Compression Options In Hadoop – A Tale of Tradeoffs

Govind Kamat, Sumeet Singh

Hadoop Summit (San Jose), June 27, 2013

Introduction

Govind Kamat

Technical Yahoo!, Hadoop

Cloud Engineering Group

701 First Avenue
Sunnyvale, CA 94089 USA

- Member of Technical Staff in the Hadoop Services team at Yahoo!
- Focuses on HBase and Hadoop performance
- Worked with the Performance Engineering Group on improving the performance and scalability of several Yahoo! applications
- Experience includes development of large-scale software systems, microprocessor architecture, instruction-set simulators, compiler technology and electronic design

Sumeet Singh

Director of Products, Hadoop

Cloud Engineering Group

701 First Avenue
Sunnyvale, CA 94089 USA

- Leads Hadoop products team at Yahoo!
- Responsible for Product Management, Customer Engagements, Evangelism, and Program Management
- Prior to this role, led Strategy functions for the Cloud Platform Group at Yahoo!

Agenda

1. Data Compression in Hadoop
2. Available Compression Options
3. Understanding and Working with Compression Options
4. Problems Faced at Yahoo! with Large Data Sets
5. Performance Evaluations, Native Bzip2, and IPP Libraries
6. Wrap-up and Future Work

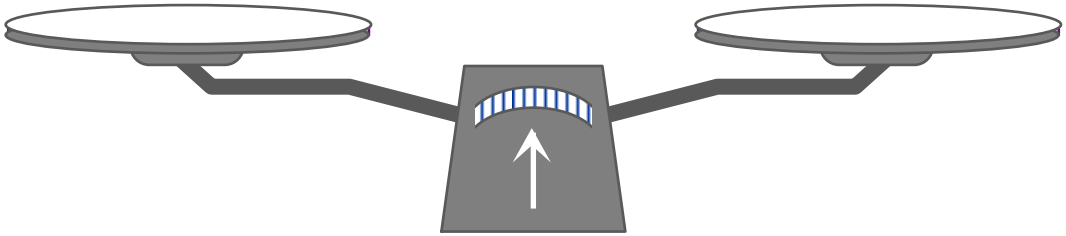
Compression Needs and Tradeoffs in Hadoop

- Hadoop jobs are data-intensive, compressing data can **speed up the I/O operations**
  - MapReduce jobs are almost always I/O bound
- Compressed data can **save storage space** and **speed up data transfers** across the network
  - Capital allocation for hardware can go further
- Reduced I/O and network load can bring **significant performance improvements**
  - MapReduce jobs can finish faster overall
- On the other hand, **CPU utilization and processing time increases** during compression and decompression
  - Understanding the tradeoffs is important for MapReduce pipeline's overall performance

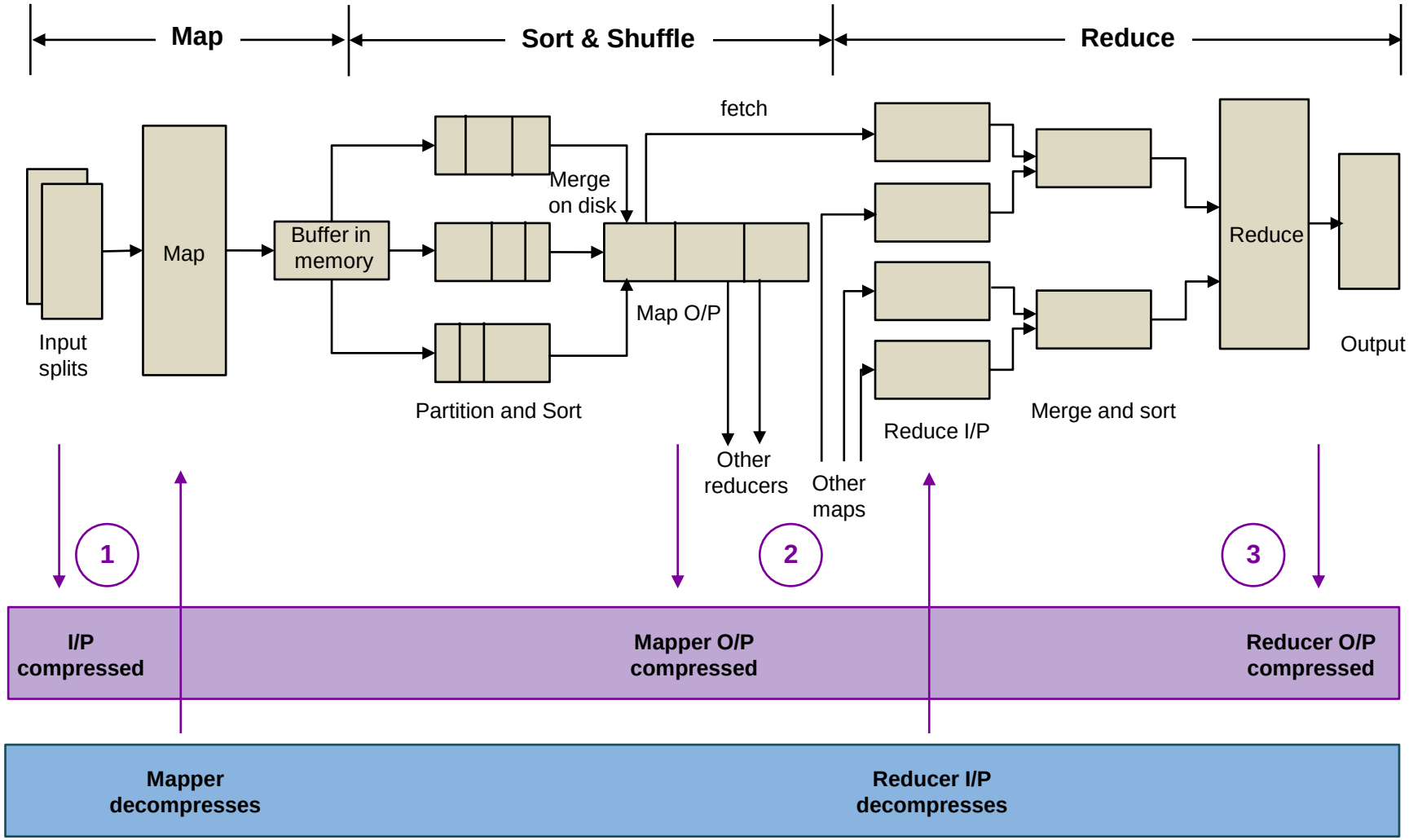
The Compression Tradeoff

- Storage
- Disk I/O
- Network bandwidth

- CPU Time



Data Compression in Hadoop's MR Pipeline



Source: Hadoop: The Definitive Guide, Tom White

Compress

Decompress

Compression Options in Hadoop (1/2)

