The Language of Compression

Leif Walsh

Two Sigma Investments
leif.walsh@gmail.com
@leifwalsh

September 22, 2015

Scope

TWO SIGMA

Today's talk is about **compression**

Scope

TWO SIGMA

Today's talk is about **compression**:

- In data storage systems (databases, filesystems)

Scope

TWO SIGMA

Today's talk is about **compression**:

- In data storage systems (databases, filesystems)
- Using general-purpose (lossless) algorithms

Scope

Today's talk is about **compression**:

- In data storage systems (databases, filesystems)
- Using general-purpose (lossless) algorithms
- On disk, not in memory or over the wire

Scope

TWO SIGMA

We'll talk about systems like:

- MySQL (InnoDB, TokuDB)
- MongoDB (WiredTiger, TokuMX, RocksDB)
- Cassandra
- PostgreSQL
- Vertica
- zfs, btrfs

Goal of the Talk

Goal

A **framework** for answering:

Goal

TWO SIGMA

A **framework** for answering:

- How do compression algorithms even work?

Goal

TWO SIGMA

A **framework** for answering:

- How do compression algorithms even work?
- How do storage systems use compression?

Goal

TWO SIGMA

A **framework** for answering:

- How do compression algorithms even work?
- How do storage systems use compression?
- How should I evaluate the compression of a storage system?

Goal

TWO SIGMA

A **framework** for answering:

- How do compression algorithms even work?
- How do storage systems use compression?
- How should I evaluate the compression of a storage system?
- How should I read articles about compression?

Goal

TWO SIGMA

A **framework** for answering:

- How do compression algorithms even work?
- How do storage systems use compression?
- How should I evaluate the compression of a storage system?
- How should I read articles about compression?
- How should I write articles about compression?

About Me

Me

TWO SIGMA

Engineer at Two Sigma

- We have a lot of data
- We care a lot about compression

Me

TWO SIGMA

Engineer at Two Sigma

- We have a lot of data
- We care a lot about compression

Previously at Tokutek

- Worked on TokuMX, TokuFT
- We thought a lot about compression
- We evaluated a lot of compression algorithms
- We wrote a lot about compression

How to Talk About Compression

TWO SIGMA

Scenario

I have a database which can store 1TB of “user data” in only 200GB of disk.

How to Talk About Compression

TWO SIGMA

Scenario

I have a database which can store 1TB of “user data” in only 200GB of disk.

How much is my database compressing?

Scenario

I have a database which can store 1TB of “user data” in only 200GB of disk.

How much is my database compressing? 80%? 20%? 5x? 1/5? 5:1?

How to Talk About Compression

TWO SIGMA

Scenario

I have a database which can store 1TB of "user data" in only 200GB of disk.

How much is my database compressing? 80%? 20%? 5x? 1/5? 5:1?

Let's talk about what this number is going to mean to us...

Why Compress?

Why Compress?

Data storage is expensive (you’ve heard this)

Why Compress?

TWO SIGMA

Data storage is expensive (you've heard this)

- Replication magnifies your data costs

Why Compress?

TWO SIGMA

Data storage is expensive (you've heard this)

- Replication magnifies your data costs
- Maintenance/operations cost scales superlinearly with hardware

Why Compress?

Data storage is expensive (you’ve heard this)

- Replication magnifies your data costs
- Maintenance/operations cost scales superlinearly with hardware
- SSD is expensive

Why Compress?

TWO SIGMA

Compression **magnifies** your capacity to store data at a fixed cost.

Compression **minimizes** your cost to provide a fixed capacity.

Why Compress?

TWO SIGMA

Compression **magnifies** your capacity to store data at a fixed cost.

We ask “by what **factor** does compression *multiply* my *capacity*?”

Compression **minimizes** your cost to provide a fixed capacity.

We ask “by what **factor** does compression *divide* my *cost*?”

Why Compress?

TWO SIGMA

Compression **magnifies** your capacity to store data at a fixed cost.

We ask "by what **factor** does compression *multiply* my *capacity*?"

Compression **minimizes** your cost to provide a fixed capacity.

We ask "by what **factor** does compression *divide* my *cost*?"

We should always talk about compression in terms of the **multiplicative factor** by which you increase your **cost-effectiveness**.

How to Talk About Compression

Say **“5x compression”**, not “80% compression”.

Cost Model

Cost Model

Compression is **more expensive** than decompression.

