An Intelligent, Adaptive, & Flexible Data Compression Framework


Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun
hdevarajan@hawk.iit.edu
Modern Big Data Applications

- Explosion of data volume, variety, and velocity
  - Facebook is storing roughly 250 billion images.
  - 80% of all the world's data is unstructured.
  - Facebook users upload more than 900 million photos a day.
  - Square Kilometer Array (SKA) is estimated to reach 10 Pb/s.

Generates tremendous stress on storage sub-system
Data Compression is popularly used to ease this stress
Data Compression

- Categories of compression techniques
  - lossy and lossless algorithms.
- The lossless algorithms are standard in scientific and cloud applications.
- Popular examples of lossless algorithms
  - General Purpose: Bzip, Zlib, 7z, etc.
  - Specialized: Snappy, SPDP, LZ0, etc.
Challenges in Data Compression

- **Data-dependency**: Each Compression library is specialized for a certain input (i.e., data-type and data-format)
- **Library-choice**: Choice of library is complex as different situations might demand different compression needs.
- **API diversity**: Each library has its own definition of Interface.

These challenges highlight there is no "one compression for all".

Can we do something better?
Outline

- Approach
- Design
- Results
- Conclusion
Approach: Overview

- Benchmark
- Analyze
- Build Ares

<table>
<thead>
<tr>
<th>Approach</th>
<th>Design</th>
<th>Results</th>
<th>Conclusion</th>
</tr>
</thead>
</table>

Hariharan Devarajan, hdevarajan@hawk.iit.edu
Approach: Benchmark

- **Library Corpus**: bzip2, zlib, huffman, brotli, bsc, lzma, lz4, lzo, pithy, snappy, and quicklz.
- **Data-Types**: characters, integers along with their modifiers (short, long, signed, unsigned), sorted integers, floating point, and double floating points.
- **Data-Formats**: binary, HDF5, csv, json, xml, and Avro, Parquet.
- **Metrics**: Compression/Decompression Speed & Compression Ratio

These total to over 1000 test cases
### Approach: Analyze

- **Workload Priority**: defines different requirement that a workload prioritizes
- **Score Formulation**:

<table>
<thead>
<tr>
<th></th>
<th>CS</th>
<th>DS</th>
<th>CR</th>
<th>Workload</th>
<th>Char</th>
<th>Integer</th>
<th>Sorted int</th>
<th>Float</th>
<th>Double</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Asynchronous communication</td>
<td>lz4</td>
<td>lz4</td>
<td>lz4</td>
<td>quicklz</td>
<td>lz4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Multicast in Network</td>
<td>lz4</td>
<td>lz4</td>
<td>pithy</td>
<td>pithy</td>
<td>brotli</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Archival Store</td>
<td>bsc</td>
<td>lzma</td>
<td>bsc</td>
<td>lzma</td>
<td>bsc</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>Synchronous Communication</td>
<td>lz4</td>
<td>lz4</td>
<td>pithy</td>
<td>pithy</td>
<td>lz4</td>
</tr>
<tr>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>Dequeue Operation</td>
<td>lz4</td>
<td>lz4</td>
<td>lz4</td>
<td>quicklz</td>
<td>pithy</td>
</tr>
<tr>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Queue Operation</td>
<td>lz4</td>
<td>lz4</td>
<td>lz4</td>
<td>pithy</td>
<td>lz4</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td>Mixed workload</td>
<td>lz4</td>
<td>lz4</td>
<td>pithy</td>
<td>pithy</td>
<td>pithy</td>
</tr>
</tbody>
</table>
Approach: Build Ares

- Goals
  - The framework should be able to learn and adjust itself to the input data compression characteristics.
  - The framework should be able to reconfigure itself, dynamically, to various compression needs of an application.
  - The framework should be able to unify all interfaces of the compression libraries it contains.
Ares Design
Design: Overview

Ares Compression Framework

Input Analyzer
- Data type identifier
- Data format identifier
- Metadata parser

Main Engine

Output Manager
- Metadata decorator
- Memory allocator
- I/O client

Library Pool
- lzop2
- zlib
- lz4
- ffmpeg
- ...lzop

Feedback

Input
- Buffer
- File
- Directory
- Ares File

Output
- Compressed Buffer
- Compressed File
- Compressed Directory
- File

Approach
Design
Results
Conclusion
Design

- Infers data type and format
- Uses a hybrid approach
  - static analysis and a dynamic feedback mechanism
- **Data-format**: mime-type, extensions, and metadata-rich.
- **Data-type**: decoding techniques, type-inference and metadata-rich.

Input Analyzer

Data type identifier
Data format identifier
Metadata parser
Design

Main Engine

Library Pool
bzlib
zlib
lzo4
lzo8
pitty
lzo0

Approach  Design  Results  Conclusion
Design

- Decorates the compressed data with headers, regarding the compression library used
  - 8 bytes header per data-type
- Checks the correctness of the format using parity checking
- Performs final I/O of the compressed/uncompressed data
The engine updates the log with actual performance results.

