StorkCloud

Data Transfer Scheduling and Optimization as a Service

Presented by: Brandon Ross

Contents

StorkCloud?

- ✓ Multi-protocol data transfer scheduler
- ✓ Remote metadata retrieval and caching service
- ✓ Dynamic, protocol-agnostic transfer optimization to improve speed
- ✓ All in the cloud – accessible through thin client GUIs and public REST API

Why StorkCloud?

- ✓ Storage and computation are both in the cloud – cloud transfer is the missing link
- ✓ Data transfer is usually inconvenient
  - Many different protocols
  - Many different applications
- ✓ Transferring large files requires monitoring whole process – StorkCloud is "fire-and-forget"

Why optimize?

- ✓ Transfers are usually suboptimal
  - Inadequacies of underlying protocols
  - End-system misconfiguration
- ✓ Not designed for high-speed networks
- ✓ Some applications can be specially configured, but it's mostly guesswork
- ✓ Network environments can vary – dynamic optimization is important
- ✓ StorkCloud aims to solve these problems

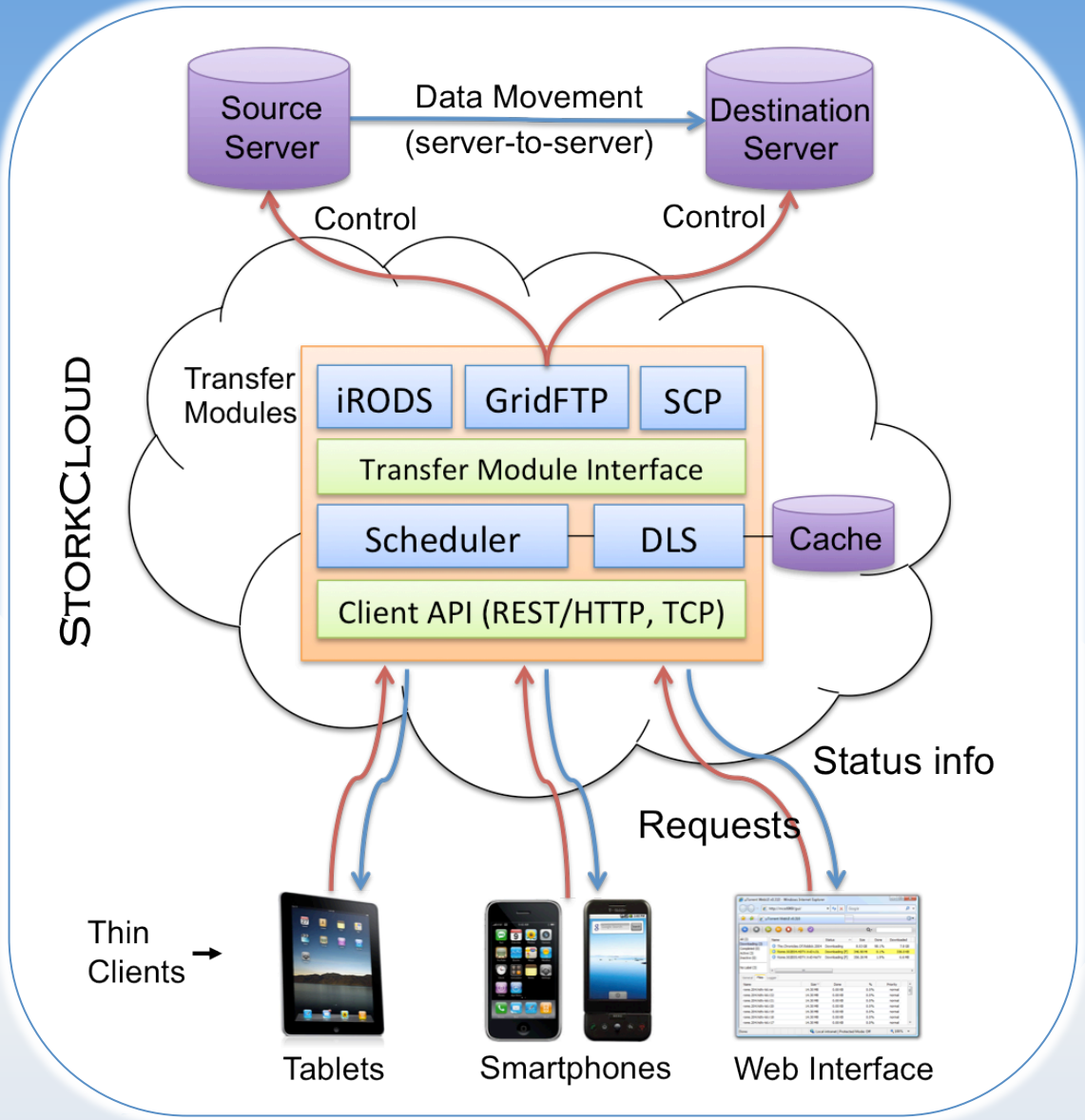
Who?

