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 Schuler[1], Nandan Hirpathak[1]

[1]University of Southern California Information Sciences Institute

[2]Lawrence Berkeley National Laboratory

Motivation

- Many science applications require staging of large datasets to prepare for analysis on shared computational resources
- Need to make efficient use of available resources
  - Avoid overprovisioning
- **ADAPT Project (Adaptive Data Access and Policy-Driven Transfers)**
  - Goal: improve transfer throughput and overall latency for large data staging operations in a resource-constrained, shared environment
  - Provide simple software path to adaptation
- **Two Techniques:**
  - **Performance-based transfer parameter adaptation**
  - **Policy-based resource allocation**

Outline

- **ADAPT Project Approach**
- **System Description**
- **Implementation**
- **Experimental Results**
- **Summary**

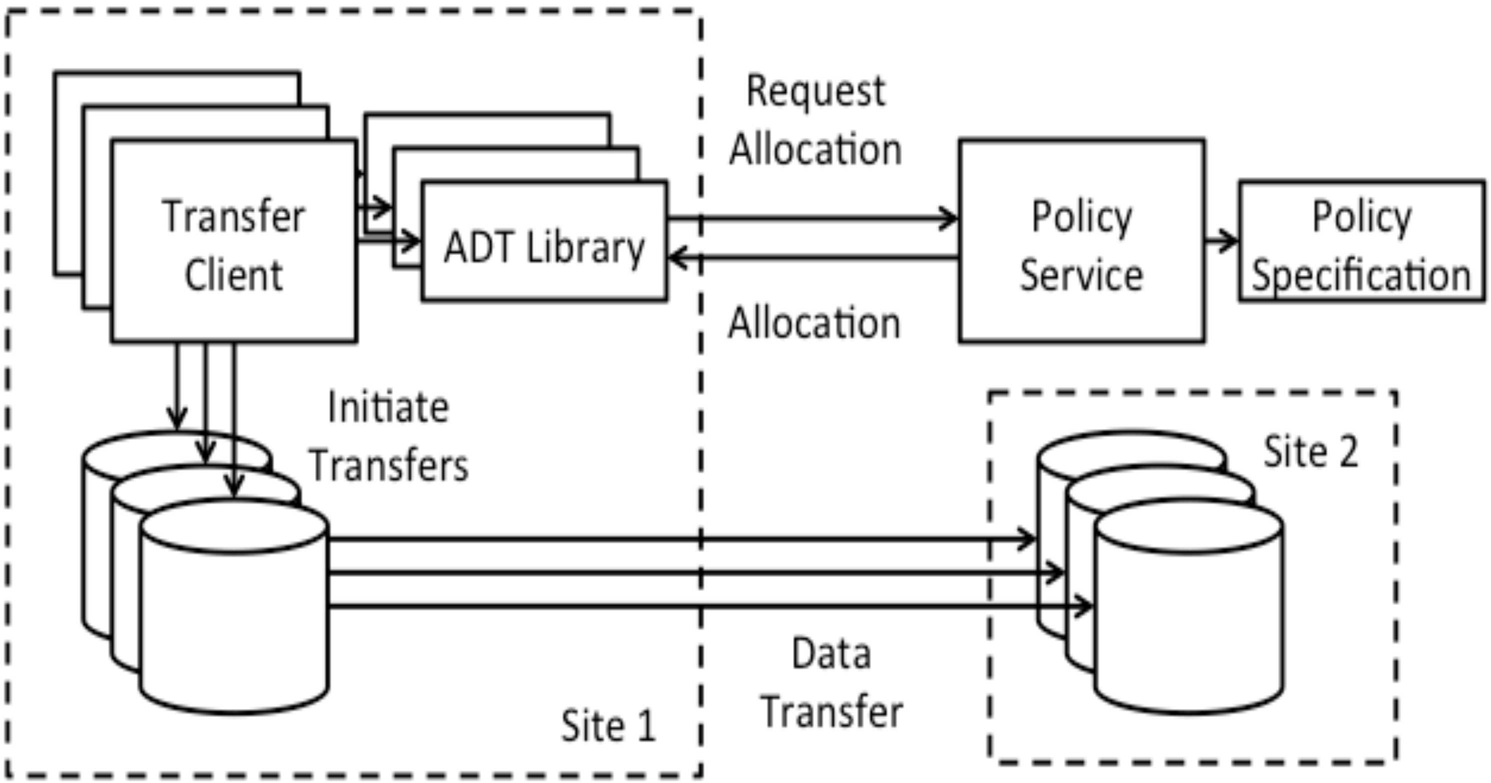
Contributions:

