The Composite Endpoint Protocol (CEP): Scalable Endpoints for Large-Scale Flows

Eric Weigle

Computer Science and Engineering
and Center for Networked Systems
University of California, San Diego

Advisor: Andrew Chien

14 April 2006
1. Introduction
   - Problem
   - Background

2. Related Work
   - Current Approaches
   - Open Research Issues

3. Approach
   - Thesis Statement
   - Transfer Scheduling
   - Evaluation
   - Algorithms

4. Research Plan

5. Initial Results
   - Greedy Algorithm
   - Simple P2P Algorithm

6. Summary/Conclusion
How to get large numbers of nodes to efficiently transfer large amounts of data over arbitrary networks; achieving high bandwidth, low latency, low jitter?
Example Application: Visualization pipeline

- Bandwidth: 10-40Gbps
- Latency: low, but can’t beat the speed of light
- Jitter: $10\mu s$-$10$ms (depends)
Introduction

- Nodes are plentiful and cheap (see: Google)
  - > 85% of Top 500 supercomputers are clusters
  - Software: Free! MPI, Linux, Rocks, Globus
  - Beowulf ≈ $ SUV
  - How many computers do you have?

- Networking is plentiful and cheap
  - Businesses, government, ISPs, telcos own networks
  - Fiber \( \frac{1}{6} \)th the cost of 5 years ago
  - 1-10Gbps++ at edge (e.g. SC), Terabits in core
  - Dells come with 1GigE onboard, standard

- Infeasible and undesirable for one node to handle transfers
  - Nodes are weak: 1/256 system capacity < 0.5%
  - Guilder versus Moore
  - >1Gbps, or gigabytes in size: better cooperate
Where Do You Want A Lambda Today?
Problem Features

Successfully use large numbers of nodes to simply & efficiently transfer lots of data, achieving high bandwidth, low latency, low jitter?

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Management</td>
<td>Spectrum of centralized-decentralized mgmt, update heuristics, data encoding, etc.</td>
</tr>
<tr>
<td>Transfer Scheduling</td>
<td>Large space of implicit/explicit algorithms, info available? environment? goals?</td>
</tr>
<tr>
<td>Robustness</td>
<td>Replication, failure model, semantics</td>
</tr>
<tr>
<td>Protocol Design</td>
<td>Many protocols, TCP/GTP/SABUL/RBUDP... protocol selection, parameters, tuning</td>
</tr>
<tr>
<td>Interface</td>
<td>User demands, how to extend existing APIs</td>
</tr>
<tr>
<td>Security</td>
<td>Trust, PKI, encryption, grid services</td>
</tr>
</tbody>
</table>
**Problem Features**

Successfully use large numbers of nodes to simply & efficiently transfer lots of data, achieving high bandwidth, low latency, low jitter?

<table>
<thead>
<tr>
<th><strong>Issue</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Information Management</strong></td>
<td>Spectrum of centralized-decentralized mgmt, update heuristics, data encoding, etc.</td>
</tr>
<tr>
<td><strong>Transfer Scheduling</strong></td>
<td>Large space of implicit/explicit algorithms, info available? environment? goals?</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td>Replication, failure model, semantics</td>
</tr>
<tr>
<td>Protocol Design</td>
<td>Many protocols, TCP/GTP/SABUL/RBUDP... protocol selection, parameters, tuning</td>
</tr>
<tr>
<td>Interface</td>
<td>User demands, how to extend existing APIs</td>
</tr>
<tr>
<td>Security</td>
<td>Trust, PKI, encryption, grid services</td>
</tr>
</tbody>
</table>
1 Introduction
   - Problem
   - Background