Format	Algorithm	Strategy	Emphasis	Comments
zlib	Uses DEFLATE (LZ77 and Huffman coding)	Dictionary-based, API	Compression ratio	Default codec
gzip	Wrapper around zlib	Dictionary-based, standard compression utility	Same as zlib, codec operates on and produces standard gzip files	For data interchange on and off Hadoop
bzip2	Burrows-Wheeler transform	Transform-based, block-oriented	Higher compression ratios than zlib	Common for Pig
LZO	Variant of LZ77	Dictionary-based, block-oriented, API	High compression speeds	Common for intermediate compression, HBase tables
LZ4	Simplified variant of LZ77	Fast scan, API	Very high compression speeds	Available in newer Hadoop distributions
Snappy	LZ77	Block-oriented, API	Very high compression speeds	Came out of Google, previously known as Zippy

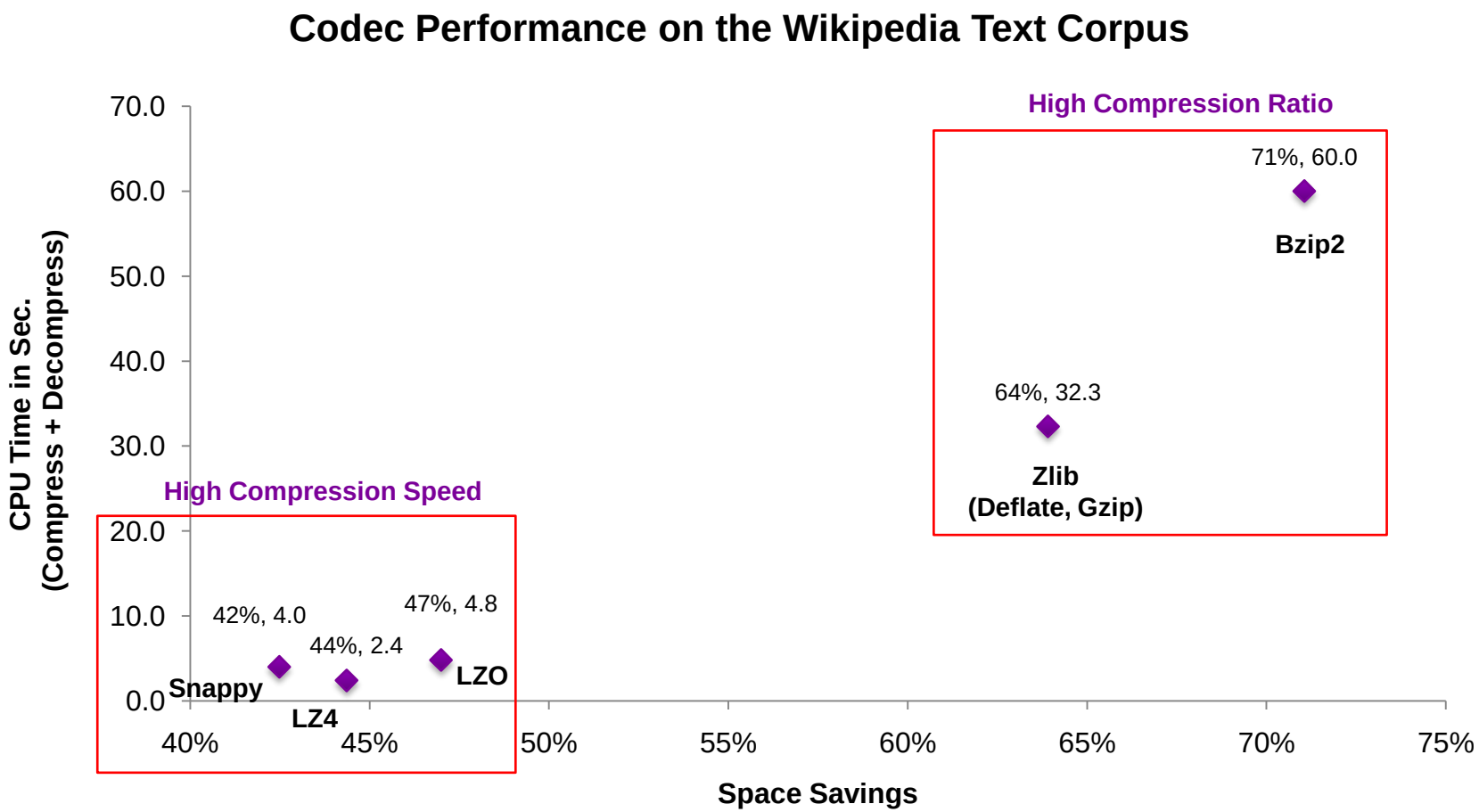
Compression Options in Hadoop (2/2)

Format	Codec (Defined in io.compression.codecs)	File Extn.	Splittable	Java/ Native
zlib/ DEFLATE (default)	org.apache.hadoop.io.compress.DefaultCodec	.deflate	N	Y/ Y
gzip	org.apache.hadoop.io.compress.GzipCodec	.gz	N	Y/ Y
bzip2	org.apache.hadoop.io.compress.BZip2Codec	.bz2	Y	Y/ Y
LZO (download separately)	com.hadoop.compression.lzo.LzoCodec	.lzo	N	N/ Y
LZ4	org.apache.hadoop.io.compress.Lz4Codec	.lz4	N	N/ Y
Snappy	org.apache.hadoop.io.compress.SnappyCodec	.snappy	N	N/ Y

NOTES:

- **Splittability –** Bzip2 is “splittable”, can be decompressed in parallel by multiple MapReduce tasks. Other algorithms require all blocks together for decompression with a single MapReduce task.
- **LZO** – Removed from Hadoop because the LZO libraries are licensed under the GNU GPL. LZO format is still supported and the codec can be downloaded separately and enabled manually.
- **Native bzip2 codec** – added by Yahoo! as part of this work in Hadoop 0.23