- Describe algorithms for policy-based resource allocation and adaptation of data transfer parameters
- Experimental results:
  - Detailed operation of transfer client adaptation and policy-based resource allocation for large, multi-file transfers
  - Approximately 20% improvement in overall intercontinental transfer completion time with our techniques

ADAPT Project Approach

- **Performance-based transfer adaptation**
  - **Adaptive data transfer client** selects transfer properties based on past performance, available resources
  - Adapts properties for subsequent transfers when observed performance changes due to dynamic load on storage, network, other resources
- **Policy-based resource allocation**
  - Based on Virtual Organization (VO and site level policies regarding resource allocation, priorities for resources, users)
  - VO-Level **Policy Service** gives data transfer clients advice on resource allocations (transfer streams)
  - Balance user requirements for data access with load on resources

System Overview



- Adaptive Data Transfer Client
- Policy Service (PS)

Policy Service (PS)

- Suggests resource allocations based on:
  - Available system resources
  - Virtual Organization or site policies for allocating resources for network data transfers
- Handles multiple resource allocation requests from clients
  - Client makes ***initial allocation*** request
  - Periodically requests ***updated allocation***
- By default: deployed with **Greedy policy**
  - Users may replace with their own policy module written in python
- Virtual Organization administrators set policies based on VO environment requirements

Policy Service Parameters

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)

Greedy Stream Allocation Algorithm for Policy Service (1)

- **Request arrives from client**
- Includes source, destination
- **PROVISION checks for available streams between source and destination**
- If enough for full initial allocation, allocate s_i streams

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)
t ← transfer resource request with (source[t], dest[t]) and steams[t])
s_a ← allocated streams between (source[t], dest[t])
s_v ← min(s_cmax – streams[t], s_pmax – s_a)      // Available streams
if s_v = 0 then
    // No available streams for transfer request
    return t
else if streams[t] = 0 and s_v > s_i then
    // Enough streams for initial allocation
    streams[t] ← s_i
    s_a ← s_a + s_i      // Update total allocated streams
```

Greedy Stream Allocation Algorithm for Policy Service (2)

else if streams[t] > 0 **and** $s_v > s_u$ **then**
    // Enough streams for *update* allocation
    streams[t] $\leftarrow$ streams[t] $+ s_u$
    $s_a \leftarrow s_a + s_u$ // Update total allocated streams
else
    // Allocate remaining available streams to *initial* or *update* request
    streams[t] $\leftarrow$ streams[t] $+ s_v$
    $s_a \leftarrow s_a + s_v$ // Update total allocated streams
end if
return t
end procedure

- **For an update request, if enough available streams, allocate s_u**
- Otherwise (for update or initial request)
  - **Allocate remaining available streams**

Adaptive Data Transfer Client

- Performs data transfers
- **Adapts within the resource allocation from Policy Service**
  - Modifies parameters for new transfers based on recent performance and resource availability
- Possible parameters for adaptation
  - concurrency, number of parallel streams, buffer size, etc.
- **Current design adapts concurrency**
  - Concurrency * parallelism = number of streams
- For long-running, multi-file transfers, a client periodically:
  - **Requests new allocation advice from the PS**
  - **Adapts its concurrency level up or down based on recent performance and current stream allocation**

USC

Adaptive Data Transfer Client Parameters

BERKELEY LAB

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.

USC

Adaptive Transfer Client Algorithm (1)

BERKELEY LAB

- Queue of files to be transferred from source to dest.
- Request initial allocation from PS
- Set concurrency based on allocation
- **Loop while Queue not empty:**
  - **If specified number of transfers are complete, update allocation**
  - **Call ADAPT function**
  - **Perform top c transfers in queue**

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)
t ← initialize a transfer request between (source, dest) of Q
Provision(t)                // request initial allocation from Policy Service
c_alloc ← floor(streams[t] / p)    // convert streams to concurrency
c ← min(c_i, c_alloc)      // limit concurrency parameter, if necessary
k ← d                      // set counter for next adaptation
while Q not empty do
    if k ≤ 0 then          // due for client adaptation
        k ← d                          // reset counter
        Provision(t)                   // request updated allocation from PS
        c_alloc ← floor(streams[t] / p) // convert streams to concurrency
        c ← Adapt(c, Δ, c_alloc)       // adapt concurrency up or down
    end if
    F ← pop at most c transfer jobs from Q
    // …perform F transfers concurrently, wait for completion…
    k ← k − c              // decrement transfer counter
end while
end procedure
```

