Adaptation and Policy-Based Resource Allocation for Efficient Bulk Data Transfers in High Performance Computing Environments

Ann L. Chervenak[1], Alex Sim[2], Junmin Gu[2], Robert E. Schuler[1], Nandan Hirpathak[1]

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

[2]Lawrence Berkeley National Laboratory
Scientific Data Management Research Group
Berkeley, CA, USA
{asim, jgu}@lbl.gov

***Abstract*—Many science applications increasingly make use of data-intensive methods that require bulk data movement such as staging of large datasets in preparation for analysis on shared computational resources, remote access to large data sets, and data dissemination. Over the next 5 to 10 years, these datasets are projected to grow to exabytes of data, and continued scientific progress will depend on efficient methods for data movement between high performance computing centers. We study two techniques that improve the use of available resources for large, long-running, multi-file transfers. First, we show the effect of adaptation of transfer parameters for multi-file transfers, where the adaptation is based on recent performance. Second, we use Virtual Organization and site policies to influence the allocation of resources such as available transfer streams to clients. We show that these techniques improve completion times for large multi-file data transfers by approximately 20% over resource constrained infrastructure.**

Keywords—bulk data transfer; performance-based adaptation; policy-based resource allocation; resource constrained; throughput

I. INTRODUCTION

Scientific activities are generating data at unprecedented volumes and rates. Over the next 5 to 10 years, applications in many domains are expected to generate exabytes of data. Many of these science domains include unique experimental facilities that generate large data sets, such as the Large Hadron Collider (LHC) [1], the Large Synoptic Survey Telescope (LSST) [2], and the Laser Interferometer Gravitational Wave Observatory (LIGO) [3]. Other domains generate large amounts of simulation data that must be shared, compared and analyzed. For example, the climate modeling community has published multiple petabytes of data produced by the Coupled Model Intercomparison Project (CMIP) through the Earth System Grid Federation (ESGF) [4, 5]; based on current growth rates, the climate community estimates that its model data will exceed 100 exabytes by 2020 [6].

For science to progress, the large and steadily increasing amount of data originating from instruments and simulations must be efficiently accessed by scientists and disseminated throughout a scientific collaboration or Virtual Organization (VO) [7] for analysis, simulation, and storage. While many large collaborations have access to high-performance networks, the growth in data volumes and demand for data movement increasingly stress networks and file transfer services. In particular, spikes in bulk data movement requests, when uncoordinated, can easily overwhelm file transfer resources, causing them to become sluggish and unresponsive, and may ultimately lead to bulk file transfer failures. A key unmet challenge for high-performance, data-instensive cyberinfrastructure is to coordinate bulk data movement to facilitate efficient transfer performance across a scientific collaboration.

Among large-scale scientific collaborations, key bulk data access operations include:

- *staging of large data sets* to computational resources before execution of a large analysis, as is typical for scientific workflows running on the Open Science Grid (OSG) [8] or XSEDE [9] computing environments;
- support for *remote data access* to large data sets stored at archive sites, for example, when a workflow running on a commercial cloud infrastructure remotely accesses data stored at science archives at NASA or the ESGF;
- *data replication and dissemination operations*, which copy data generated at an instrument or simulation site to sites around the world based on a VO's data dissemination policies (e.g., distributing data sets from the LHC to Tier 1 and Tier 2 sites in the LHC community [10]); and
- *bulk data access requests* by scientists within the VO across shared resources.

Our goal is to improve transfer throughput and latency for bulk data transfer operations. We study two techniques to improve the performance of long running, multi-file data transfers in resource constrained environments: *policy-based allocation of data transfer resources* based on Virtual Organization (VO) and site level policies and *client-side adaptation of transfer parameters* based on recent transfer performance. Using a testbed that models wide-area, bulk data staging over high-performance networks to a supercomputing facility, we measure the performance of these techniques separately and in combination and demonstrate that they provide significant throughput and completion time improvements when there is contention for resources.

This paper extends our previous work [11, 12] with the following contributions:

- We describe our algorithms for policy-based resource allocation and adaptation of data transfer parameters in our revised design and implementation (Section II),
- We present experimental results that show the detailed operation of transfer client adaptation and policy-based resource allocation for large, multi-file wide area transfers for slow and fast rates of adaptation (Section III), and
- We show approximately 20% improvement in overall transfer completion time with our techniques (Section III).

Finally, we review related work in Section IV and conclude in Section V.

II. System Design and Implementation

Fig. 1 shows the system design for our transfer client adaptation and policy-based resource allocation techniques. The Adaptive Data Transfer (ADT) Library includes client-side functionality for requesting, updating, and releasing resource allocations. The ADT Library also tracks the performance of transfers and makes dynamic adjustments to increase or decrease the number of transfer streams in use. The Transfer Client (a modified SRM-Copy client) integrates with the ADT Library and is responsible for managing and controlling active data transfers. These client side components are distributed across a cluster of nodes, and each client adapts independently.

On the server side, a Policy Service (PS) handles multiple resource allocation requests from the client-side ADT Library. By default, the Policy Service (PS) is deployed with a Greedy policy specification (Fig. 2). The PS may also be configured to work with other user-defined policy specifications. Users may specify their policies in the form of a python module that implements the policy interface for adding, updating, and removing resource allocations. We anticipate that the VO administrator would deploy the default policy or modify it to meet specific goals of that VO environment. For a bulk transfer operation, the client will first request an initial resource allocation and then adapt its parameters within the bounds of the allocation (i.e., increase or decrease the streams in use). Periodically, the client will request an updated resource allocation from the PS. The client releases its resources once a bulk transfer has completed.