Cost Model

TWO SIGMA

Compression is **more expensive** than decompression.

- Compression is **searching** for repeated patterns in data. Searching is expensive.

Cost Model

TWO SIGMA

Compression is **more expensive** than decompression.

- Compression is **searching** for repeated patterns in data. Searching is expensive.
- Decompression is **copying** bytes out in the order described by encoding, which isn't very hard.

Cost Model

TWO SIGMA

Compression is **more expensive** than decompression.

- Compression is **searching** for repeated patterns in data. Searching is expensive.
- Decompression is **copying** bytes out in the order described by encoding, which isn't very hard.

Bandwidth speeds for typical compression algorithms (`cp` is a no-op)
(my laptop, Haswell CPU, Samsung SSD, 362MB tarball of `/usr/include`):

(MB/s)	zlib	bz2	lzma	lzo	lz4	zstd	cp
Compress	39	8	3	366	405	293	1466
Decompress	179	28	138	395	774	500	1466

(higher is better)

Cost Model

How does compression impact **perceived** performance?

Cost Model

TWO SIGMA

How does compression impact **perceived** performance?

Compression:

- Usually infrequent and done in the background
- Can reduce overall **throughput**

Cost Model

TWO SIGMA

How does compression impact **perceived** performance?

Compression:

- Usually infrequent and done in the background
- Can reduce overall **throughput**

Decompression:

- More frequent ("Write Once, Read Many") and on the critical path
- High impact on user-visible **latency**

Cost Model: Corollaries

Cost Model: Corollaries

TWO SIGMA

1. Do compression in the background and in large batches
   - Implement backpressure to avoid falling behind
   - If backpressure reaches users, try a faster compressor

Cost Model: Corollaries

TWO SIGMA

1. Do compression in the background and in large batches
    - Implement backpressure to avoid falling behind
    - If backpressure reaches users, try a faster compressor
2. Be sensitive to decompression latency
    - Hit the highest nail: other latency sources may be more important
    - Experiment with block sizes and faster compression algorithms

How Compression Even Works

How Compression Even Works

All* compression algorithms, at their core, use a form of *dictionary encoding*:

- Write down a dictionary of “common phrases” with shorter names
- Encode the input stream by referencing the short names in the dictionary

How Compression Even Works

TWO SIGMA

All* compression algorithms, at their core, use a form of *dictionary encoding*:

- Write down a dictionary of “common phrases” with shorter names
- Encode the input stream by referencing the short names in the dictionary

***A Universal Algorithm for Sequential Data Compression*, J. Ziv, A. Lempel 1977

How Compression Even Works

TWO SIGMA

All* compression algorithms, at their core, use a form of *dictionary encoding*:

- Write down a dictionary of "common phrases" with shorter names
- Encode the input stream by referencing the short names in the dictionary

```
abbabbabbcdcdcdcdabb => abb|abb|abb|cd|cd|cd|cd|abb
```

**A Universal Algorithm for Sequential Data Compression*, J. Ziv, A. Lempel 1977

How Compression Even Works

All* compression algorithms, at their core, use a form of *dictionary encoding*:

- Write down a dictionary of "common phrases" with shorter names
- Encode the input stream by referencing the short names in the dictionary

```
abbabbabbcdcdcdcdabb => abb|abb|abb|cd|cd|cd|cd|abb
```

Symbol	Phrase
x	abb
y	cd

, xxxyyyyx

**A Universal Algorithm for Sequential Data Compression*, J. Ziv, A. Lempel 1977

How Compression Even Works

TWO SIGMA

All* compression algorithms, at their core, use a form of *dictionary encoding*:

- Write down a dictionary of "common phrases" with shorter names
- Encode the input stream by referencing the short names in the dictionary

```
abbabbabbcdcdcdcdabb => abb|abb|abb|cd|cd|cd|cd|abb
```

Symbol	Phrase
x	abb
y	cd

, xxxyyyyx

To decompress: read the dictionary, use it to interpret the compressed stream.

A Universal Algorithm for Sequential Data Compression, J. Ziv, A. Lempel 1977

How Compression Even Works

Most compressors have a dynamic dictionary which is modified (optimized) as it compresses the input.

How Compression Even Works

Most compressors have a dynamic dictionary which is modified (optimized) as it compresses the input.

The dictionary takes up some space in the file header, so to be worthwhile, we want to compress a lot of input with it at once.

