Reducing Replication Bandwidth for Distributed Document Databases

Lianghong Xu[1], Andy Pavlo[1], Sudipta Sengupta[2]
Jin Li[2], Greg Ganger[1]
Carnegie Mellon University[1], Microsoft Research[2]

Andy Pavlo
@andy_pavlo

Today I am visiting @eliothorowitz at @mongodbinc to try to convince them to ditch MMAP & switch to anti-caching.

RETWEETS 3 FAVORITES 3

9:57 AM - 3 Dec 2013

New York, NY

#1 – You can sleep with grad students but <u>not</u> undergrads.

#2 – Keep a bottle of water in your office in case a student breaks down crying.

#3 – Kids love MongoDB, but they want to go work for Google.

System Votes

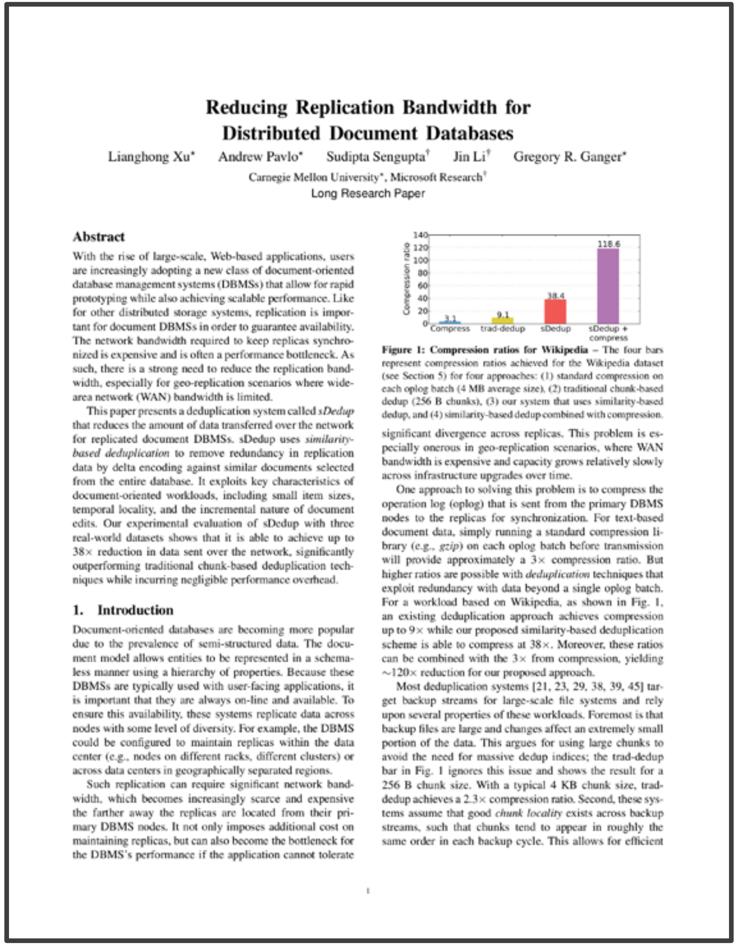
Google Spanner	**24**
mongoDB	**23**
redis	**10**
amazon DynamoDB	**5**
MySQL	**2**
APACHE HBASE	**1**
dbShards	**1**

Reducing Replication Bandwidth for Distributed Document Databases

In ACM Symposium on Cloud Computing, pg. 1-12, August 2015.

More Info:

http://cmudb.io/doc-dbs



Reducing Replication Bandwidth for Distributed Document Databases

Lianghong Xu* Andrew Pavlo* Sudipta Sengupta† Jin Li† Gregory R. Ganger*
Carnegie Mellon University*, Microsoft Research†

Long Research Paper

Abstract

With the rise of large-scale, Web-based applications, users are increasingly adopting a new class of document-oriented database management systems (DBMSs) that allow for rapid prototyping while also achieving scalable performance. Like for other distributed storage systems, replication is important for document DBMSs in order to guarantee availability. The network bandwidth required to keep replicas synchronized is expensive and is often a performance bottleneck. As such, there is a strong need to reduce the replication bandwidth, especially for geo-replication scenarios where wide-area network (WAN) bandwidth is limited.

This paper presents a deduplication system called *sDedup* that reduces the amount of data transferred over the network for replicated document DBMSs. sDedup uses *similarity-based deduplication* to remove redundancy in replication data by delta encoding against similar documents selected from the entire database. It exploits key characteristics of document-oriented workloads, including small item sizes, temporal locality, and the incremental nature of document edits. Our experimental evaluation of sDedup with three real-world datasets shows that it is able to achieve up to 38× reduction in data sent over the network, significantly outperforming traditional chunk-based deduplication techniques while incurring negligible performance overhead.

1. Introduction

Document-oriented databases are becoming more popular due to the prevalence of semi-structured data. The document model allows entities to be represented in a schema-less manner using a hierarchy of properties. Because these DBMSs are typically used with user-facing applications, it is important that they are always on-line and available. To ensure this availability, these systems replicate data across nodes with some level of diversity. For example, the DBMS could be configured to maintain replicas within the data center (e.g., nodes on different racks, different clusters) or across data centers in geographically separated regions.

Such replication can require significant network bandwidth, which becomes increasingly scarce and expensive the farther away the replicas are located from their primary DBMS nodes. It not only imposes additional cost on maintaining replicas, but can also become the bottleneck for the DBMS's performance if the application cannot tolerate

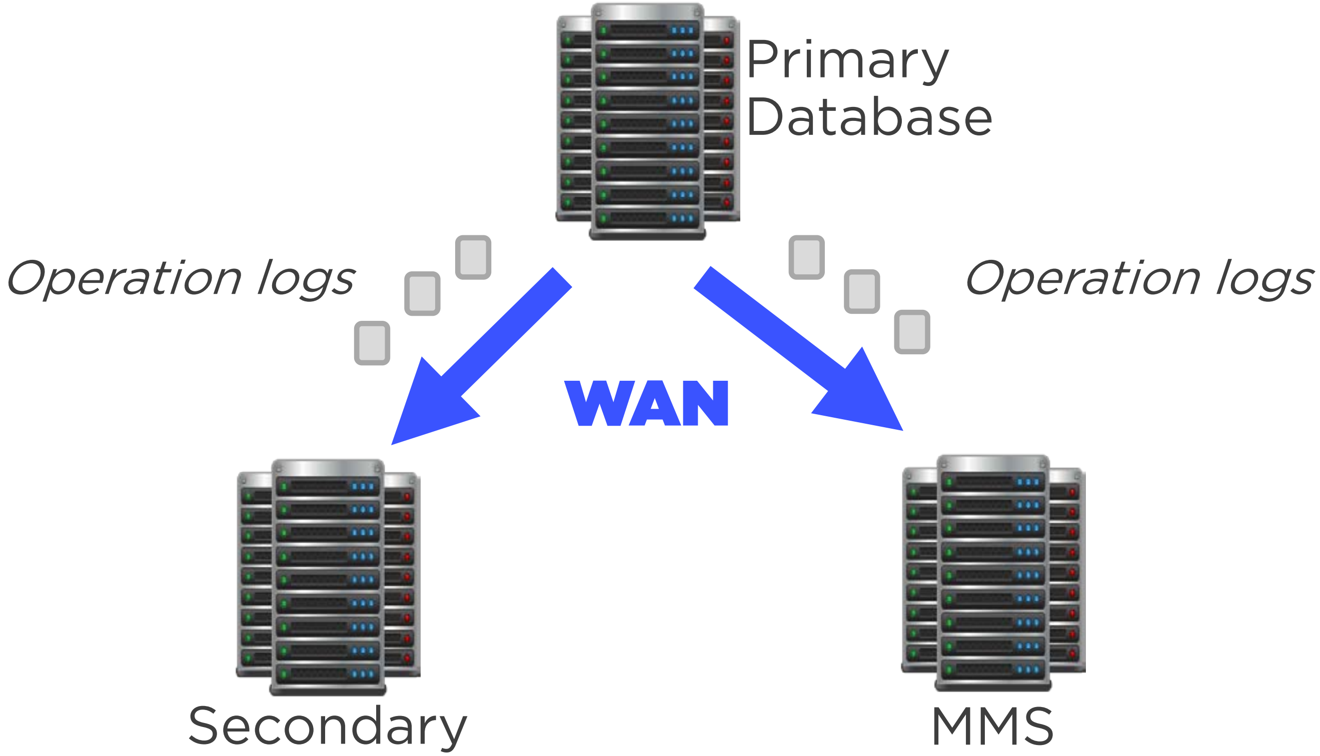
Figure 1: Compression ratios for Wikipedia – The four bars represent compression ratios achieved for the Wikipedia dataset (see Section 5) for four approaches: (1) standard compression on each oplog batch (4 MB average size), (2) traditional chunk-based dedup (256 B chunks), (3) our system that uses similarity-based dedup, and (4) similarity-based dedup combined with compression.