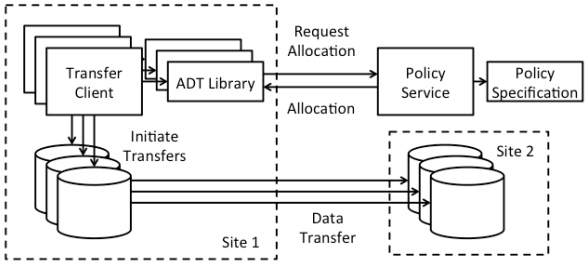

Fig. 1. Service interactions for bulk remote data access use cases.

A. Resource Allocation Algorithm

Pseudocode for the resource allocation algorithm implemented by the Greedy policy specification is shown in Fig. 2. The algorithm depends on a few parameters, shown in Table I, which an administrator may specify in order to tune the algorithm to their requirements. The pseudocode illustrates the primary functionality of the algorithm, which is to provision resources either for a new or *initial* allocation or to *update* an existing allocation. For brevity, we do not illustrate the resource allocation release, which simply serves to return resources to the pool of available resources without making adjustments to other allocations.

The Provision function is passed a resource request parameter t, a structure indicating the source and destination of the transfer (e.g., either a hostname or URL) and the currently allocated streams (streams[t]). For an initial transfer request, streams[t] = 0, whereas in the case of an update on an existing allocation, streams[t] > 0. The algorithm must first determine how many streams are currently allocated between source[t] and destination[t] (line 2), because stream allocations are managed between endpoints and are limited by the policy specification parameter, s_{pmax}.

Require: s_i: initial streams allocation specified by policy; s_u: update increment streams allocation specified by policy; s_{pmax}: maximum streams allowed between endpoints specified by policy; s_{cmax}: maximum streams allowed for a single client, specified by policy.

procedure PROVISION(t)

01: $t \leftarrow$ transfer resource request with (source[t], dest[t]) and steams[t])
02: $s_a \leftarrow$ allocated streams between (source[t], dest[t])
03: $s_v \leftarrow \min(s_{cmax} - \text{streams}[t], s_{pmax} - s_a)$ // Available streams
04: **if** $s_v = 0$ **then**
05: // No available streams for transfer request
06: **return** t
07: **else if** streams[t] = 0 **and** $s_v > s_i$ **then**
08: // Enough streams for *initial* allocation
09: streams[t] $\leftarrow s_i$
10: $s_a \leftarrow s_a + s_i$ // Update total allocated streams
11: **else if** streams[t] > 0 **and** $s_v > s_u$ **then**
12: // Enough streams for *update* allocation
13: streams[t] $\leftarrow$ streams[t] + s_u
14: $s_a \leftarrow s_a + s_u$ // Update total allocated streams
15: **else**
16: // Allocate remaining available streams to *initial* or *update* request
17: streams[t] $\leftarrow$ streams[t] + s_v
18: $s_a \leftarrow s_a + s_v$ // Update total allocated streams
19: **end if**
20: **return** t

end procedure

Fig. 2. Resource allocation algorithm used by the PS to implement Greedy Policy for an initial or update allocation request by a client.

TABLE I. Greedy Policy Parameters

Greedy Policy Parameter	*Definition*
Maximum total streams for source/destination pair, s_{pmax}	Maximum concurrent streams active between a pair of source/destination sites.
Maximum streams per client, s_{cmax}	Maximum allocation to a single client from the Policy Service.
Initial stream allocation, s_i	On a new client request, the Policy Service attempts to allocate this many streams (subject to resource availability).
Update increment stream allocation, s_u	On an update request, the Policy Service attempts to increment the allocation by this many streams (subject to availability)

The algorithm then computes the maximum available streams for the allocation request (line 3), which is the minimum of two values: (1) the remaining streams that may be allocated between the source and destination, which is determined by the difference between s_{pmax} and the current allocation for that source/destination pair, and (2) the number of streams that may be allocated to this client, specified by the difference between the policy maximum per client (s_{cmax}) and the streams[t] already allocated to the client. If no streams are available to satisfy the request, the algorithm terminates (line 6).

If the current request is an initial request and the available streams exceed the policy parameter for initial stream allocations (s_i) (line 7), then the algorithm sets streams[t] to s_i. If the current request is an update request and the available streams exceed the policy parameter for update allocation increment (s_u) (line 11), the algorithm increments streams[t] by s_u. If the previous conditions were unmet (line 15), then the available streams are below the policy values s_i for initial requests or s_u for update requests. In that case, the algorithm allocates the remaining stream resources to the request, whether it is an initial or update request. In each case, the algorithm updates the state of the currently allocated requests between the pair of source and destintation endpoints. Finally, it terminates by returning the allocated request to the client.

B. Client Transfer Adaptation Algorithm

Pseudocode for the client side transfer adaptation is shown in Fig. 3. The AdaptTransferClient and Adapt functions depend on several client policy parameters shown in Table II. The AdaptTransferClient is passed a queue Q of files to be transferred between a pair of source and destination endpoints and several policy parameters, including the initial concurrency for a client transfer c, the increment/decrement value Δ that specifies how much concurrency is increased or decreased for a client adaptation, the adaptation delay parameter d that specifies how frequently adaptation occurs (after d transfers complete), and the parallelism or number of data streams p used per transfer.