How Compression Even Works

We cannot seek directly to an offset in the decompressed output because:

How Compression Even Works

TWO SIGMA

We cannot seek directly to an offset in the decompressed output because:

- We need to read the compressed stream to modify the dictionary

How Compression Even Works

TWO SIGMA

We cannot seek directly to an offset in the decompressed output because:

- We need to read the compressed stream to modify the dictionary
- We don't know how much output any given chunk of input will produce

How Compression Even Works

TWO SIGMA

We cannot seek directly to an offset in the decompressed output because:

- We need to read the compressed stream to modify the dictionary
- We don't know how much output any given chunk of input will produce

We also can't update a compressed file without recompressing the whole thing.

How Compression Even Works

Systems that provide seeking in compressed data do so by dividing the input into **blocks**, and compressing them individually.

How Compression Even Works

TWO SIGMA

Systems that provide seeking in compressed data do so by dividing the input into **blocks**, and compressing them individually.

- When writing, recompress the whole block being written (but not the whole data set)

How Compression Even Works

Systems that provide seeking in compressed data do so by dividing the input into **blocks**, and compressing them individually.

- When writing, recompress the whole block being written (but not the whole data set)
- When reading, decompress the whole block being read

How Compression Even Works

TWO SIGMA

Systems that provide seeking in compressed data do so by dividing the input into **blocks**, and compressing them individually.

- When writing, recompress the whole block being written (but not the whole data set)
- When reading, decompress the whole block being read
- Overall compression ratio depends on the size of the blocks

Block Sizes

Block Sizes

TWO SIGMA

Compression algorithms are pattern finders. Give them more data to search in, and they find more patterns.

Block Sizes

TWO SIGMA

Compression algorithms are pattern finders. Give them more data to search in, and they find more patterns.

Compressors use **block sizes** to limit their runtime and memory usage.

Block Sizes

TWO SIGMA

Compression algorithms are pattern finders. Give them more data to search in, and they find more patterns.

Compressors use **block sizes** to limit their runtime and memory usage.

As block size increases:

Block Sizes

Compression algorithms are pattern finders. Give them more data to search in, and they find more patterns.

Compressors use **block sizes** to limit their runtime and memory usage.

As block size increases:

- Compression throughput decreases

Block Sizes

TWO SIGMA

Compression algorithms are pattern finders. Give them more data to search in, and they find more patterns.

Compressors use **block sizes** to limit their runtime and memory usage.

As block size increases:

- Compression throughput decreases
- Compression and decompression memory usage increases

Block Sizes

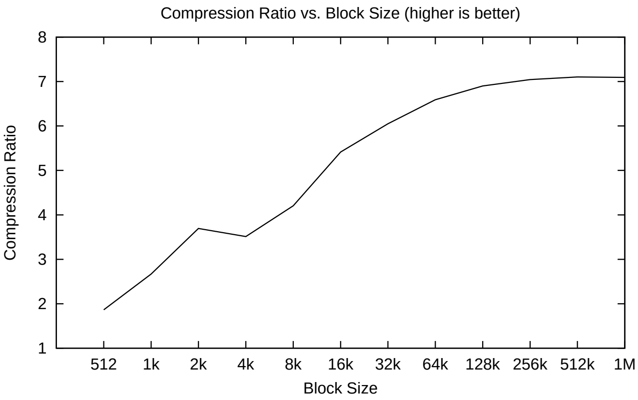
Compression algorithms are pattern finders. Give them more data to search in, and they find more patterns.

Compressors use **block sizes** to limit their runtime and memory usage.

As block size increases:

- Compression throughput decreases
- Compression and decompression memory usage increases
- Decompression throughput may increase if disk throughput increases

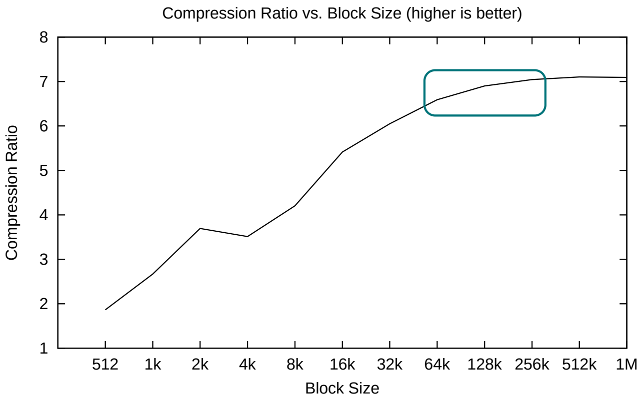
Block Sizes



Compression Ratio vs. Block Size (higher is better)
Compression Ratio
Block Size
1
2
3
4
5
6
7
8
512
1k
2k
4k
8k
16k
32k
64k
128k
256k
512k
1M