Adaptive Transfer Client Algorithm (2)

```
procedure ADAPT(c, Δ, c_alloc)
T ← user specified transfer rate adaptation threshold
r_last ← state of last recorded transfer rate  // between source-destination
r_rate ← test of current transfer rate         // between source-destination
r_delta ← r_rate - r_last
if abs(r_delta) > T then                // change exceeds threshold
    if r_delta < 0 then
        c ← max(0, c – Δ)               // decrease concurrency
    else
        c ← min(c + Δ, c_alloc, c_max)  // increase concurrency
    end if
end if
return c
end procedure
```

- Compares current transfer rate between source, dest with last measured transfer rate
- **If difference exceeds a threshold T,**
  - **Increase concurrency by Δ if transfer rate has increased**
  - **Decrease concurrency by Δ if transfer rate has decreased**
- New concurrency must be non-negative and <= c_{max}

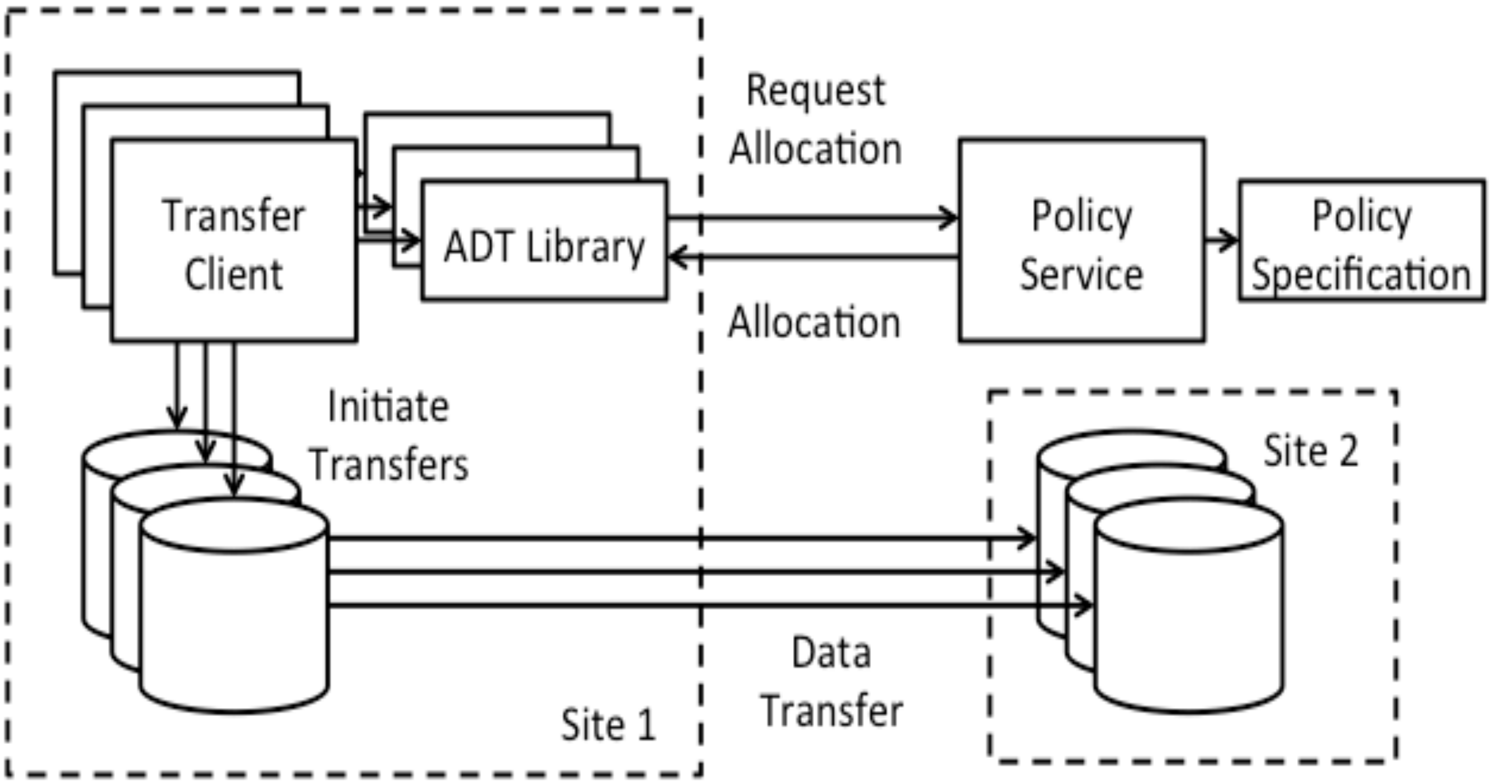
Implementation

- **Policy Service (PS)**
  - RESTful Web service implemented in Python
    - Webpy framework, CherryPy embedded HTTP server
  - Open source implementation available at: http://github.com/robes/adapt-policy-service