Space-Time Tradeoff of Compression Options

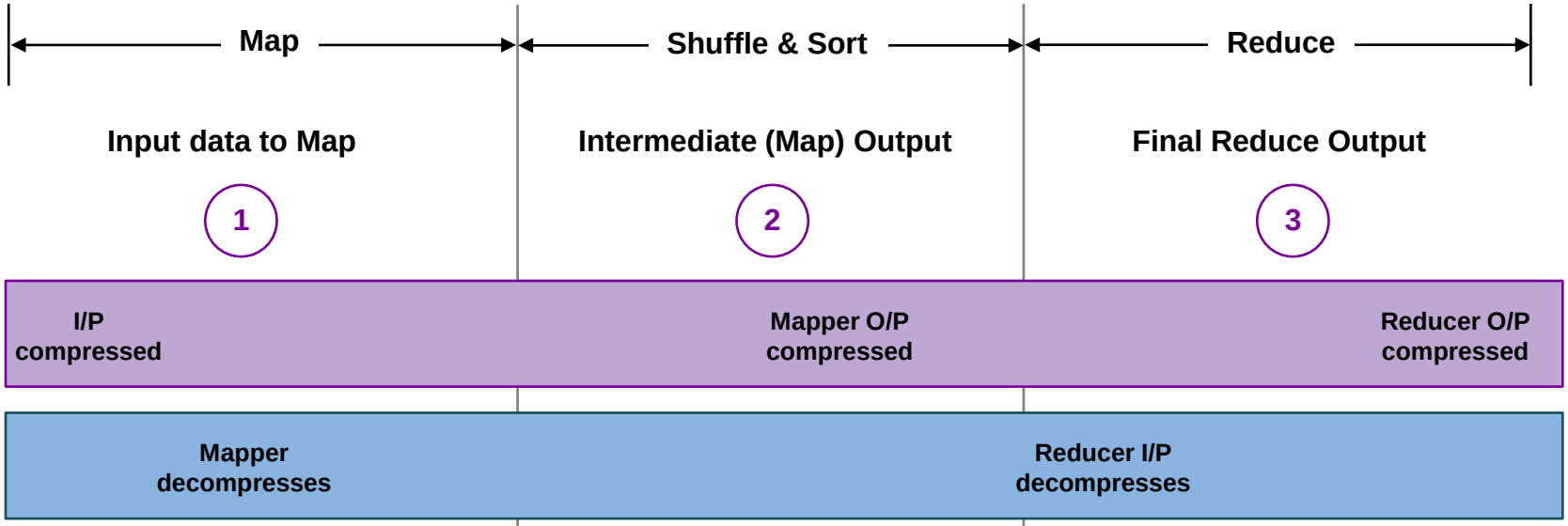


Note:
A 265 MB corpus from Wikipedia was used for the performance comparisons.
Space savings is defined as [1 – (Compressed/ Uncompressed)]

Using Data Compression in Hadoop

	Phase in MR Pipeline	Config	Values
1	Input data to Map	File extension recognized automatically for decompression	File extensions for supported formats Note: For SequenceFile, headers have the information [compression (boolean), block compression (boolean), and compression codec]
		One of the supported codecs	one defined in io.compression.codecs
2	Intermediate (Map) Output	mapreduce.map.output.compress	false (default), true
		mapreduce.map.output.compress.codec	one defined in io.compression.codecs
3	Final (Reduce) Output	mapreduce.output.fileoutputformat. compress	false (default), true
		mapreduce.output.fileoutputformat. compress.codec	one defined in io.compression.codecs
		mapreduce.output.fileoutputformat. compress.type	Type of compression to use for SequenceFile outputs: NONE, RECORD (default), BLOCK

When to Use Compression and Which Codec



- Compress the input data, if large
- Use splittable algo such as bzip2, or use zlib with SequenceFile format

- Always use compression, particularly if spillage or slow network transfers
- Use faster codecs such as LZO, LZ4, or Snappy

- Compress for storage/ archival, better write speeds, or between MR jobs
- Use standard utility such as gzip or bzip2 for data interchange, and faster codecs for chained jobs

Compress Decompress

Compression in the Hadoop Ecosystem

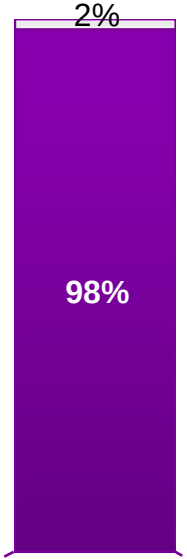
Component	When to Use	What to Use
Pig	▪ Compressing data between MR job ▪ Typical in Pig scripts that include joins or other operators that expand your data size	Enable compression and select the codec: pig.tmpfilecompression = true pig.tmpfilecompression.codec = gzip, lzo
Hive	▪ Intermediate files produced by Hive between multiple map-reduce jobs ▪ Hive writes output to a table	Enable intermediate or output compression: hive.exec.compress.intermediate = true hive.exec.compress.output = true
HBase	▪ Compress data at the CF level (support for LZO, gzip, Snappy, and LZ4)	List required JNI libraries: hbase.regionserver.codecs Enabling compression: create ’table', { NAME => 'colfam', COMPRESSION => ’LZO' } alter ’table', { NAME => 'colfam', COMPRESSION => ’LZO' }

Compression in Hadoop at Yahoo!

1 Map

Input data to Map

380M Files on Jun 16, 2013 (/data, /projects)

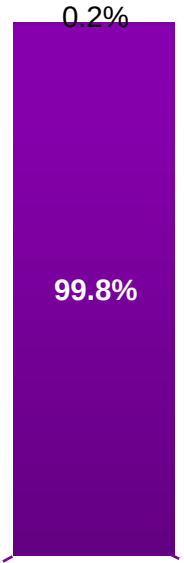


zlib / default	**73%**
gzip	22%
bzip2	4%
LZO	1%

2 Shuffle & Sort

Intermediate (Map) Output

4.2M Jobs, Jun 10-16, 2013

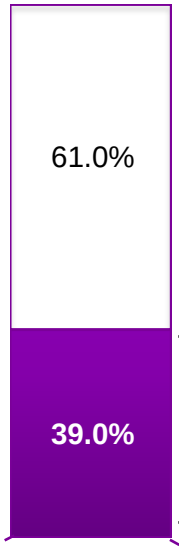


LZO	**98.3%**
gzip	1.1%
zlib / default	0.5%
bzip2	0.1%

3 Reduce

Final Reduce Output

4.2M Jobs, Jun 10-16, 2013



Includes intermediate Pig/ Hive compression

LZO	55%
gzip	**35%**
bzip2	5%
zlib / default	5%

Pig Intermediate

Compressed

Compression for Data Storage Efficiency

- DSE considerations at Yahoo!
- RCFile instead of SequenceFile
- Faster implementation of bzip2
- Native-code bzip2 codec
- HADOOP-8462[1], available in 0.23.7
- Substituting the IPP library

