Application-specific Compression for Remote Visualization of Genomics Applications

Lars Ailo Bongo, Kai Li, Olga Troyanskaya, Tore Larsen and Grant Wallace
Outline

- Motivation
  - Genomics applications
  - WAN challenges
- Compression
- Methodology
- Compression results
- System
- Conclusion and future work
Functional Genomics

- Describe the function and interaction of genes.
- Search for patterns in microarray data.
- Hundreds of measurements for thousands of genomes.
- Visualizations important.
Remote Collaboration

- Important.

- Challenges:
  - Performance: bandwidth and latency.
  - Privacy.
  - Security.
  - Ease of use.
Goal
Thin-client Remote Visualization

- Only share pixels
  - Put rectangle of pixel data at a given x, y position
- Examples
  - VNC
  - Microsoft Remote Desktop
  - Microsoft Livemeeting
- Advantages:
  - Only share visualizations, not raw data.
  - Very simple clients → portable
  - Thick servers → easy data management
- Disadvantage
  - Low Performance
  - High bandwidth requirements
Bandwidth Requirements

Region size (pixels)

Bandwidth (Mbytes/sec)

GigaEther

100 Ether

PU-MIT

PU-GA

PU-UiT

1/32x 1/4x 1/2x 1x 2x 4x

1Hz

6.4Hz

10Hz

20Hz

70Hz
Outline

- Motivation
- Compression
  - Lossless compression algorithms
  - Rabin fingerprints
  - 2D anchoring schemes
- Methodology
- Compression results
- System
- Conclusion and future work
Compression

- **Lossy**
  - Update frequency reduction
  - Color reduction
  - Jpeg (frequency downsampling)

- **Lossless**
  - Diff
  - Run-length encoding (RLE)
  - Fingerprinting (FP)

- **Our approach**: diff + FP + RLE
Diff

- Only send what has been changed since last update
- VNC does this: works well for text editing
Diff (2)

- Problem: how to detect what has been changed?
- Scrolling only moves pixels.
- Scrolling important for Genomics applications.
Bits Changed Between Synchronization Events
Run-length encoding (RLE)

- Zlib
  - DEFLATE = LZ77 + Huffman

- Example:
  - AAAAABBBBBCCCCDDE → 5*A 4*B 3*C DDE
  - A = 10001001 → A = 01

- VNC Hextile
  - Raw pixels and encoded pixels
    - Rectangles with a single color
    - Rectangles with same color as previous
Fingerprinting - Example

- All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy.
Select Anchor Points

- All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy.
Calculate Hash for Regions

- \( \text{hash}("\text{All work and no play makes Jack a dull boy.}\) "\) \(\rightarrow 0x12ad82b3\)
- Send:
  - \((0x12ad82b3, "\text{All work and no play makes Jack a dull boy.}\) "\)
  - 0x12ad82b3
  - 0x12ad82b3
  - ...
Rabin Fingerprints

- Fast sliding window algorithm
- All work and no play makes Jack a dull boy.

0x83af
Rabin Fingerprints

- Fast sliding window algorithm
- All work and no play makes Jack a dull boy.

0x83af, 0x3241
Rabin Fingerprints

- Fast sliding window algorithm
- All work and no play makes Jack a dull boy.

0x83af, 0x3241, 0x31fa
Rabin Fingerprints (2)

- All work and no play makes Jack a dull boy.

- Fixed window size [Spring and Wetherall]:
  - Use Rabin fingerprints as cache index
  - Select fingerprints based on $k$ least significant bits.