- ✓ Scientists in data-oriented fields (astrophysics, genomics, climatology, biochemistry, etc.)
- ✓ Data centers looking to outsource replication and data placement
- ✓ Application developers who might want to offload data transfer tasks
- ✓ Anyone with a lot of data to move

Similar Work

- ✓ Only similar service we know of is Globus Online
  - Mature, popular service
    - Over 18.3 *petabytes* transferred!
  - Designed to support FTP and GridFTP
  - Statically optimized transfers
  - No prefetching/caching available for directory listings

Contents


Source Server
Data Movement
(server-to-server)
Destination Server
Control
Control
StorkCloud
Transfer Modules
iRODS
GridFTP
SCP
Transfer Module Interface
Scheduler
DLS
Cache
Client API (REST/HTTP, TCP)
Status info
Requests
Thin Clients
Tablets
Smartphones
Web Interface

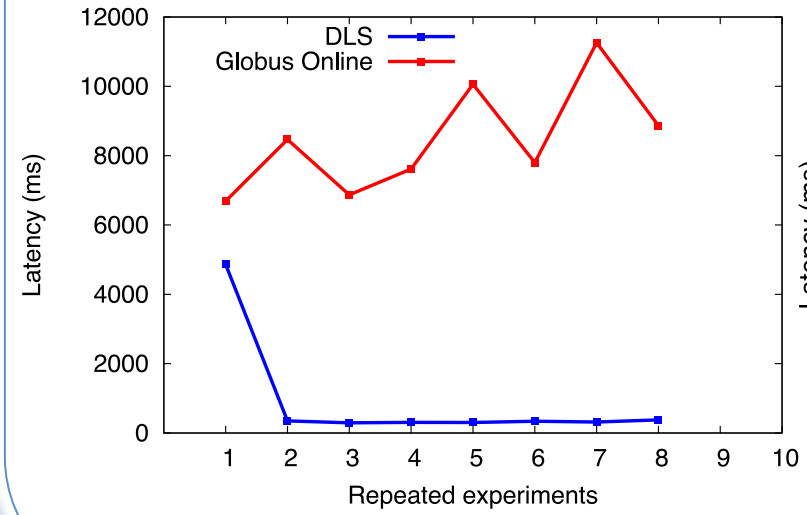
Stork Data Scheduler

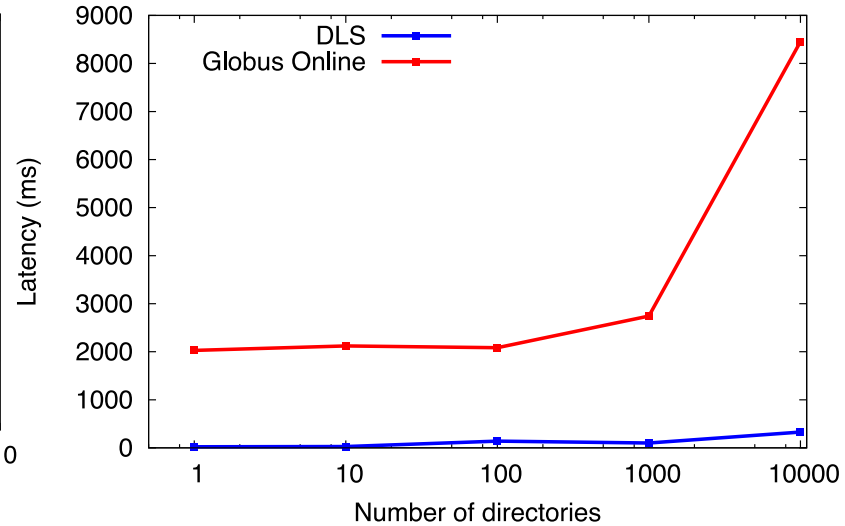
- ✓ Accepts file transfer jobs – source, destination, and other options
- ✓ Places job in queue for processing
- ✓ Passes jobs off to transfer module
- ✓ Job status can be queried by clients
- ✓ Currently first come, first served

Directory Listing Service

- ✓ Conceptually: unified metadata interface to many file systems
- ✓ Retrieves file and directory metadata from remote file systems
- ✓ Returns results as JSON
- ✓ Uses caching and prefetching to improve responsiveness

DLS Performance





Transfer Modules

- ✓ Pluggable transfer modules perform transfers for specific protocols
- ✓ Communicate progress and messages back to scheduler
- ✓ Either Java bytecode or external executable – can be any language!

