Peer-to-Peer: Part III continued

Theophilus Benson
CS 0138 Spring ‘18
Matching Issues to Techniques

Issues
• What is the unit of replication
• Finding a nearby copy
• Updating all copies
• Access control
• Integrity
• Fault tolerance

Techniques
• End-to-end encryption, self-certifying data
• Tapestry self-organizing routing infrastructure
• Erasure coding for durability (m-of-n)
• Byzantine update commitment
• Dynamic replica management
Benefits of Replication

• Example with 2 replicas
  – \( N = \) total number of servers = \(10^6\)
  – \( M = \) number of servers expected to be down = \(10^5\) (10%)

• What is the probability that at least one copy of a replicated block is available?

\[
\sum_{i=0}^{1} \binom{M}{i} \binom{N-M}{2-i} \frac{1}{\binom{N}{2}}
\]

• = 0.99
Erasure Codes

• Divide an object into $m$ equal-size fragments and code them as $n$ equal-size fragments, $n > m$
  – $m/n = \text{rate of encoding} = r$
  – $1/r = \text{multiplicative storage increase}$

• Original object can be reconstructed from any $m$ out of $n$ fragments
  – handles $n-m$ “erasures”
In Practice

- Original block is stored in Tapestry by its BGUID

- Erasure-coded fragments stored by id \( f(\text{BGUID}, k) \), where \( k \) is the block number

- Can be replicated at will

- These are called *archival copies*
The Down Side …

• Fetching and assembling the fragments is expensive
  – how can it be made cheaper (on average)?

```
Get (f(BGUID,0))
Get (f(BGUID,1))
Get (f(BGUID,2))
```
Caching

• If you’ve reconstructed a block
  – publish it in tapestry
  – available for others until you delete it
Matching Issues to Techniques

**Issues**
- What is the unit of replication
- Finding a nearby copy
- Updating all copies
- Access control
- Integrity
- Fault tolerance

**Techniques**
- End-to-end encryption, self-certifying data
- Tapestry self-organizing routing infrastructure
- Erasure coding for durability (m-of-n)
- Byzantine update commitment
- Dynamic replica management
Distributing a Mutable File

- updates can take a while to propagate
Distributing a Mutable File

- Updates at any computer
- Collectively determine effective order
Distributing a Mutable File

- Updates just at one computer
- It propagates them to the others
- Read from any replica
Issues

• Should we trust the server holding the primary copy?
  – should it be special?
    - No!
A Compromise

- Updates go to master group
- Master group = Inner Ring
- It propagates them to the others
- Read from any replica
Inner Ring

• A collection of servers are responsible for the primary copy
  – called the *inner ring*

• Collectively make updates
  – together they hold the *primary replica*

• File is identified by its *active GUID (AGUID)*
  – secure hash of application-specific name and owner’s public key

• Each inner-ring server publishes the AGUID in Tapestry
  – each holds a copy of the current VGUID
OceanStore File

BGUID  block GUID  secure hash of a block of data
VGUID  version    secure hash of the root block of a version
AGUID  active     Effective name of file

AGUID  active GUID  Effective name of file

VGUID  version GUID  secure hash of the root block of a version

BGUID  block GUID  secure hash of a block of data
Byzantine Generals Problem

Attack!

She said Attack!

Attack!

She said Retreat!

She said Retreat!

She said Attack!

She said Attack!
OceanStore File

BGUID: block GUID
  secure hash of a block of data

VGUID: version
  secure hash of the root block of a version

AGUID: active
  effective name of file

AGUID: active GUID
  effective name of file

Heartbeat Certificate

VGUID

Root Block

Indirect Block

Data Block 1
  Data Block 2
  Data Block 3
  Modified Data

Indirect Block

Data Block 4
  Data Block 1
  Modified Data
  Data Block 4
  Modified Data Block 4

Root Block

Indirect Block

Data Block 4
  Data Block 1
  Modified Data
  Data Block 4
  Modified Data Block 4