2 Related Work
   - Current Approaches
   - Open Research Issues

3 Approach
   - Thesis Statement
   - Transfer Scheduling
   - Evaluation
   - Algorithms

4 Research Plan

5 Initial Results
   - Greedy Algorithm
   - Simple P2P Algorithm

6 Summary/Conclusion
Overview of Current Approaches

- Relevant work has a many-to-many abstraction and behavior
  - P.Sockets, GridFTP, BitTorrent, P2P File-Sharing, Bullet, MPI, MetaChaos, Circuit schedulers
- Not distributed memory/filesystems
- Characteristics/Mechanisms:
  - Users: Real-world vs. research systems
  - Management: Centralized vs decentralized/distributed
  - Data transfer: Striped, parallel, greedily (RRF, linear), app driven
  - Error correction: Forward, backward (erasure codes/rexmit)
  - Network Structure: 1-1, 1-M, N-N, M-N
- Target environment:
  - Internet: Slow access links, congestion, $1000\times$ speed range
  - Next-Gen: Fast links, no congestion, homogeneous nodes
## Design Evaluation

<table>
<thead>
<tr>
<th>System Name</th>
<th>Bandwidth</th>
<th>Efficiency</th>
<th>Robustness</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.Sockets</td>
<td>low</td>
<td>low</td>
<td>low</td>
<td>med</td>
</tr>
<tr>
<td>GridFTP</td>
<td>high</td>
<td>med</td>
<td>low</td>
<td>med</td>
</tr>
<tr>
<td>BitTorrent</td>
<td>med</td>
<td>med</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>Other P2P</td>
<td>med</td>
<td>med</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>Bullet</td>
<td>high</td>
<td>med</td>
<td>med</td>
<td>high</td>
</tr>
<tr>
<td>MPI</td>
<td>med</td>
<td>n/a</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>MetaChaos</td>
<td>high</td>
<td>n/a</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>Circuits</td>
<td>high</td>
<td>high</td>
<td>low</td>
<td>med</td>
</tr>
<tr>
<td>Ideal</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>
## Summary of (Most Popular) Systems’ Features

<table>
<thead>
<tr>
<th>Features</th>
<th>Type</th>
<th>Support</th>
<th>Grid-FTP</th>
<th>Meta-Chaos</th>
<th>Bit-Torrent</th>
<th>CDN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allows Data Replication</td>
<td>BR</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Uses Dynamic Feedback</td>
<td>B</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>±</td>
<td></td>
</tr>
<tr>
<td>Dual Cli+Srv Operation</td>
<td>B</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>±</td>
<td></td>
</tr>
<tr>
<td>Arbitrary Cli/Srv Mapping</td>
<td>BG</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td>Tolerates Heterogeneity</td>
<td>EG</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Scales to Large # Nodes</td>
<td>SB</td>
<td>±</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Tolerates Failures</td>
<td>R</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Allows Missing Data</td>
<td>R</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>±</td>
<td></td>
</tr>
<tr>
<td>Low-latency Operation</td>
<td>LE</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>±</td>
<td></td>
</tr>
<tr>
<td>Allows Variable-Sized Blocks</td>
<td>G</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td>Allows Non-Constant Data</td>
<td>G</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td></td>
</tr>
<tr>
<td>Needs No Special Encoding</td>
<td>G</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>±</td>
<td></td>
</tr>
<tr>
<td>Works in WANs</td>
<td>GB</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>±</td>
<td></td>
</tr>
</tbody>
</table>

(Bandwidth, Efficiency, Robustness, Latency, Scalability, Generality)
No clear solution to all the challenges.
Can’t necessarily just pick and choose: mechanisms don’t mesh

Measure all relevant (distributed) variables?
Collect/maintain distributed metadata with:
  - Maximal accuracy?
  - Maximal up-to-date-ness?
  - Minimal storage overhead?
  - Minimal network load?

Process metadata to perform transfers?
Open Research Issues (2): Transfer Scheduling

- Transfer scheduling problem is oversimplified
  - Two extremes- centralized vs. distributed
  - Large space of problems, solutions; not very well explored
  - Some versions are NP hard– but this is irrelevant (insufficient information!)