significant divergence across replicas. This problem is especially onerous in geo-replication scenarios, where WAN bandwidth is expensive and capacity grows relatively slowly across infrastructure upgrades over time.

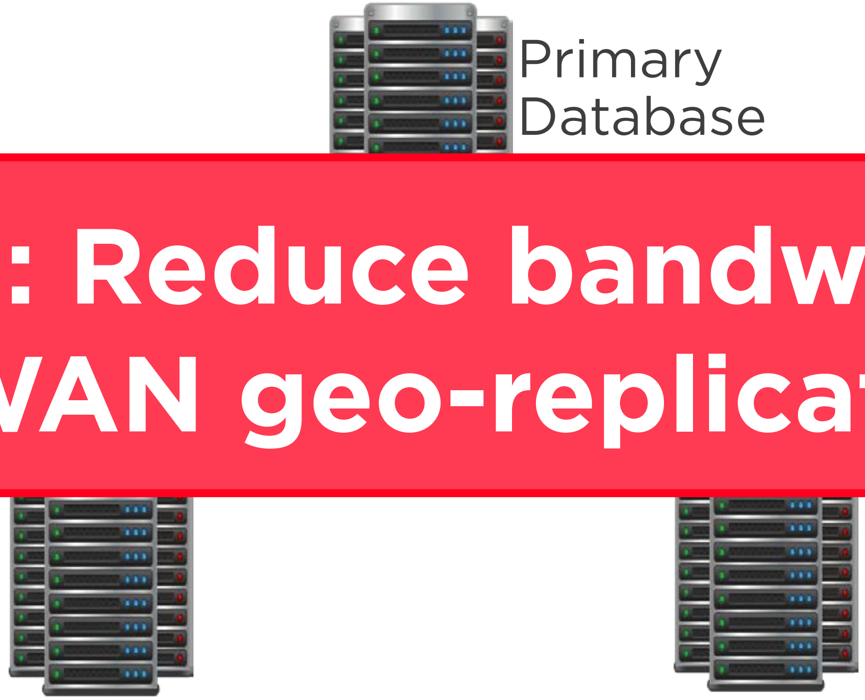
One approach to solving this problem is to compress the operation log (oplog) that is sent from the primary DBMS nodes to the replicas for synchronization. For text-based document data, simply running a standard compression library (e.g., *gzip*) on each oplog batch before transmission will provide approximately a 3× compression ratio. But higher ratios are possible with *deduplication* techniques that exploit redundancy with data beyond a single oplog batch. For a workload based on Wikipedia, as shown in Fig. 1, an existing deduplication approach achieves compression up to 9× while our proposed similarity-based deduplication scheme is able to compress at 38×. Moreover, these ratios can be combined with the 3× from compression, yielding ~120× reduction for our proposed approach.

Most deduplication systems [21, 23, 29, 38, 39, 45] target backup streams for large-scale file systems and rely upon several properties of these workloads. Foremost is that backup files are large and changes affect an extremely small portion of the data. This argues for using large chunks to avoid the need for massive dedup indices; the trad-dedup bar in Fig. 1 ignores this issue and shows the result for a 256 B chunk size. With a typical 4 KB chunk size, trad-dedup achieves a 2.3× compression ratio. Second, these systems assume that good *chunk locality* exists across backup streams, such that chunks tend to appear in roughly the same order in each backup cycle. This allows for efficient

Replication Bandwidth



Replication Bandwidth



Goal: Reduce bandwidth for WAN geo-replication.

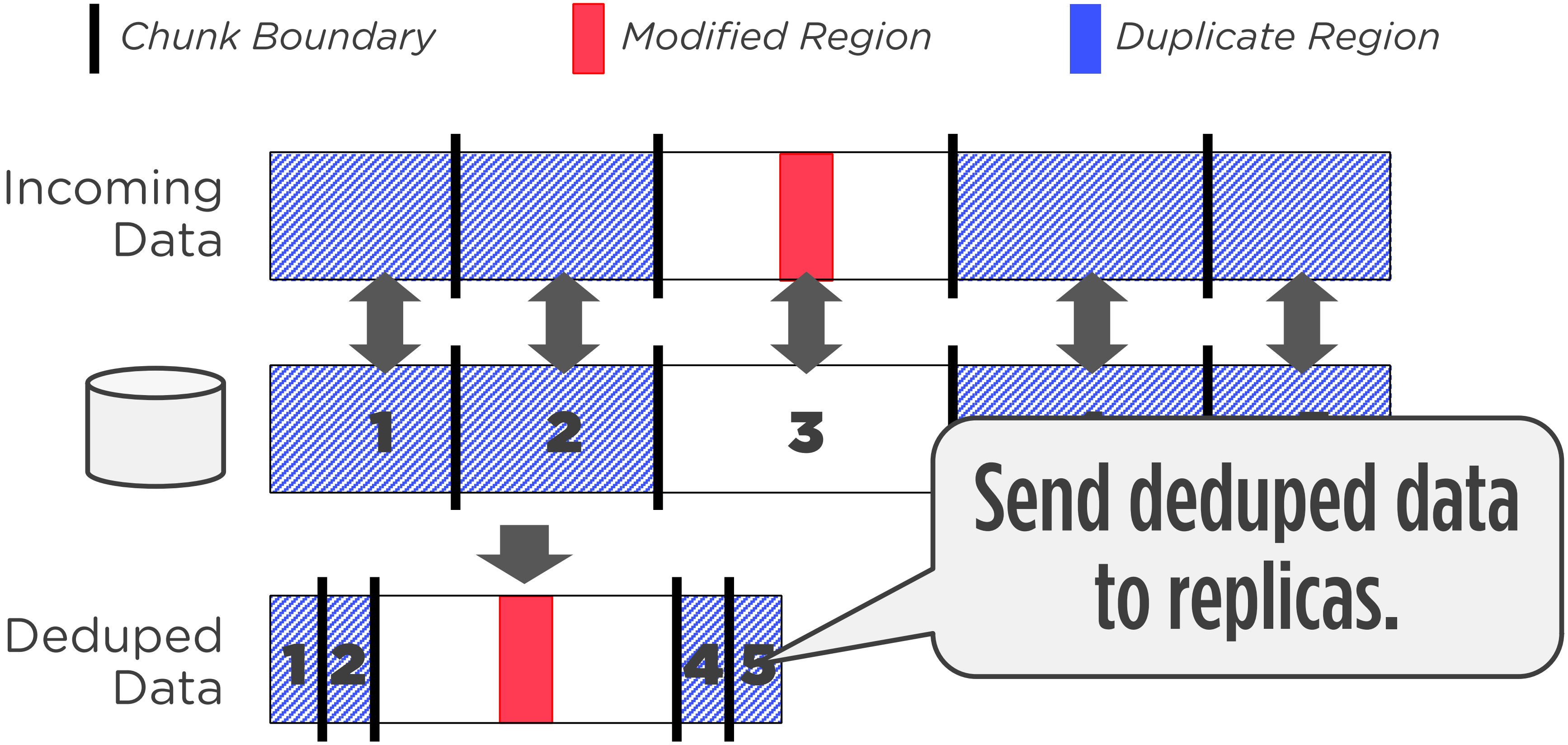
Why Deduplication?