[1] Native-code bzip2 implementation done in collaboration with Jason Lowe, Hadoop Core PMC member

IPP Libraries

- Integrated Performance Primitives from Intel
- Algorithmic and architectural optimizations
- Processor-specific variants of each function
- Applications remain processor-neutral
- Compression: LZ, RLE, BWT, LZO
- High level formats include: zlib, gzip, bzip2 and LZO

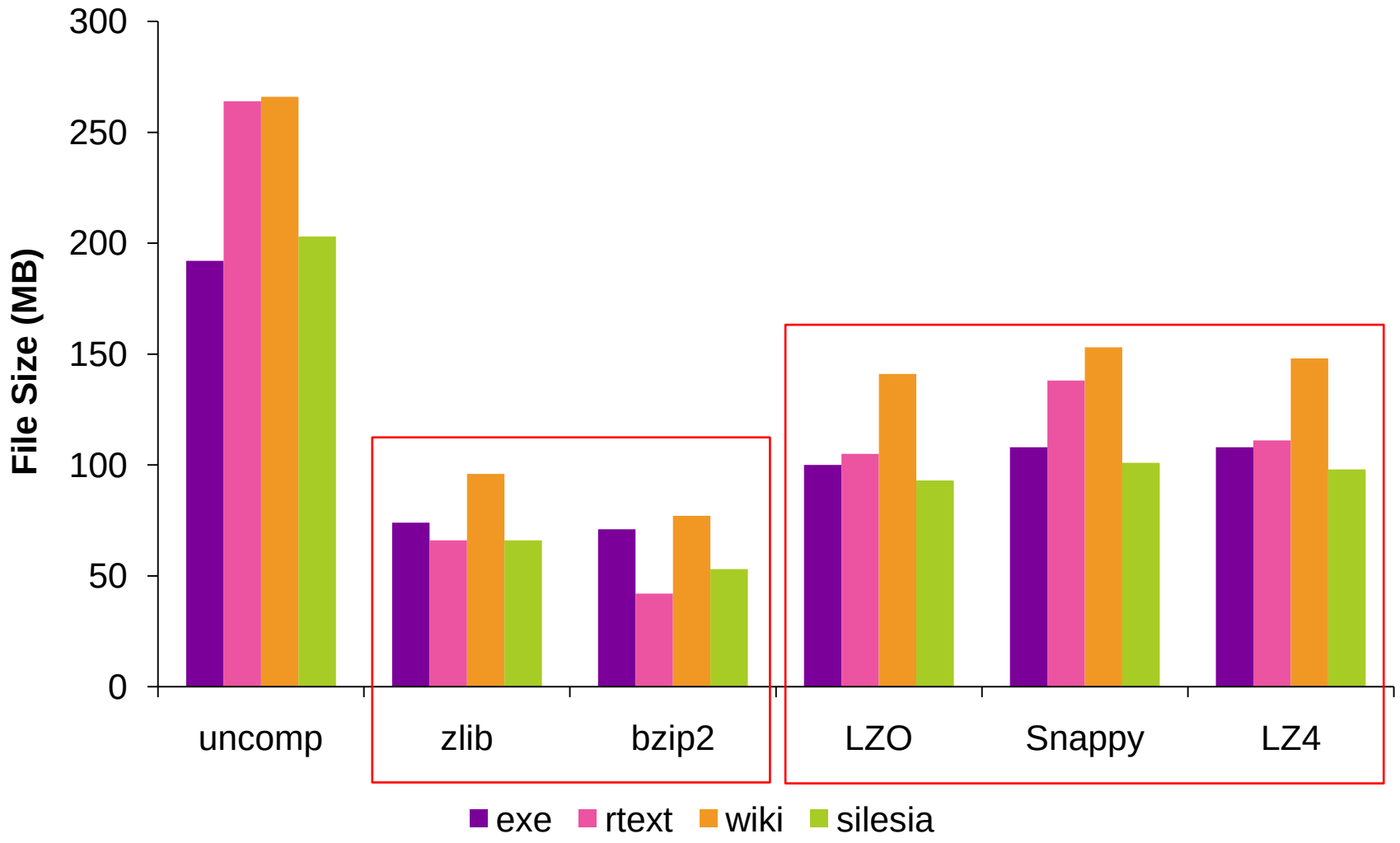
Measuring Standalone Performance

- Standard programs (gzip, bzip2) used
- Driver program written for other cases
- 32-bit mode
- Single-threaded
- JVM load overhead discounted
- Default compression level
- Quad-core Xeon machine

