Adaptive Data Transfers that Utilize Policies for Resource Sharing

Junmin Gu[1], David Smith[2], Ann L. Chervenak[2], Alex Sim[1]

[1] Lawrence Berkeley National Laboratory
Berkeley, CA, USA
{jgu, asim}@lbl.gov

[2] Information Sciences Institute
University of Southern California
Marina del Rey, CA, USA
{smithd, annc}@isi.edu

***Abstract*—With scientific data collected at unprecedented volumes and rates, the success of large scientific collaborations requires that they provide distributed data access with improved data access latencies and increased reliability to a large user community. The goal of the ADAPT (Adaptive Data Access and Policy-driven Transfers) project is to develop and deploy a general-purpose data access framework for scientific collaborations that provides fine-grained and adaptive data transfer management and the use of site and VO policies for resource sharing. This paper presents our initial design and implementation of an adaptive data transfer framework. We also present preliminary performance measurements showing that adaptation and policy improve network performance.**

***Index Terms*—data transfers, adaptation, resource sharing, policy, passive performance monitoring.**

I. Introduction

Large-scale science applications are expected to generate exabytes of data over the next 5 to 10 years. With scientific data collected at unprecedented volumes and rates, the success of large scientific collaborations requires that they provide distributed data access with improved data access latencies and increased reliability to a large user community. To meet these requirements, scientific collaborations are increasingly transferring large datasets over high-speed networks to multiple sites.

The goal of the ADAPT (Adaptive Data Access and Policy-driven Transfers) project is to develop and deploy a general-purpose data access framework for scientific collaborations that provides fine-grained and adaptive data transfer management, passive performance monitoring, and use of site and Virtual Organization (VO) policies for resource sharing. Passive monitoring mechanisms collect information about transfer performance from data movement tools without putting extra loads on shared resources. Transfer management mechanisms select transfer properties based on past performance and available resources between the source and destination, and they adapt those properties when the observed performance changes due to the dynamic load on storage, network and other resources. Finally, policy-driven resource management uses Virtual Organization and site policies regarding replication and resource allocation to balance user requirements for data freshness with the load on resources.

This paper presents our initial design and implementation of an adaptive data transfer framework. We also present preliminary performance measurements that demonstrate that adaptation and policies for resource sharing improve network performance for multi-client experiments.

II. System Overview

Our approach includes three components: the Data Movement Service (DMS), the Passive Measurement Archive (PMA) and the Policy Service (PS). The relationship among these components is shown Figure 1. The Data Movement Service (DMS) performs data transfers and records information about the performance of completed transfers in the Passive Measurement Archive (PMA). Before initiating a new data transfer, the DMS requests advice on parameters for that transfer, such as the recommended number of parallel streams, concurrency or buffer size, from the Policy Service (PS). The PS suggests transfer parameter values that are based on past performance measurements stored in the PMA (if available), available system resources, and site or Virtual Organization policies, which might specify the default parallelism for transfers or the maximum number of streams that may be allocated between each source and destination pair. The PS returns this advice to the DMS, which may modify the recommended transfer parameters before initiating the transfers. For long-running, multi-file transfers, the DMS also periodically requests new advice from the PS and adapts transfer parameters to accommodate changes in resource availability and to optimize throughput. When a user community is large and active, such as Open Science Grid (OSG) [1] or Earth System Grid Federation (ESGF) [2], the Passive Measurement Archive should have enough historical measurements to provide a good basis for transfer advice by the Policy Service.

III. Architecture and Implementation

In this paper, we describe the initial phase of the ADAPT project. In this initial design, we modify a commonly used data transfer client, srm-copy [3], to provide client-side adaptation that is transparent to the user. Storage Resource Manager (SRM) client tools, based on the SRM standard [4], are currently used in production in large-scale scientific grids, including ESG and OSG. One goal of our approach is

to provide a simple transition from current data movement practices in Virtual Organizations like OSG and ESG toward the use of adaptive data movement. In Section V, we describe our plans for the second phase of the ADAPT project, which will focus on providing service-oriented functionality and a more generalized adaptive data transfer framework.

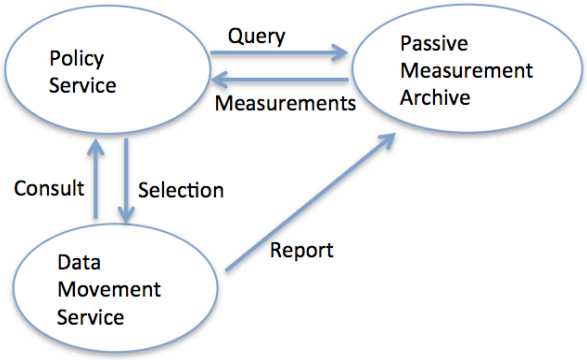

Fig. 1. Interaction between ADAPT components

In the first phase of the ADAPT project, we enhanced the srm-copy data movement client with an Adaptive Data Transfer (ADT) library, a Passive Measurement Archive (PMA), and a Policy library. Our enhancements to the srm-copy client optimize transfer throughput performance adaptively. The policy library provides initial advice on transfer parameters, and the ADT library adapts transfer parameters to increase the performance of long-running, multi-file transfers. Such long-running large transfers are typical of replication and download operations performed in many distributed science collaborations.

