Contents
1. Introduction
2. Components
3. Optimization
4. Conclusion
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

1. Introduction
2. Components
3. Optimization
4. Conclusion
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

![Graph 1: DLS Performance vs. Repeated Experiments](image1)

![Graph 2: DLS Performance vs. Number of Directories](image2)
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

1. Introduction
2. Components
3. Optimization
4. Conclusion
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

a) Regular Transfer

File 1  File 2  File 3

Network Pipe

b) Pipelining (pp=3)

File 1  File 2  File 3

Network Pipe

c) Parallelism (p=3)

File 1  File 2  File 3

Network Pipe

d) Concurrency (c=3)

File 3
File 2
File 1

Network Pipe
Algorithms

✓ Optimal parallelism prediction:
  – Samples points, performs regression analysis to predict optimal parallelism
  – $2^{nd}$ 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
1. Introduction
2. Components
3. Optimization
4. Conclusion
SCC/MCC Performance

(a) (XSEDE)

(b) (LONI)
Effects of Parameters

(a) 10000*1M files (XSEDE)

(c) 10*10G files (XSEDE)
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?