- 1-to-1 or 1-to-N aggregates ≠ native N-to-M transfers
  - Aggregates move towards *local* maxima
  - Aggregates can not be managed uniformly

- Possible directions
  - Hybrid approaches?
  - Iterative approximation techniques?
  - General heuristics appropriate for most environments?
Algorithm Design Space

Other dimensions: robustness, calculation cost, efficiency, etc. Expected contribution: exploration of this space and methods for improving performance.
Algorithm Design Space

Other dimensions: robustness, calculation cost, efficiency, etc. Expected contribution: exploration of this space and methods for improving performance.
Open Research Issues (3): Fault Tolerance

- All “high performance” systems fail easily
- Need a random-rarest-first analogue that:
  - Maintains some robustness under failure
  - Works with arbitrary data segments (not equal-sized blocks)
  - Works in the WAN with heterogeneous nodes/links
  - Does not preclude low-latency transfers

Possible directions
- Apply traditional fault tolerance approaches
- RAID-style FEC (checksum blocks)
- Hierarchical stream encoding
Remaining problem features: Leverage existing work!

Protocol design:
- Tune protocols based on system knowledge (e.g., NWS, detected state).
- Don’t worry about interaction with the network; taken care of by lower-level

Interface:
- Extend existing APIs (sockets, file transfer)
- Glue under existing applications

Security:
- Grid security services
- Public key infrastructure
- Encryption algorithms
1 Introduction
   - Problem
   - Background

2 Related Work
   - Current Approaches
   - Open Research Issues

3 Approach
   - Thesis Statement
   - Transfer Scheduling
   - Evaluation
   - Algorithms

4 Research Plan

5 Initial Results
   - Greedy Algorithm
   - Simple P2P Algorithm

6 Summary/Conclusion
Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Thesis Statement

Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Fully utilizing 10Gb-1Tb links across the wide area for large transfers (10s of Gigabytes or more) requires multiple nodes performing simultaneous transfers. When network state is only partially known, composite endpoints using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics provide a general, high-performance and robust approach.
Transfer Scheduling Framework

1. Information Management: local, global
   - Metadata is any useful information: bandwidth/latency, data location, etc.

2. Transfer Scheduling: local, global
   - Choose Host (replica) + Data
   - Explicit or implicit

- What does a transfer scheduler look like?
- It comes down to managing constraints...
Transfer Scheduling Constraints (1): Data

Servers

- "Memory at 0xC03f0000"
- "File /usr/File1"
- "File /usr/File2"

Clients

- "File /tmp/FullCopy"
- "File /tmp/dat"
- "File /tmp/dat"
- "File /tmp/dat"
- "File /tmp/dat"

Existing Data

Desired
Transfer Scheduling Constraints (1): Data

Existing DesiredLogical

<table>
<thead>
<tr>
<th>File Blocks</th>
<th>Servers</th>
<th>File Blocks</th>
<th>Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 5</td>
<td>&quot;Memory at 0xC03f0000&quot;</td>
<td>&quot;File /tmp/FullCopy&quot;</td>
</tr>
<tr>
<td>2</td>
<td>2 6</td>
<td>&quot;File /tmp/dat&quot;</td>
<td>&quot;File /tmp/dat&quot;</td>
</tr>
<tr>
<td>3</td>
<td>3 7</td>
<td>&quot;File /tmp/dat&quot;</td>
<td>&quot;File /tmp/dat&quot;</td>
</tr>
<tr>
<td>4</td>
<td>4 8</td>
<td>&quot;File /tmp/dat&quot;</td>
<td>&quot;File /tmp/dat&quot;</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>&quot;File /tmp/dat&quot;</td>
<td>&quot;File /tmp/dat&quot;</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>&quot;File /tmp/dat&quot;</td>
<td>&quot;File /tmp/dat&quot;</td>
</tr>
<tr>
<td>7</td>
<td>7 11</td>
<td>&quot;File /usr/File1&quot;</td>
<td>&quot;File /usr/File1&quot;</td>
</tr>
<tr>
<td>8</td>
<td>8 12</td>
<td>&quot;File /usr/File1&quot;</td>
<td>&quot;File /usr/File1&quot;</td>
</tr>
<tr>
<td>9</td>
<td>9 13</td>
<td>&quot;File /usr/File1&quot;</td>
<td>&quot;File /usr/File1&quot;</td>
</tr>
<tr>
<td>10</td>
<td>10 14</td>
<td>&quot;File /usr/File1&quot;</td>
<td>&quot;File /usr/File1&quot;</td>
</tr>
<tr>
<td>11</td>
<td>11 15</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>12</td>
<td>12 16</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>13</td>
<td>13 17</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>14</td>
<td>14 18</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>15</td>
<td>15 19</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>16</td>
<td>16 20</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>&quot;File /usr/File2&quot;</td>
<td>&quot;File /usr/File2&quot;</td>
</tr>
</tbody>
</table>