Figure 2 illustrates the use case for the first phase of ADAPT, where a data movement client (srm-copy) interacts with a data storage site on a grid running the BeStMan Storage Resource Manager server [4-7]. This scenario is a typical OSG use case for data staging, for example, when one or more large data sets must be staged to an OSG computational cluster before analysis on those datasets begins, or when data must be staged off OSG resources once analysis completes. Next, we describe the components of the ADAPT design in more detail.

A. Data Movement: Adding Adaptation to srm-copy

We modified the standard srm-copy data movement client to adapt transfer parameters during the operation of a transfer using an Adaptive Data Transfer (ADT) library.

Using multiple data transfer streams is a common technique applied in the application layer to increase network bandwidth utilization [8-10]. To achieve high throughput, the number of connections needs to be adjusted according to the capacity of the underlying environment. Our work builds on earlier research on an Adaptive Data Transfer (ADT) algorithm that calculates dynamic transfer parameters such as the concurrency level from a simple throughput prediction model with information from current data transfer operations [11-15]. This is in contrast to predictive sampling proposed in other research [9, 16], and it does not depend on external profilers for active measurements.

The ADT library [11-15] increases the number of streams adaptively and dynamically toward an optimal level. This method also enables adaptation to varying environmental conditions for improved system and network resource utilization. By gradually adjusting the number of streams, the ADT library is able to identify a near optimal level for the number of parallel streams without requiring complex identification of transfer bottlenecks. In this adaptive algorithm, when a change in the performance is detected over the course of a long-running, multi-file transfer, the number of concurrent connections is adjusted accordingly for subsequent transfers, as shown in Figure 3. Adaptive transfer management by adjusting the concurrency level dynamically provides higher throughput, and the ADT algorithm is applied in user space on application level auto-tuning methodologies.

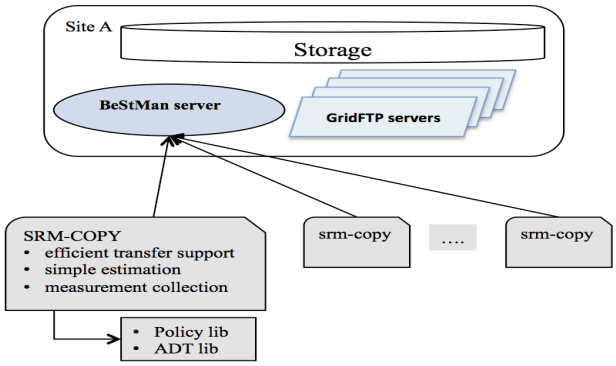

Fig. 2. ADAPT use case

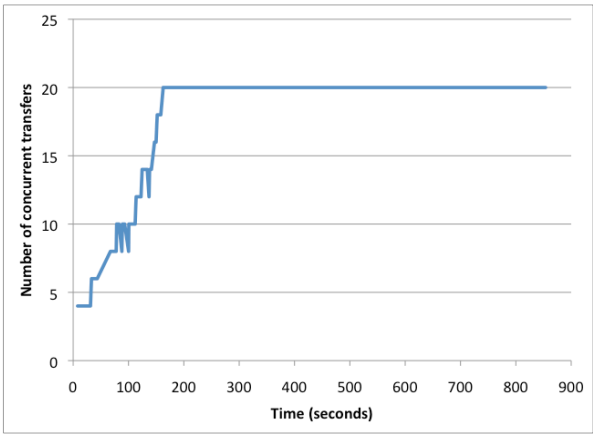

Fig. 3. Gradual adaptation in total number of streams over time

Earlier work on ADT [11-15] addressed two challenges in adaptive data movement: estimating the initial number of streams and determining the interval between transfer parameter adjustment points. Network latency directly affects throughput, and more transfer streams are needed to occupy the available network bandwidth when latency is higher. A challenge in ADT is to estimate the initial number of streams between the source and destination hosts, and a simple model has been studied for the estimation [12]. This estimate will then be adjusted gradually within maximum

resource limits that are determined based on advice from the Policy module.

While we can measure the instant throughput for each transfer, it may not be appropriate to adjust the number of transfer streams after every measurement point. Given the possibility of minor fluctuations in network throughput, a threshold value is needed based on some transfer property such as the transferred data size and time interval before determining changes in the achievable throughput and adjusting the number of concurrent streams. As a future research topic, we plan to experiment the threshold values using policy rules as well as historical measurements stored in the Passive Measurement Archive (PMA). For our experiments presented below, we use a simple metric of adjusting the transfer parameters for a client after five transfers have completed.

Our work extends this earlier work and further explores these challenges within the ADAPT framework.

B. Passive Measurement Archive

The ADAPT architecture is shown in Figure 4. In our system, the srm-copy data movement clients collect performance information on data transfers and record the collected information in the Passive Measurement Archive (PMA). When an srm-copy client requests advice on transfer parameters, the Policy library accesses the PMA to obtain past transfer performance and uses that information in its policy rules.

In the first phase of the ADAPT project, we implemented a Passive Transfer Monitor (PTM) module as a library of the SRM data movement client. Once an srm-copy data transfer completes, the srm-copy client writes performance information about that transfer through the PTM into the Passive Measurement Archive (PMA), which is implemented as a storage archive. This archive is labeled in Figure 4 as the PTM log. The performance information recorded by the srm-copy client includes the transfer rate, total transferred data, and number of transfer streams used.

