introduce this interface in Section 3.4.

3.3. Data transfer relay

In some cases, it's required to transfer data between two sites which do not have a common transfer protocol. As illustrated in Figure 3, site A has a deployment of GridFTP, and site B has HTTP only. Obviously, the two sites can not establish a connection and transfer data directly. In order to address this issue, a "relay" mechanism is designed in DataMover. As in the example, an intermediate site is employed, which has both GridFTP and HTTP, to relay the data movement. As illustrated in the figure, a third-party transfer controlled by DataMover is first performed between site A and C over GridFTP connection, and then another third-party transfer moves data from C to its destination B over HTTP. Logically, data is moved from A to B directly.

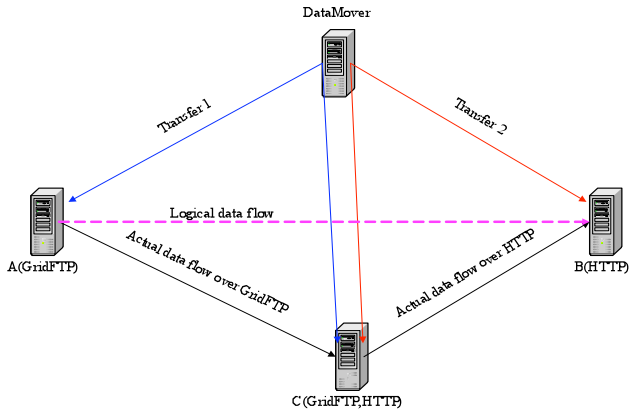

Figure 3 Data transfer relay

Generally speaking, the process of finding an intermediate site requires interaction with grid information service, such as Globus MDS [15]. If more than one eligible

intermediate node is available, a policy should be employed to select an optimal node. For simplicity, our current implementation uses a static configuration file, instead of grid information service, to get the suitable sites. And only "FirstFit" selection policy is implemented, which means the first site that can relay the transfer is picked up.

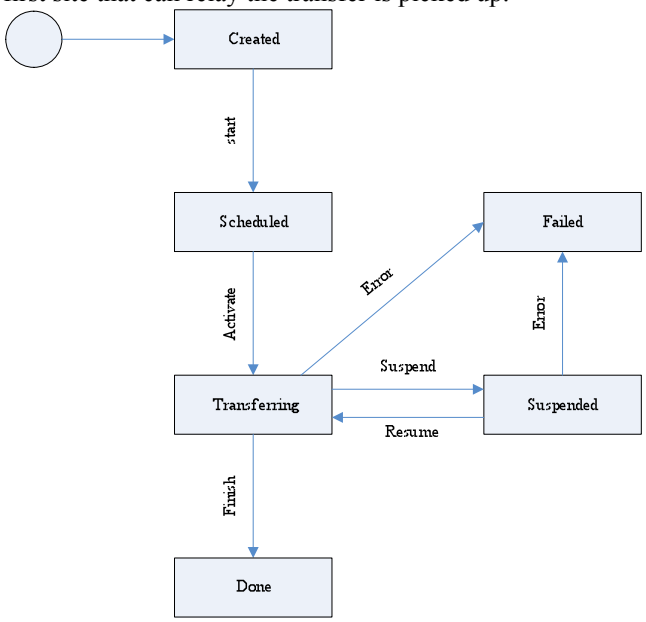

Figure 4 DataMover internal state transit graph

3.4. Application programming interface

As mentioned above, DataMover is composed of two web services: FactoryService and InstanceService. FactoryService is a plain web service, it has only one operation:

- TransferInstance getInstance(GetInstanceRequest param)

The source and destination DEPR as well as other transfer requirements are encapsulated into type GetInstanceRequest.

InstanceService is a WSRF web service, it utilizes resource properties to maintain transfer state, and notification mechanism is utilized as well to notify users about transfer state timely. A new resource will be created for each data transfer. There are five operations in InstanceService:

- getStatus( );
- start( );
- stop( );
- suspend( );
- resume( ).

Figure 4 is DataMover's internal state transit graph. State transit takes place at the trigger of operation invocation or specified events. It's "Created" state when a new resource is created and initiated. When start( ) is invoked, the created transfer will be put into the proper place of a queue according to specific scheduling policy, and the state transits into "Scheduled". It will go into "Transferring" state if the transfer is at the head of the queue and actual data transfer begins. If suspend( ) operation is invoked, the state will switch into "Suspended", and resume( ) operation will make the state come back to "transferring". If data transfer finishes successfully, the state will be "Done"; otherwise, it will be "Failed".