- Variable window size [Manber]:
  - Use Rabin fingerprints as anchor points
  - Select fingerprints based on $k$ least significant bits.
  - Calculate SHA-1 hash value for region between anchor points
  - Use SHA-1 value as cache index
2D Data

- Previous work mostly for 1D bytestreams.
- How to vectorize 2D array?
- Our approach: anchor then vectorize
Anchoring Schemes

- Fixed to glass (VNC hextile with caching)
- Splatter (fixed window size)
- (Supertile)
- Supercolumn
Fixed to Glass
Splatter
Outline

- Motivation
- Compression
- Methodology
  - Trace capturing and playback
- Compression results
- System
- Conclusion and future work
Methodology

- Record user interaction, playback trace
- Trace capturing
  - Java GUI instrumentation
  - VNC client recorder
- Trace playback
  - Simulation: compress screenshots
  - Java Robot class
    - Pixel based synchronization (slow down emulated user if system is slow)
  - VNC Playback
Traces

- Java GUI events for three Genomics applications
  - Synthetic traces
  - ~10-15 minutes each
  - Single screen and on display wall
- VNC client recordings
  - Six biologist doing real work
  - Several hour long traces
  - Work in progress
Outline

- Motivation
- Compression
- Methodology
- Compression results
  - Applications
  - Compression ratios
- System
- Conclusion and future work
## Compression Results

<table>
<thead>
<tr>
<th>Compression</th>
<th>GeneVaND</th>
<th>TreeView</th>
<th>Tigr MeV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hextile*</td>
<td>1 (1.6)</td>
<td>1 (6.3)</td>
<td>1 (3.2)</td>
</tr>
<tr>
<td>Zlib</td>
<td>13.6</td>
<td>19.2</td>
<td>14.8</td>
</tr>
<tr>
<td>Fixed-to-glass</td>
<td>15.9</td>
<td>30.6</td>
<td>16.1</td>
</tr>
<tr>
<td>Splatter</td>
<td>9.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supertile</td>
<td>18.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supercolumn</td>
<td>23.8</td>
<td>90.0</td>
<td>17.3</td>
</tr>
</tbody>
</table>
## Compression Cost Breakdown (Server)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Zlib</th>
<th>Supertile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rabin fingerprint</td>
<td>0 ms</td>
<td>26 ms (34%)</td>
</tr>
<tr>
<td>Select anchors</td>
<td>0 ms</td>
<td>1 ms (2%)</td>
</tr>
<tr>
<td>Diff</td>
<td>18 ms (18%)</td>
<td>18 ms (23%)</td>
</tr>
<tr>
<td>SHA-1</td>
<td>0 ms</td>
<td>2 ms (3%)</td>
</tr>
<tr>
<td>Cache lookup</td>
<td>0 ms</td>
<td>0.06 ms (0%)</td>
</tr>
<tr>
<td>Zlib</td>
<td>88 ms (82%)</td>
<td>30 ms (38%)</td>
</tr>
<tr>
<td><strong>SUM</strong></td>
<td><strong>106 ms</strong></td>
<td><strong>77 ms</strong></td>
</tr>
</tbody>
</table>
Bandwidth Reduction

![Graph showing bandwidth reduction over time with various methods compared.](image-url)
Outline

- Motivation
- Compression
- Methodology
- Compression results
- System
  - Design
- Conclusion and future work
System Architecture

- VNC Server
- Data Wrapper
- Anchor
- FP & Cache
- Custom/WAN
- Client Cache
- DLS2VNC
- VNC/LAN
- VNC Client

- VNC Server
- VNC/LAN
- VNC2DSL
- Data Storage
- Anchor
- FP & Cache
- Custom/WAN
- Client Cache
- DSL2VNC
- VNC/LAN
- VNC Client
Outline

- Motivation
- Compression
- Methodology
- Compression results
- System
- Conclusion and future work
Conclusion

- Genomics applications have higher bandwidth requirements than Office applications.
- Fingerprint + Zlib compression ratio is up to 4.7 times better than Zlib.
  - Anchoring scheme important.
  - Works best for scrolling.
- Fingerprint + Zlib compression can have higher throughput than Zlib.
- VNC bandwidth usage limited by VNC server performance.
Current and Future Work

- Improve fingerprinting performance
  - Faster Rabin implementation
  - Reorder compression operations
- Full system performance evaluation
- Real trace evaluation
- Planetlab deployment
- Access control