Existing Data Logical Desired
Transfer Scheduling Constraints (2): Physical

Servers

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Network

10Gbps

<table>
<thead>
<tr>
<th>7</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
</tr>
</tbody>
</table>

10Gbps

Clients

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

1Gbps

<table>
<thead>
<tr>
<th>6</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

1Gbps

<table>
<thead>
<tr>
<th>11</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>15</td>
</tr>
</tbody>
</table>

1Gbps

<table>
<thead>
<tr>
<th>16</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>20</td>
</tr>
</tbody>
</table>
Problem Representation: Transfer Constraint Graph

- General, powerful, simple, efficient to modify/update
- Allows rigorous analysis and efficient implementation
- Segment formulation versus Block/bitfield/stateless:
  - Segments have high overhead for small ranges
  - Blocks have problems with strided access, irregular data
  - Segments more general
  - Stateless: does not capture problem semantics
- Best representation is application specific
- → Use segments, optimized to blocks when possible
- Structure
  - Vertices: Nodes, Data ranges
  - (Directed) Edges: Data Supply, Demand, Network Bandwidth, Latency
Sample Constraint Graph

**Servers**

- "Memory at 0xC03f0000"
- "File /usr/File1"

**Data Segments**

Edges: Data Location

- 1–5
- 6
- 7–8
- 9–10
- 11–12
- 13–14

- 10Gbps, 30ms
- 622Mbps, 10ms
- 1Gbps, 60ms

**Clients**

- "File /tmp/FullCopy"
- 132.239.226.8
- Linux 2.4/x86
- "File /tmp/dat"
- "File /tmp/dat"

Edges: Network Performance

- 1Gbps, 60ms
- 622Mbps, 10ms
- 10Gbps, 30ms
Evaluate Claims: General, High Performance, Robust

- **Generality**: test wide variety of configurations, applications
- **Bandwidth**: aggregate transfer rate for all nodes
- **Latency**: responsiveness and jitter
  - Request Latency: data request → initiation of transfer
  - Block Delay: expected → actual data arrival time
- **Scalability**: a ratio; speedup, or old/new latency.
  Large numbers of nodes (10k goal), data segments
- **Efficiency**: % capacity we acquire, subject to user constraints
- **Robustness**: Drop, delay, corrupt packets. Links/nodes up and down.
- **Not fairness**: global performance trumps nodes’.
  Transports (e.g. TCP, GTP) can guarantee fairness if required.

Analysis → Simulation → Emulation → Real-world tests
Other dimensions: robustness, calculation cost, efficiency, etc. Expected contribution: exploration of this space and methods for improving performance.
Centralized, Total Information: Linear Programming

- Transfer scheduling: a maximization problem
- Objective: maximal rate
- Constraints: sender/receiver data, termination, network speeds
- Many problems:
  - Doesn’t allow any dynamic behavior: network bandwidth, receiver sharing, etc.
  - Conversion to equations is expensive
  - Algorithm itself is expensive

Constraints Overview:
Data demand:
- \( r_i > 0 \) for links to a range.

Capacity:
- \( 0 \leq \sum s_i \leq 1 \) where links \( s_i \) from one server
- \( 0 \leq \sum r_i \leq 1 \) where links \( r_i \) supplies some range

Supply and Demand:
- \( \sum \text{ingress capacity} = \sum \text{egress capacity} \)
Little or No State Information

- Flooding and Epidemic-based algorithms
- Useful when most/all nodes require the same data
- Erasure codes: encode data so any subset $n$ of $k$ encoded blocks allow reconstruction of original data
- Nearly any host can provide some useful data
- Good aspects
  - Low management overhead
  - (Relatively) simple to implement
  - Efficient in common cases
- Bad aspects:
  - Overhead for encoding/decoding
  - High latency, decoding issues with large files
  - Problems optimizing for bottlenecks (stragglers)
- How best to encode over segments is an open question.
(de)Centralized, Partial Information: Greedy

