Adaptive Online Data Compression

Emmanuel Jeannot (Univ. H. Poincaré – LORIA-INRIA ; France)
ejeannot@loria.fr

Bjorn Knutsson (Univ. Pennsylvania ; USA)
bjornk@dsl.cis.upenn.edu

Mats Bjorkman (Mälardalens högskola ; Sweden)
Mats.Bjorkman@mdh.se

WEB Presentation
Outline

- Introduction/problem
- Simple solution
- AdOC algorithm
- Experimental results
- Conclusion
Introduction

Performances of grid middlewares depend on their ability to efficiently transfer huge chunk of data

Transfer data:
- of large size (some Mbytes \(\leq\) X Gb/Tb),
- as fast as possible,
- on standard network,
- while managing heterogeneity,
- adaptive service for grid applications/middleware (but not only),
Simple Idea
Simple Idea
Simple Idea
Simple Idea
Simple Idea

Internet
Simple Idea

Does not work if:

\[ T_{\text{comp}}(\text{FILE}) + T_{\text{send}}(\text{FILE}) + T_{\text{uncomp}}(\text{FILE}) > T_{\text{send}}(\text{FILE}) \]
Parameters that Play a Role

- Network speed,
- current network load,
- current sender and receiver CPU speed,
- data type (ASCII, binary, .gz, etc.),
- data size.
Adaptive On-line Compression Algorithm.

Two ideas:
1. Compression/communication overlap,
2. Dynamic adaptation of compression level.
Basic Algorithm [kb99]

Data to be sent are stored in an output queue.

Foreach data block to be sent
1. Update compression level according to the size of the output queue
2. Compress the block at current compression level
3. Add compressed data to output queue

Originally put into the transport layer (TCP).
AdOC Implementation

User level algorithm/library

1. Multithreading: compression/communication overlap
2. FIFO/queue where are stored compressed packets (monitored for change)
AdOC in a Nutshell

Data to be transmitted

Adaptive compression thread

Packets compressed at Different level

Sending thread

Network

FIFO size
AdOC in a Nutshell

Data to be transmitted → Adaptive compression thread → Packets compressed at Different level → Sending thread → Network

FIFO size

FIFO

Compression based on the zlib
AdOC in a Nutshell

Data to be transmitted

Adaptive compression thread

Packets compressed at Different level

FIFO

Sending thread

Network

FIFO size

Compression based on the zlib
Symmetric mechanisms on the receiver side
Compression Thread

1. Splits the file into buffers
2. Compresses each buffers at a constant compression level into packets of a given size
3. Stores each packet into the FIFO
Buffer Compression

Uses the Zlib library

Buffer to be compressed

FIFO
Buffer Compression

Uses the Zlib library

Buffer to be compressed

FIFO
Buffer Compression

Uses the *Zlib library*

Buffer to be compressed

FIFO
Buffer Compression

Uses the Zlib library

Buffer to be compressed

FIFO
Buffer Compression

Uses the Zlib library
Buffer Compression

Uses the Zlib library
Buffer Compression

Uses the Zlib library
Buffer Compression

Uses the Zlib library
Buffer Compression

Uses the Zlib library

Buffer to be compressed

packet 1 FIFO
Buffer Compression

Uses the *Zlib library*
Buffer Compression

Uses the Zlib library

Buffer to be compressed

packet 1  FIFO  packet 2

Same size
Buffer Compression

Uses the *Zlib library*

- Buffer to be compressed
- Packet 1
- Packet 2
- FIFO

Same size

Remark:

Packet size = Size of the compressed buffer
Tuning AdOC: packet size

Each buffer is compressed into packets of fixed size stored into the FIFO.

**Small** packets have more overhead for managing the FIFO

**Large** packets can have bad overlap
Tuning AdOC: packet size

Each buffer is compressed into packets of fixed size stored into the FIFO.

Small packets □ more overhead for managing the FIFO
Large packets □ bad overlap

Theoretical result: \[ B(e + e') \sqrt{ \frac{1}{2(C + C')} } \]
Tuning AdOC: packet size

Each buffer is compressed into packets of fixed size stored into the FIFO. **Small** packets have more overhead for managing the FIFO. **Large** packets have bad overlap.

Theoretical result: 

$$ \square = \sqrt{\frac{B(e+e)}{2(C+C')}} $$

Experiments: 500 ≤ 173550: we choose 8 Kbytes.
Tuning AdOC: buffer size

level $= 0$ → no compression.

Default case (beginning of the algorithm).