- Why not just compress?
  - *Oplog batches are small and not enough overlap.*

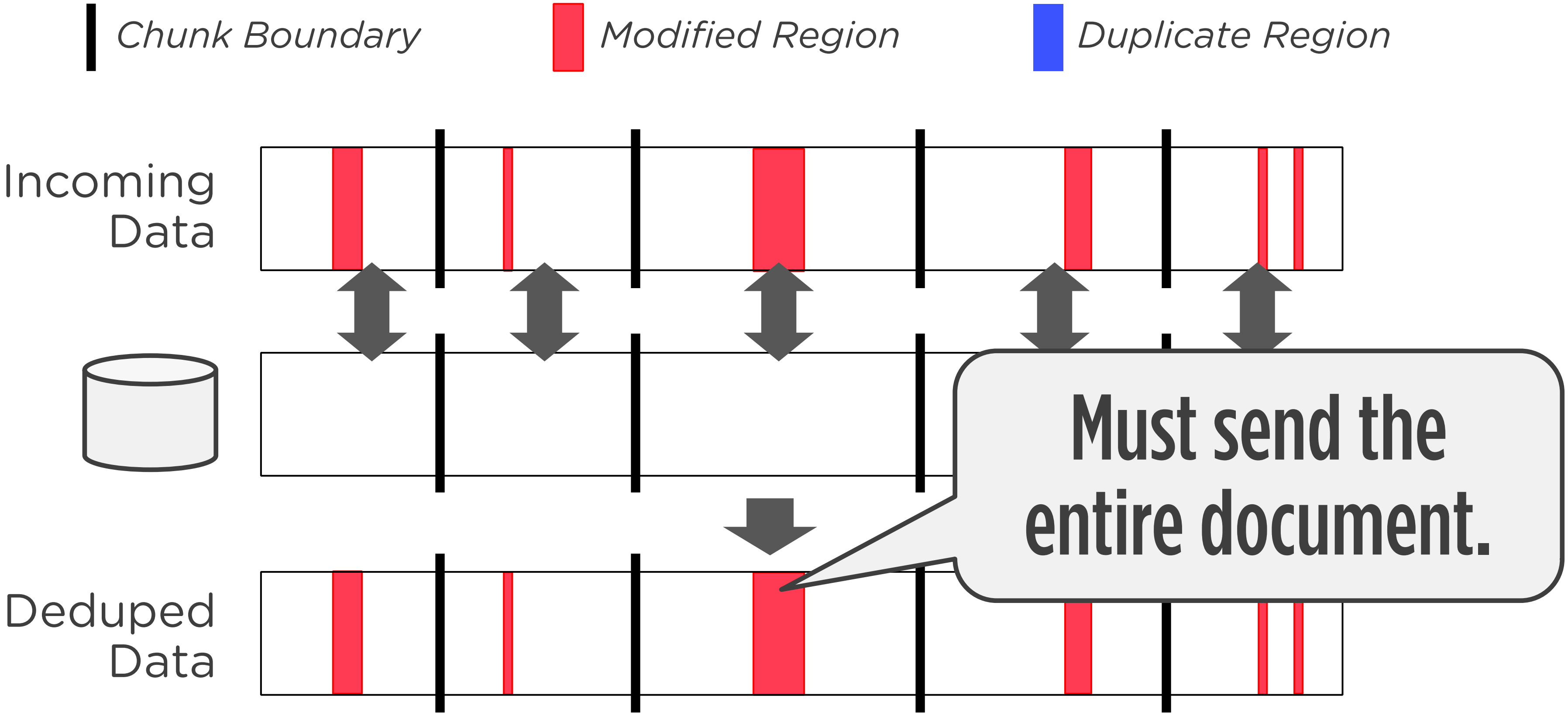
- Why not just use diff?
  - *Need application guidance to identify source.*

- Deduplication finds and removes redundancies.