- **Adaptive Data Transfer Client**
  - Modified conventional srm-copy data movement client
  - Stand-alone, command-line client implemented in Java
  - Added an **Adaptive Data Transfer (ADT) library**
  - Open source Adaptive Data Transfer Client available at: https://codeforge.lbl.gov/projects/adapt/

Evaluation: Experimental Set up

- Example scenario: Users want to run analysis on an Open Science Grid site
- Must first stage large multi-file data set from a remote data source

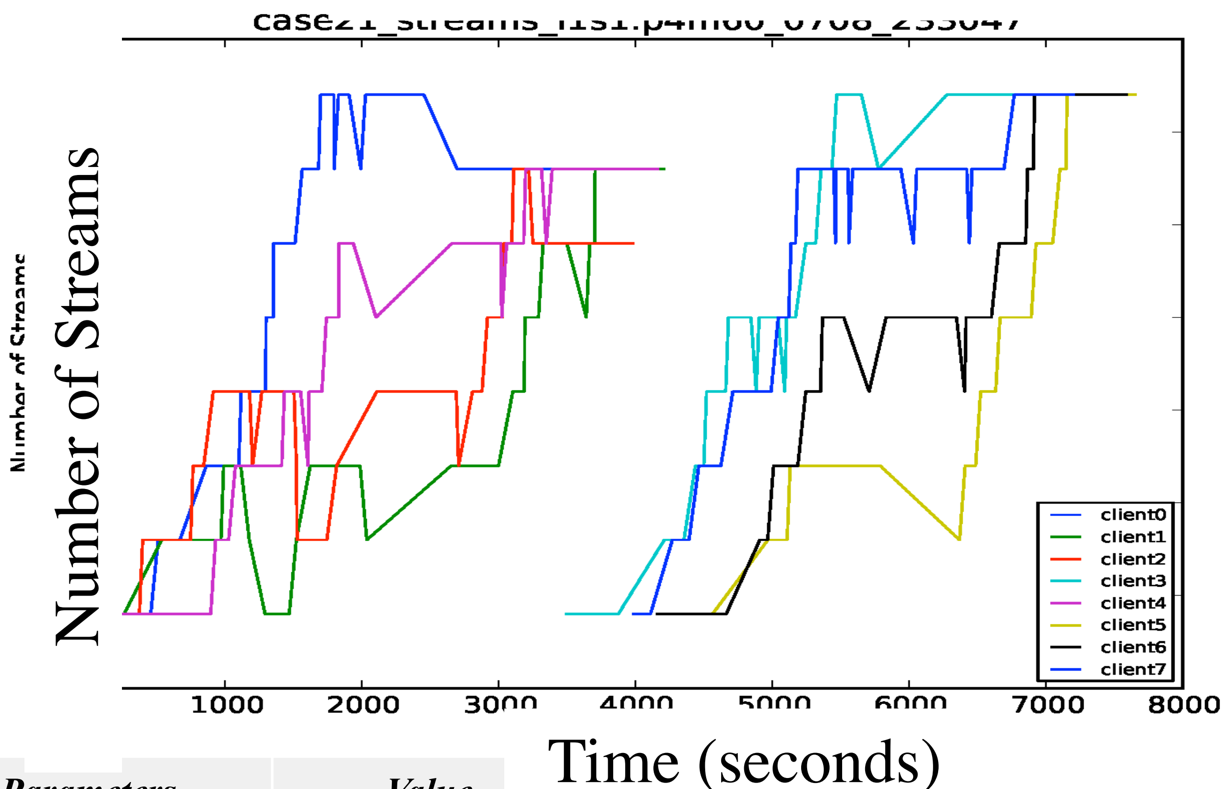


Experiments: Testbed 1

- Performance of adaptive data transfers with srm-copy client
- Transfer data from NERSC in Oakland, CA to Open Science Grid site at University of Nebraska at Lincoln (UNL), over 10Gbps link
- 8 srm-copy clients performing multi-file transfers: 260 Gbytes / 488 files
- Long-running, multi-file transfers; adapt between completed transfers
- Default common parameters