- Transfer scheduling: too hard!
- Most projects just use heuristics:
  - Block scheduling, such as random-rarest-first (e.g. BitTorrent)
    - Maximizes objective locally (most rare)
  - Graph-based scheduling (e.g. GTP, CEP)
    - Calculate supply and demand for data
    - Locally divide supply to fulfill demand
    - Good for inhomogeneous supply/demand (hot blocks)
- Hybrid Satellite/Terrestrial Scheduling
  - Specialized multicast/recovery
  - Allocate transfers inversely proportional to current load
  - Heuristics for congestion avoidance, metadata update
The Primary CEP Greedy Algorithm (1/2)

1. Calculate (proportional) demand:
   - For each receiver:
     - Sum total size = total data required = size of all linked segments.
     - Label each link with (receiver speed * (segment size)/(total size)).
   - For each segment, label with total demand (Mbps):
     - Sum receiver demands for that segment (marked on links).

2. Calculate proportional supply. For each sender:
   - Sum demand for all segments sender provides (mark on segments).
   - For each linked segment, provide to that segment bandwidth:
     (sender speed)*(segment demand / total demand)
The Primary CEP Greedy Algorithm (1/2)

1. Calculate (proportional) demand:
   - For each receiver:
     - Sum total size = total data required = size of all linked segments.
     - Label each link with (receiver speed * (segment size)/(total size)).
   - For each segment, label with total demand (Mbps):
     - Sum receiver demands for that segment (marked on links).

2. Calculate proportional supply. For each sender:
   - Sum demand for all segments sender provides (mark on segments).
   - For each linked segment, provide to that segment bandwidth:
     (sender speed)*(segment demand / total demand)
Create rate schedule. For each sender:
- For each segment they provide: equally allocate rate to receivers. Send at rate \( \min(\text{total segment rate}/(\text{# of receivers}), \text{receiver demand}) \) to each receiver.

Execute transfer, and reallocate flow as required:
- Update the graph as flows progress
- Detect stragglers/bottlenecks (last to finish)
- Allocate flow from any path finishing earlier.
Create rate schedule. For each sender:
- For each segment they provide: equally allocate rate to receivers. Send at rate \( \min\left(\frac{\text{total segment rate}}{\text{# of receivers}}, \text{receiver demand}\right) \) to each receiver.

Execute transfer, and reallocate flow as required:
- Update the graph as flows progress
- Detect stragglers/bottlenecks (last to finish)
- Allocate flow from any path finishing earlier.
Greedy Algorithm Example

1. **Calculate proportional Demand**
   - D has speed 3, ABC speed 1.
   - \(\frac{9-0+1}{99-0+1} \times 3 = 0.3\)
   - \(\frac{69-10+1}{99-0+1} \times 3 = 1.8\)
   - \(\frac{99-70+1}{99-0+1} \times 3 = 0.9\)

2. **Calculate proportional Supply**
   - Readjust demand
   - \(\frac{0.8}{0.8+0.9} = 0.47\)
   - \(\frac{0.9}{0.8+0.9} = 0.53\)

3. **Reset proportions:**
   - Supply and Demand, weight vs. Mbps

4. **Iteratively transfer**
Greedy Algorithm Example

1. Calculate proportional Demand
   - D has speed 3, ABC speed 1.
   - \((9-0+1)/(99-0+1) \times 3 = 0.3\)
   - \((69-10+1)/(99-0+1) \times 3 = 1.8\)
   - \((99-70+1)/(99-0+1) \times 3 = 0.9\)

2. Calculate proportional Supply
   - Readjust demand
     - \(0.8/(0.8+0.9)=.47\)
     - \(0.9/(0.8+0.9)=.53\)

3. Reset proportions: Supply and Demand, weight vs. Mbps

4. Iteratively transfer
Greedy Algorithm Example

1. Calculate proportional Demand
   - D has speed 3, ABC speed 1.
   - \((9-0+1)/(99-0+1) \times 3 = 0.3\)
   - \((69-10+1)/(99-0+1) \times 3 = 1.8\)
   - \((99-70+1)/(99-0+1) \times 3 = 0.9\)

