Using Data Compression to Improve Application Server Performance

Peter Haggar
Senior Software Engineer
Emerging Technology
IBM Corporation
Research Triangle Park, NC
haggar@us.ibm.com
About Me

- Practicing Software Engineer for 17+ years
- Focus on high performance Web Services
- IBM Representative to W3C XML Binary Characterization Working Group
- Work with Web Services on mobile devices
- Please email me with any questions about this presentation
Agenda

- Desire/Need for Compression
- Compression Techniques
- Sample Application and Data
- Sample GZIP Compression Ratios
- Performance with GZIP Compression
- Performance Conclusions
- Enabling GZIP Compression
- W3C XML Binary Characterization Working Group
- References
Desire/Need for Compression

- According to a May 2003 study:
  - 13% of U.S. has broadband
  - 87% of U.S. has narrowband (dial-up)
    - Majority at 56kbps
- Broadband growing rapidly
  - But, most people don’t have optimal connections
- Not all businesses can operate over optimal connections
Desire/Need for Compression

- As computing grows
  - More and more data being sent
- Computers are more sparsely connected
- Connections are not always optimal
- Making matters worse (for performance)
  - Many new(er) technologies are ASCII based
    - HTTP, XML, HTML, etc.
    - More verbose than binary counterparts
      - Typically compress very well, however
Desire/Need for Compression

- New(er) ASCII based technologies
  - Less brittle
  - Easier to program and maintain
  - But...
  - More verbose
  - Require more processing power
  - Require more bandwidth
Desire/Need for Compression

- Many existing applications use brittle binary protocols
  - Over less than optimal connections
  - Acceptable performance achieved due to
    - Efficient processing of binary data
    - Efficient use of existing bandwidth
Desire/Need for Compression

Challenge:

- Upgrade existing applications to less brittle ASCII based technologies such as XML and Web Services
  - Maintain performance
    - At least make it acceptable
  - Don’t require upgrade to network
    - Expensive...sometimes impossible
  - Don’t use remaining bandwidth
    - Chokes off other applications
Desire/Need for Compression

- Can’t expect people/businesses to upgrade connections when you deploy “better” technology
Compression Techniques

- Alternate encodings
  - proprietary

- Non-proprietary compression algorithms
  - GZIP
  - deflate
  - compress

- Technique doesn’t matter
  - Goal is to send less bytes
Compression Techniques

- GZIP supported by most modern browsers
  - via "Accept-Encoding" HTTP header
  - EX:
    - Accept-Encoding: gzip, deflate
    - Means browser can accept data encoded in indicated forms and decompress those forms

- However, most application servers don’t support compression by default
  - Therefore, adding "Accept-Encoding" header has no effect
Compression Techniques

- Solution is to add compression to server
  - If client is not a browser... add it there too
- GZIP compression is solid and pervasive
  - We will use it in examples later
Sample Application and Data

- Web Services application

  - Client
  - Server
    - Application Server == WebSphere
      - Could be any other Application Server though

  - Four different payload sizes
    - 1K
    - 10K
    - 100K
    - 1Meg
Sample Application and Data

- Data is comprised of banking messages
- Conforms to IFX Schema
- Data is sufficiently randomized to give realistic compression and performance numbers
  - Larger payloads were not created with copy/paste
Sample Application and Data

- Determine how differing payloads perform using varying line speeds
  - 100 Mbps
    - Excellent network
  - 2.4 Mbps
    - ~ Cable Modem speed
  - 56 kbps
    - Dial-up speed
XML Based Flow

Client

Generate XML

Process XML

Server

Parser/ SOAP Engine

XML

XML

Parser/ SOAP Engine

Generate XML

Process XML

Parser/ SOAP Engine
GZIP Based Flow

Client

Generate XML
Run GZIP
Send binary form

Parser/ SOAP Engine/ GZIP

Receive binary form Uncompress-GZIP Process XML

Binary - GZIP encoded XML

Parser/ SOAP Engine/ GZIP

Binary - GZIP encoded XML

Parser/ SOAP Engine/ GZIP

Server

Receive binary form Uncompress - GZIP Process XML

Generate XML Run GZIP Send binary form

Parser/ SOAP Engine/ GZIP
Sample GZIP Compression Ratios
1K and 10K Compression Ratios

<table>
<thead>
<tr>
<th>Payload Size</th>
<th>1K Request</th>
<th>1K Response</th>
<th>10K Request</th>
<th>10K Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Size</td>
<td>1153</td>
<td>1079</td>
<td>10139</td>
<td>9977</td>
</tr>
<tr>
<td>GZIP Size</td>
<td>526</td>
<td>550</td>
<td>2472</td>
<td>2528</td>
</tr>
</tbody>
</table>
Data Compression Analysis

- **GZIP achieves excellent compression**
  - 82% reduction best case
  - 72% reduction average
- **Not unexpected since XML messages are ASCII based**
- **GZIP doesn’t always work this well**
  - Example: large sets of floating point numbers don’t compress very well
Performance with GZIP Compression
Topology

- 2 Windows Machines
- 1 Linux machine
- Private network
Topology – 100 Mbps Network