Buffer size $= \text{packet size}$ (good reactivity).
Tuning AdOC: buffer size

AdOC: splits the file into buffers and compress them.
Small buffer → bad compression ratio
Large buffer → bad reactivity

Experimental results:
**buffer size = 200 Kb** (less than 5% from optimal compression ratio).
Updating Compression Level

Before compressing a buffer: update compression level

Compression level

1. Size of compressed file
2. Compression time
Updating Compression Level

Before compressing a buffer: update compression level

Compression level

1. Size of compressed file
2. Compression time

Basic idea:
Before compressing a buffer: update compression level

Compression level

1. Size of compressed file
2. Compression time

Basic idea:
1. fifo size growing: increase compression level
Updating Compression Level

Before compressing a buffer: update compression level

Compression level
1. Size of compressed file
2. Compression time

Basic idea:
1. fifo size growing: increase compression level
2. fifo size shrinking: decrease compression level.
Updating Compression Level

**Before compressing a buffer: update compression level**

Compression level

1. Size of compressed file
2. Compression time

**Basic idea:**
1. fifo size **growing**: increase compression level
2. fifo size **shrinking**: decrease compression level.

**Threshold (fifo size):**
Updating Compression Level

Before compressing a buffer: update compression level

Compression level
1. Size of compressed file
2. Compression time

Basic idea:
1. fifo size growing: increase compression level
2. fifo size shrinking: decrease compression level.

Threshold (fifo size):
1. Avoiding compressing when the fifo is small (less than 10)
Updating Compression Level

Before compressing a buffer: update compression level

1. Size of compressed file
2. Compression time

Basic idea:
1. fifo size growing: increase compression level
2. fifo size shrinking: decrease compression level.

Threshold (fifo size):
1. Avoiding compressing when the fifo is small (less than 10)
2. Fast increase when the fifo is large
Updating Compression Level

**Before compressing a buffer : update compression level**

Compression level

1. Size of compressed file
2. Compression time

**Basic idea:**

1. fifo size **growing** : increase compression level
2. fifo size **shrinking** : decrease compression level.

**Threshold (fifo size):**

1. Avoiding compressing when the fifo is small (less than 10)
2. Fast increase when the fifo is large
3. Fast decrease when the fifo is small
Experiments

Two bench files:

- `Oilpann.hb`: ASCII sparse matrix (Harwell-Boeing format)
- `Bin.tar`: tarball of binaries
Compression Level and FIFO Size
Average Timing Ratio: ASCII

<table>
<thead>
<tr>
<th>Networks</th>
<th>FTP</th>
<th>lvl=1</th>
<th>lvl=9</th>
<th>scp</th>
<th>scp -C</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 Mbps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 mbps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56Kbps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tm/Tadoc
Average Timing Ratio : binaries

Networks

- FTP
- lvl=1
- lvl=9
- scp
- scp -C

- 100 Mbps
- 10 Mbps
- Internet
- ADSL
- 56Kbps
AdOC Library

Library that implements the AdOC algorithm:
1. Multithreaded receiver
2. Optimized FIFO («zerocopy»)
3. API: `sendFile/receiveFile` + `adoc_read/adoc_write` + `setMaxLevel/setMinLevel`
4. Available at: 
   `http://www.loria.fr/~ejeannnot/adoc`
   (under LGPL license)
Plugging AdOC into NetSolve

Remote matrix multiplication on Internet

- Time
- Matrix size
- NetSolve
- NetSolve + AdOC
Plugging AdOC into NetSolve

Remote matrix multiplication on Internet

- **Time**
- **Matrix size**

NetSolve vs. NetSolve + AdOC

- NetSolve: 340% improvement
Plugging AdOC into NetSolve

Remote matrix multiplication on 100 Mbps

NetSolve
NetSolve + AdOC
Plugging AdOC into NetSolve

Remote matrix multiplication on 100 Mbps

1. AdOC guesses that compression is not very useful
Plugging AdOC into NetSolve

1. AdOC guesses that compression is not very useful
2. No performance degradation
Conclusion/Future Work
Conclusion/Future Work

Conclusion/Future Work

- Improves middlewares.
Conclusion/Future Work

- Improves middlewares.
- No performance degradation.
Conclusion/Future Work

- Improves middlewares.
- No performance degradation.

Future work :
- Incorporate very fast compression algorithms (done),
- incorporate more efficient compression algorithms,
- test on faster network (1Gbps),
- feedback mechanism from the receiver.