2. Calculate proportional Supply
   - Readjust demand
   - \(0.8/(0.8+0.9) = 0.47\)
   - \(0.9/(0.8+0.9) = 0.53\)

3. Reset proportions:
   Supply and Demand, weight vs. Mbps

4. Iteratively transfer
Greedy Algorithm Example

1. Calculate proportional Demand
   - D has speed 3, ABC speed 1.
   - \( \frac{9-0+1}{99-0+1} \times 3 = 0.3 \)
   - \( \frac{69-10+1}{99-0+1} \times 3 = 1.8 \)
   - \( \frac{99-70+1}{99-0+1} \times 3 = 0.9 \)

2. Calculate proportional Supply
   - Readjust demand
   - \( \frac{0.8}{0.8+0.9} = 0.47 \)
   - \( \frac{0.9}{0.8+0.9} = 0.53 \)

3. Reset proportions: Supply and Demand, weight vs. Mbps

4. Iteratively transfer

Eric Weigle
The Composite Endpoint Protocol (CEP)
Greedy Algorithm Example

1. Calculate proportional Demand
   - D has speed 3, ABC speed 1.
   - \( \frac{9-0+1}{99-0+1} \times 3 = 0.3 \)
   - \( \frac{69-10+1}{99-0+1} \times 3 = 1.8 \)
   - \( \frac{99-70+1}{99-0+1} \times 3 = 0.9 \)

2. Calculate proportional Supply
   - Readjust demand
   - \( \frac{0.8}{0.8+0.9} = 0.47 \)
   - \( \frac{0.9}{0.8+0.9} = 0.53 \)

3. Reset proportions: Supply and Demand, weight vs. Mbps

4. Iteratively transfer
Greedy Algorithm Example

1. Calculate proportional Demand
   - D has speed 3, ABC speed 1.
   - \( \frac{9-0+1}{99-0+1} \times 3 = 0.3 \)
   - \( \frac{69-10+1}{99-0+1} \times 3 = 1.8 \)
   - \( \frac{99-70+1}{99-0+1} \times 3 = 0.9 \)

2. Calculate proportional Supply
   - Readjust demand
   - \( \frac{0.8}{0.8+0.9} = 0.47 \)
   - \( \frac{0.9}{0.8+0.9} = 0.53 \)

3. Reset proportions: Supply and Demand, weight vs. Mbps

4. Iteratively transfer
A Simple Peer-to-Peer Greedy Algorithm

Satellites: fast, good for broadcast, but errors
Internet: slower, bad for broadcast, more reliable
Broadcast with satellite, recover errors with Internet

Terrestrial transfer: *transfer scheduling problem!*
- Provide, request metadata from master
- Master replies with least-loaded node
- Request blocks from peers
Constraint Weakening (Relaxation)

- Powerful optimization techniques
- On-demand links (UCLP) change structure of network constraints
  - Remove bottlenecks by requesting a lambda
- Erasure codes change structure of data constraints
  - Data segments no longer one-to-one
- Applications semantics may change effective data constraints
  - e.g. MPEG encoded files, redundancy, data regeneration
- How to best exploit these features?
Addressing Research Questions (What is “CEP”)...

- **Information Management:**
  - Preferentially initialize state using:
    - NWS, cached data, config file, or interface speeds.
  - Measure and update speed/latency as transfer progresses (piggyback)
  - Simple flow control (Source-quench) avoids update overload

- **Transfer Scheduling:**
  - Greedy algorithms + heuristics are the way to go.
  - Centralized bottleneck detection & global optimization
  - Decentralized data transfers, and constrained optimization
  - “Locally optimal” \( \equiv \) There exists no path from a receiver to a sender with desired data and available capacity

- **Robustness:**
  - Tolerate failure of any node whose data is replicated or unnecessary.
  - Lots of engineering
## Calendar