Later, the Policy module may retrieve information about past transfers from the log file via the PTM.

In the second phase of the project, a perfSONAR-based measurement repository service will be used as the PMA, so that historical information can be archived and queried through a common, standardized interface.

C. The Policy Library

We implemented a Policy library for the srm-copy data movement client that provides advice on transfer parameters and resource allocations for client sessions. This policy work builds on earlier work on data staging, placement and policy [8, 17-22]. Advice generated by the policy engine is based on historical information from the Passive Measurement Archive, system resource availability, and Virtual Organization and site policies that place limits on transfer parameters and overall research usage. For the first phase of the ADAPT project, the Policy logic was implemented as a library module that can optionally be included with the srm-copy client to provide advice on the number of transfer streams and other transfer rate parameters to use for the data transfer request. In the second phase of the ADAPT project, we plan to provide a hosted Policy Service that can be shared by multiple data movement clients.

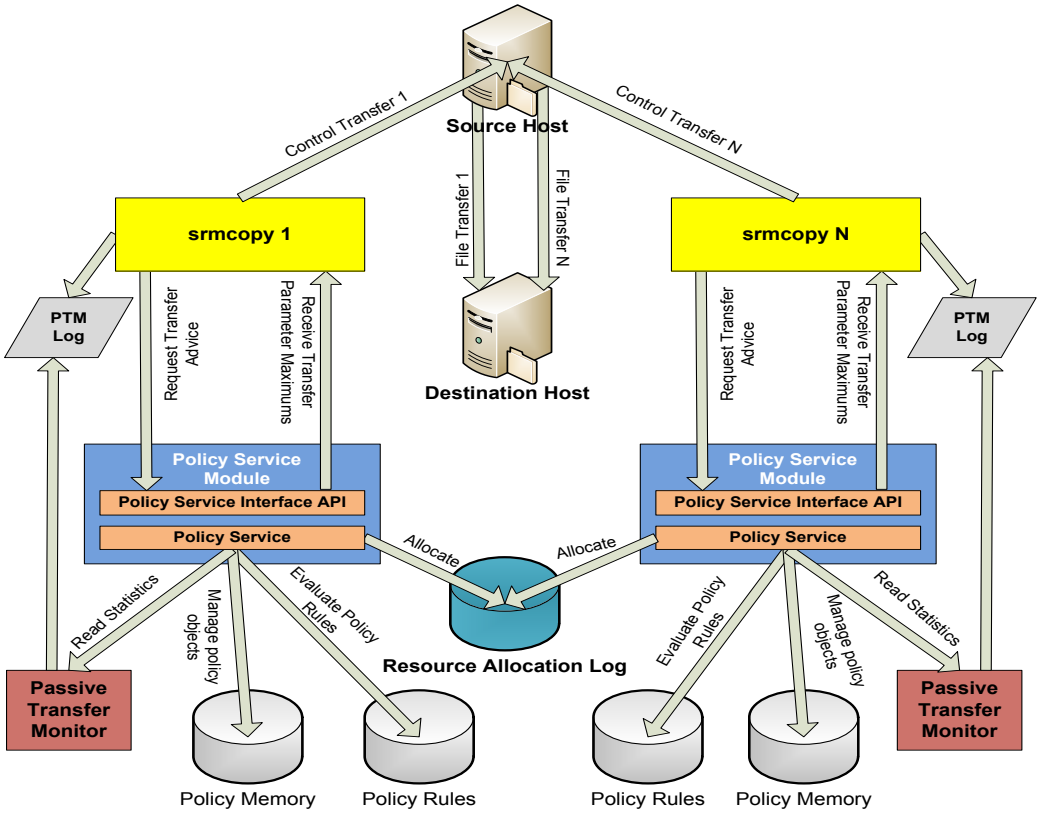

Fig. 4. Architecture of Policy Service

The overall goal of the ADAPT project is to achieve higher throughput for data transfers in a distributed environment. Each data movement client, srm-copy, contains its own Policy Module. Network administrators at the site and Virtual Organization (VO) level define the overall resource limits between pairs of hosts based on their hardware resources (e.g. memory size, I/O rates, and storage system bandwidth) and the network bandwidth between them. These resource limits are included in a configurable policy file that is provided as input to the policy library.

Within the policy library, an open source policy engine (Drools [23]) is used to enforce policy rules. The policy library receives requests from the srm-copy data movement client for advice on transfer parameters, either for new transfers or to refresh advice for ongoing transfers. The policy engine uses the site- and VO-specified policy rules as well as its knowledge of the resources already allocated to ongoing transfers (which are recorded in the shared Resource Allocation Log) and the performance of past transfers (recorded in the Passive Measurement Archive) to recommend parameters to the srm-copy client for the current transfer. After receiving advice from the policy library, the srm-copy client may modify the transfer parameters based on additional logic in the Adaptive Data Transfer library. The srm-copy client then makes another call to the Policy library to inform it of the transfer parameters that were actually used for the transfer. The Policy library records the information about the resources that were actually allocated in the shared Resource Allocation Log.

The Policy Service Module is implemented as a Java library that contains the policy logic. The Policy Service manages policy sessions that contain policy rules and persistent policy memory, manages a Passive Transfer Monitor (PTM) instance, and records resource allocation information in a shared Resource Allocation Log. The Policy Service Interface API is a Java API used to communicate with the Policy Service. The Policy rules specify how transfers should be handled in the system. Policy Memory retains current knowledge of information stored in policy; its state persists across transfer requests.