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)

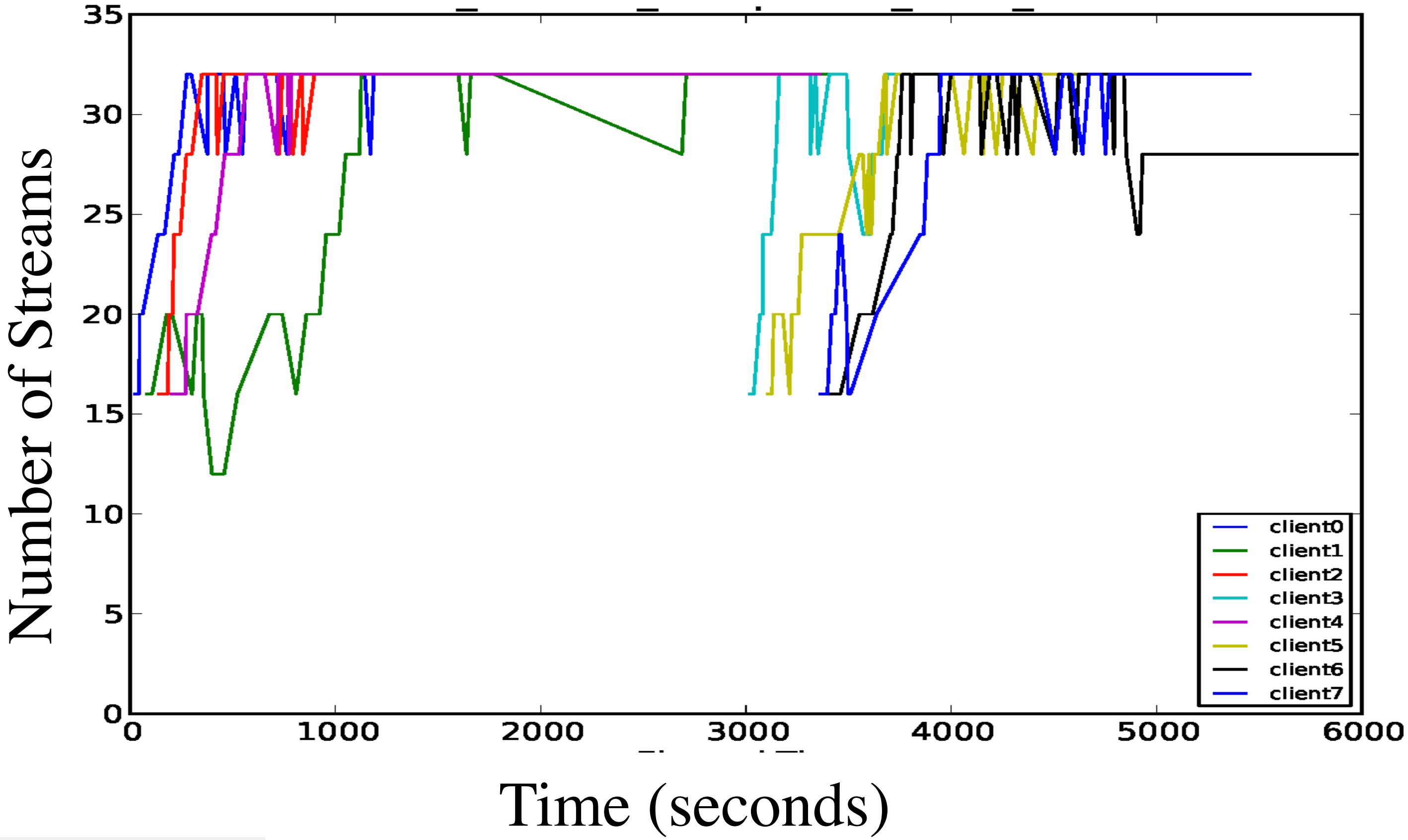
Slow Client-side Adaptation



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

- All adaptation takes place on the client side
- Each client slowly adapts the concurrency of its transfers up to the allocation given by the PS