Windows 2000

Client

100 Mbps Network

Server

WebSphere

Windows 2000

Web Services Application
100 Mbps Network - Network Utilization

- XML Network Utilization
- GZIP Network Utilization

<table>
<thead>
<tr>
<th>Payload Size</th>
<th>XML Utilization</th>
<th>GZIP Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>21.68%</td>
<td>8.65%</td>
</tr>
<tr>
<td>10K</td>
<td>59.38%</td>
<td>12.14%</td>
</tr>
<tr>
<td>100K</td>
<td>94.74%</td>
<td>17.93%</td>
</tr>
<tr>
<td>1 Meg</td>
<td>85.50%</td>
<td>14.50%</td>
</tr>
</tbody>
</table>

Percentage of Network Used
100 Mbps Network Summary

- GZIP is slower because the network is not the bottleneck
  - The cost of compressing and uncompressing the data does not pay off
- GZIP uses the least network due to its excellent compression
- XML uses almost the entire 100 Mbps network for 100K and 1Meg payloads
  - 12,500,000 bytes
Topology – 2.4 Mbps Network

Windows 2000 → NistNet → WebSphere → Windows 2000

Client → Simulated 2.4 Mbps Network → Server

Windows 2000

Linux

NistNet

Web Services Application

WebSphere

Client

Simulated 2.4 Mbps Network

Server
2.4 Mbps Network - Performance

Requests/sec vs Payload Size

- XML Requests/Second
- GZIP Requests/Second

XML Response Time vs Payload Size
- GZIP Response Time

- Payload Size: 1K, 10K, 100K, 1 Meg
- Requests/Second: 0, 57, 90, 114, 46, 3419, 665, 36966
- Response Time: millisecond values are not shown in the diagram.
2.4 Mbps Network Summary

- GZIP faster in all cases with the lower response time
  - Better compression enables this as the network becomes the bottleneck

- GZIP
  - Up to 5x faster than XML
  - 3x faster than XML on average
Topology – 56 kbps Network

Windows 2000

Client

Simulated 56 kbps Network

NistNet

Linux

WebSphere

Server

Windows 2000

Web Services Application
56 kbps Network - Performance

<table>
<thead>
<tr>
<th>Payload Size</th>
<th>XML Requests/Second</th>
<th>GZIP Requests/Second</th>
<th>XML Response Time</th>
<th>GZIP Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>178</td>
<td>3</td>
<td>0.5</td>
<td>0.16</td>
</tr>
<tr>
<td>10K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Meg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Peter Haggar — Using Data Compression to Improve Application Server Performance
56 kbps Network Summary

- Similar to 2.4 Mbps case
  - GZIP better performer
  - Up to 5x faster than XML
  - About 3x faster than XML on average
XML and GZIP CPU Cost

- Quantify the server CPU utilization for both XML and GZIP at the same transaction rate
Server CPU utilization at a Constant Req/Sec Rate

<table>
<thead>
<tr>
<th>Payload Size</th>
<th>XML Server CPU</th>
<th>GZIP Server CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>59</td>
<td>31</td>
</tr>
<tr>
<td>100K</td>
<td>51</td>
<td>74</td>
</tr>
<tr>
<td>1Meg</td>
<td>24</td>
<td>29</td>
</tr>
</tbody>
</table>
Performance Conclusions

- When the network is not the bottleneck
  - GZIP is slower

- When the network is the bottleneck
  - GZIP is significantly faster than XML
  - GZIP the better performer
    - 3x faster than XML on average
Performance Conclusions

- GZIP uses more CPU at the same tran rate
  - 1K
    - 90% more Server CPU
  - 100K
    - 45% more Server CPU
  - 1Meg
    - 21% more Server CPU
Enabling GZIP Compression

- Need to enable it on client and server
- If client is browser and it supports GZIP
  - Just worry about server
- Otherwise, need to handle GZIP on both client and server
HTTP Headers

- **Accept-Encoding:**
  - Tells receiver that you accept the comma separated list of encodings

- **Content-Encoding:**
  - Tells receiver the encoding of what you sent

- **Content-Type:**
  - Tells receiver the type of data being sent
    - Ex: text/xml
HTTP Headers

- When you add code to your server to handle GZIP
  
  ➢ Only encode response with GZIP if the request header contains
    
    • Accept-Encoding: gzip
      - Means requester supports GZIP
    
    • If you encode the response with GZIP, include
      - Content-Encoding: gzip
        - Alerts requester of the encoding used
HTTP Headers

Also add to the response header:
- Accept-Encoding: gzip
- This notifies the requester that the server handles gzip

- For XML, Content-Type is “text/xml”
  - For gzip, set it to “binary/gzip”

- For performance, add additional header:
  - Uncompressed-Content-Length:
    - Tells receiver the size of the buffer they need to create to store the message after they uncompress it
Enabling GZIP Compression

- For Server, implement a Servlet Filter
  - Performs GZIP decompression on requests
  - Performs GZIP compression on responses

- Implement the `javax.servlet.Filter` interface
  - `public void init()`
  - `public void destroy()`
  - `public void doFilter()`
Servlet Filter