Client Interfaces

- ✓ Thin clients communicate with server REST API using JSON
  - Starting or canceling transfers, or querying transfer status
  - Browsing remote directories
  - Initializing credentials (e.g. GSI proxies)
- ✓ Currently have Android and web applications, and command line tools
- ✓ Our GUI applications can browse remote files and check transfer progress

Contents

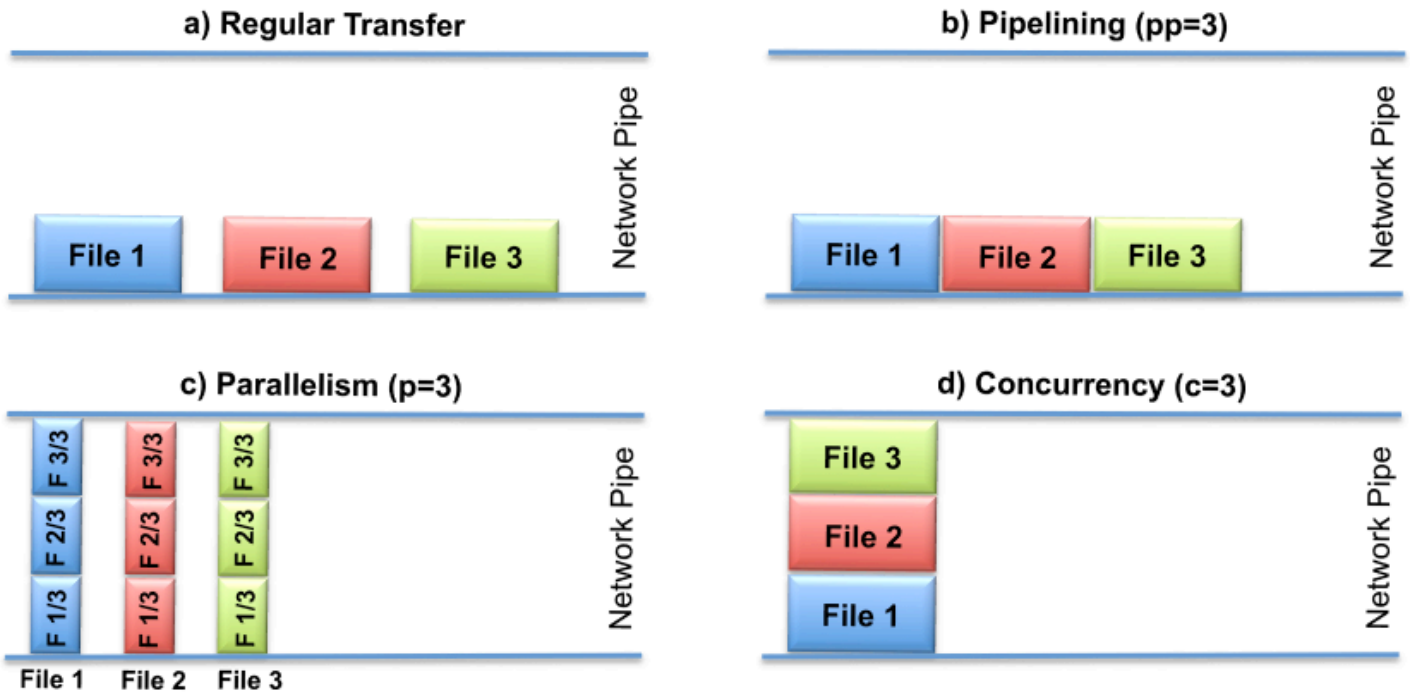
Optimization in StorkCloud

- ✓ Each pluggable optimizer is an implementation of a different algorithm
- ✓ Each targets a set of file transfer parameters
- ✓ Feedback loop; optimizer and TM work together to optimize transfer
- ✓ Optimizers are protocol-agnostic – only care about whether transfers support targeted parameters

Parameters and Techniques

- ✓ Pipelining – "queuing up" transfer commands at a remote system
- ✓ Parallelism – transferring file data over multiple connections
- ✓ Concurrency – transferring multiple files at a time

