Enabling near-VoD via P2P Networks

Siddhartha Annapureddy
Saikat Guha, Dinan Gunawardena
Christos Gkantsidis, Pablo Rodriguez

World Wide Web, 2007
Motivation

- Growing demand for distribution of videos
  - Distribution of home-grown videos
  - BBC wants to make its documentaries public
  - People like sharing movies (in spite of RIAA)
- Video distribution on Internet very popular
  - About 35% of the traffic

- What we want?
  - *(Near) Video-on-Demand (VoD)*
  - User waits a little, and starts watching the video
First Generation Approaches

- **Multicast support in network**
  - Hasn’t taken off yet 😞
  - Research yielded good ideas
    - Pyramid schemes

- **Dedicated Infrastructure**
  - Akamai will do
  - Costs big bucks
Recent Approaches

- **BitTorrent Model**
  - P2P mesh-based system
  - Video divided into blocks
  - Blocks fetched at random
    - Preference for rare blocks

- High throughput through cooperation
- Cannot watch until entire video fetched
Current Approaches

- Distribute from a server farm
  - YouTube, Google Video

- Scalability problems
  - Bandwidth costs very high

- Poor quality to support high demand

- Censor-ship constraints
  - Not all videos welcome at Google video
  - Costs to run server farm prohibitive
Live Streaming Approaches

- Provides a different functionality
  - User arriving in middle of a movie cannot watch from the beginning

- Live Streaming
  - All users interested in same part of video
  - Narada, Splitstream, PPLive, CoolStreaming
  - P2P system caching windows of relevant chunks
Challenges for VoD

Given this motivation for VoD, what are the challenges?

- **Near VoD**
  - Small *setup time* to play videos

- **High sustainable goodput**
  - Largest slope osculating the block arrival curve
  - Highest video encoding rate system can support

**Goal**: Do block scheduling to achieve low setup time and high goodput
System Design – Principles

- Keep bw load on server to a minimum
  - Server connects and gives data only to few users
    - Leverage participants to distribute to other users
  - Maintenance overhead with users to be kept low

- Keep bw load on users to minimum
  - Each node knows only a small subset of user population (called neighbourhood)
    - Bitmaps of blocks
System Design – Outline

- Components based on BitTorrent
  - Central server (tracker + seed)
  - Users interested in the data
- Central server
  - Maintains a list of nodes
- Each node finds neighbours
  - Contacts the server for this
  - Another option – Use gossip protocols
- Exchange data with neighbours
  - Connections are bi-directional
Overlay-based Solutions

- Tree-based approach
  - Use existing research for IP Multicast
  - Complexity in maintaining structure

- Mesh-based approach
  - Simple structure of random graphs
  - Robust to churn
Simulator – Outline

- All nodes have unit bandwidth capacity
  - 500 nodes in most of our experiments
  - Heterogeneous nodes not described in talk
- Each node has 4-6 neighbours
- Simulator is round-based
  - Block exchanges done at each round
  - System moves as a whole to the next round
- File divided into 250 blocks
  - Segment is a set of consecutive blocks
  - Segment size = 10
- Maximum possible goodput – 1 block/round
Why 95\textsuperscript{th} percentile?

- \((x, y)\)
  - y nodes have a goodput which is at least \(x\).

- Red line at top shows 95\textsuperscript{th} percentile

- 95\textsuperscript{th} percentile a good indicator of goodput
  - Most nodes have at least that goodput
Feasibility of VoD – What does block/round mean?

- 0.6 blocks/round with 35 rounds setup time

- Two independent parameters
  - 1 round ≈ 10 seconds
  - 1 block/round ≈ 512 kbps

- Consider 35 rounds a good setup time
  - 35 rounds ≈ 6 min

- Goodput = 0.6 blocks/round
  - Size of video = 250 / 0.6 ≈ 70 min
  - Max encoding rate = 0.6 * 512 > 300 kbps
Naïve Scheduling Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>3 6 8 1 4 2 7 5</td>
</tr>
<tr>
<td>Sequential</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
<tr>
<td>Segment-Random</td>
<td>4 1 3 2 6 5 8 7</td>
</tr>
</tbody>
</table>

Segment-Random Policy

- Divide file into segments
- Fetch blocks in random order inside same segment
- Download segments in order.
Naïve Approaches – Random

- Each node fetches a block at random
- Throughput – High as nodes fetch disjoint blocks
  - More opportunities for block exchanges
- Goodput – Low as nodes don’t get blocks in order
  - 0 blocks/round
Naïve Approaches – Sequential

- Each node fetches blocks in order of playback
- **Throughput** – Low as fewer opportunities for exchange
  - Increase this

< 0.25 blocks/round

Sequential
Random
Naïve Approaches – Segment-Random Policy

- **Segment-random**
  - Fetch random block from the segment the earliest block falls in
  - Increases the number of blocks in progress

< 0.50 blocks/round
Throughput is number of block exchanges system-wide

Period between two valleys corresponds to a segment

Throughput could be improved at two places – Reduce time:
- To obtain last blocks of current segment
- For initial blocks of segment to propagate
Mountains and Valleys

- Local rarest
  - Fetch the rarest block in the current segment
  - System progress improves 7% (195.47 to 208.61)
Naïve Approaches – Rarest-client

- Segment-Random
- Local-rarest

- How can we improve this further?
- Pre-fetching
  - Improve availability of blocks from next segment
**Pre-fetching**

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>6</th>
<th>8</th>
<th>1</th>
<th>4</th>
<th>2</th>
<th>7</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Sequential</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Segment-Random</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Pre-fetching</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

- **Pre-fetching**
  - Fetch blocks from the next segment with a small probability
  - Creates seeds for blocks ahead of time
  - Trade-off: block is not immediately useful for playback
Pre-fetching

- Pre-fetching with probability 0.2

- Throughput:
  - 9% improvement
  - 195.47 to 212.21

- Get better with Network Coding!!!
NetCoding – How it Helps?