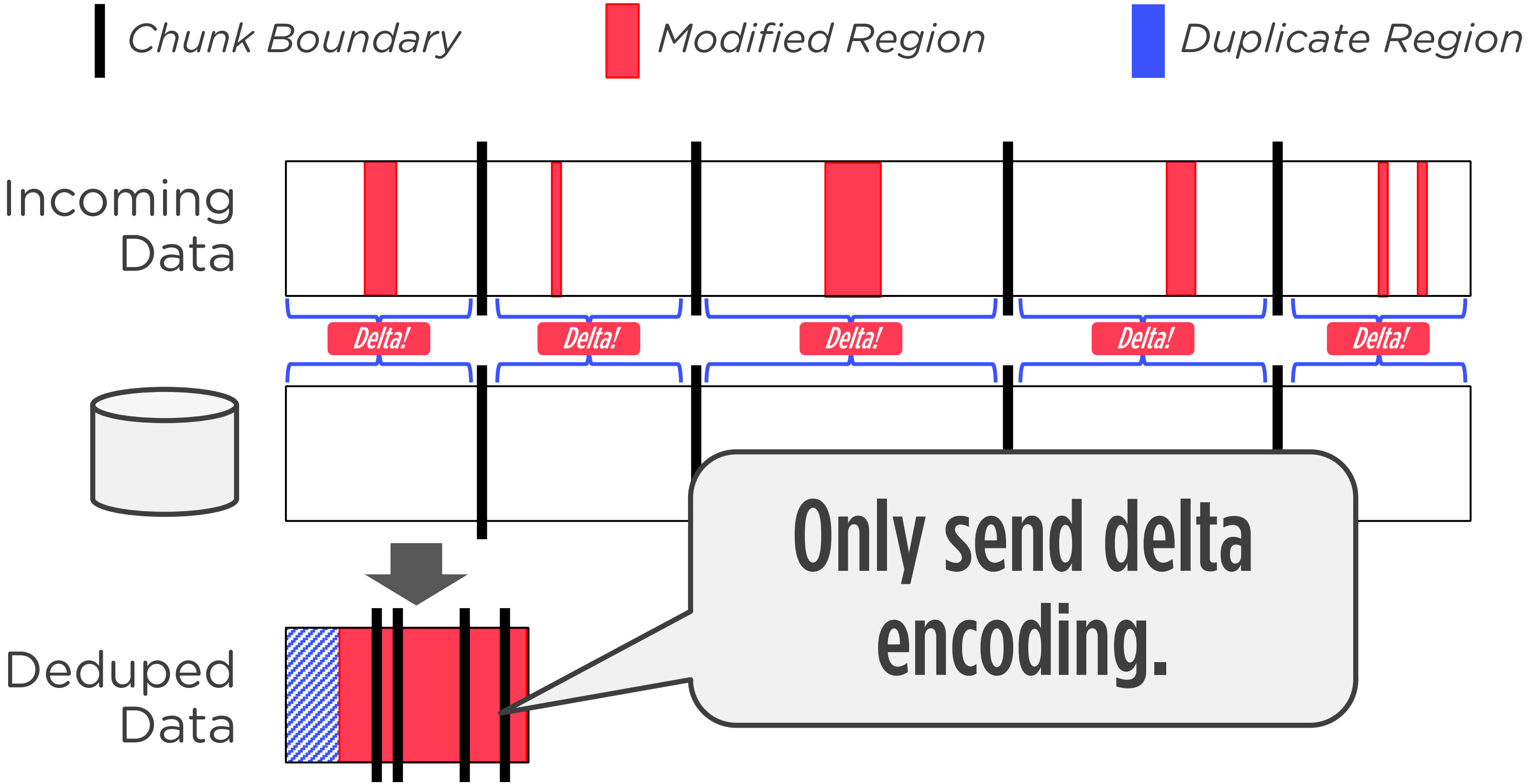
Traditional Dedup



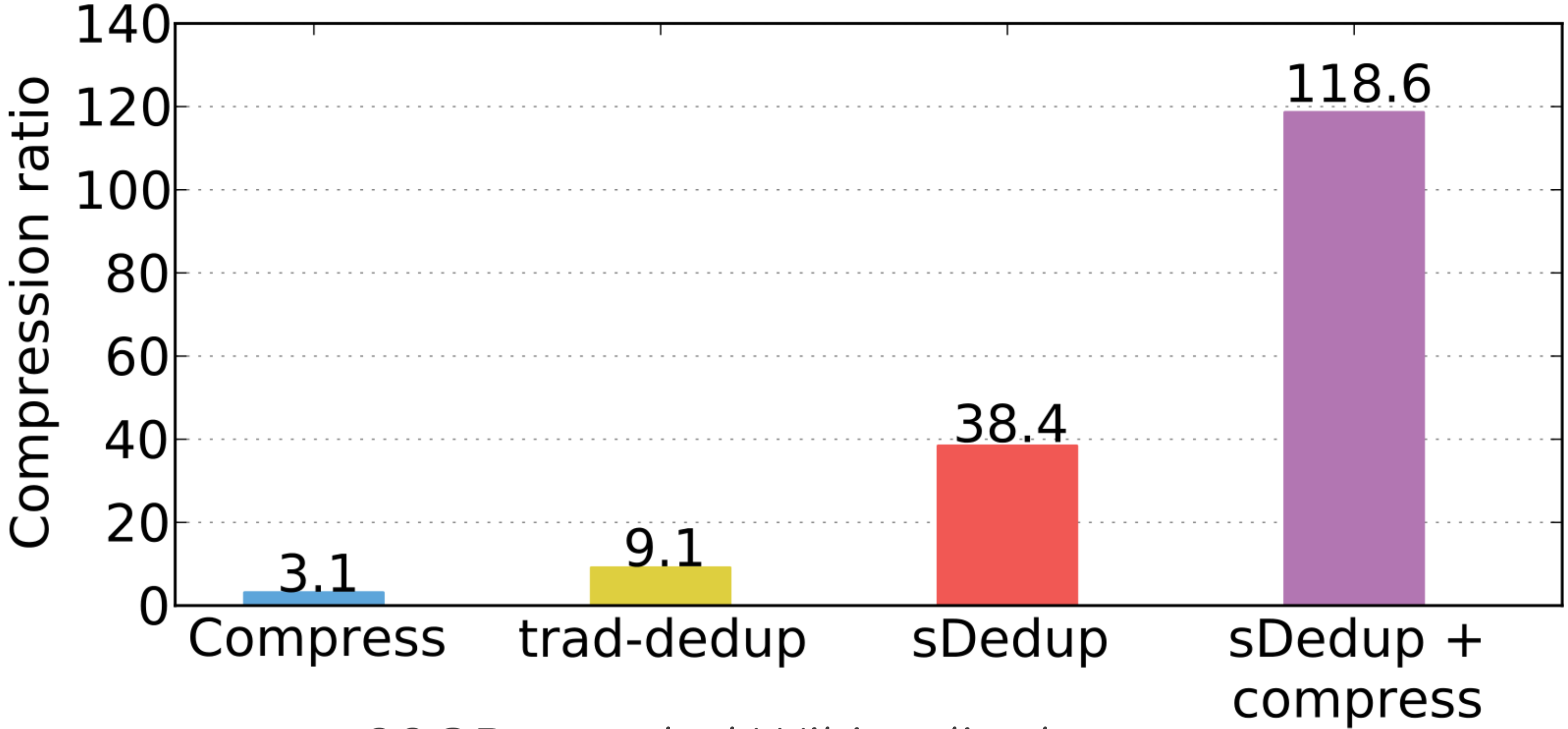
Traditional Dedup



Similarity Dedup

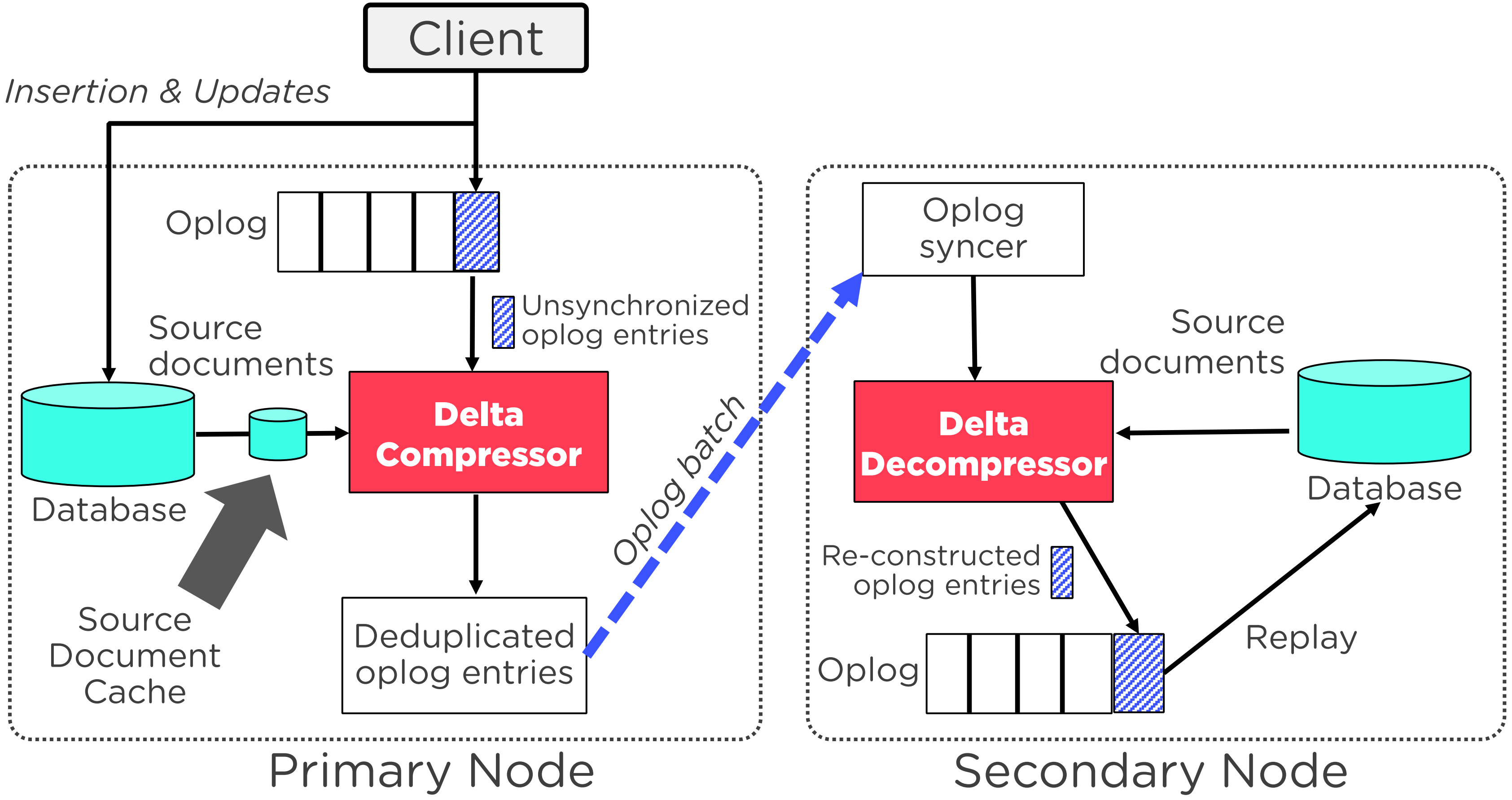


Compress vs. Dedup



20GB sampled Wikipedia dataset.
MongoDB v2.7 // 4MB Oplog batches

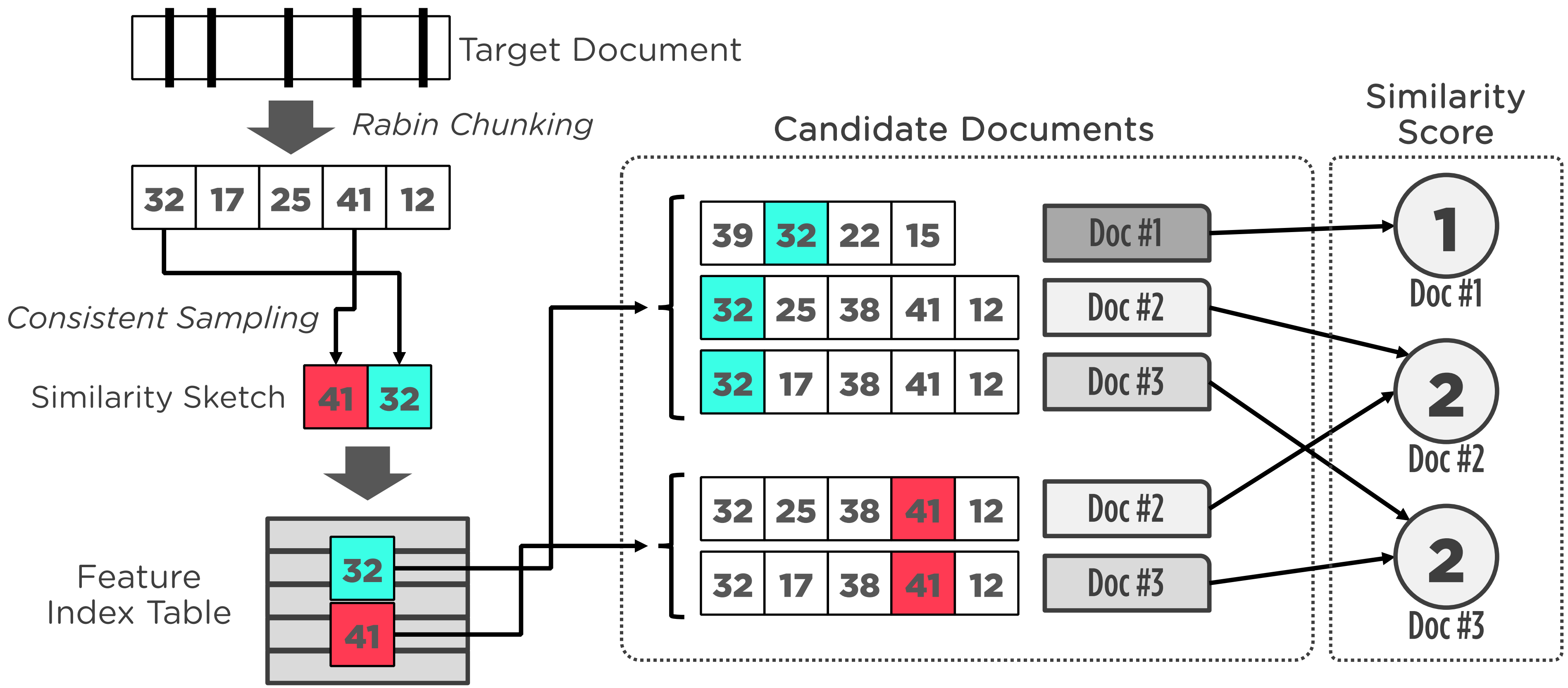
sDedup: Similarity Dedup