Parameters Visualized



Algorithms

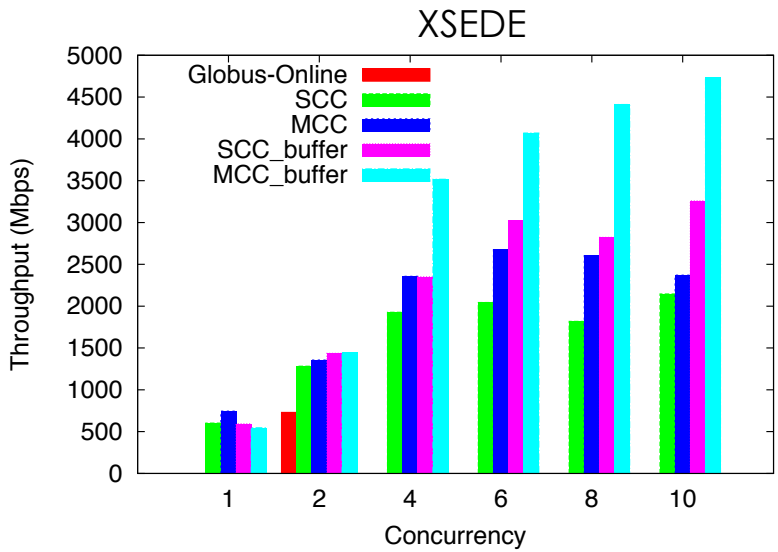
- ✓ Optimal parallelism prediction:
  - Samples points, performs regression analysis to predict optimal parallelism
  - 2nd order and c-order analysis
- ✓ Parallelism-Concurrency-Pipelining
  - Uses historical database and clustering

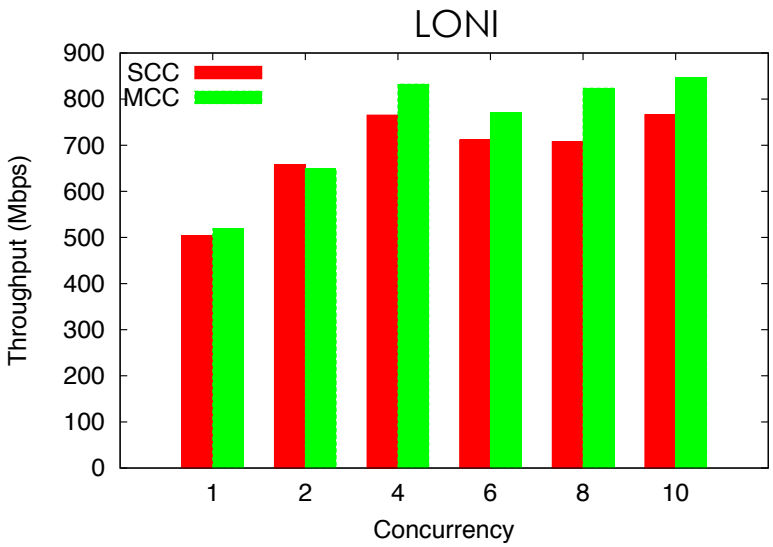
Algorithms

- ✓ Single/Multi Chunk Concurrency
  - Newest algorithms, designed for multi-file transfers with mixed sizes
  - Partition file sets into "chunks" based on file size and transfer concurrently on multiple channels
  - Each channel is configured heuristically
  - SCC: transfer chunks one at a time, split across all concurrent channels
  - MCC: each chunk gets a dedicated channel; reduces effect of small files