D. Policy Overview

The Policy library for the ADAPT project is implemented using the Drools open source policy engine [23]. We implemented a greedy allocation algorithm that enforces policies related to controlling the number of streams and the bandwidth allocated between a source and destination pair, with the goal of increasing the efficiency of data transfers. A site or Virtual Organization administrator provides as input to the Policy Service a threshold number of streams and threshold bandwidth that should be allowed between a source and destination pair, as well as a default number of streams and bandwidth for individual transfers.

When a new request comes in for advice on parameters for a transfer between a source and destination site, the policy engine first checks its state in the Resource Allocation Log to determine how many streams or how much bandwidth have already been allocated for ongoing transfers between those sites. If there are sufficient streams or bandwidth remaining below the threshold to satisfy the new request, then the policy engine returns advice to the srm-copy client that the transfer should receive the default number of streams/ bandwidth for the new transfer.

However, if a new transfer allocation will exceed the threshold set by the site or VO administrator for either number of streams or bandwidth allocated, then the advice for the new transfer recommends that the transfer be allocated fewer resources. If the threshold value has not been reached, then the policy service will recommend allocating the remaining number of streams or available bandwidth. If the threshold value has been reached, then the policy engine will recommend allocating a minimal number of resources for the transfer. This avoids starvation of new requests.

The goal of the greedy algorithm is to allocate to all transfers the default number of streams/bandwidth up to the point where the specified threshold is reached. After that point, additional transfers are allowed to proceed with a minimal number of streams/amount of bandwidth to avoid starvation. As earlier transfers complete and free up available streams, those streams may be allocated to other transfers via the adaptation process already described.

E. Policy Implementation

When the Policy Engine receives a transfer from the srm-copy client for evaluation, it applies the policies shown in Table I to each transfer. As described earlier, the policy service uses information about past transfer performance stored in the Passive Measurement Archive and about resources that have already been allocated, which is recorded in the shared Resource Allocation Log.

TABLE I. ADAPT POLICY RULES (GREEDY ALGORITHM)

Insert a new transfer into policy memory: Adds an object corresponding to a new transfer to policy memory.
Set default max streams for new transfer when no Passive Measurement Archive (PMA) record is available for this source and destination pair: Set the max_streams property to the default value.
Set default max bandwidth for new transfer when no PMA record is available for this source and destination pair: Set the max_rate property to the default value.
Set max streams to number of streams recorded in the PMA for the new transfer: When the PMA has measurements from a previous transfer between the source and destination, set the max_streams property to the value stored in the PMA if it exceeds the default value. Otherwise, set max_streams property to the default value.
Set max rate to PMA rate for new transfer: When the PMA has measurements from a previous transfer between the source and destination, set the max_rate property to the value stored in the PMA if it exceeds the default value. Otherwise, set max_rate property to the default value.
Read the resource allocation log: Read the resource allocation log to determine how many transfer instances are utilizing throughput and parallel streams on the sites.
Enforce limit for max_streams_on a transfer using greedy allocation: If the number of advised streams would not exceed the maximum streams threshold between the source and destination, then allocate max_streams. If the

number of advised streams would exceed the maximum streams threshold between the source and destination, then allocate only the number of streams that does not exceed the threshold. If the threshold has been reached or exceeded, allocate the minimum number of streams for the new transfer.
Enforce limit for bandwidth on transfer using greedy allocation: If the amount of advised bandwidth would not exceed the maximum rate threshold between the source and destination, then allocate max_rate. If the amount of advised bandwidth would exceed the maximum rate threshold between the source and destination, then allocate only the amount of bandwidth that does not exceed the threshold. If the threshold has been reached or exceeded, allocate the minimum rate for the new transfer.
Create resource allocation entry for transfer: Create a new transfer record in the resource allocation log recording the max_streams and max_rate properties for the transfer.
Update resource allocation entry for transfer: Update an existing entry in the resource allocation log with the actual transfer parameters used for a transfer: adjusted_streams and adjusted_rate.
Remove resource allocation entry for completed transfer: Once a transfer is complete, remove the resource allocation log record for that transfer to reflect that the resources used in the transfer are available.
Remove resource allocation entry for failed transfer: If a transfer fails, remove its record from the resource allocation log and remove the transfer object from policy memory.
Record completed transfer: Record information about a transfer that completed successfully in policy logs. This information includes rate, streams, source and destination. Remove the transfer object from policy memory.

IV. Evaluation

Next, we present initial performance results for adaptive data transfers and compare the performance of adaptive transfers with unmodified srm-copy data transfer clients. We run these tests between two sites: the Lawrence Berkeley National Laboratory (LBNL) in Berkeley, California, and the Open Science Grid site at the University of Nebraska at Lincoln (UNL).

Our tests are intended to model a common Open Science Grid use case as in Figure 5, in which OSG users who want to run an analysis job on OSG compute resources must first stage data from a remote location to the OSG site. Files used in our data transfer experiments are from an existing data set stored at LBNL.

Our experiments are multi-client tests that show transfers from 6 srm-copy clients running at LBNL that stage data to UNL. Table II shows the number of files and file sizes transferred by each client. These files from an existing data set at LBNL are either 132 MB or 957 MB in size. Multiple clients transferring files of different sizes is a realistic data staging use case.