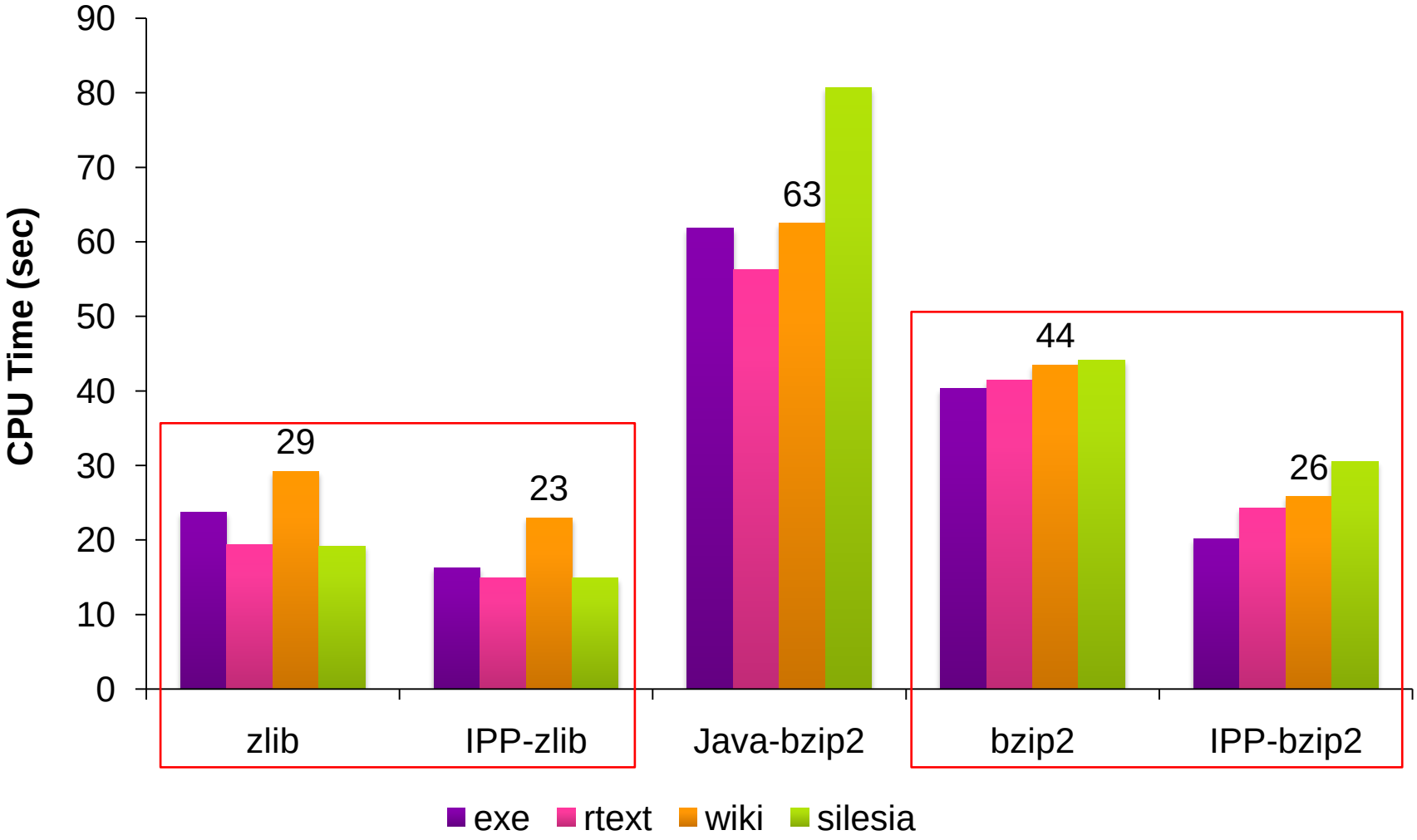
Data Corpuses Used

- Binary files
- Generated text from randomtextwriter
- Wikipedia corpus
- Silesia corpus