The function begins by initializing a new transfer request t between the source and destination of Q and calling the Provision function described in the last section to request a resource allocation for t (line 2). The stream allocation provided by the PS is converted to a concurrency c_{alloc} and further limited to the initial concurrency c_i set by the client configuration (line 4).

The function then begins a loop that continues while the transfer queue Q is not empty and adapts every time d transfers complete. If adaptation is due (line 7), the client calls Provision again to acquire an updated resource allocation c_{alloc} (line 9). The client calls the Adapt function with the parameters c_{alloc},, current concurrency c and the increment/decrement parameter Δ. If adaptation is not due or after the Adapt call completes (line 13), the client takes c transfers from the queue, performs the transfers and updates the counter as they complete.

The Adapt function compares the current transfer rate r_{rate} between the source and destination with the last measured rate r_{last}. If the difference exceeds a specified threshold T (line 24), the Adapt function changes the concurrency for subsequent transfers. The function reduces the concurrency by Δ if the transfer rate has decreased by more than T and increases the concurrency by Δ if the transfer rate has increased by more than T. The new concurrency must be non-negative (line 26) and may not exceed the maximum allocation c_{alloc} or the client maximum concurrency c_{max} (line 28).

```
Require: Q: queue of files to be transferred between source and dest.; c_i: initial client concurrency; Δ: adaptation increment/decrement delta; d: adaptation delay; p: parallel streams per file transfer.
 procedure AdaptTransferClient(Q, c, Δ, d, p)
01: t ← initialize a transfer request between (source, dest) of Q
02: Provision(t)          // request initial allocation from Policy Service
03: c_alloc ← floor(streams[t] / p)   // convert streams to concurrency
04: c ← min(c_i, c_alloc)   // limit concurrency parameter, if necessary
05: k ← d                   // set counter for next adaptation
06: while Q not empty do
07:    if k ≤ 0 then        // due for client adaptation
08:       k ← d                      // reset counter
09:       Provision(t)               // request updated allocation from PS
10:       c_alloc ← floor(streams[t] / p)  // convert streams to concurrency
11:       c ← Adapt(c, Δ, c_alloc)   // adapt concurrency up or down
12:    end if
13:    F ← pop at most c transfer jobs from Q
14:    // …perform F transfers concurrently, wait for completion…
15:    k ← k – c           // decrement transfer counter
16: end while
 end procedure
 procedure Adapt(c, Δ, c_alloc)
20: T ← user specified transfer rate adaptation threshold
21: r_last ← state of last recorded transfer rate  // between source-destination
22: r_rate ← test of current transfer rate         // between source-destination
23: r_delta ← r_rate - r_last
24: if abs(r_delta) > T then          //  change exceeds threshold
25:    if r_delta < 0 then
26:       c ← max(0, c – Δ)          // decrease concurrency
27:    else
28:       c ← min(c + Δ, c_alloc, c_max)          // increase concurrency
29:    end if
30: end if
31: return c
end procedure
```

Fig. 3. Adaptation algorithm used by Adaptive Transfer Client. The AdaptTransferClient procedure processes a batch of transfer jobs and uses the Adapt procedure to adjust concurrency up or down.

TABLE II. ADAPTIVE TRANSFER CLIENT PARAMETERS

Adaptive Transfer Client Parameter	***Definition***
Initial concurrency, c_i	Number of active transfers initiated by a client when it begins transferring data.
Maximum concurrency, c_{max}	Maximum number of active file transfers by a client; this value may be reached by adaptation.
Parallelism, p	Number of parallel streams per file transfer
Adaptation delay time, d	How often the client requests an updated resource allocation from the PS; expressed as number of completed transfers before adaptation occurs.
Adaptation increment/ decrement, Δ	How much the concurrency level increases/ decreases when the client adapts up or down within its resource allocation.
Threshold, T	Difference between current and past performance that triggers adaptation of concurrency level.

C. Implementation

The Policy Service implements a RESTful Web service in the Python programming language on the Web.py framework (webpy.org) deployable in the lightweight CherryPy embedded

HTTP server (www.cherrypy.org). The open source implementation of the PS is available at: http://github.com/robes/adapt-policy-service.

The Adaptive Data Transfer Client is a standalone, command-line client implemented in the Java programming language. It extends the conventional SRM-Copy command-line client (sdm.lbl.gov/srmclients/) with the adaptive data transfer (ADT) library. ADT library includes the implementation of the adjustment decisions on the transfer parameters based on the observed throughput performance differences. The open source Adaptive Data Transfer Client may be found at: https://codeforge.lbl.gov/projects/adapt/.

III. EVALUATION

We present our evaluation results in two parts. In the first part of our evaluation, we illustrate the operation and performance of client side adaptation and policy-based resource allocation techniques on long-running data transfers over a relatively high-performance network between the Parallel Distributed Systems Facility at the National Energy Research Scientific Computing Center (NERSC) in Oakland, CA and the University of Nebraska at Lincoln (UNL). This first set of results show the tradeoffs and capabilities of adaptive transfers and policy-based resource allocation. In the second part of the evaluation, we use a testbed with constrained resources to model bulk data movement between HPC facilities with contention for available resources. For these experiments, a client node at National Institute of Supercomputing and Networking (NISN) in Daejon, Korea transferred data to NERSC in Oakland, CA. These experiments show the advantage of our techniques in realistic, resource constrained environments. All experiments transfer a 260 GByte data set made up of 488 files.