Block Sizes

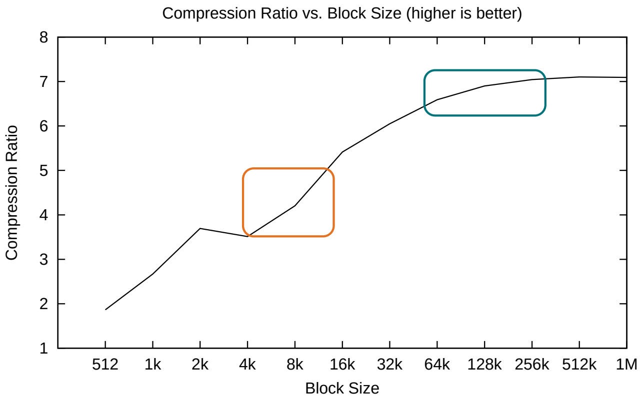
TWO SIGMA



The compression ratio sweet spot is ~**128k**, for gzip on this data set.

Block Sizes

TWO SIGMA

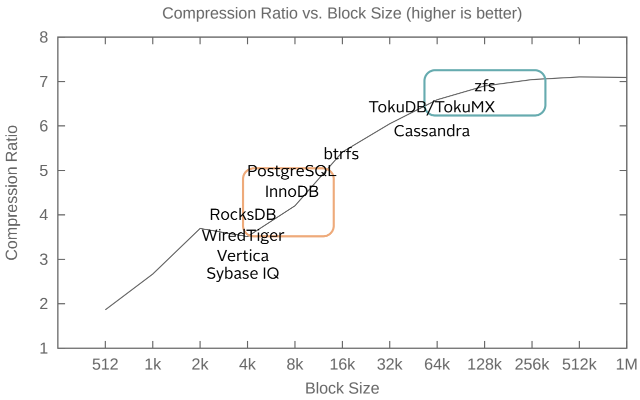


The compression ratio sweet spot is ~**128k**, for gzip on this data set.

Most systems use small blocks ~**8k**, to reduce decompression latency.

Block Sizes

TWO SIGMA



The compression ratio sweet spot is ~**128k**, for gzip on this data set.

Most systems use small blocks ~**8k**, to reduce decompression latency.

Fragmentation

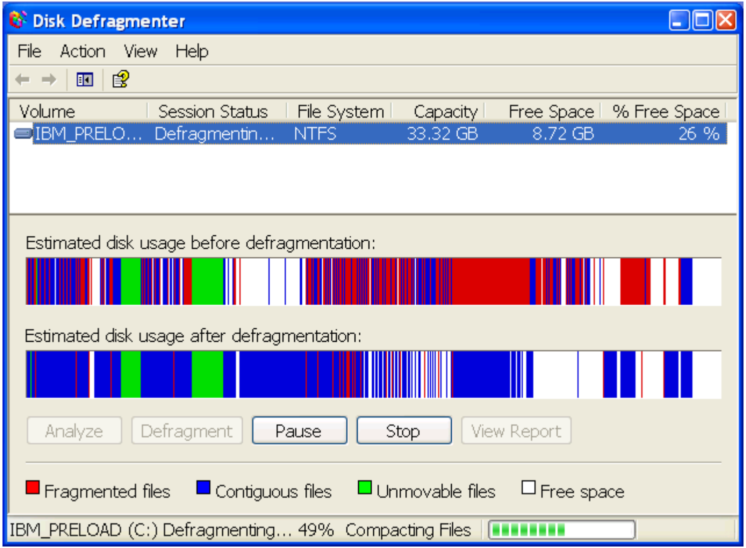
Another corollary of compressing in blocks is **fragmentation**.

Fragmentation

Another corollary of compressing in blocks is **fragmentation**.

Blocks need to be allocated locations on disk. As the data grows, shrinks, and moves around, these locations (and for some systems, allocation sizes) change.