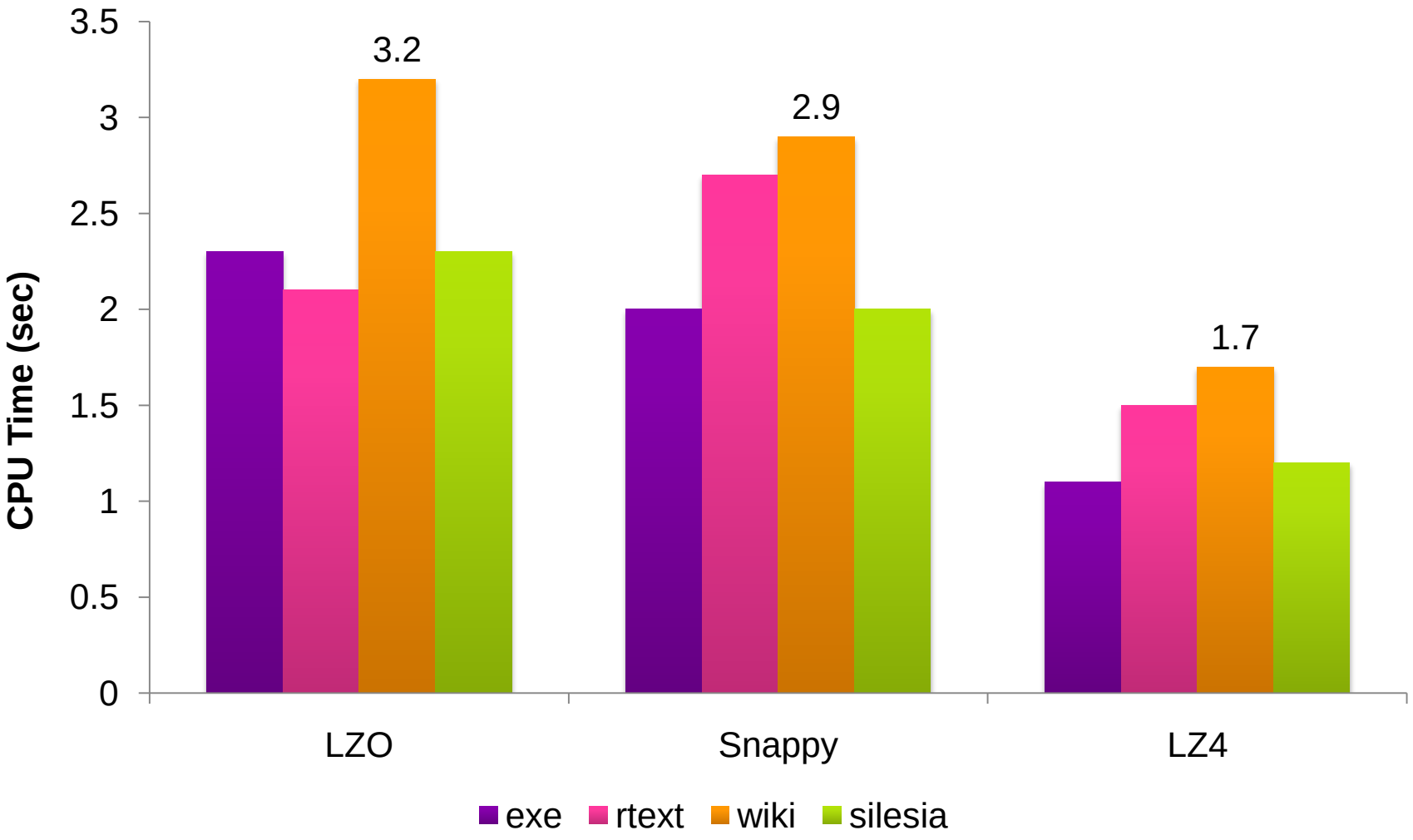
Compression Ratio



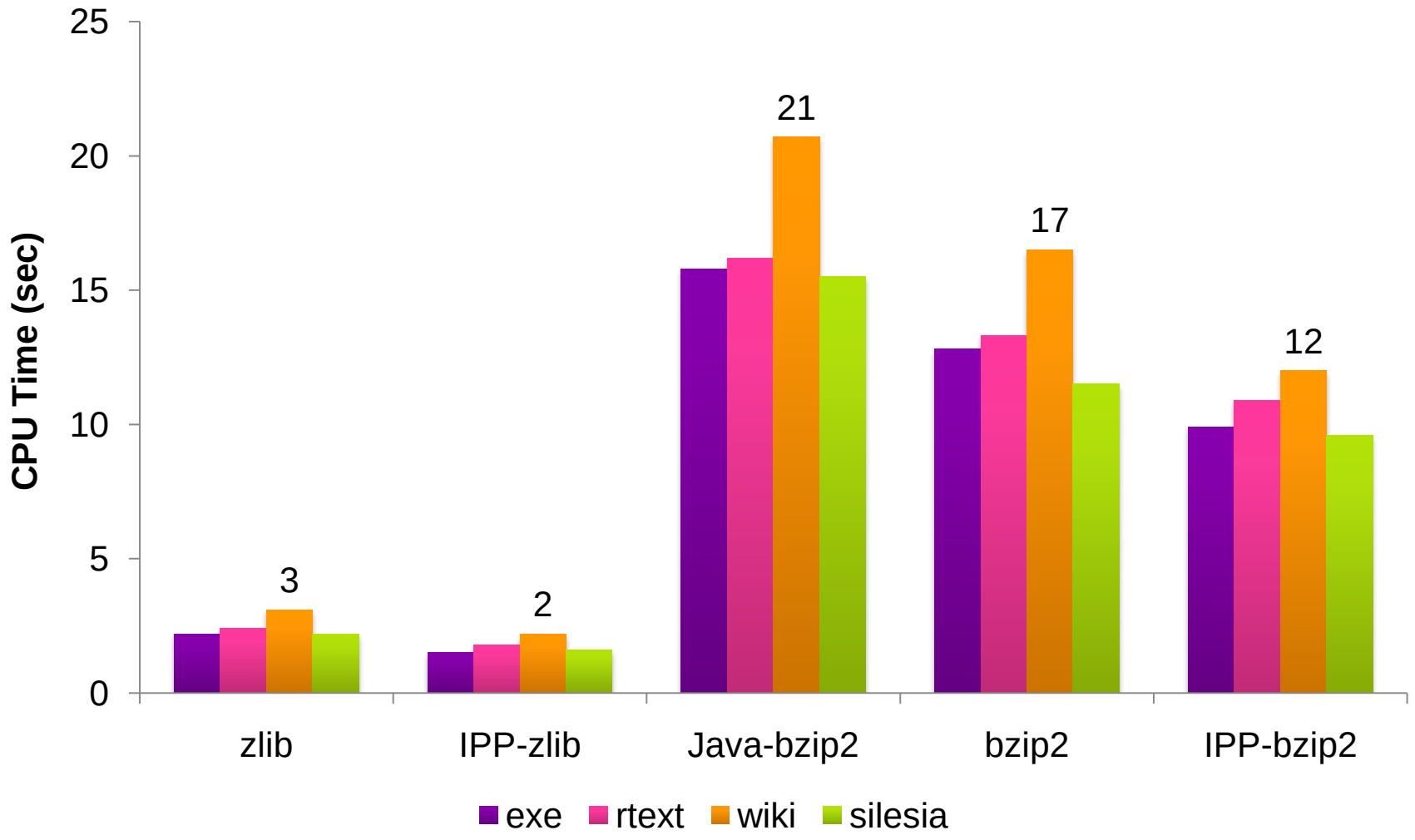
Compression Performance



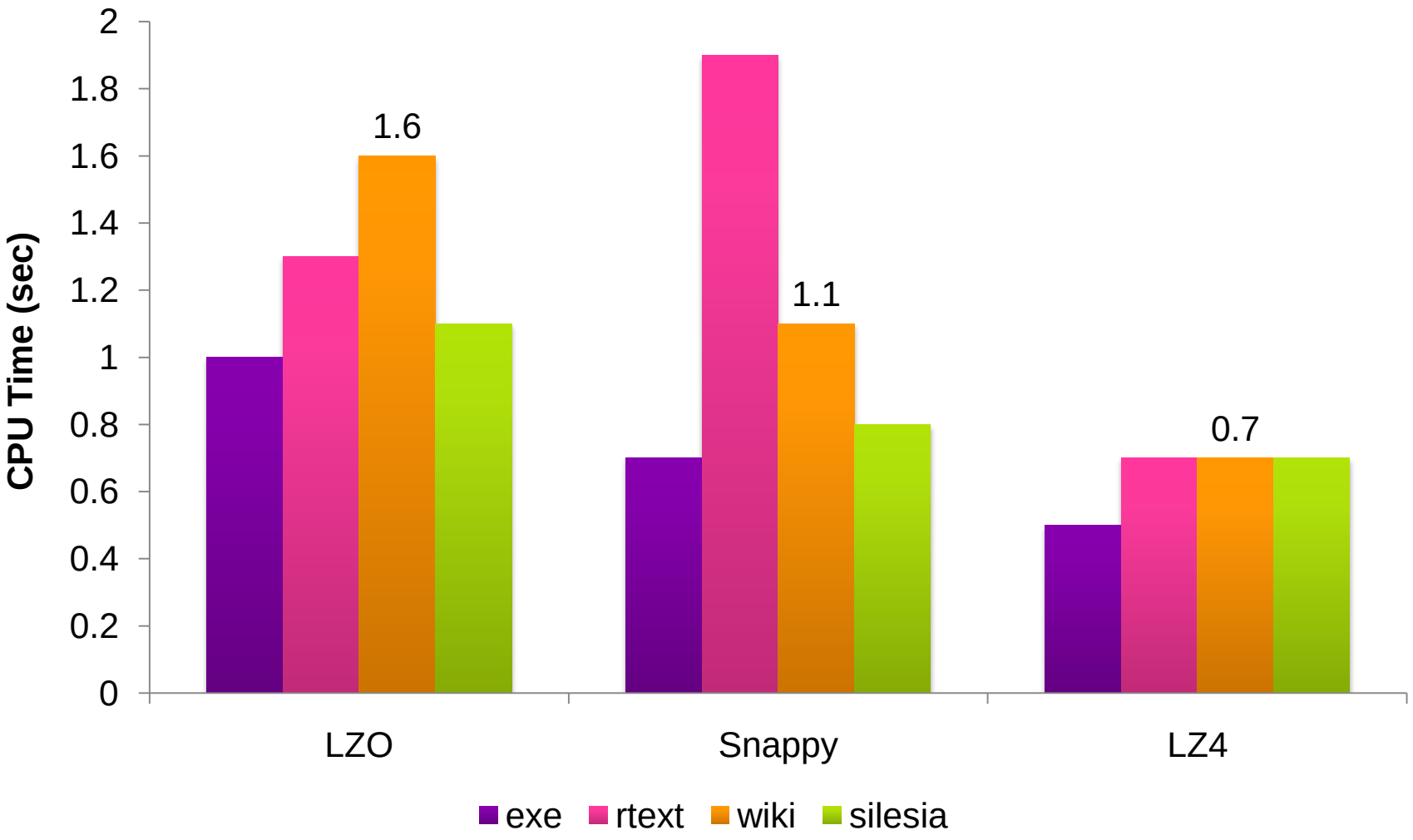
Compression Performance (Fast Algorithms)