Figure 6 shows the number of streams allocated to each client over time using the adaptive srm-copy client with the greedy allocation algorithm. In this experiment, the overall threshold for the streams that may be allocated between the source and destination sites was 100. The requested default allocation for any transfer was 80 streams. At the start of the experiment, the srm-copy client for client 1 requests advice on the allocation of streams for its transfer session and receives the requested allocation of 80 streams, since no other resources have been allocated. As the graph shows, client 1 begins transferring data with its minimum allocation of 4 streams, and the adaptive algorithm adjusts over time toward the maximum allocated value of 80 streams. The current allocation is the sum of all concurrent file transfers for that client. When srm-copy client 2 requests advice on its session allocation, it is given an allocation of 20 streams, which is the value remaining under the allowed threshold of 100 streams. Again, client 2 begins its session with the minimum allocation of 4 streams and adjusts over time toward its allowed allocation of 20 streams.

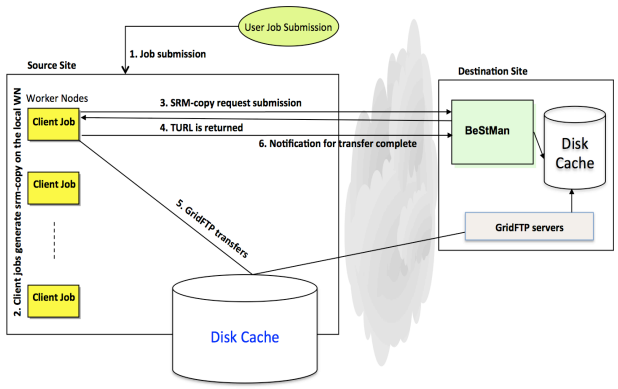

Fig. 5. Test use case that is common in Grid environment

TABLE II. Data transferred by each client

Client	Total data transferred	Total files	File sizes
1	45749 MB	84	45 files of size 957MB, 39 files of size 132MB
2	43571 MB	80	40 files of size 957MB, 40 files of size 132MB
3	21786 MB	40	20 files of size 957MB, 20 files of size 132MB
4	33768 MB	62	31 files of size 957MB, 31 files of size 132MB
5	31589 MB	58	29 files of size 957MB, 29 files of size 132MB
6	33768 MB	62	31 files of size 957MB, 31 files of size 132MB

Clients 3 through 6 also request advice from the policy engine and, since the threshold number of streams has been allocated to the first two clients, they receive the minimum allocation of 4 streams for their sessions. This allows transfers for those clients to proceed and avoids starvation of requests.

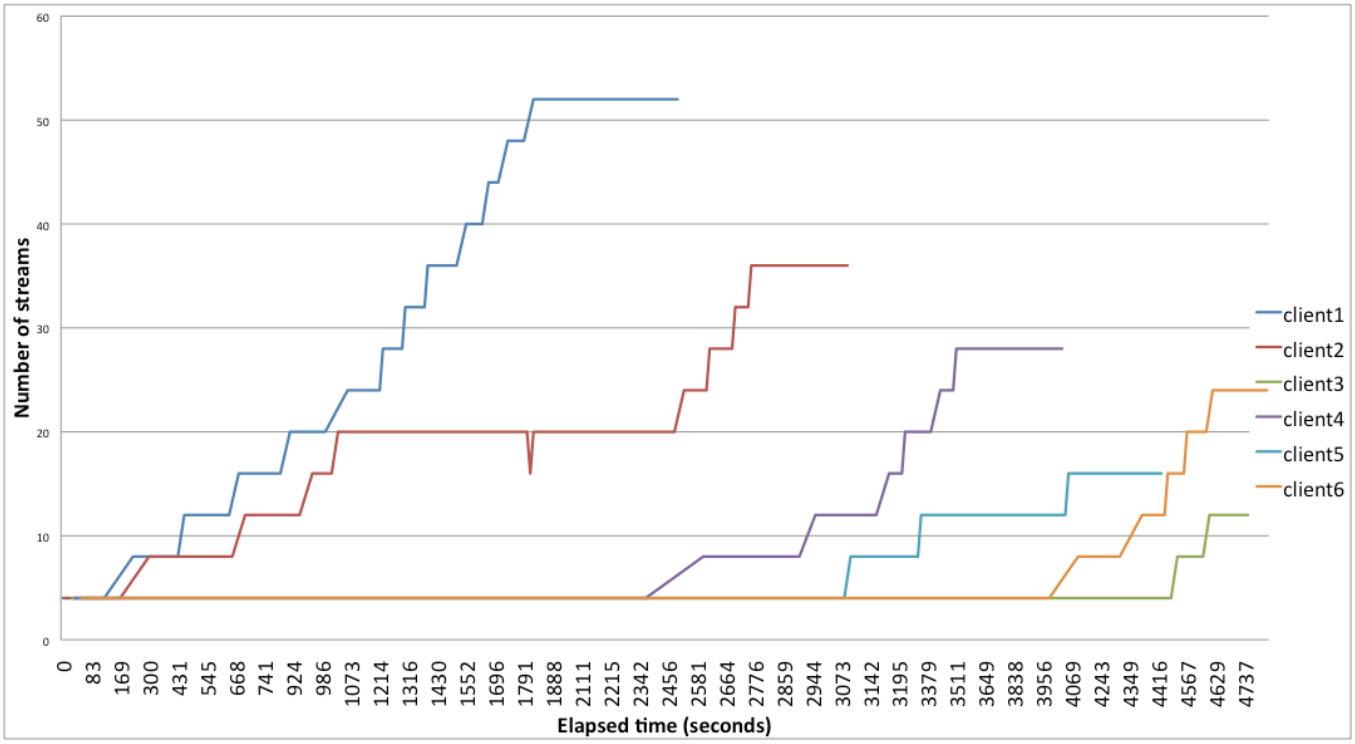

Fig. 6. Total number of streams for each ADAPT-enhanced client when multiple clients are running.