A. Evaluation for Experimental Testbed 1: NERSC to UNL over High Bandwidth Network

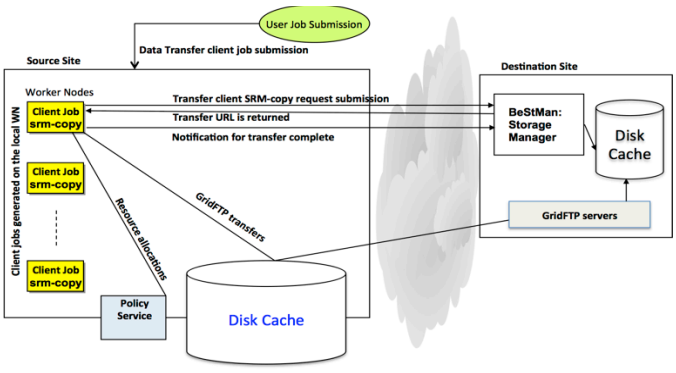

Fig. 4. Experimental testbed setup

We ran experiments to investigate the tradeoffs of a range of stream allocation policies and faster vs. slower client-side transfer parameter adaptation. We transferred the 260 Gbyte data set from NERSC to UNL. Our experiments used 8 job submission nodes at the source site at NERSC, each of which runs SL6 with General Parallel File System (GPFS) backend storage. Both NERSC and UNL have 10 Gbps connections to the wide area network, which crosses the ESnet and Internet2 domains. The network and resources at both ends are shared with other traffic, which causes performance variations in our results. Fig. 4 shows this setup. In our study, we assume that the main constrained resource is the shared network; however, in some cases, the end systems could be the performance bottleneck. Our algorithm addresses end system performance issues by detecting poor transfer performance and adjusting transfer parameters, which in turn adjusts end system resource usage.

Table III shows common parameters for the following experiments that use adaptation or policy-based allocation.

TABLE III. COMMON PARAMETERS USED FOR ADAPTIVE EXPERIMENTS

Common Parameters for all Adaptive Experiments	***Value***
Maximum total streams between source/destination	128
Number of clients	8
Maximum streams per client	32
Parallel streams per file	4
Adaptation increment/decrement	1 concurrency (4 streams)

1) Slow Client Side Adaptation

First, we isolate the effect of slow client side adaptation. Parameters for this experiment are shown in Table IV. In this experiment, the PS has no role beyond its initial allocation to each client. All adaptation takes place on the client side.

TABLE IV. PARAMETERS USED FOR SLOW CLIENT SIDE ADAPTATION

Client Parameters	***Value***
Initial concurrency	1
Maximum concurrency	8
Adaptation delay time (update after how many transfers)	4
Policy Service Parameters	***Value***
Initial stream allocation	32
Update allocation increment	N/A

Fig. 5 shows the performance for one of the three runs for this experiment. This figure shows the number of streams being used by each client on the vertical axis, with the horizontal axis showing elapsed time. Based on the parameters in Table IV, the first four clients that consulted the Policy Service were allocated 32 streams out of the 128 total streams available between the source and destination; the remaining 4 clients had to wait until one or more of those clients completed their transfers and released streams for the remaining clients.

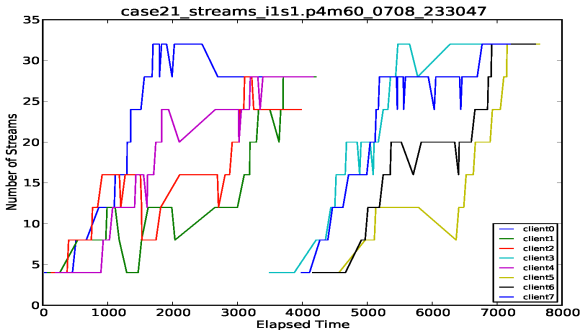

Fig. 5. Number of streams used for slow client side adaptation

Within the 32 allocated streams, each client slowly adapts the concurrency of its transfers, beginning with one transfer that uses 4 parallel streams, and increasing to a maximum of 8 concurrent transfers that each use 4 streams, or a maximum of

32 streams per client. Each client is configured to adapt its concurrency after 4 transfers complete; it may then increase or decrease its concurrency by one transfer (4 streams) based on recent performance. Fig. 5 shows each client slowly adapting its concurrency up or down, with some clients eventually reaching the maximum of 8 transfers (32 streams). Once a client completes its transfers, the PS frees up the client's stream allocation and allocates those 32 streams to one of the waiting clients, which then performs its transfers, adapting in the same manner.

We ran these experiments three times on different days and times of the day. We observed a range of performance based on the load on the infrastructure, as expected when using shared infrastructure at the source and destination sites and shared networking betweeen sites. The experiment completion times were 127 minutes at 11:31pm on 7/8/13, 141 minutes at 2:59am on 7/9/13 and 159 minutes at 11:00am on 7/11/13.

2) Fast Client Side Adaptation

In the next set of experiments, we measured faster client side adaptation, where the client increases or decreases its concurrency by one transfer after every 2 completed transfers. In this scenario, the Policy Service again allocates 32 streams per client when it first gets a request for a client allocation, and then it has no further role in the adaptation. The experimental parameters are summarized in Table V.