<table>
<thead>
<tr>
<th>Stage</th>
<th>Deliverables</th>
<th>Len</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analysis</strong></td>
<td>State problem, objectives</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>Background literature search</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>Theoretical background &amp; algorithms</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td><strong>Design/Impl.</strong></td>
<td>Develop Transfer Scheduling Algorithms</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>File transfer, Sockets, XIO, Fuzzy APIs</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>Sockets, XIO, GTP implementation</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>Implement different approaches in $\text{ns}$</td>
<td>1.5</td>
<td>Sp ’06</td>
</tr>
<tr>
<td></td>
<td>Assemble emulation infrastructure</td>
<td>1</td>
<td>Su ’06</td>
</tr>
<tr>
<td></td>
<td>Distributed Information Management</td>
<td>1.5</td>
<td>Su ’06</td>
</tr>
<tr>
<td></td>
<td>Integration with GridFTP or others</td>
<td>1</td>
<td>Fa ’06</td>
</tr>
<tr>
<td><strong>Eval.</strong></td>
<td>Paper - initial implementation (CCGRID05)</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>High-level simulation</td>
<td>3</td>
<td>*Sp ’06</td>
</tr>
<tr>
<td></td>
<td>Paper - wireless+loss simulations (P2P06?)</td>
<td>-</td>
<td>*Sp ’06</td>
</tr>
<tr>
<td></td>
<td>Large scale emulation tests</td>
<td>1</td>
<td>Su ’06</td>
</tr>
<tr>
<td></td>
<td>Real-world tests, GridFTP integration</td>
<td>2</td>
<td>Fa ’06</td>
</tr>
<tr>
<td></td>
<td>Demos at SC2006</td>
<td>1</td>
<td>Fa ’06</td>
</tr>
<tr>
<td><strong>Thesis</strong></td>
<td>Write Dissertation, Prepare Defense</td>
<td>6</td>
<td>Wi/Sp ’07</td>
</tr>
</tbody>
</table>
Design & Implementation

- Initial implementations: SPED vs Threads
- Sockets, GTP, XIO Networking Stacks
- Support x86, Opteron, PPC hardware
- Implementation of core features in ns
- Erasure coding in transfer scheduling
- Build emulation infrastructure
  - Xen/ModelNet/microgrid technology...
- Integration with GridFTP
- DHT-based version for metadata management
  - Pastry, Chord, or Bamboo? MACEDON/MACE
- Demonstrations for SC2006
Evaluation

(Initial results: next section)

✓ Regression testing framework for CEP functionality
✓ Performance across campus
   (RockStar, CSAG, NCMIR, clusters)
≈ Performance across WAN
   (CaveWave, clusters in Chicago & Amsterdam)
≈ Simulations for all realistic target configurations
≈ Robustness under faults (packet, link, node failure)
≈ GridFTP(CEP) versus other versions, transports
× Evaluate centralized and DHT environments above
× Scalability with virtual nodes, various topologies
   (star, BRITE, NLR)
1 Introduction
   - Problem
   - Background

2 Related Work
   - Current Approaches
   - Open Research Issues

3 Approach
   - Thesis Statement
   - Transfer Scheduling
   - Evaluation
   - Algorithms

4 Research Plan

5 Initial Results
   - Greedy Algorithm
   - Simple P2P Algorithm

6 Summary/Conclusion
Basic goal: compose multiple flows into a logical connection

2 Gigabytes of data to/from user-level memory

Aggregate bandwidth for last node to terminate
Demonstrate value in exploiting heterogeneity

16 nodes with Ethernet, 10 slow (100Mbps) 6 fast (1Gbps)

1GB transfer
Exploiting Flexibility in Problem Constraints

- Performance benefit from access flexibility (via replica selection)
- 8 heterogeneous nodes, four slow and four fast.
- Vary the amount of data each may access
ns-2 simulations with AT&T core network

Satellite has a 2% loss rate

Error correction can be changed on-the-fly
Scatter plot of block delays for a 100MB transfer
- Latency dominated by network delay (4-80ms)
- $\approx \frac{2}{3}$ of packets received in 200ms.
Summary & Conclusion

- Clear requirements for high-performance communication
  - High bandwidth, efficient, scalable, robust, low cost, simple, general, (locally) optimal
- Vast body of past and present work, lots of approaches, but they target different environments, problems.

These requirements can be met by composite endpoints, using a hybrid centralized/decentralized transfer scheduling approach via graph-structured algorithms and feedback heuristics.
Questions?