Figure 6 shows the progression of these transfers over time and how adaptation affects the number of streams allocated for each client session. In practice, we do two types of adaptation: one that uses the srm-copy client's adaptation library to adjust the number of streams for client transfers within the client's current maximum stream allocation, and one that periodically consults the policy engine to determine whether new resources have become available that increase the maximum stream allocation for the client's session. In our experiments, the adaptation algorithm is executed every time five transfers from a client complete; the adaptation gradually increases or decreases the use of streams within the allocated maximum to find the stream allocation that gives the highest throughput (which is often achieved by allocating more streams). The adaptation module also periodically consults the policy module for fresh advice on the stream allocation limit for a client session. If the stream allocation limit changes, the adaptation module then adjusts the number of streams for subsequent transfers within this new allocation limit. This allows the adaptation logic to take advantage of resources as they become available, for example, when earlier transfers complete and free up available streams.

Performing adaptation after a fixed number of completed transfers is the reason for the step-wise increase in the number of streams shown in the graph. Client 1 increases its bandwidth toward its maximum allocation of 80 streams (eventually reaching 52 streams), and Client 2 acquires additional streams until it reaches its maximum allocation of 20 streams at around 1000 seconds of elapsed time.

Once the transfers associated with client 1 complete at around 2500 seconds, the resources associated with that client are freed and can be allocated to other clients during adaptation. When the adaptation module requests fresh advice from the policy module, the policy engine applies its policy rules, including checking the Resource Allocation Log to determine whether additional resources have become available. Based on its rules, the policy engine then recommends the allocation of those freed resources to other client sessions.

Thus, after the completion of client 1's transfers, client 2 consults the policy module and acquires a larger stream allocation. The graph shows the subsequent adaptation as the total number of streams for client 2 increases from its original allocation of 20 streams to a total of 36 streams. Similarly, following the completion of client 1's transfers, client 4 adapts to increase its number of allocated streams toward a total of 28 streams. The graph shows that this process continues for the remaining clients, with client 5 increasing its stream allocation after client 2 completes its transfers and frees up its resources. Client 6 increases its stream allocation after client 4 completes its transfers, etc.

Figure 7 shows the throughput achieved by each client in this multi-client adaptation experiment. As would be expected based on Figure 6, the first two clients achieve the highest bandwidth because of their greater allocation of streams. Once each client completes its transfers, the remaining clients achieve higher throughput as they acquire resources that have been freed. Because our policy rules avoid starvation by allocating a minimum of four streams to

each client, all clients in this experiment achieve throughput of at least 3.7 MB/sec between LBNL and UNL.

For comparison, Figure 8 shows the performance of standard srm-copy clients that do not use the adaptation or policy logic that we developed. 6 clients transfer the same files from LBNL to UNL. Figure 8 shows the throughput achieved by each client, where each client requests an allocation of 80 streams at the start of the transfer. The network resources quickly become oversubscribed, and the clients cannot adapt, so the overall throughput goes down significantly. When some of the client transfers complete, the rest of the clients achieve a throughput increase. Overall, the non-ADAPT srm-copy clients took 30% more time than ADAPT-enhanced client to complete the same transfers in our experiments. These early results demonstrate the potential advantages of adaptation and policy for improving transfer performance.