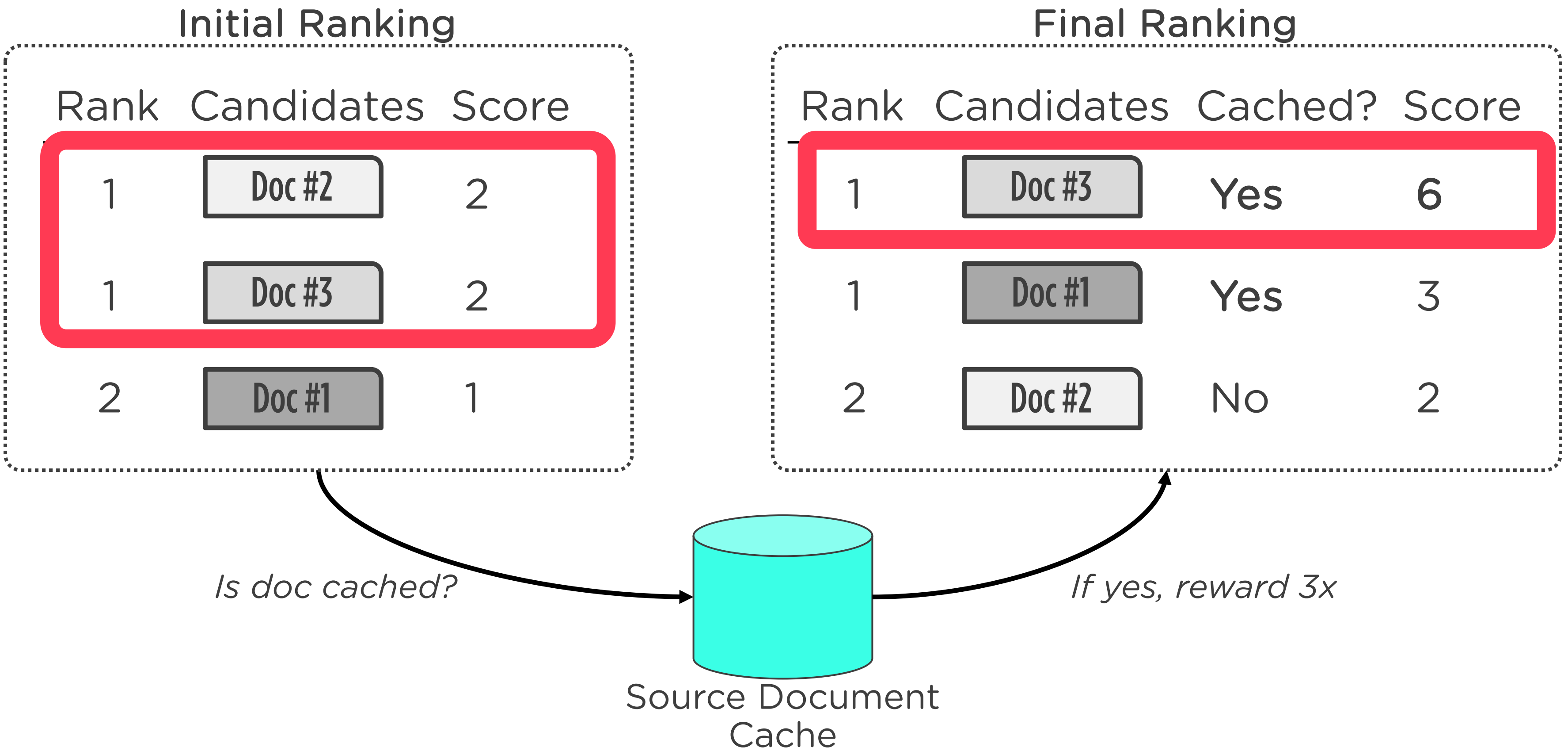
Encoding Steps

- Identify Similar Documents
- Select the Best Match
- Delta Compression

Identify Similar Documents



Selecting the Best Match



Delta Compression

- Byte-level diff between source and target docs:
  - *Based on the xDelta algorithm*
  - *Improved speed with minimal loss of compression*