Analyzer processes the log to identify the difference between expected and actual measurements.

This makes the analyzer improve its predictions over time.
Evaluation
Evaluation: Testbed

Machine Configuration (Per Node)
- dual Intel(R) Xeon(R)
  - CPU E5-2670 v3
  - 2.30GHz
  - 48 cores
- 128 GB RAM
- 10 Gbit Ethernet,
- 200 GB HDD

Deployment
- Scientific Setup:
  - 32 client nodes
  - 8 PFS nodes
- Cloud Setup:
  - 40-node Hadoop cluster
  - 1 Namenode
Evaluation: Goals

- Overheads and Resource Utilization
  - Ares’s analysis overheads + CPU + Memory utilization

- Compression/Decompression Intelligence
  - Data type and format aware data compression

- Compression/Decompression Adaptiveness
  - Workflow-specific data compression

- Compression/Decompression Flexibility
  - Ares for various real applications
Evaluation: Overheads

- Description:
  - 64GB HDF5 input with four datasets: characters, integers, sorted integers, and doubles.
  - **Workflow**: read input data -> compress data -> write compressed data -> read compressed data -> decompress the data.
  - Metrics overall time and utilization

![Graph showing overheads and metrics for different compression libraries]

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Design</th>
<th>Results</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Observations

- libraries demonstrates different overheads
- Ares balances the tradeoff between CT/DT and CR by analyzing the input data with a 10% overhead.
- Ares performs better as it uses a collection of libraries where they have strength.
Evaluation: Intelligence

- Description:
  - Different Data-Types
  - 64 GB of buffer input
  - Configurations of this buffer:
    - Characters, integers, floats, doubles, and a mixed case
  - We measure the CT, DT and CR.
Observation:
- Different libraries excel in different data types.
- Trade-off between CT and CR
- For mixed input each library takes a hit in performance
  - Ares optimizes by using best library for given data-type.
Evaluation: Intelligence

- Description:
  - Different Data-Format
  - 64 files (each 1 GB) in a directory
  - Composition of this folder:
    - POSIX, HDF5, pNetCDF, HTML, XML, JSON, Avro, and Parquet
  - We measure the CT, DT and CR.
Evaluation: Intelligence

- Observation:
  - Different libraries excel in different data formats.
  - trade-off between CT and CR
  - For mixed directory each library takes a hit in performance
    - Ares optimizes by using best library for given data-format.
Evaluation: Adaptiveness

- **Description:**
  - Different Workflow Priorities
  - 64 GB of CSV file input
  - Four columns of this file:
    - Index (sorted integer), location (char), population size (integer), income (double)
  - We measure the CT, DT and CR.

[Diagram showing compression time, decompression time, and compression ratio for different compression libraries and data sizes.]
Evaluation: Adaptiveness

- Observation:
  - This multi type data has an effect on every compression library
  - Different prioritization of Ares results in difference in performance metrics
  - In Balanced mode, Ares is a Jack of all trades.
Evaluation: Scientific Application (VPIC)

- VPIC simulation
  - Each process is producing 1 GB at each time step.
  - The overall data size is 1.5 TB
  - HDF5 file is organized with 7 datasets
    - two datasets of integers, two of floats and three of doubles.
- We show Compression Time, Compression Ratio and I/O Time.
Evaluation: Scientific Application (VPIC)

- Observation
  - Compression reduces the I/O time.
  - Heavy compression is costly, and a balance must be found to be beneficial to the application.
  - Opposite picture can be seen when using lz4, lzo, and snappy as compression filters.
  - Ares prioritizes both CT and CR.
Evaluation: Hadoop Application (Word Count)

- Map-Reduce implementation of the word-count kernel 32 mappers and 8 reducers
  - 1.5 TB of HTML files (Wikipedia articles)
  - Workflow:
    - MAP: reads its input data and counts individual word occurrences and create intermediate files
      - a compressed input and a high DS.
    - SHUFFLE: all intermediate files are sorted
      - quick compression to minimize I/O traffic.
    - REDUCE: merge the final count across all intermediate files and write the final word count back to a file in HDFS
Observations:
- compression on the input data reduces the I/O time in map phase.
- tradeoff CT/DT and CR
- Ares achieving the best overall performance
- highlights the importance of striking a balance of compression speed and ratio.
  - Compression libraries do not offer dynamic adaptiveness based on the workload type.
Conclusions

- We investigated how different data-types, data-format, and workload characteristics affect the choice of the “ideal” compression library.

- We have developed Ares, a dynamic, adaptive, and flexible compression framework, that can transparently meet various compression needs of big data applications.

- Under real world applications Ares performed 2-6x faster than competitive solutions with a 10% analysis cost.
Hariharan Devarajan, hdevarajan@hawk.iit.edu

Q & A