Fragmentation



Fragmentation

Fragmentation hurts you in two ways:

Fragmentation

TWO SIGMA

Fragmentation hurts you in two ways:

1. Fragmented files occupy more **effective space** than defragmented ones

Fragmentation

TWO SIGMA

Fragmentation hurts you in two ways:

1. Fragmented files occupy more **effective space** than defragmented ones
2. Fragmentation degrades range query throughput by reducing data locality

Fragmentation

TWO SIGMA

Fragmentation hurts you in two ways:

1. Fragmented files occupy more **effective space** than defragmented ones
2. Fragmentation degrades range query throughput by reducing data locality

For some systems, the overall compression ratio will be reduced once fragmentation develops.

Entropy

Entropy

Not all data compresses equally!

Entropy

Not all data compresses equally!

*Information Theory** can tell us how much **real information** is present in a set of data ("bits of entropy").

Entropy

TWO SIGMA

Not all data compresses equally!

*Information Theory** can tell us how much **real information** is present in a set of data ("bits of entropy").

**A Mathematical Theory of Communication*, C. E. Shannon, 1948

Entropy

TWO SIGMA

Not all data compresses equally!

*Information Theory** can tell us how much **real information** is present in a set of data ("bits of entropy").

A general-purpose, lossless compression algorithm can't hope to compress data smaller than that.

**A Mathematical Theory of Communication*, C. E. Shannon, 1948

Entropy

Not all data compresses equally!

*Information Theory** can tell us how much **real information** is present in a set of data ("bits of entropy").

A general-purpose, lossless compression algorithm can't hope to compress data smaller than that.

If it could, it would have to produce the same compressed output for multiple inputs, which would mean it isn't *lossless*.

**A Mathematical Theory of Communication*, C. E. Shannon, 1948

Not all data compresses equally!

*Information Theory** can tell us how much **real information** is present in a set of data ("bits of entropy").

A general-purpose, lossless compression algorithm can't hope to compress data smaller than that.

If it could, it would have to produce the same compressed output for multiple inputs, which would mean it isn't *lossless*.

High entropy data is *highly uncompressible*.
Low entropy data is *easily compressed*.

**A Mathematical Theory of Communication*, C. E. Shannon, 1948

Entropy: Experiment

Entropy: Experiment

Built 8 data sources ($\sim$50k each):