3.5. Implementation

Due to the modular design of DataMover, Java is a pretty appropriate option to implement it. The following features of Java language are utilized to make our implementation easier:

- Object oriented: Entities and concepts in the architecture are wrapped into different classes to make the logic clearer; Complex classes are assembled from simple ones.
- Interface mechanism: It's the key feature required to reach the extensible objective. New protocol adaptor and scheduling policy is incorporated into the architecture through interface, other components only interact with their interface no matter how they are implemented.
- Performance: The execution efficiency of Java is not so satisfactory at its beginning days; whereas, with boost of hardware performance and improvement of JVM, now Java can meet performance requirements of most cases.
- Cross-platform: DataMover is expected to be able to run at different platforms, through Java's platform independent feature, the same code can compile and execute in the same way at different platforms.

DataMover has been implemented using Java language. Since WSRF and WS-N specifications are involved, Globus Toolkit ws-core is selected as the web service container. At present, we implements two protocol adaptors: GridFTP adaptor and HTTP adaptor. GridFTP adaptor is based on CoG jglobus library [21], and HTTP adaptor is programmed as a web application deployed into Apache Tomcat.

4. Performance Evaluation

In this section, performance evaluation results are presented. To evaluate the performance of DataMover and overhead incurred by web service invocation, we carried out the following experiments.

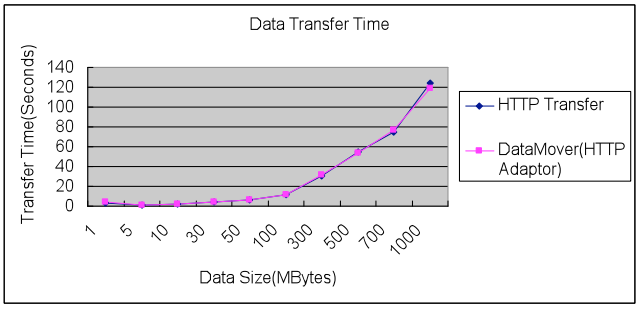

Figure 5 Data transfer time comparison between locally invoked HTTP third-party transfer and DataMover using HTTP adaptor. Data is moved from ANL to ORNL, RTT is about 25 ms

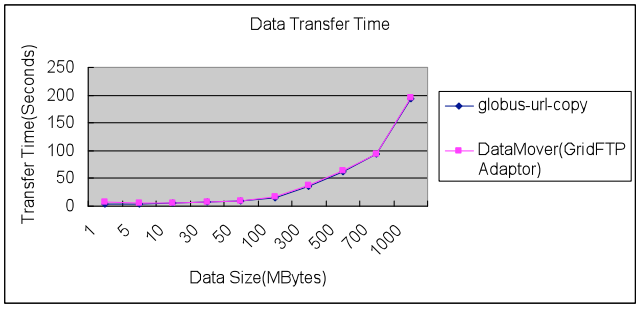

Figure 6 Data transfer time comparison between globus-url-copy third-party transfer and DataMover using GridFTP adaptor. Data is moved from ANL to ORNL, RTT is about 25 ms

Figure 5 and Figure 6 demonstrate the performance of DataMover with different data transfer protocol adaptors. The data is transferred from a machine in Argonne National Lab to a machine in Oak Ridge National Lab. Round Trip Time (RTT) between source and destination is about 25ms. DataMover is deployed into a GT4.0.8 ws-core container. In order to reduce network delay for web service invocation as much as possible, test client was running at the same machine as DataMover. Ten files of sizes 1MB, 5MB, 10MB, 30MB, 50MB, 100MB, 300MB, 500MB, 700MB, and 1000MB were transferred over the network. Each size of data was transferred five times, and the average value of each transfer time is calculated to draw these graphs. In Figure 5, DataMover uses HTTP adaptor, and compares with HTTP third-party transfer, invoked by a command-line program. From this graph, we can see that the data transfer takes longer time with the increase of data size, and the two curves are very close to each other; this demonstrates that the overhead introduced by DataMover is very little. In Figure 6, GridFTP adaptor is employed in DataMover, and the data transfer time is in comparison with that of globus-url-copy, which is a command-line tool distributed with Globus GridFTP. The result shows similar trend with Figure 5.

To investigate the overhead imposed by web service invocation, we invoked a "dummy" DataMover 10 times in a loop continuously, here "dummy" means no actual data transfer is performed, but the other logic is identical with that of DataMover. The "dummy" DataMover and client are deployed in the same machine. The result shows in Figure 7. Each DataMover invocation actually is composed of two web service invocation: FactoryService and InstanceService. As shown in the figure, the first invocation of "dummy" DataMover takes about 4 seconds, while others only cost about 100ms. That's because for the first time, client side Axis SOAP engine requires an initiation process, and Axis maintains a client cache as well. To verify this, we carried out this experiment using other web service, and the results confirmed our conclusion. Due to this phenomenon, the average overhead by DataMover is less than 1 second.