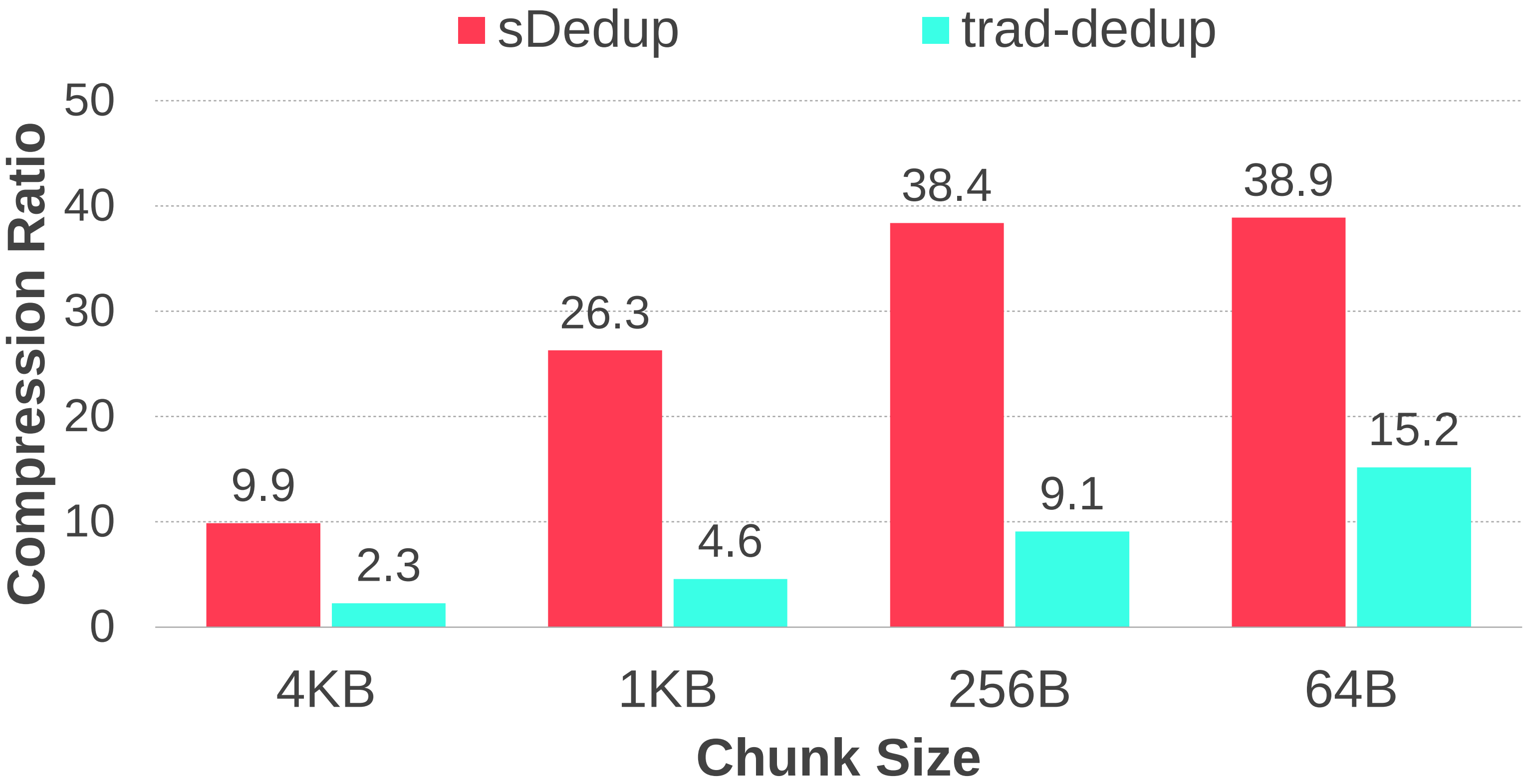
- **Encoding**:
  - *Descriptors about duplicate/unique regions + unique bytes*
- **Decoding**:
  - *Use source doc + encoded output*
  - *Concatenate byte regions in order*

Evaluation

- MongoDB setup (v2.7)
  - *1 primary, 1 secondary node, 1 client*
  - *Node Config: 4 cores, 8GB RAM, 100GB HDD storage*