TABLE V. PARAMETERS USED FOR FAST CLIENT SIDE ADAPTATION

Client Parameters	***Value***
Initial concurrency	4
Maximum concurrency	8
Adaptation delay time (update after how many transfers)	2
Policy Service Parameters	***Value***
Initial stream allocation	32
Update allocation increment	N/A

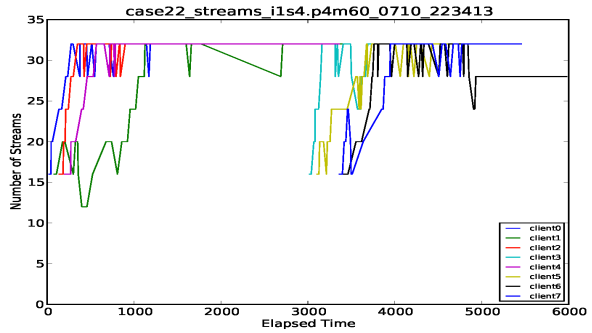

Fig. 6. Number of streams used for fast client side adaptation

Fig. 6 shows the streams used during one run of these experiments. The first four clients receive an allocation of 32 streams from the PS and begin transferring data with a concurrency of 4 transfers (16 streams). The figure shows that each client then quickly adapts its concurrency up or down by 1 transfer (4 streams) each time two transfers complete. The result of this fast adapation is that the four clients quickly increase their concurrency to utilize the allocated maximum of 32 streams per client. Note that each adaptive transfer client sometimes reduces its concurrency based on recent performance, utilizing fewer streams when performance drops. The adaptive transfer client thus avoids overprovisioning constrained resources.

We ran this experiment three times, and the completion times were 100 minutes at 11:34pm on 7/10, 88 minutes at 1:16am on 7/11/13 and 90 minutes at 8:32am on 7/11/13.

Because of the large variations in load on our shared infrastructure, it is challenging to do direct comparisons of experiments (e.g., slow vs. fast client adaptation). We limit our comparative conclusions to Section III.B and focus in this section on the tradeoffs of adaptation and allocation.

3) Policy Service Resource Allocation: Slow Increases

Next, we isolated the effect of the Policy Service (PS), which provides an allocation of streams to each client. In this experiment and the next, the data transfer client does no performance-based adaptation of the number of streams. After it sends an initial or update request for an allocation to the PS, the transfer client simply sets its concurrency level based on the allocation it receives.

The current implementation of the PS only increases the allocation to each client if additional resources are available; it does not decrease the allocation, but instead waits for the data transfer client to release streams if they are no longer needed. In future work, we will modify the PS to decrease allocations based on transfer performance or on VO policies and to handle exceptional situations such as non-responsive clients.

Experimental parameters for slow increases in resource allocation by the PS are summarized in Table VI. Fig. 7 shows the streams used by clients for this experiment. The PS initially allocates 4 streams to each client (or concurrency = 1). A client requests an updated allocation after 4 transfers complete; the PS then allocates 4 additional streams if they are available. When a client receives an allocation from the PS, it initiates transfers at the maximum concurrency allowed by that allocation (up to a concurrency of 8 for this experiment). The clients do not adapt based on performance.

TABLE VI. PARMETERS USED FOR SLOW INCREASES IN PS ALLOCATION

Client Parameters	***Value***
Initial concurrency	1
Maximum concurrency	8
Adaptation delay time (update after how many transfers)	4
Policy Service Parameters	***Value***
Initial stream allocation	4
Update allocation increment	4

Fig. 7 shows that each client receives an initial allocation of 4 streams from the Policy Service. Several of the clients quickly request additional PS allocations until they reach the maximum concurrency of 8 (or 32 streams). Since the overall maximum number of streams allowed between the source and destination is 128 (from Table III), several clients must wait until those first clients finish their transfers and release their allocated resources before the later clients receive increased stream allocations (e.g., the dark blue, purple, black and yellow lines in Fig. 7.)

The three runs of these experiments had completion times of 73 minutes at 4:23am on 7/20/13, 83 minutes at 5:54am on 7/22/13, and 85 minutes at 8:38am on 7/22/13.

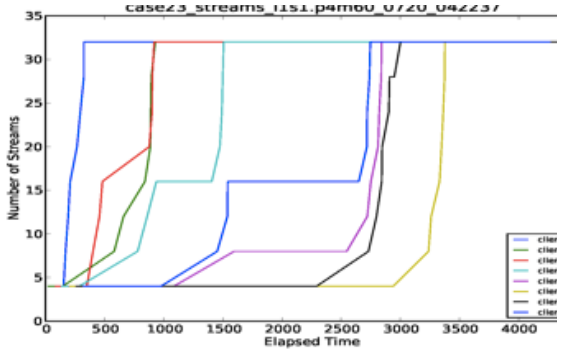

Fig. 7. Stream allocation for slow increases in policy service allocation

4) Policy Service Resource Allocation: Fast Increases

In the next experiment, we again isolate the effect of increasing allocations by the PS, this time using faster increases in those allocations. When a new request arrives from a data transfer client to the PS, the PS allocates 16 streams (concurrency level of 4) to the client. Each time a client completes two transfers, it requests an updated allocation from the PS. When additional resources are available, the PS provides 4 additional streams, allowing the client to increase its concurrency by 1. As in the last experiment, the transfer client does no performance-based adaptation. It sets its concurrency level to use the allocated streams provided by the PS. These parameters are summarized in Table VII.