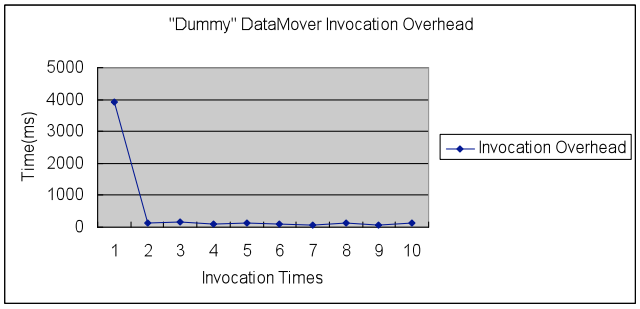

Figure 7 "dummy" DataMover invocation overhead. Both DataMover and client are deployed in the same machine

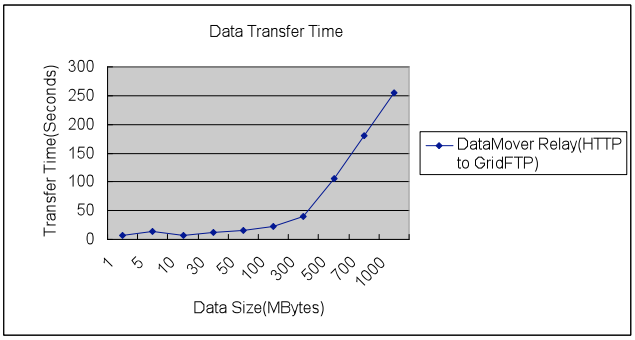

Figure 8 Data transfer time relayed by DataMover, source site locates in ANL using HTTP protocol, and destination site locates in ORNL using GridFTP protocol, the transfer is relayed by an intermediate machine in ANL, which has both GridFTP and HTTP

The next experiment shows the performance of the transfer relayed by an intermediate site. In this test, both source site and intermediate site are located in Argonne National Lab, and destination site is in Oak Ridge National Lab. The data is first transferred from source to intermediate site through HTTP protocol, and then moved from intermediate site to destination over GridFTP protocol. The result is shown in Figure 8.

5. Future Work

In future, we plan to add connection splitting feature into DataMover to split a long, poor performance connection into a series of short, good performance connections. We plan to research into how to efficiently split a connection and develop an algorithm to solve it. Route planner now only supports "FirstFit" policy to pick up an intermediate site. We plan to develop some more sophisticated alternative policies in which historical information is used to evaluate which site is suitable. We intend to add more advanced scheduling policies with consideration of transfer requirement, such as QoS requirement, cost requirement, etc.

6. Conclusion

In this paper, we presented the design and implementation of a lightweight extensible data movement framework: DataMover. It performs third-party transfers between two servers, and hides the complexity and heterogeneity of various underlying data transfer protocols. We depicted the principles which guided the design of DataMover and its architecture. We evaluated the performance of DataMover and the results show that it is efficient.

7. Acknowledgements

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357 and in part by National Science Foundation's CDIGS.

REFERENCES

[1] J. Postel and J. Reynolds, "File Transfer Protocol," IETF, RFC 959, 1985.

[2] Allcock, W. GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-RP. 020, 2003.

[3] Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., and Stockinger, K. (2000) "Data Management in an International Data Grid Project," IEEE/ACM International Workshop on Grid Computing Grid'2000, Bangalore, India 17-20 December 2000.

[4] Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[5] http://www.sdsc.edu/srb/index.php/Main_Page

[6] http://www.cagrid.org/wiki/CaGridTransfer

[7] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in SC'05, ACM Press, 2005

[8] http://globus.org/toolkit/docs/latest-stable/data/rft/#rft.

[9] http://globus.org/toolkit/

[10] S. Bhattacharjee, W. Cheng, Chou C., Golubchik L., and Khuller S. Bistro: A Framework for Building Scalable Wide-Area Upload Applications. Performance Evaluation Review, 28(2):29-35, 2000.

[11] Golubchik, L. Cheng, W.C., Chou, C-F, Khuller, S., Samet, H. and Wan, Y.C.J. "Bistro: A Scalable and Secure Data Transfer Service For Digital Government Applications," Communications of the ACM (46:1), 2003, pp. 50-51.

[12] www.oasis-open.org/committees/wsrf/

[13] www.oasis-open.org/committees/wsn/

[14] www.w3.org/Submission/ws-addressing/

[15] http://globus.org/toolkit/docs/latest-stable/info/#info

[16] http://forge.gridforum.org/sf/projects/ogsa-dmi-wg

[17] http://cagrid.org/wiki/BulkDataTransport

[18] T. Kosar and M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid. In Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS 2004), Tokyo, Japan, March 2004.

[19] Condor. The Directed Acyclic Graph Manager. http://www.cs.wisc.edu/condor/dagman/

[20] J. Gosling, B. Joy, and G. Steele, The Java Language Specification, Addison Wesley, 1996

[21] http://dev.globus.org/wiki/CoG_jglobus

[22] http://www.w3.org/Protocols/

[23] http://www.eos.ncsu.edu/remoteaccess/man/scp.html

[24] http://www.ogf.org/