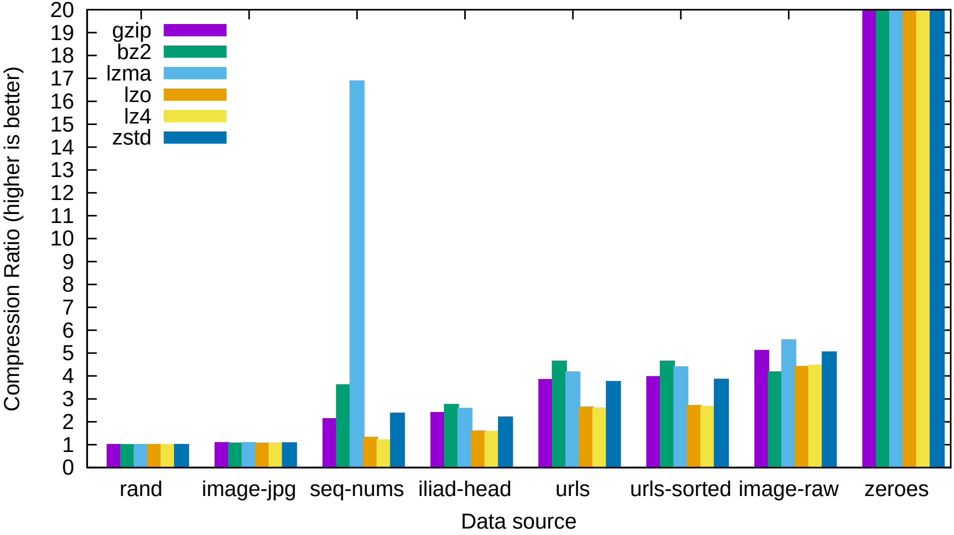
Entropy: Experiment

TWO SIGMA

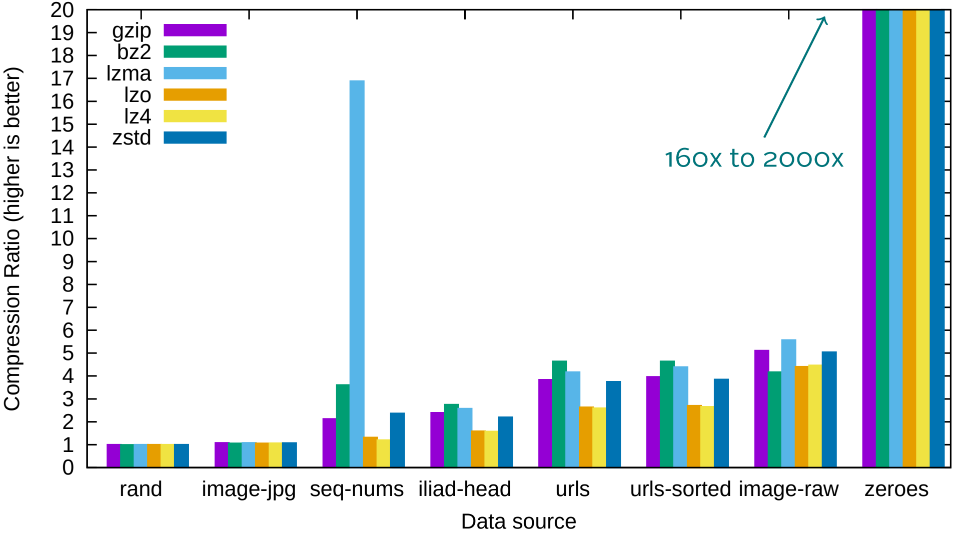
Built 8 data sources (∼50k each):

1. Random bytes
2. Sequential numbers, encoded as ASCII decimals
3. All zeroes
4. The beginning of *The Iliad*
5. 1000 random Wikipedia URLs
6. 1000 random Wikipedia URLs, sorted
7. RAW image (CR2)
8. JPEG-compressed image

Entropy: Experiment

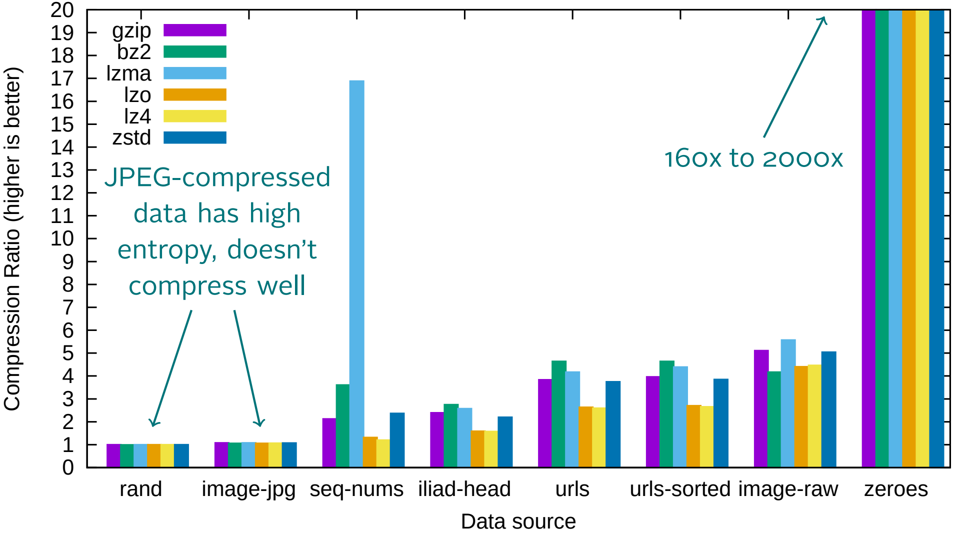


Entropy: Experiment



Entropy: Experiment

TWO SIGMA



Entropy

TWO SIGMA

Homogeneous data has lower entropy than *heterogeneous* data.

Entropy

Homogeneous data has lower entropy than *heterogeneous* data.

- Integers compress better than documents with complex interal structure

Entropy

TWO SIGMA

Homogeneous data has lower entropy than *heterogeneous* data.

- Integers compress better than documents with complex interal structure

Column stores have a compression advantage over **row stores**.

Know your data!

Don’t waste your time compressing JPEG blobs

TWO SIGMA

Know your data!
Don't waste your time compressing JPEG blobs

Some compressors are fantastic in specific data domains
(VLQ, delta coding, JPEG, MP3, ...)