Indirect Block

Data Block 2
  Data Block 3
Reading from Oceanstore

Get (BGUID) → Tapestry

AGUID ←

Get (AGUID) →

VGUID ←

Get (VGUID) →

RootBlock ←
Almost There …

• What about:
  – getting notified about file updates?
  – finding the AGUID in the first place?
Putting it all together: the Path of a Write

![Diagram showing the path of a write involving you, primary replicas, archival servers, and your friends.](image-url)
## Performance

<table>
<thead>
<tr>
<th>Phase</th>
<th>LAN</th>
<th></th>
<th>WAN</th>
<th></th>
<th>Predominant operations in benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linux NFS</td>
<td>Pond</td>
<td>Linux NFS</td>
<td>Pond</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>1.9</td>
<td>0.9</td>
<td>2.8</td>
<td>Read and write</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
<td>11.0</td>
<td>9.4</td>
<td>16.8</td>
<td>Read and write</td>
</tr>
<tr>
<td>3</td>
<td>1.1</td>
<td>1.8</td>
<td>8.3</td>
<td>1.8</td>
<td>Read [status only]</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>1.5</td>
<td>6.9</td>
<td>1.5</td>
<td>Read</td>
</tr>
<tr>
<td>5</td>
<td>2.6</td>
<td>21.0</td>
<td>21.5</td>
<td>32.0</td>
<td>Read and write</td>
</tr>
<tr>
<td>Total</td>
<td>4.5</td>
<td>37.2</td>
<td>47.0</td>
<td>54.9</td>
<td></td>
</tr>
</tbody>
</table>
Secondary Replicas

• May be stored on any server
• Hold copies of heartbeat
  – new copies pushed to them by inner-ring servers
    - new secondary replicas find and link to existing ones via Tapestry
    - forms tree of replicas
What Else?

• Not described here
  – all file blocks are encrypted by clients
  – directories
  – access control

• Not implemented in Pond
  – automatic repair
    - periodic verification of data
    - periodic copying of data to new disks
  – introspection
    - adaptive system management

• Not clear
  – management of inner rings
Topics

• Layering

• Network Metrics

• Reliability

• Congestion Control

• Routing
Fallacies

• The network is reliable
• Latency is zero
• Bandwidth is infinite
• The network is secure
• Topology doesn’t change
• There is one administrator
• Transport cost is zero
• The network is homogeneous
Analogy to Delivering a Letter

• **Messages are self-contained**
  – Post: a message in an envelope
  – Internet: data in a packet

• **Interior routers forward based on destination address**
  – Post: zip code, then street, then building, then apartment number (then the right individual)
  – Internet: progressively smaller blocks of IP addresses, then your computer (then the right application)

• **Simple, robust.**
  – More sophisticated things go at the *ends of the network*
Internet layering = “Protocol stack”

- **Separation of concerns**
  - Break problem into separate parts
  - Solve each one independently
  - Tie together through common interfaces: abstraction
  - Encapsulate data from the layer above inside data from the layer below
  - Allow independent evolution
Layers, Services, Protocols

Layer N+1

Layer N

Layer N-1

**Service**: abstraction provided to layer above
**API**: concrete way of using the service

**Protocol**: rules for communication within same layer

Layer N uses the services provided by N-1 to implement its protocol and provide its own services
Layering + Encapsulation

Application
Transport
Network
Link
Physical
Layering + Encapsulation

Application
Transport
Network
Link
Physical

Application
Transport
Network
Link
Physical
Layering + Encapsulation
Layering + Encapsulation

Application
Transport
Network
Link
Physical

VIDEO  T  N  L

Application
Transport
Network
Link
Physical
Layering + Encapsulation

Application
Transport
Network
Link
Physical

VIDEO  T  N  L  P
Layering + Encapsulation

Physical
Link
Network
Transport
Application

VIDEO
T
N
L
P
Layering + Encapsulation

Application
Transport
Network
Link
Physical

Application
Transport
Network
Link
Physical

VIDEO
TNL
Layering + Encapsulation
Layering + Encapsulation
Layering + Encapsulation
Layer 7: Application layer