TABLE VII. PARMETERS USED FOR FAST INCREASES IN PS ALLOCATION

Client Parameters	***Value***
Initial concurrency	4
Maximum concurrency	8
Adaptation delay time (update after how many transfers)	2
Policy Service Parameters	***Value***
Initial stream allocation	16
Update allocation increment (streams)	4

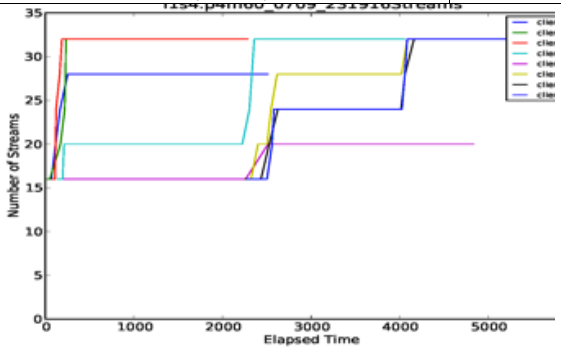

Fig. 8. Stream allocation for fast increases in policy service allocation

Fig. 8 shows the stream usage for one run of this experiment. Initially, five clients consult the PS, receive an allocation of 16 streams (concurrency = 4) and begin transferring data. These initial allocations consume 80 of the 128 available streams between the source and destination. Before the remaining three clients receive a stream allocation, the initial clients request updated allocations from the PS. The graph shows that those clients eventually receive allocations of 32 streams (client1 and client2), 28 streams (client0), 20 streams (client3) and 16 streams (client4). These allocations consume all of the 128 available streams between the source and destination. The last three clients must wait until some of the earlier clients complete their transfers and release resources before receiving a stream allocation.

We ran these experiments three times and observed these completion times: 92 minutes at 6:41pm, 107 minutes at 7:56pm and 88 minutes at 11:20pm, all on 7/9/13.

5) Summary of Tradeoffs Illustrated by Testbed 1

The experiments on Testbed 1 illustrate the operation and tradeoffs of client-side adaptation and policy-based stream allocation techniques.

The results show that the use of fast client-side adaptation allows transfer clients to quickly saturate high bandwidth networks (perhaps with a few competing clients) without overprovisioning resources, while slow adaptation is better suited to scenarios with network contention where the goal is to share bandwidth more fairly among clients.

For the Policy Service, the results demonstrate that granting larger allocations with a first-come-first-served strategy can significantly increase resource consumption for the earliest requesters. This approach may inform the design of VO policies that seek to minimize the makespan for early requesters while delaying (effectively queueing) the start time for later requesters. Conversely, VO policies that allocate fewer resources may be better suited to minimize overall makespan (for all requesters), which may be especially beneficial where jobs can be parallelized and start when a subset of input files have been transferred.

These results can thus be used to inform the specification of VO policies and for tailoring the resource allocations to the needs of VO clients and their transfer resources.

B. Evaluation for Experimental Testbed 2: NISN to NERSC over Constrained Resources

In earlier results [12], we ran experiments in a highly resource-constrained environment that consisted of a single node at Lawrence Berkeley National Laboratory running 8 adaptive data transfer clients and transferring a 260 Gbyte data set to the University of Nebraska at Lincoln. We compared the performance of fast increases in policy based allocation and fast data transfer client adaptation with the performance of non-adaptive transfers. For a configuration with 640 total streams between the source and destination, 32 initial streams per client and 160 streams maximum per client, we measured a reduction in overall transfer time of the data set of approximately 20% using our fast client-based transfer adaptation and policy-based resource allocation techniques.

Next, we describe an experiment that used a more powerful client to transfer the same data set over an inter-continental network from the National Institute of Supercomputing and Networking (NISN) in Daejon, Korea to the National Energy Research Scientific Computing Center (NERSC) in Oakland, CA. The source and destination share a 10 Gbps inter-domain network. Data are transferred from local disk on the NISN node to a GPFS project directory on the NERSC PSDF networked distributed computing cluster. The data set for these experiments is the same 260 Gbyte data set consisting of 488 files. Maximum throughput achieved between the source and

destination site is approximately 450 MB/sec or 3.6 Gbps. Because the NISN node is not a cluster, we run a single client that issues transfers at the designated concurrency level and parallelism using multiple threads.

TABLE VIII. PARAMETERS FOR COMPARATIVE EXPERIMENT

Parameters for all Comparative Experiments	*Value*
Maximum total streams between source/destination	1024
Number of clients	1
Maximum streams per client (for adaptation)	1024
Parallel streams per file (parallelism)	8
Adaptation increment/decrement (concurrency/streams)	4/32
Initial concurrency/streams for adaptation	20/160
Maximum concurrency/streams per client for adptation	128/1024
Adaptation delay time (update after how many transfers)	2
Non-adaptive concurrency/streams	128/1024

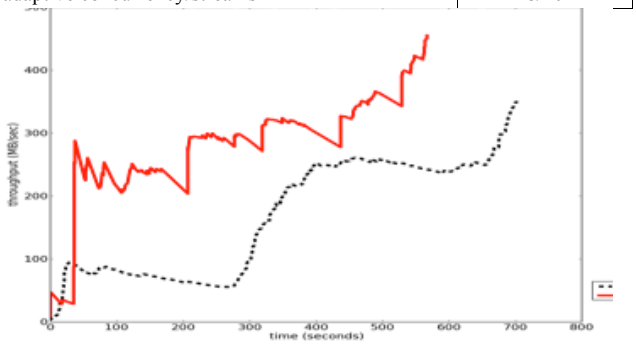

Fig. 9. Comparison of adaptive vs. non-adaptive data transfer performance for a 260 GByte data set from NISN to NERSC.