Know your data!
Don't waste your time compressing JPEG blobs

Some compressors are fantastic in specific data domains
(VLQ, delta coding, JPEG, MP3, ...)

(But 95% of the time, gzip is fine)

Before we use compression, we need to understand the **costs** and **benefits** to our application.

Benchmarking

Benchmarking

When designing a compression benchmark, you should consider:

Benchmarking

TWO SIGMA

When designing a compression benchmark, you should consider:

- Execution

Benchmarking

TWO SIGMA

When designing a compression benchmark, you should consider:

- Execution
- Measurement

Benchmarking

TWO SIGMA

When designing a compression benchmark, you should consider:

- Execution
- Measurement
- Presentation

Benchmarking: Execution

TWO SIGMA

Main Question

Is the workload representative of a real-world use-case?

Benchmarking: Execution

TWO SIGMA

1. Sample real data if you can get it.

Benchmarking: Execution

TWO SIGMA

1. Sample real data if you can get it.

   If not, generate *plausibly realistic* data:

Benchmarking: Execution

TWO SIGMA

1. Sample real data if you can get it.

   If not, generate *plausibly realistic* data:

   - Zeroes: bad
   - Random: bad
   - 25% random and 75% zeroes: meh
   - JSON blobs: good

Benchmarking: Execution

TWO SIGMA

2. Use a realistic read/insert/update mixture:

Benchmarking: Execution

TWO SIGMA

2. Use a realistic read/insert/update mixture:
   - Most applications are read-heavy
   - Favors fast decompressors

Benchmarking: Execution

TWO SIGMA

3. Use a realistic insert/update distribution:

Benchmarking: Execution

TWO SIGMA

3. Use a realistic insert/update distribution:
   - Most applications don't write uniformly over the keyspace

Benchmarking: Execution

3. Use a realistic insert/update distribution:
   - Most applications don't write uniformly over the keyspace
   - Zipfian or Pareto (or sometimes sequential, or nearly) distributions are more realistic, and cache-friendlier

Benchmarking: Execution

TWO SIGMA

3. Use a realistic insert/update distribution:
   - Most applications don't write uniformly over the keyspace
   - Zipfian or Pareto (or sometimes sequential, or nearly) distributions are more realistic, and cache-friendlier
   - Vadim wrote a `sysbench` workload generator that uses a Zipfian distribution: `http://j.mp/sysbench-zipf`

Benchmarking: Execution

TWO SIGMA

4. To measure **latency**, throttle your workload.

Benchmarking: Execution

TWO SIGMA

4. To measure **latency**, throttle your workload.
   - Full-throughput workloads will induce artificial latency spikes (fsyncs, GC)

Benchmarking: Execution

TWO SIGMA

4. To measure **latency**, throttle your workload.
   - Full-throughput workloads will induce artificial latency spikes (fsyncs, GC)

To measure **max throughput**, run at full speed.

Benchmarking: Execution

TWO SIGMA

4. To measure **latency**, throttle your workload.
   - Full-throughput workloads will induce artificial latency spikes (fsyncs, GC)

To measure **max throughput**, run at full speed.

You should do both.

Benchmarking: Execution

TWO SIGMA

5. Run for a *long time*. Lots of important properties don't become visible immediately (e.g. fragmentation), and you need to understand them.

Benchmarking: Execution

TWO SIGMA

5. Run for a *long time*. Lots of important properties don't become visible immediately (e.g. fragmentation), and you need to understand them.

   Your application is hopefully going to run for months or years. You don't want to be surprised by degradation after you think everything's stable.

Benchmarking: Execution

6. Parameterize your workload:

Benchmarking: Execution

TWO SIGMA

6. Parameterize your workload:
   - Read/insert/update mixture

Benchmarking: Execution

6. Parameterize your workload:
   - Read/insert/update mixture
   - Write distribution

Benchmarking: Execution

6. Parameterize your workload:
    - Read/insert/update mixture
    - Write distribution
    - Number of threads

Benchmarking: Execution

TWO SIGMA

6. Parameterize your workload:
   - Read/insert/update mixture
   - Write distribution
   - Number of threads
   - Data size

Benchmarking: Execution

TWO SIGMA

6. Parameterize your workload:
    - Read/insert/update mixture
    - Write distribution
    - Number of threads
    - Data size
    - Throttling

Benchmarking: Execution

TWO SIGMA

6. Parameterize your workload:
   - Read/insert/update mixture
   - Write distribution
   - Number of threads
   - Data size
   - Throttling
   - Duration