- Communication of whatever you want
- Can use whatever transport(s) is(are) convenient/appropriate
- Freely structured
- Examples:
  - Skype (UDP)
  - SMTP = email (TCP)
  - HTTP = web (TCP)
  - Online games (TCP and/or UDP)
Layer 4: Transport layer

- End-to-end communication between **processes**
- Different types of services provided:
  - UDP: unreliable **datagrams**
  - TCP: **reliable** byte stream
- “Reliable” = keeps track of what data were received properly and retransmits as necessary
Layer 3: (Inter)network layer

- Bridges multiple “subnets” to provide *end-to-end internet* connectivity between nodes
- Provides *global* addressing (IP addresses)
- Only provides *best-effort* delivery of data (i.e., no retransmissions, etc.)
- Works across different link technologies

Different for each Internet “hop”

Lowercase-i “internet” = network of networks.
Uppercase-i Internet = “*the Internet*”
All ASes are not equal
Layer 2: Link layer

- Framing and transmission of a collection of bits into individual messages sent across a single subnetwork (one physical topology)
- Provides local addressing (MAC)
- May involve multiple physical links
- Often the technology supports broadcast: every “node” connected to the subnet receives

Examples:
- Modern Ethernet
- WiFi (802.11a/b/g/n/etc)
Layer 1: Physical layer

- **Encoding** of bits to send over a single physical link
- **Examples:**
  - Voltage levels
  - RF modulation
  - Photon intensities

*FSK: Frequency Shift Keying*
Layers, Services, Protocols

**Application**
- **Service**: user-facing application.
- **Functions**: Application specific

**Transport**
- **Service**: multiplexing applications
- **Functions**: Connection establishment/termination, error control, flow control

**Network**
- **Service**: move packets to any other node in the network
- **Functions**: Routing, addressing

**Link**
- **Service**: move frames to other node across link.
- **Functions**: Framing, media access control, error checking

**Physical**
- **Service**: move bits to other node across link
- **Functions**: Convert bits to signal
Topics

• Layering

• Network Metrics

• Reliability

• Congestion Control

• Routing
Latency Breakdown

Transmission Delay

Queuing Delay

Processing Delay

Switch

CPU

NIC

Switch

NIC

NIC

NIC

NIC
Latency

• Processing
  – Per message, small, limits throughput
  – e.g. \( \frac{100 \, Mb}{s} \times \frac{pkt}{1500 \, B} \times \frac{B}{8b} \approx 8333 \, pkt/s \) or 120\( \mu \)s/pkt

• Queue
  – Highly variable, offered load vs outgoing b/w

• Transmission
  – Size/Bandwidth

• Propagation
  – Distance/Speed of Light
Sending Frames Across

Transmission Delay

Propagation Delay

Latency
Sending Frames Across

Throughput: bits / s
Which matters most, bandwidth or delay?

- How much data can we send during one RTT?
- *E.g.*, send request, receive file

- For small transfers, latency more important, for bulk, throughput more important
Performance Metrics

• **Throughput** - Number of bits received/unit of time
  – *e.g.* 10Mbps

• **Goodput** - *Useful* bits received per unit of time

• **Latency** – How long for message to cross network
  – Process + Queue + Transmit + Propagation

• **Jitter** – Variation in latency
If you send 30 Mbps... but every packet contains:
  50 bits of data and
  50 bits of header.

What is your GoodPut?
Maximizing Throughput

• Can view network as a pipe
  – For full utilization want bytes in flight $\geq$ bandwidth $\times$ delay
  – But don’t want to overload the network (future lectures)
How To Fill The Pipe

• Blast packets into the Network

• What if protocol doesn’t involve bulk transfer?
  – Get throughput through concurrency – service multiple clients simultaneously
Topics

• Layering

• Network Metrics

• Reliability

• Congestion Control

• Routing