- A has blocks 1 and 2
- B gets 1 or 2 with equal prob. from A
- C gets 1 in parallel
- If B downloaded 1
- Link B-C becomes useless

- Network coding routinely sends $1 \oplus 2$
Network Coding – Mechanics

- Coding over blocks B1, B2, ..., Bn
- Choose coefficients c1, c2, ..., cn from finite field
- Generate encoded block: \( E1 = \sum_{i=1..n} c_i B_i \)
- Without n coded blocks, decoding cannot be done
  - Setup time at least the time to fetch a segment
Benefits of NetCoding

Throughput:
- 39% improvement
- 271.20 with NC
- 212.21 with pre-fetching and no NC
- 195.47 initial

Pre-fetching segments moderately beneficial
Implementation

- With the above insights, built a C# prototype
  - 25K lines of code

- Rest of talk
  - Identify problems when nodes arrive at different times, nodes with heterogeneous capacities
  - Address these issues with improvements to the basic Netcoding scheme
  - Note – All evaluation done with the prototype
Segment Scheduling

- NetCoding good for fetching blocks in a segment, but cannot be used across segments
  - Because decoding requires all encoded blocks

- **Problem**: How to schedule fetching of segments?
  - Until now, sequential segment scheduling
    - Segment pre-fetching beneficial
    - Results in rare segments when nodes arrive at different times

- Segment scheduling algorithm to avoid rare segments
Segment Scheduling

- A has 75% of the file
- Flash crowd of 20 Bs join with empty caches
- Server shared between A and Bs
  - Throughput to A decreases, as server is busy serving Bs
- Initial segments served repeatedly
  - Throughput of Bs low because the same segments are fetched from server
Segment Scheduling – Problem

- 8 segments in the video; A has 75% = 6 segments, Bs have none
- Green is popularity 2, Red is popularity 1
- Popularity = # full copies in the system

<table>
<thead>
<tr>
<th>Segment #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popularities</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
## Segment Scheduling – Problem

### Table

<table>
<thead>
<tr>
<th>Segment #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popularities</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- Instead of serving A ...
## Segment Scheduling – Problem

The server gives segments to Bs

<table>
<thead>
<tr>
<th>Segment #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popularities</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Segment Scheduling – Problem

- A’s goodput plummets
- Get the best of both worlds – Improve A and Bs
  - Idea: Each node serves only rarest segments in system
Segment Scheduling – Algorithm

<table>
<thead>
<tr>
<th>Segment #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popularities</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- When dst node connects to the src node...
  - Here, dst is B, src is Server
**Segment Scheduling – Algorithm**

<table>
<thead>
<tr>
<th>Segment #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟡</td>
<td>🟡</td>
</tr>
<tr>
<td>A</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popularities</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- src node sorts segments in order of popularity
  - Segments 7, 8 least popular at 1
  - Segments 1-6 equally popular at 2
### Segment Scheduling – Algorithm

<table>
<thead>
<tr>
<th>Segment #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Popularities</strong></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- src considers segments in sorted order one-by-one, and serves dst
  - Either completely available at src, or
  - First segment required by dst
### Segment Scheduling – Algorithm

<table>
<thead>
<tr>
<th>Segment #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Popularities</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Server injects segment needed by A into system
- Avoids wasting bandwidth in serving initial segments multiple times
Segment Scheduling – Popularities

- How does src figure out popularities?
  - Centrally available at the server
    - Our implementation uses this technique

- Each node maintains popularities for segments
  - Could use a gossiping protocol for aggregation
Segment Scheduling

- Note that goodput of both A and Bs improves
Topology Management

- A has 75% of the file
- Flash crowd of 20 Bs join with empty caches
- Limitations of segment scheduling
  - Bs do not benefit from server, as they get blocks for A
  - A delayed too because of re-routing of blocks from Bs
- Present an algorithm that avoids these problems
Topology Management – Algorithm

- Cluster nodes interested in the same part of the video
  - Retain connections which have high goodput

- When dst approaches src, src serves dst only if
  - src has spare capacity, or
  - dst is interested in the worst-seeded segment per src’s estimate
    - If so, kick a neighbour with low goodput
Topology Management – Algorithm

- When Bs approach server...
  - Server does NOT have spare capacity
  - B is NOT interested in the worst-seeded segment per server’s estimate (segment 7)

- When dst approaches src, src serves dst only if
  - src has spare capacity, or
  - dst is interested in the worst-seeded segment per src’s estimate
    - If so, kick a neighbour with low goodput
Topology Management

- Note that both A and the Bs see increased goodput
Heterogeneous Capacities

- Above algorithms do not work well with heterogeneous nodes

- A is a slow node
- Flash crowd of 20 fast Bs
- Bs get choked because A is slow
Heterogeneous Capacities

- Server refuses to give initial segments to Bs
  - Popularity calculation takes A to be a source

- Adjust segment popularity with weight proportional to the capacity of the node
  - Initial segments have < 2 popularity, rounded to 1
  - Server then gives to Bs
Heterogeneous Capacities

- Note the improvement in the fast/slow nodes
Conclusions

- System designed to support near-VoD using peer-to-peer networks
  - Hitherto, only large file distribution possible

- Three techniques to achieve low setup time and sustainable goodput
  - NetCoding – Improves throughput
  - Segment scheduling – Prevent rare blocks
  - Topology management – Cluster “similar” nodes
Questions?