Table VIII shows parameters of our experimental scenario. In Fig. 9, we show one of two experimental run for this scenario. The red line shows the aggregate throughput in MBytes/second when using both fast client side adaptation and fast increases in resource allocation; the black line shows the throughput of non-adaptive transfer clients that use no policy-based resource allocation.

In the adaptive case, the NISN client begins with an initial concurrency of 20 and adapts the concurrency up or down after every 2 transfers complete by an increment of 32 streams (or concurrency of 4). Maximum overall concurrency is 128 (or 1024 streams) between NISN and NERSC. For the non-adaptive case, the client initiates 128 concurrent transfers with parallelism of 8 for a total of 1024 streams.

Intiuitively, if available bandwidth is not limited, the non-adaptive transfers should have higher throughput, since they consistently use 1024 streams to transfer data, while the adaptive case starts its transfers with only 160 streams (concurrency of 20, parallelism of 8). Instead, Fig. 9 shows that the throughput for adaptive transfers (shown in red) is significantly higher than the non-adaptive transfers (shown in black), indicating that the test environment is resource-constrained and that the adaptive transfer client and policy-based resource allocation make more effective use of available resources without overprovisioning.

The overal time to transfer the data set from NISN to NERSC is approximately 20% shorter in the adaptive case, which is similar to the benefit we observed in our earlier experiments between LBNL and UNL. These experiments show a significant advantage in throughput and overall transfer time for adaptive, policy-based transfers compared to non-adaptive transfers on resource-constrained infrastructure.

IV. Related Work

Policies for data staging and resource management: In earlier work, we studied policy-based data staging for scientific applications [13-17]. Large scientific collaborations have used VO policy-based data dissemination and replication, including the Physics Experiment Data Export (PheDEx) [18, 19] system and the Lightweight Data Replicator (LDR) [20]. The integrated Rule-Oriented Data System (iRODS) [21] uses a rule based system to implement data management policies

Data transfer adaptation: Data transfer parameter estimations have been studied based on the active profiling measurements from the sample transfers [22-28] or a simple throughput prediction model with passsive information from current data transfer operations [29, 30]. Application-level auto-tuning techniques [23, 31-33], including adaptive parameter tuning [23, 24, 29] only consider network conditions. Multiple data transfer streams is a common way of increasing the network throughput performance in client applications [22, 24, 34]. Our work does not make data transfer throughput predictions or try to model the performance; rather, it reflects dynamic throughput performance changes into data transfer management.

V. Conclusions

We presented two techniques to adapt the use of resources for long running, multi-file data transfers: (1) policy-based allocation of data transfer resources at the Virtual Organization level based on VO and site level policies, and (2) adaptation of transfer parameters by each transfer client based on recent performance. We showed that these techniques provide significant improvements in throughput and overall transfer completion time (up to 20% in our experiments) in resource constrained environments. We also demonstrated the tradeoffs of these techniques. Fast client side adaptation and VO-level policies that quickly increase resource allocation consume available resources aggressively without overprovisioning the resources, while slower adaptation and VO-level allocation policies share available resources more fairly among clients.

We plan to extend the current work to provide richer policies and adaptation to further improve network utilization and transfer performance. Our current work uses static policies based on VO preferences and priorities; we will extend this work to study whether a dynamic policy model that adapts to changes in resource demand and availability would provide additional improvements. In addition, our current work adapts transfer parameters such as concurrency based on the transfer client's limited knowledge of recent performance between a specific pair of source and destination hosts. We will extend this work to study the impact of moving performance-based adaptation to the VO level and developing adaptation algorithms that incorporate knowledge of transfers throughout the VO as well as performance measurements and resource availability information from monitoring systems. Finally, we will study extending policies to incorporate novel techniques for performance estimation and prediction.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation (NSF) Office of Cyberinfrastructure under awards 1127101 (USC/ISI) and 1127039 (LBNL) and by the U.S. Department of Energy (DOE) Office of Advanced Scientific Computing Research, Office of Science, under Contract DE-AC02- 05CH11231. This work used resources from the Open Science Grid, which is supported by NSF and the DOE Office of Science. We thank Brian Bockelman, Garhan Attebury and Carl Lundstedt at U. Nebraska, Lincoln; Buseung Cho and Jysoo Lee at NISN; Eli Dart at ESnet; K. John Wu at LBNL; C. S. Chang at PPPL; and Iwona Sakrejda at NERSC for their support for our experiments.

REFERENCES

[1] *LHC – The Large Hadron Collider*. Available: http://lhc.web.cern.ch/lhc/

[2] *Large Synoptic Survey Telescope (LSST)*. Available: http://www.lsst.org/lsst

[3] LIGO Project. *LIGO - Laser Interferometer Gravitational Wave Observatory*. Available: http://www.ligo.caltech.edu/

[4] A. B. Rosen, M. B. Hamel, M. C. Weinstein, D. M. Cutler, A. M. Fendrick, and S. Vijan, "Cost-effectiveness of full Medicare coverage of angiotensin-converting enzyme inhibitors for beneficiaries with diabetes," *Annals of internal medicine,* vol. 143, pp. 89-99, 2005.