Decompression Performance



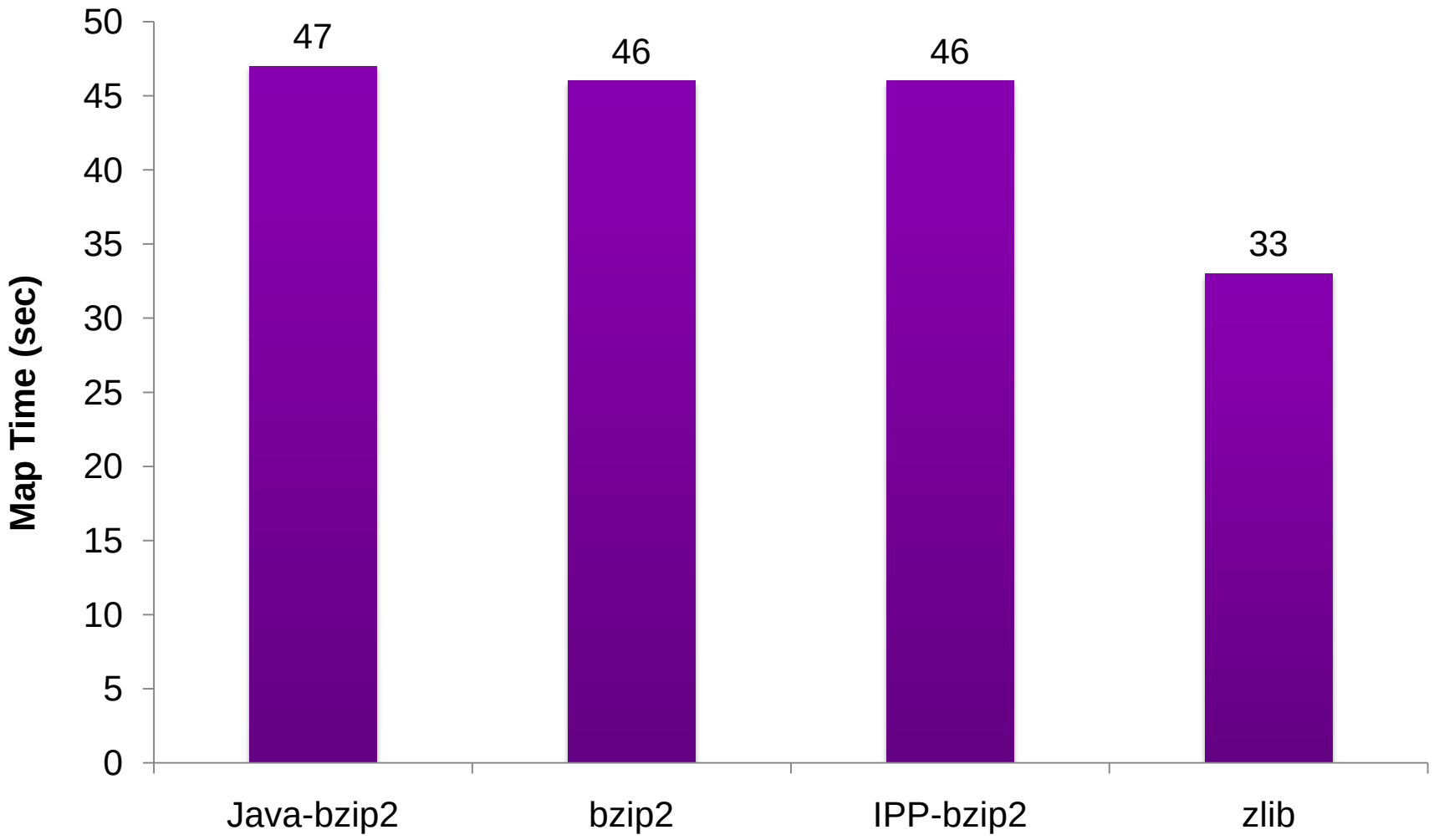
Decompression Performance (Fast Algorithms)