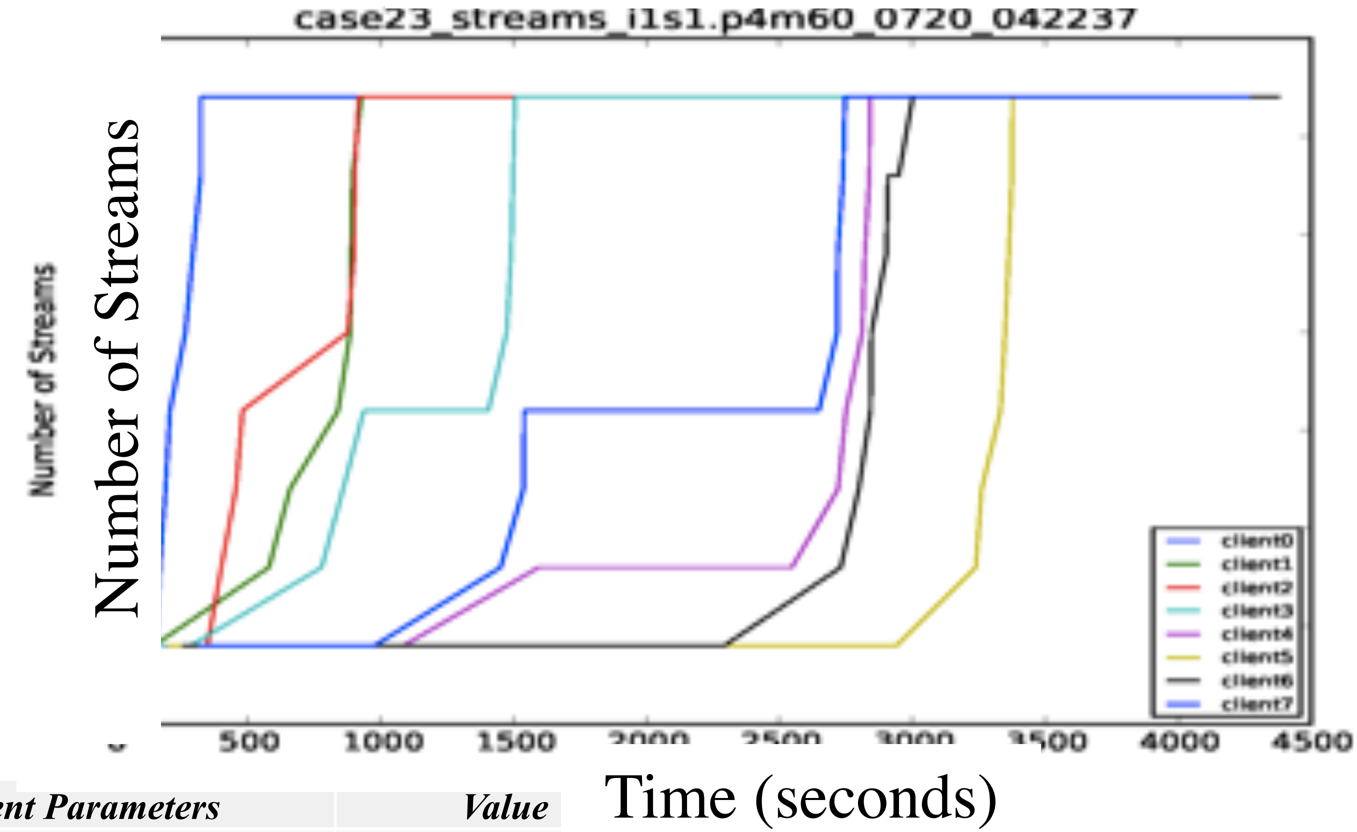
Fast Client-side Transfer Adaptation



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

- The higher initial concurrency and the faster adaptation rate have a significant effect on the performance of the transfers.