[5] D. N. Williams, R. Ananthakrishnan, D. E. Bernholdt, S. Bharathi, D. Brown, M. Chen*, et al.*, "The Earth System Grid: Enabling Access to Multi-Model Climate Simulation Data," *Bulletin of the American Meteorological Society (BAMS),* vol. 90, no. 2, pp. 195–205, February 2009.

[6] "Biological and Environmental Research Network Requirements Workshop, November 2012 - Final Report," DOE Office of Science and the Energy Sciences Network, http://www.es.net/assets/pubs_presos/BER-Net-Req-Review-2012-Final-Report.pdf20112012.

[7] J. Cummings, T. Finholt, I. Foster, C. Kesselman, and K. A. Lawrence, "Beyond Being There: A Blueprint for Advancing the Design, Development, and Evaluation of Virtual Organizations " Final report from the NSF workshop on Building Effective Virtual Organizations, http://www.ci.uchicago.edu/events/VirtOrg2008/, 2008.

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

[9] *XSEDE Extreme Science and Engineering Discovery Environment*. Available: https://http://www.xsede.org/

[10] J. Shiers, "The Worldwide LHC Computing Grid (Worldwide LCG)," *Computer Physics Communications,* vol. 177, pp. 219-223, 2007.

[11] J. Gu, D. Smith, A. L. Chervenak, and A. Sim, "Adaptive Data Transfers that Utilize Policies for Resource Sharing," presented at the 2nd Int'l Workshop on Network Aware Data Management (NDM 2012), in conjunction with SC12 Conference, Salt Lake City, UT, 2012.

[12] A. L. Chervenak, A. Sim, J. Gu, R. Schuler, and N. Hirpathak, "Efficient Data Staging Using Performance-Based Adaptation and Policy-Based Resource Allocation," presented at the 22nd Euromicro Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2014), Turin, Italy, 2014.

[13] 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.

[14] 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.

[15] S. Bharathi and A. Chervenak, "Scheduling Data-Intensive Workflows on Storage Constrained Resources," in *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science (WORKS09), in conjunction with Supercomputing (SC09)*, Portland, Oregon, 2009.

[16] 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.

[17] A. L. Chervenak, D. E. Smith, W. Chen, and E. Deelman, "Integrating Policy with Scientific Workflow Management for Data-Intensive Applications," presented at the 7th Workshop on Workflows in Support of Large-Scale Science (WORKS 12), in conjunction with SC12 Conference, Salt Lake City, UT, 2012.

[18] T. A. Barrass, et al., "Software Agents in Data and Workflow Management," in *Computing in High Energy and Nuclear Physics (CHEP) 2004*, Interlaken, Switzerland, 2004.

[19] J. Rehn, T. Barrass, D. Bonacorsi, J. Hernandez, I. Semeniouk, L. Tuura*, et al.*, "PhEDEx high-throughput data transfer management system," in *Computing in High Energy and Nuclear Physics (CHEP) 2006*, Mumbai, India, 2006.

[20] (2004). *Lightweight Data Replicator*. Available: http://www.lsc-group.phys.uwm.edu/LDR/

[21] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder, "A prototype rule-based distributed data management system," in *HPDC Workshop on Next-Generation Distributed Data Management*, ed, 2006.

[22] T. Dunigan, M. Mathis, and B. Tierney, "A tcp tuning daemon," in *SuperComputing: High-Performance Networking and Computing*, 2002.

[23] W. Feng, M. Fisk, M. Gardner, and E. Weigle, "Dynamic right sizing:An automated, lightweight, and scalable technique for enhancing grid performance," in *the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks*, 2002.

[24] T. Ito, H. Ohsaki, and M. Imase, "On parameter tuning of data transfer protocol gridftp in wide-area grid computing," in *Second Int'l Workshop on Networks for Grid Applications, GridNets*, 2005.

[25] T. J. Hacker, B. D. Noble, and B. D. Athey, "Adaptive data block scheduling for parallel TCP streams," in *the High Performance Distributed Computing*, 2005.

[26] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to TCP throughput prediction," *SIGMETRICS Perform. Eval. Rev.,* vol. 35, pp. 97-108, 2007.

[27] E. Yildirim, M. Balman, and T. Kosar, "Dynamically Tuning Level of Parallelism in Wide Area Data Transfers," in *DADC'08*, 2008.

[28] D. Yin, E. Yildirim, and T. Kosar, "A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing," in *MTAGS'09*, 2009.

[29] M. Balman and T. Kosar, "Dynamic Adaptation of Parallelism Level in Data Transfer Scheduling," in *Second International Workshop on Adaptive Systems in Heterogeneous Environments*, 2009.

[30] 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.

[31] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster, "GridFTP Pilelining," in *Teragrid 2007 Conference*, 2007.

[32] M. Gardner, S. Thulasidasan, and W. Feng, "User-space auto tuning for tcp flow control in computational grids," *Computer Communications,* vol. 27, pp. 1364-1374, 2004.

[33] S. Soudan, B. Chen, and P. V.-B. Primet, "Flow scheduling and endpoint rate control in grid networks," *Future Gener. Comput. Syst.,* vol. 25, pp. 904-911, 2009.

[34] 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.