- Implement the doFilter() method
  - Check and set HTTP headers
  - Uncompress the request
  - Compress the response
Uncompress the Request

//Read the uncompressed content length from the header
int length = 0;
int bytesread;
String uncompressedContentLength = req.getHeader("Uncompressed-Content-Length");
if (uncompressedContentLength != null)
    length = Integer.parseInt(uncompressedContentLength.trim());

//create a buffer to hold the uncompressed data
byte[] uncompressedData = new byte[length];
try {
    //unzip the message
    GZIPInputStream gzipIS = new GZIPInputStream(req.getInputStream(), length);
    bytesRead = gzipIS.read(uncompressedData, 0, length);
    while (bytesRead != length)  //1
    {
        int numbytes = gzipIS.read(uncompressedData, bytesRead, (length - bytesRead));
        if (numbytes == -1)
            break;
        bytesRead += numbytes;
    }
}

catch (Exception e) {
    System.out.println("Exception caught: ");
    e.printStackTrace();
}
Compress Response

//Get the uncompressed data and its length
byte[] uncompressedData = getBytes();
uncompressedContentLength = uncompressedData.length;

//Create a byte array buffer to hold the compressed content
ByteArrayOutputStream baos = new ByteArrayOutputStream(uncompressedContentLength);
try {
    //zip the data and write it to the stream
    GZIPOutputStream gzipOS = new GZIPOutputStream(baos, uncompressedContentLength);
    gzipOS.write(uncompressedData, 0, uncompressedContentLength); //1
    gzipOS.finish(); //2
} catch (IOException ioe) { ioe.printStackTrace(); }
compressedData = baos.toByteArray();

//Use length when you set the Content-Length header
length = compressedData.length;
res.addHeader(HTTPConstants.HEADER_UNCOMPRESSED_CONTENT_LENGTH,
String.valueOf(uncompressedContentLength));
java.io.OutputStream out = res.getOutputStream();
//Write the compressed data to the stream
out.write(compressedData);
Uncompressed-Content-Length

- Important to use this
- Without it, code must guess at buffer size
  - If wrong:
    - Waste memory if too big
    - Waste time creating new buffers and copying data if too small
  - Usage of Uncompressed-Content-Length avoids both issues
JDK GZIP Streams

- java.util.zip package
- Works differently than other streams
  - Don’t know why
- When writing data to a GZIPOutputStream
  - You MUST call finish(); after calling write();
    - See //1 and //2 on previous slide
  - finish() completes writing data to the stream and writes the trailer
    - Does not close the underlying stream
GZIPOutputStream – 3 steps to make it work as you want

1) Invoke constructor
   - Creates stream
   - Writes header

2) Invoke write()
   - Writes data

3) Invoke finish()
   - Completes the write (it’s buffered)
   - Writes trailer
JDK GZIP Streams

- When reading into a GZI Pistons
  - Make sure you call read() in a loop until it returns -1
    - -1 indicates you are done
    - See //1 on “Uncompress the Request” slide
W3C XML Binary Characterization Working Group

- Formed in March 2004 to investigate the benefits and drawbacks to a binary representation of XML
  - [http://www.w3.org/XML/Binary/](http://www.w3.org/XML/Binary/)
  - Not chartered to write a spec
  - Chartered to make a recommendation on whether another group should be commissioned to produce a spec
W3C XML Binary Characterization Working Group

- Group has one year charter
  - Work ends March 2005

- Output
  - Use Case Document
  - Properties Document
  - Measurements Document
  - Recommendation for creating a standard
W3C XML Binary Characterization Working Group

- Current draft of Use Case document
  - http://www.w3.org/XML/Binary/UseCases/xbc-use-cases.html

- Current draft of Properties document
  - http://www.w3.org/XML/Binary/Properties/xbc-properties.html

- Current draft of Measurements document
  - Not available at the time of this writing
W3C XML Binary Characterization Working Group

- Diverse interests represented
  - Business to Business
  - 3D Graphics
  - Mobile devices
  - Huge sets of scientific data (floating point)
- Will a binary representation satisfy the needs of everyone?
  - And still accomplish goals?
W3C XML Binary Characterization Working Group

- Everyone agrees:
  - Need smaller representation of XML
  - Need a representation that is faster to
    - Transmit
    - Process

- Unclear whether you can optimize small footprint and fast parse speed at the same time

- When you attempt to increase parsing speed
  - You often also increase generation speed
IBM has experimented with an alternate encoding of XML
- Mixed results
- Not a clear winner
- More work needed
W3C XML Binary Characterization Working Group

- Differing needs/desires
  - Server
    - Fast generation
    - Fast parsing/processing
  - Client
    - Small message size due to power constraints and/or connection
    - Cheap to process due to power constraints

- Can one solution address all needs and desires and still meet goals?
References

- [http://www.w3.org/XML/Binary/UseCases/xbc-use-cases.html](http://www.w3.org/XML/Binary/UseCases/xbc-use-cases.html) – W3C XML Binary Working Group Use Case Document draft