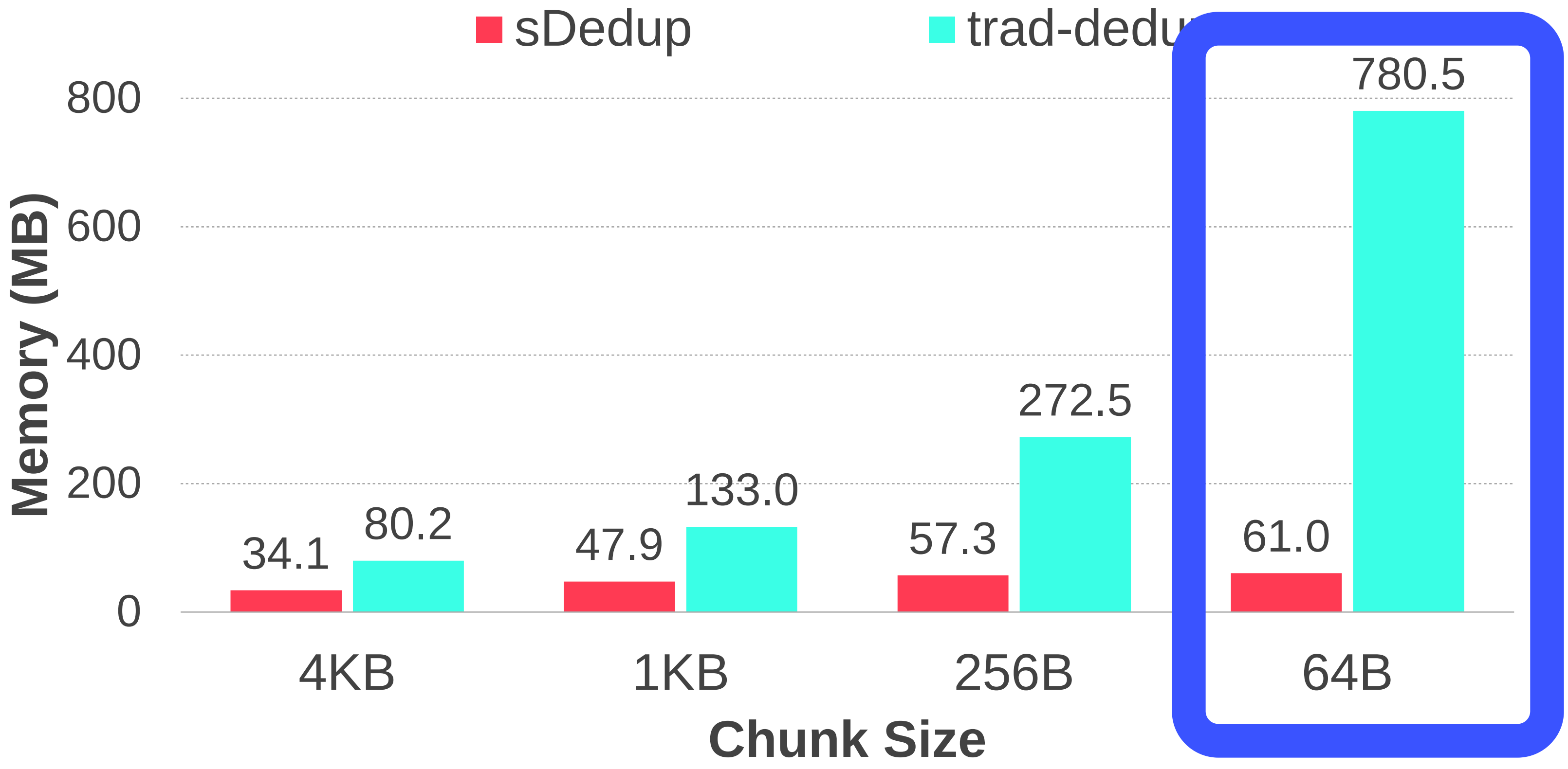
- Datasets:
  - *Wikipedia dump (20GB out of ~12TB)*
  - *Stack Exchange data dump (10GB out of ~100GB)*