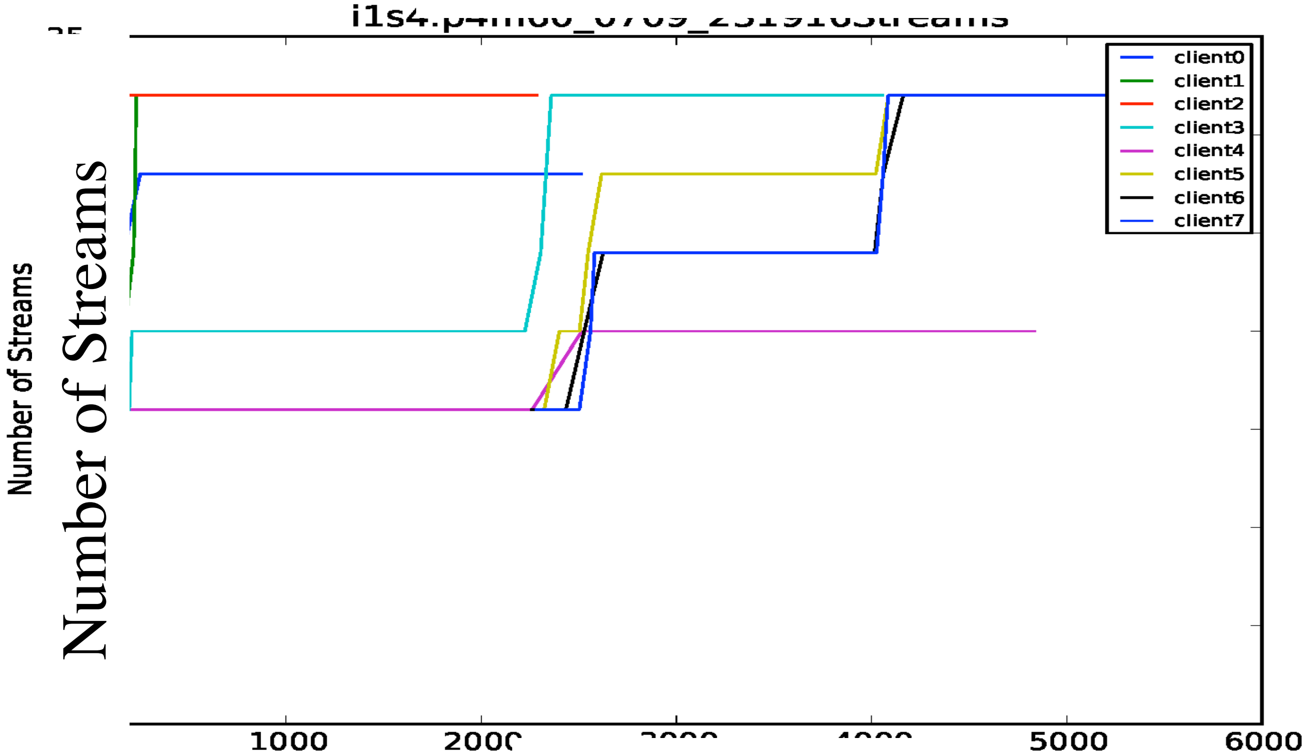
Policy-based Resource Allocation
Slow Increase



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

- No client side adaptation
- Higher number of streams used after clients update their allocations and increase their concurrency several times.

Policy-based Resource Allocation Fast Increase



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

- Several clients adapt up to 20, 28 and 32 streams
- This forces the last three clients to wait until those first clients complete their transfers and release the streams

Experimental Testbed 2

- Transferred 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 transferred from local disk on NISN node to a GPFS project directory on NERSC PSDF networked distributed computing cluster
- Data set: **same 260 Gbyte data set consisting of 488 files**
- 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**