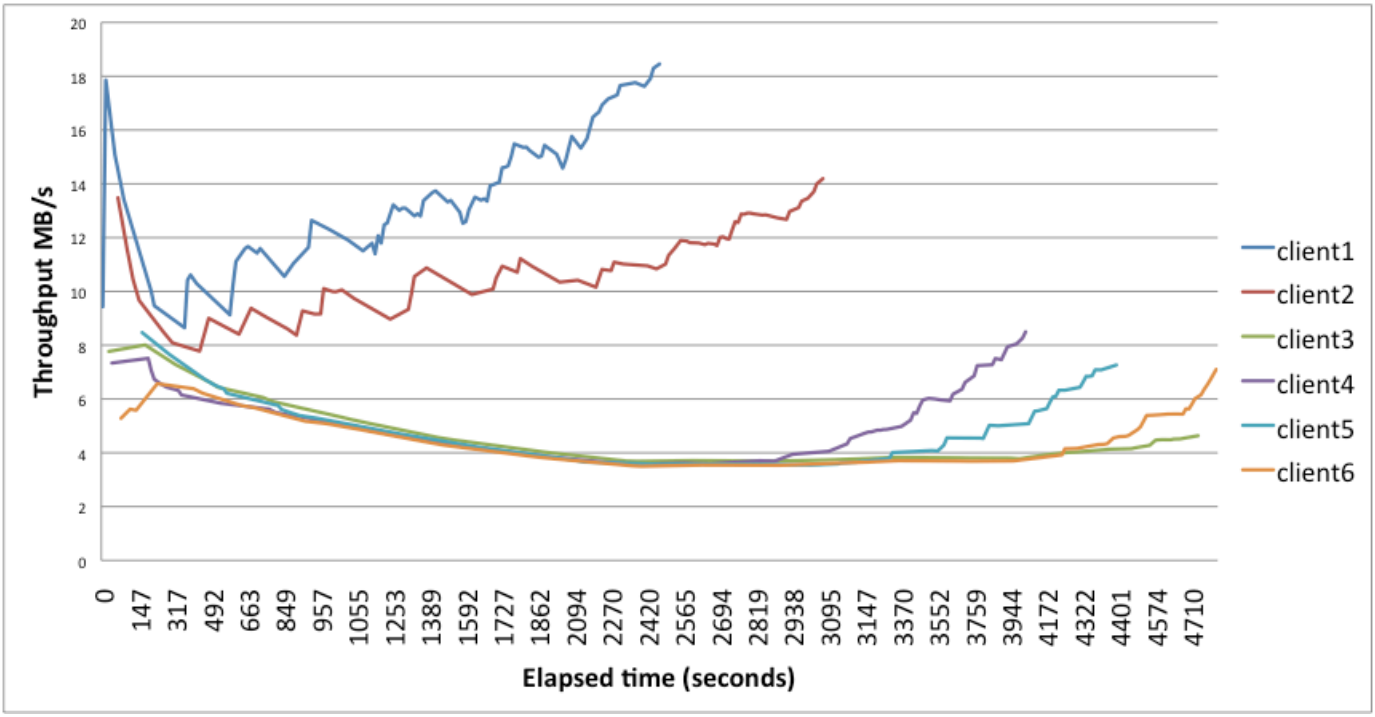

Fig. 7. Throughput performance of each ADAPT-enhanced client when multiple clients are running.

V. Summary and Future Plans

In this paper, we have described the design and implementation of the first phase of data transfer adaptation for the ADAPT project. This phase of the project has focused on modifying a commonly used data transfer client, srm-copy, to provide client-side adaptation that is transparent to the user. The goal of this approach is to provide a simple transition from current data movement practices to adaptive data movement.

Our early performance results show significant benefits in using adaptation and resource allocation policies to improve the performance of multi-file data transfers in wide area environments.

Our plans for the second phase of the project include enhancing the data movement client integrated with ADT as a library module; deploying the policy logic as a Policy Service that can easily be shared by multiple data movement clients; integrating perfSONAR as the Passive Measurement Archive; and conducting extensive performance and scalability tests with multiple data movement clients moving data among multiple storage sites. We also plan to integrate adaptation and policy logic into other data movement tools, such as the Bulk Data Mover (BDM) and Data Mover Lite (DML) tools.

We will also conduct research to explore richer policies for managing transfer resources and supporting the needs of site administrators and Virtual Organizations. We will conduct an experimental evaluation to determine which of these policies are most effective in reducing transfer times and increasing overall throughput.

Finally, we plan to work more closely with application communities to assist them in deploying and using the transfer adaptation software to improve the performance of distributed scientific collaborations that replicate and download large amounts of data.

VI. Acknowledgments

This work was supported in part by the National Science Foundation Office of Cyberinfrastructure under award 1127101 (USC/ISI) and award 1127039 (LBNL).