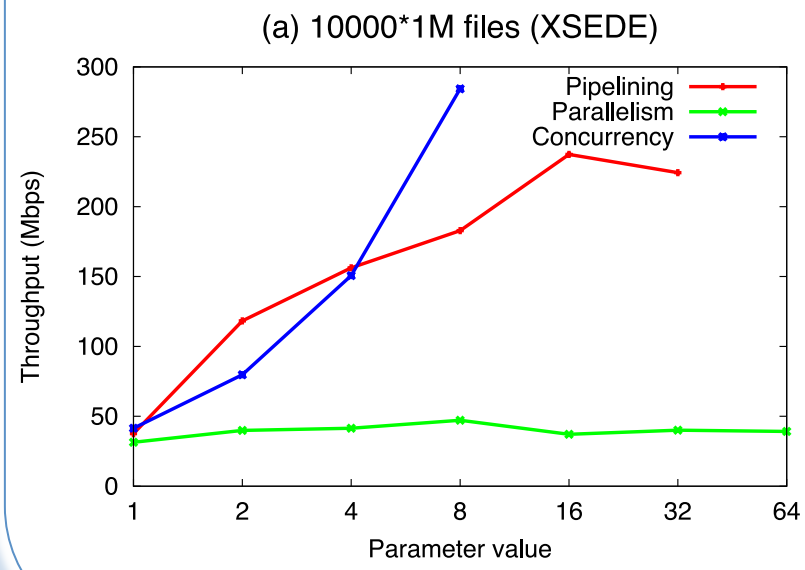
Contents

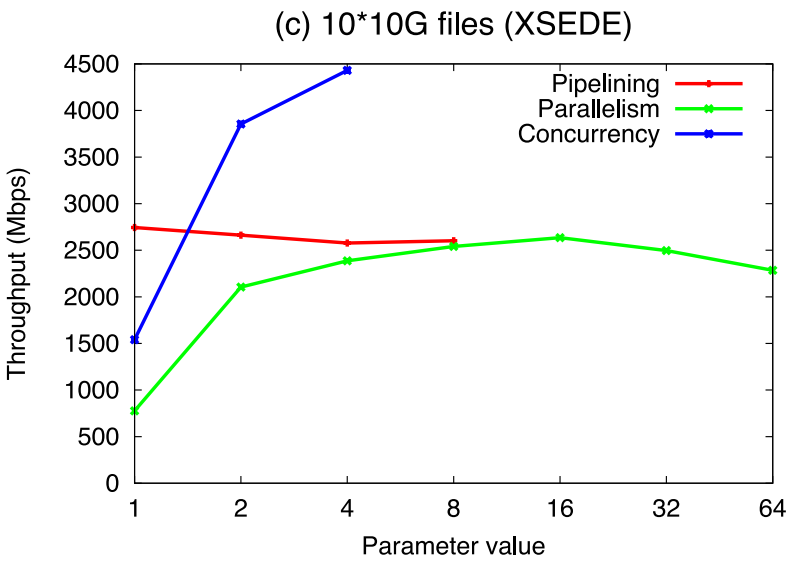
SCC/MCC Performance





Effects of Parameters





Future Work

- ✓ Additional scheduling algorithms and priority
- ✓ Date-based scheduling and routine jobs
- ✓ Direct file upload/download through client interfaces – currently only 3rd party transfers
- ✓ SSL/TLS for secure communications (HTTPS)
- ✓ Additional protocols (SFTP, HTTP, AFP, etc.)
- ✓ Temporary file parking – storage on server in case of destination issues
- ✓ Historical performance database

The End

Thank you! Any questions?

Home | Transfer | testuser | Log out

Browse and Transfer Files

ftp://didclab-ws8/~/stuff

- ftp://didclab-ws8/~/stuff
  - 10
  - 11
  - 12
  - 13
  - 14
  - 15
  - 16
  - 17
  - 18
  - 19
  - 20

ftp://didclab-ws9/~globus

- .globus
- .ssh
- globus
- stuff
- sub1
- sub2
- .bash_history
- .bash_logout
- .bashrc
- .lesshst
- .profile
- 00

Transfer Options

Transfer Progress and Queue

	ID	Progress	Speed	Source/Destination
i	4	complete	84.56MB/s	ftp://didclab-ws8/~/stuff/04 ftp://didclab-ws8/~globus/stuff/
i	3	83%	76.27MB/s (93.15MB/s)	ftp://didclab-ws8/~globus/stuff/ ftp://didclab-ws8/~/stuff/sub2/
i	2	complete	0.00 B/s	ftp://didclab-ws8/~/stuff/00 ftp://didclab-ws8/~globus/00
i	1	failed		ftp://didclab-ws8/~/stuff/99 ftp://ftp.example.com/fake/

Refresh Auto-refresh table? ☑

Submit Advanced Transfer Job


11:31
Stork Client
tg-login.spur.tacc.teragrid.org
stork-2.0.1
CVS
condorlib
login1.ls4.tacc.utexas.edu
.py
createFolders.pl
externals.tar.gz
Disconnect All
10:31
Job Progress
2 Finished ftp:/didc-ws8.buffalo.edu irods:/didc-ws7.buffalo.edu
3 70% gsiftp:/tg-login.spur.tacc.teragrid.org/etc/1.dat gsiftp:/nbirn.org
5 Queued gsiftp:/loni.org gsiftp:/dest.dsl-stork.org/home/sivahpc/test/dest
6 67% gsiftp:/dest.dsl-stork.org/home/sivahpc/test/dest gsiftp:/loni.org
10 Removed sftp:/tg-login.spur.tacc.teragrid.org/stork-2.0.1.tar. sftp:/login1.ls4tacc.utexas.edu/stork-2.0.1/