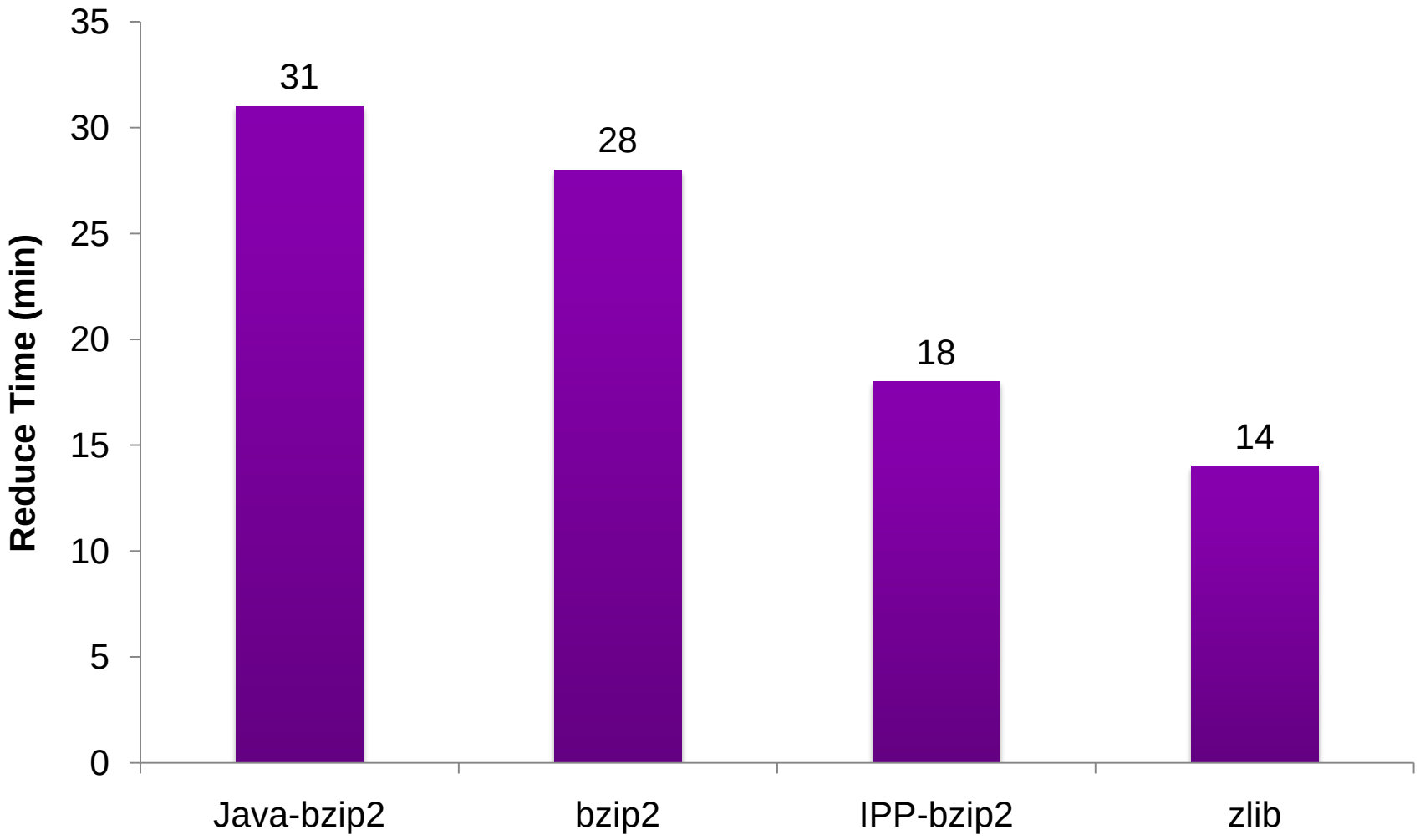
Compression Performance within Hadoop

- Daytona performance framework
- GridMix v1
- Loadgen and sort jobs
- Input data compressed with zlib / bzip2
- LZO used for intermediate compression
- 35 datanodes, dual-quad-core machines

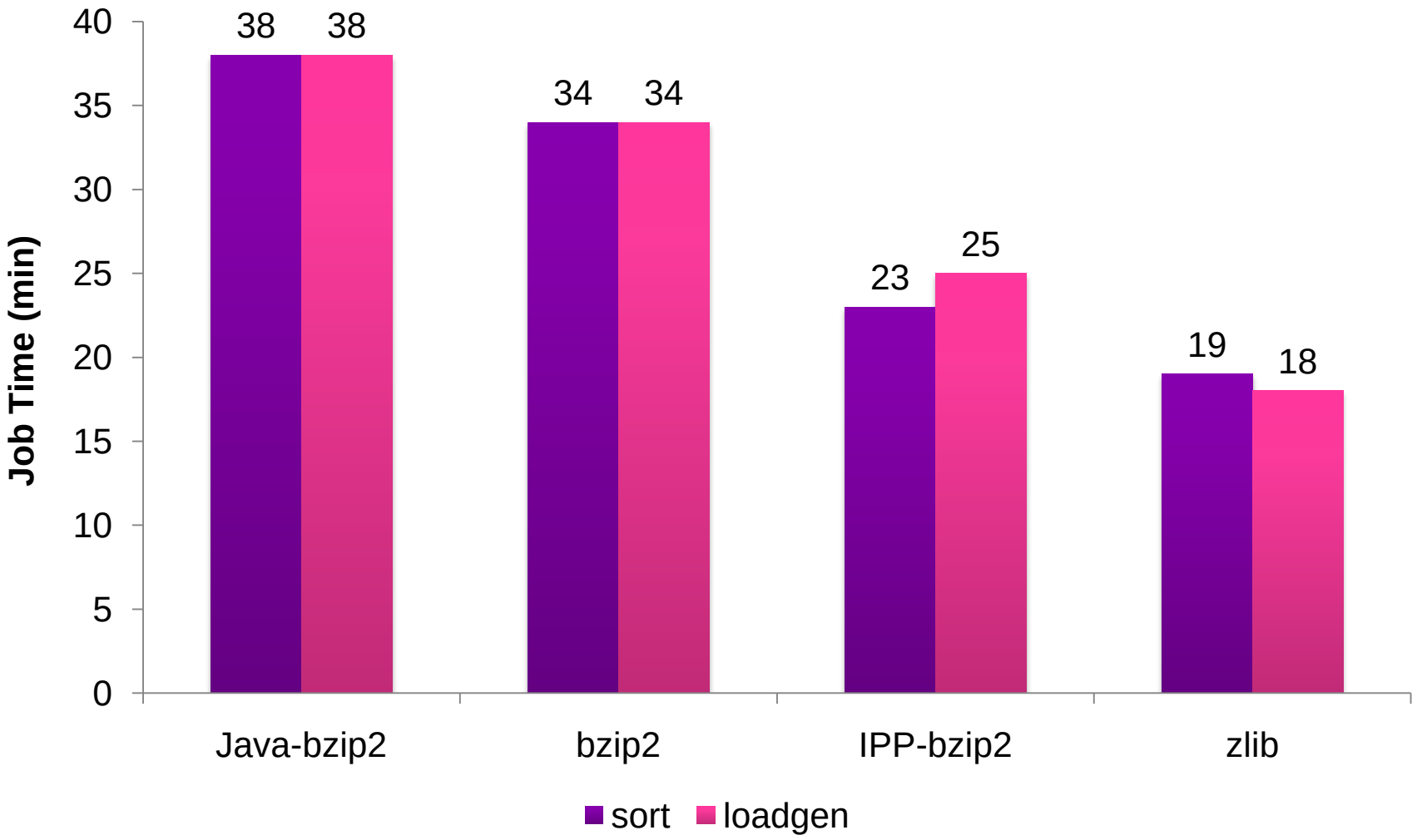
Map Performance



Reduce Performance



Job Performance



Future Work

- Splittability support for native-code bzip2 codec
- Enhancing Pig to use common bzip2 codec
- Optimizing the JNI interface and buffer copies
- Varying the compression effort parameter
- Performance evaluation for 64-bit mode
- Updating the zlib codec to specify alternative libraries
- Other codec combinations, such as zlib for transient data
- Other compression algorithms

Considerations in Selecting Compression Type

- Nature of the data set
- Chained jobs
- Data-storage efficiency requirements
- Frequency of compression vs. decompression
- Requirement for compatibility with a standard data format
- Splittability requirements
- Size of the intermediate and final data
- Alternative implementations of compression libraries

YAHOO!®