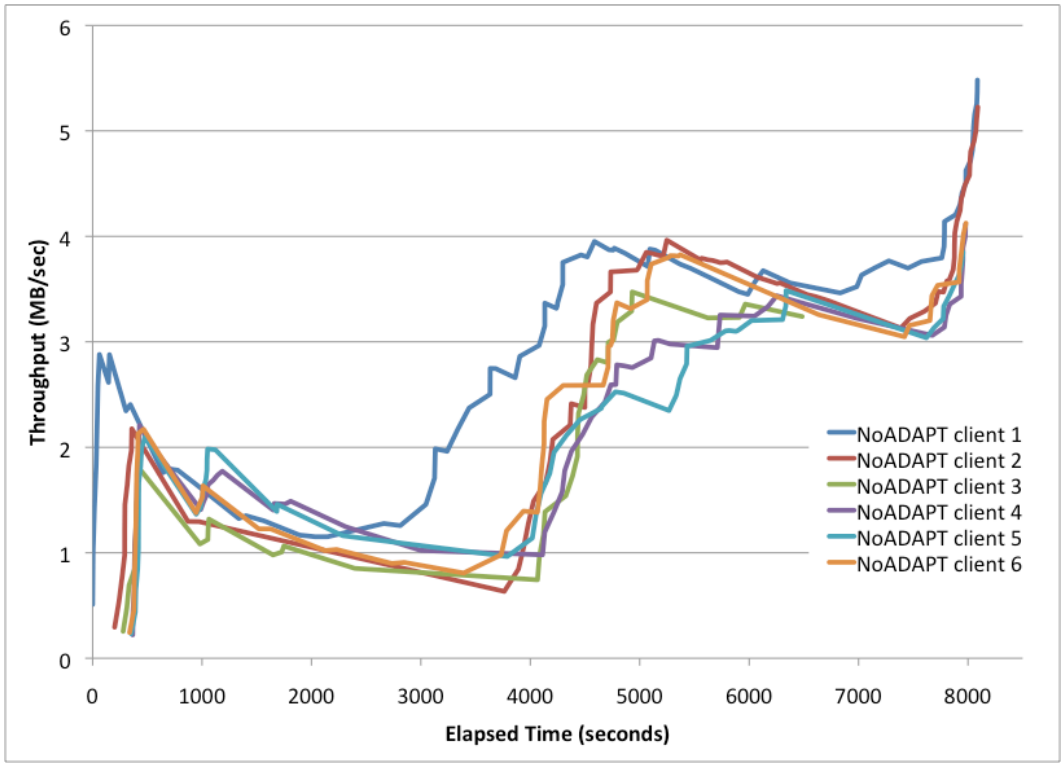

Fig. 8. Throughput performance of each Non-ADAPT client when multiple clients are running.

REFERENCES

[1] . *Open Science Grid*. Available: http://www.opensciencegrid.org/

[2] . *The Earth System Grid (ESG)*. Available: http://www.earthsystemsgrid.org

[3] . *SRM Clients*. Available: http://sdm.lbl.gov/srmclients/

[4] A. Sim, A. Shoshani, et al., "The Storage Resource Manager Interface Specification Version 2.2," in *GFD.129*, ed: Open Grid Forum, 2008.

[5] . *Berkeley Storage Manager (BeStMan)*.

[6] L. Abadie, P. Badino, J. P. Baud, E. Corso, M. Crawford, S. De Witt, *et al.*, "Storage resource managers: Recent international experience on requirements and multiple co-operating implementations," in *23rd IEEE Symposium on Mass Storage Systems and Technologies (MSS '07)*, 2007, pp. 47-59.

[7] A. Shoshani, A. Sim, and J. Gu, "Storage Resource Managers: Essential Components for the Grid," in *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds., ed: Kluwer Academic Publishers, 2003.

[8] M. Balman and T. Kosar, "Data Scheduling for Large Scale Distributed Applications," in *the 9th International Conference on Enterprise Information Systems Doctoral Symposium (DCEIS 2007)*, 2007.

[9] T. Dunigan, M. Mathis, and B. Tierney, "A TCP tuning daemon," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing (Supercomputing '02)*, 2002, pp. 1-16.

[10] T. Ito, H. Ohsaki, and M. Imase, "On parameter tuning of data transfer protocol gridftp in wide-area grid computing," in *Second International Workshop on Networks for Grid Applications (GridNets)*, 2005.

[11] M. Balman and T. Kosar, "Dynamic Adaptation of Parallelism Level in Data Transfer Scheduling," in *International Conference on Complex, Intelligent and Software Intensive Systems 2009 (CISIS '09)*, 2009, pp. 872 - 877

[12] A. Sim, M. Balman, D. Williams, A. Shoshani, and V. Natarajan, "Adaptive Transfer Adjustment in Efficient Bulk Data Transfer Management for Climate Datasets," in *the 22nd International Conference on Parallel and Distributed Computing and Systems (PDCS 2010)*, 2010.

[13] M. Balman, *Data Placement in Distributed Systems: Failure Awareness and Dynamic Adaptation in Data Scheduling*: VDM Verlag, 2009.

[14] M. Balman. (2011). *Dynamic Adaptation for High-Performance Data Transfers*. Available: csc.lsu.edu/~balman/pdfs/DynamicAdaptation-Balman.pdf

[15] M. Balman, "Data Transfer Scheduling with Advance Reservation and Provisioning," Louisiana State University, 2010.

[16] E. Yildirim, M. Balman, and T. Kosar, "Dynamically Tuning Level of Parallelism in Wide Area Data Transfers," in *2008 international workshop on Data-aware distributed computing (DADC '08)*, 2008, pp. 39-48.

[17] A. Chervenak, E. Deelman, M. Livny, M. Su, R. Schuler, S. Bharathi, G. Mehta, K. Vahi, "Data Placement for Scientific Applications in Distributed Environments," in *8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, Austin, Texas, 2007.

[18] S. Bharathi and A. Chervenak, "Data Staging Strategies and Their Impact on the Execution of Scientific Workflows," presented at the Proceedings of the Second International Workshop on Data-Aware Distributed Computing (DADC), in association with High Performance Distributed Computing (HPDS) Conference, Garching, Germany, 2009.

[19] M. Amer, W. Chen, and A. L. Chervenak, "Improving Scientific Workflow Performance using Policy Based Data Placement," in *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2012)*, Chapel Hill, North Carolina USA, 2012.

[20] J. Feng, L. Cui, G. Wasson, and M. Humphrey, "Policy-Directed Data Movement in Grids," in *12th International Conference on Parallel and Distributed Systems (ICPADS 2006)*, 2006, pp. 12-15.

[21] T. Kosar and M. Livny, "A framework for reliable and efficient data placement in distributed computing systems," *J. of Parallel and Distributed Computing,* vol. 65, pp. 1146-1157, 2005.

[22] T. Kosar and M. Livny, "Stork: making data placement a first class citizen in the grid," in *24th International Conference on Distributed Computing Systems*, 2004, pp. 342-349.

[23] Drools project, "Drools, http://www.jboss.org/drools/," ed.