Experimental Parameters

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

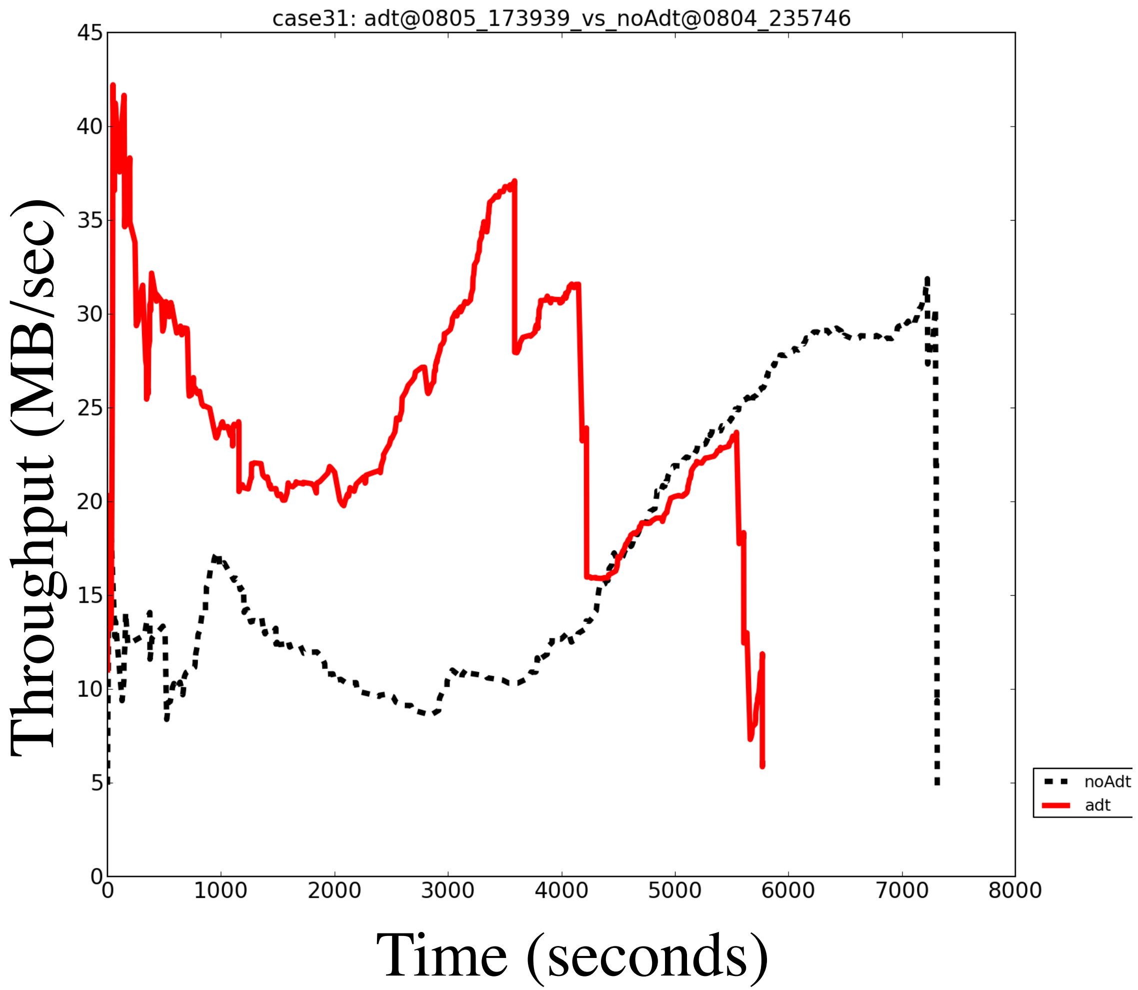
Adaptive Case: NISN client has initial concurrency of 20 and adapts concurrency after every 2 transfers complete by an increment of 32 streams (concurrency of 4). Maximum overall concurrency is 128 (or 1024 streams) between NISN and NERSC

Non-adaptive case: NISN client initiates 128 concurrent transfers with parallelism of 8 for a total of 1024 streams

Throughput Results

Cumulative throughput for adaptive vs. non-adaptive transfers

Maximum 1024 total streams between NISN and NERSC



Contention for available resources

Red line shows adaptive performance
Black line non-adaptive

Significant advantage in throughput and overall transfer runtime for the adaptive, policy-based transfers

Total transfer time reduced by approximately 20%

Remarks

- **Intuitively, 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) and adapts to increase concurrency
- **Instead, the experiment shows higher throughput for adaptive transfers compared to non-adaptive transfers**
- This indicates that:
  - The test environment is resource-constrained
  - **The adaptive transfer client and policy-based resource allocation make more effective use of available resources without overprovisioning**

Summary

- **ADAPT project goals**
  - Avoid overprovisioning of resources that results in suboptimal transfer throughput
  - Adaptive transfer parameters, policy-based allocation advice
- **ADAPT software stack**
  - Provides significant throughput and completion time improvements in resource constrained environments
  - Provide simple transition from current data movement practices to policy-based, adaptive data movement
- **Plans for next phase:**
  - Explore richer policies for managing resources, adaptation
  - Design adaptive policies that change with conditions, performance
  - Move performance-based adaptation to the VO level to incorporate knowledge of transfers, resources throughout the VO
  - Research on how to set policy parameters automatically
  - **Work with application communities to deploy and evaluate ADAPT software**

Acknowledgments

This work was supported in part by:

- The U.S. National Science Foundation Office of Cyberinfrastructure under award 1127101 (USC/ISI) and award 1127039 (LBNL)
- The Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02- 05CH11231
- This work used resources provided by the Open Science Grid, which is supported by NSF and the U.S. DOE Office of Science
  - We thank Brian Bockelman, Garhan Attebury and Carl Lundstedt at University of Nebraska, Lincoln and Iwona Sakrejda at National Energy Research Scientific Computing Center, Buseung Cho, Jysoo Lee at NISN/Kreonet and Eli Dart at ESnet for their support on our experiments.