Benchmarking: Execution

TWO SIGMA

6. Parameterize your workload:
    - Read/insert/update mixture
    - Write distribution
    - Number of threads
    - Data size
    - Throttling
    - Duration
    - System configuration (cache size, isolation levels, log commit)

Benchmarking: Execution

TWO SIGMA

6. Parameterize your workload:
   - Read/insert/update mixture
   - Write distribution
   - Number of threads
   - Data size
   - Throttling
   - Duration
   - System configuration (cache size, isolation levels, log commit)

You are going to want to explore these parameter spaces. Save yourself the pain later and think about parameterization up front.

Benchmarking: Execution

TWO SIGMA

Great example: `https://github.com/ParsePlatform/flashback`

Captures a MongoDB workload with profiling, then replays operations either at their original timestamps, or at full speed.

Benchmarking: Measurement

Benchmarking: Measurement

TWO SIGMA

Benchmarking: Measurement

Application metrics:

Benchmarking: Measurement

TWO SIGMA

Application metrics:

- Throughput
- Latency
- Aborted/retried transactions

Benchmarking: Measurement

TWO SIGMA

Application metrics:

- Throughput
- Latency
- Aborted/retried transactions

Instrument your application so you know *which* operations are expensive.

Benchmarking: Measurement

TWO SIGMA

System metrics:

Benchmarking: Measurement

System metrics:

- CPU
- Memory (RSS)
- I/O
- Network
- Actual storage usage (`du`)

Benchmarking: Measurement

TWO SIGMA

System metrics:

- CPU
- Memory (RSS)
- I/O
- Network
- Actual storage usage (`du`)

`perf(1)`, `iostat(1)`, `dstat(1)`, `oprofile(1)`, `collectd(1)`, Datadog, Librato, ...

Benchmarking: Measurement

Database/filesystem metrics (product-specific):

Benchmarking: Measurement

Database/filesystem metrics (product-specific):

- Cache hits/misses
- Replication lag
- Checkpoint lag

Benchmarking: Measurement

Database/filesystem metrics (product-specific):

- Cache hits/misses
- Replication lag
- Checkpoint lag

Talk to your storage vendor about what’s important.

Benchmarking: Presentation

TWO SIGMA

Benchmarking: Presentation

1. Describe the workload, and make a case for why it's **realistic**

Benchmarking: Presentation

TWO SIGMA

1. Describe the workload, and make a case for why it's **realistic**
2. Choose **key metrics** that reflect the *benefits* of compression (e.g. users stored per TB) as well as the *costs* (e.g. operation latency)

Benchmarking: Presentation

TWO SIGMA

1. Describe the workload, and make a case for why it's **realistic**
2. Choose **key metrics** that reflect the *benefits* of compression (e.g. users stored per TB) as well as the *costs* (e.g. operation latency)
3. Demonstrate which **parameter choices** influence the costs and benefits you think are important

Benchmarking: Presentation

TWO SIGMA

1. Describe the workload, and make a case for why it's **realistic**
2. Choose **key metrics** that reflect the *benefits* of compression (e.g. users stored per TB) as well as the *costs* (e.g. operation latency)
3. Demonstrate which **parameter choices** influence the costs and benefits you think are important
4. **Explain** which parameters have little or no effect on your metrics
5. Explain how much of your measurement is **overhead**.
6. *If* you show charts, **normalize** your data. Only present important differences.

Review

TWO SIGMA

Say **“5x compression”**

TWO SIGMA

Compression is **slower** than decompression, but decompression is **more frequent**

TWO SIGMA

Large blocks compress better

TWO SIGMA

Fragmentation degrades
effective compression over time

High entropy data is **less compressible**

Benchmark **realistic workloads** over a long period

TWO SIGMA

Present **responsibly**
(and distrust benchmarketers who don't)

Thanks!

TWO SIGMA

- Tim and Mark Callaghan for being exemplar benchmarkers (`http://acmebenchmarking.com` and `http://smalldatum.blogspot.com`)
- Bohu Tang for introducing me to zstd
- Andrew Bolin, Corey Milloy, Effie Baram, Li Jin, Wil Yegelwel for making this talk better
- Tokutek engineering
- Percona (they're also good benchmarkers)

Questions?

TWO SIGMA

Leif Walsh
leif.walsh@gmail.com
@leifwalsh