Compression: Wikipedia



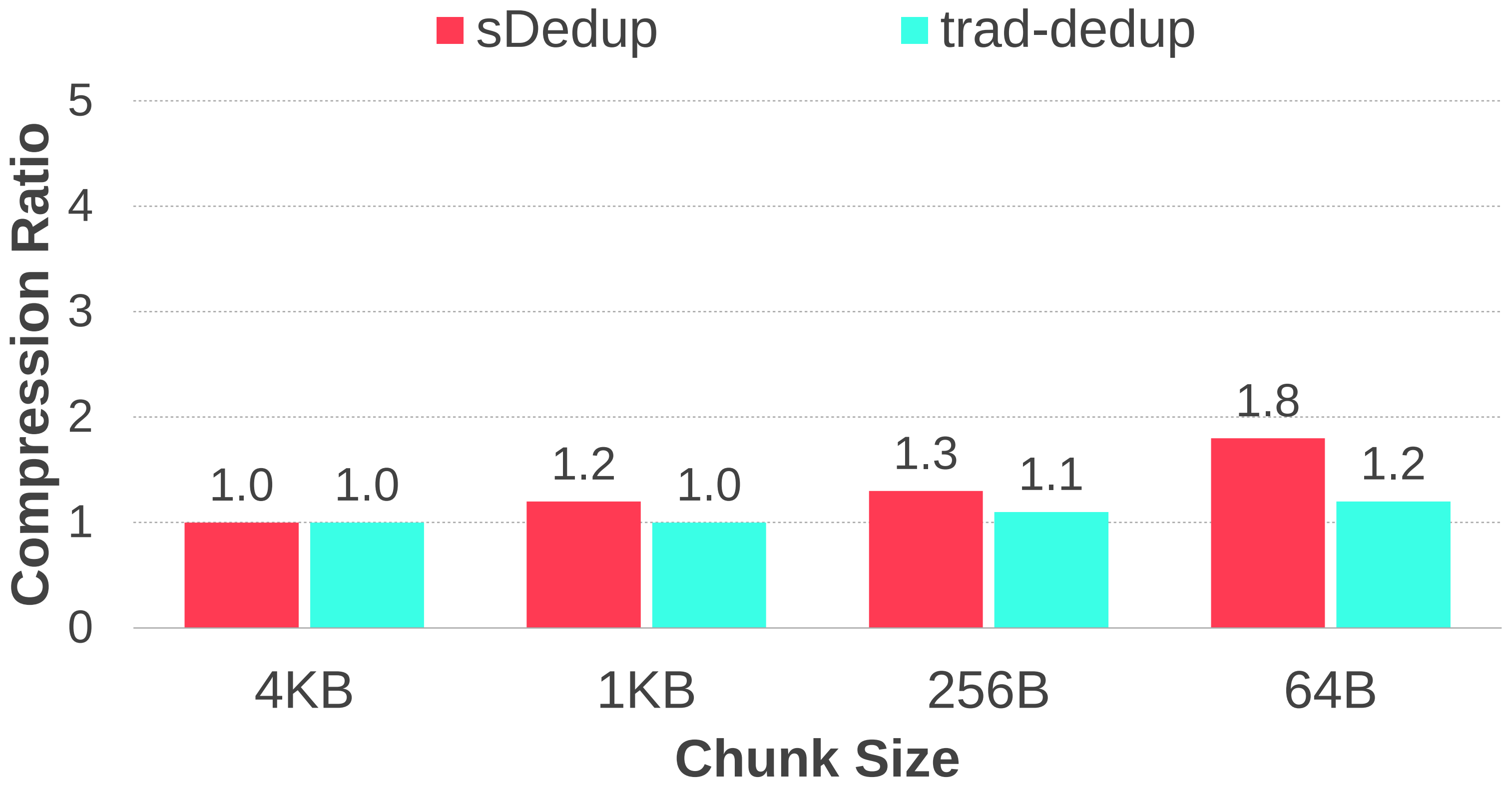
20GB sampled Wikipedia dataset

Memory: Wikipedia



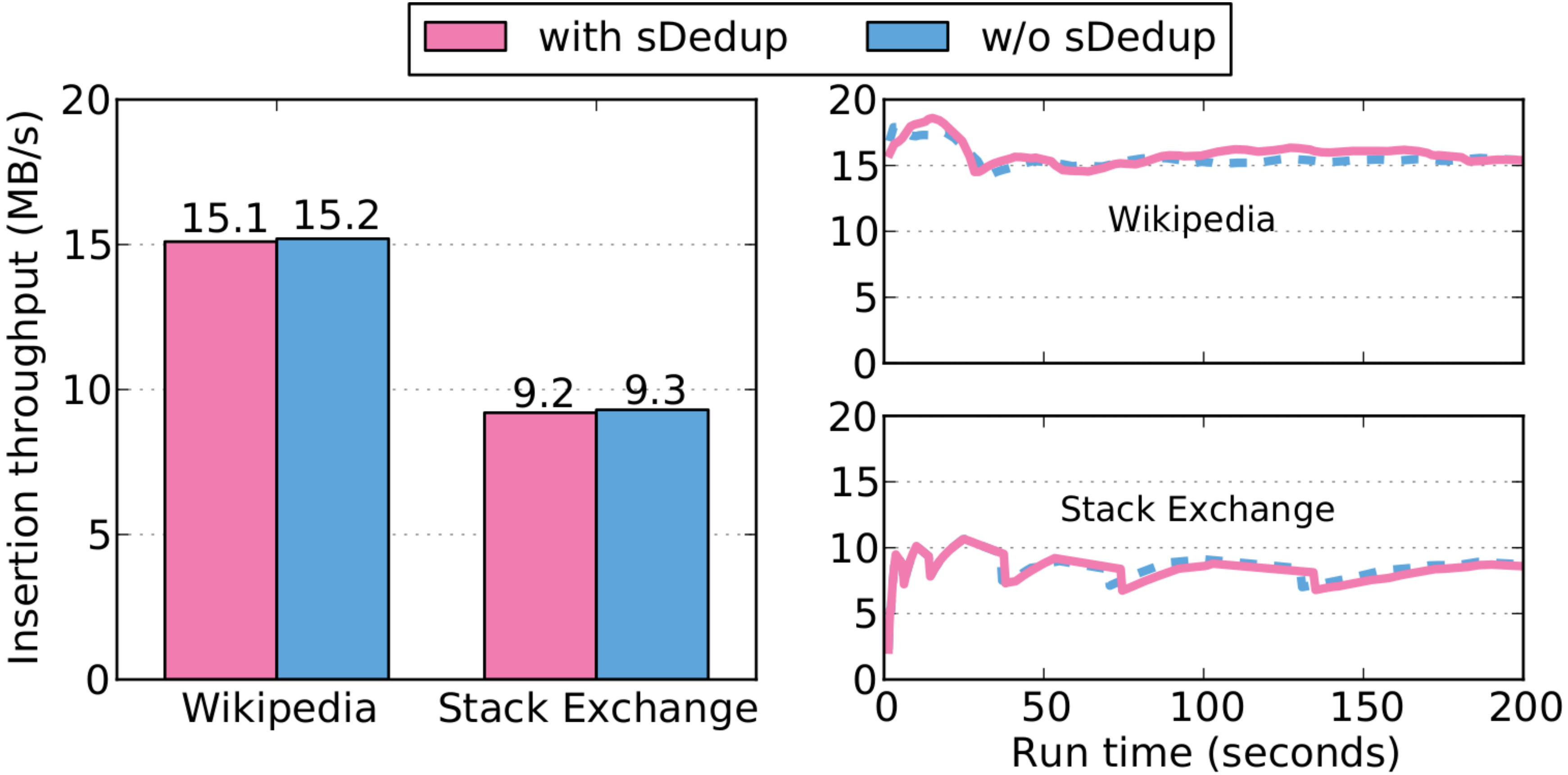
20GB sampled Wikipedia dataset

Compression: StackExchange

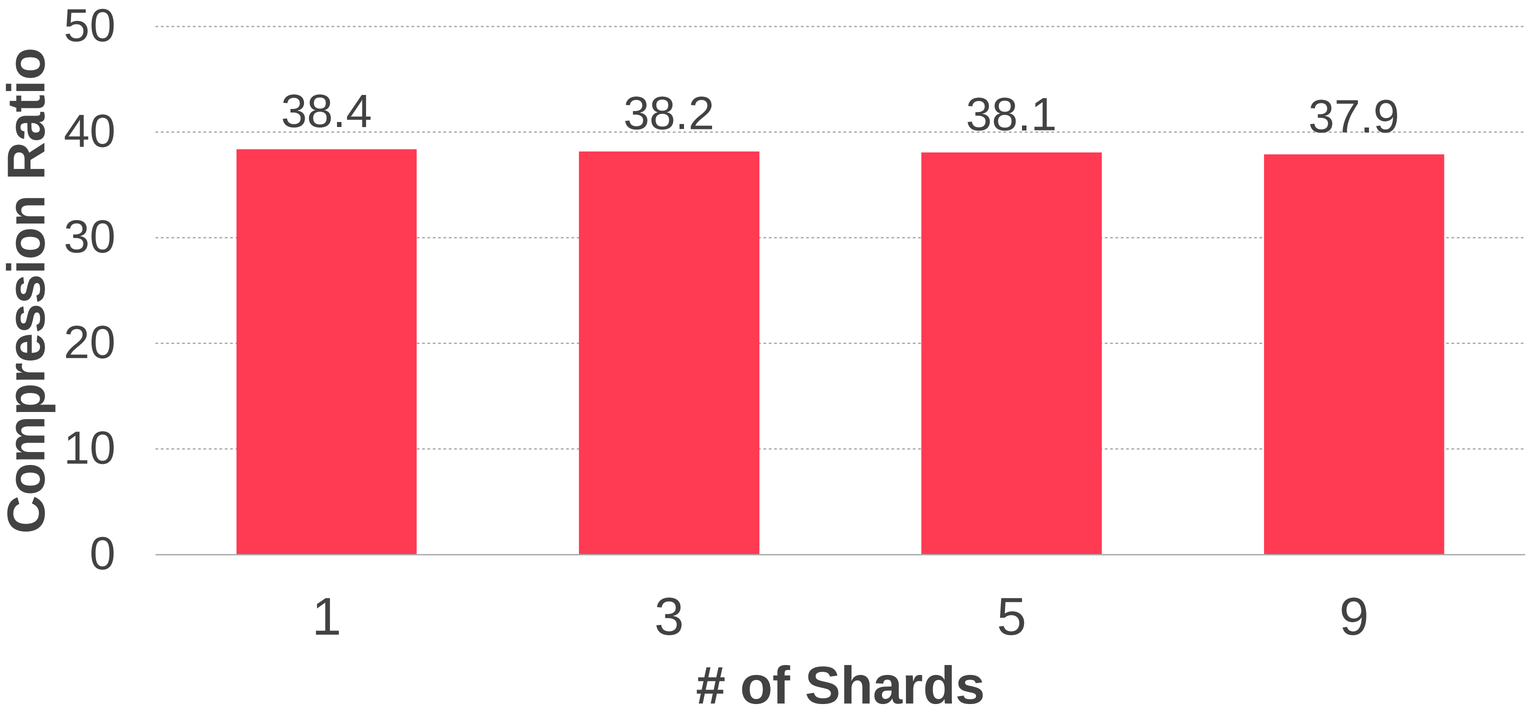


10GB sampled StackExchange dataset

Throughput Overhead

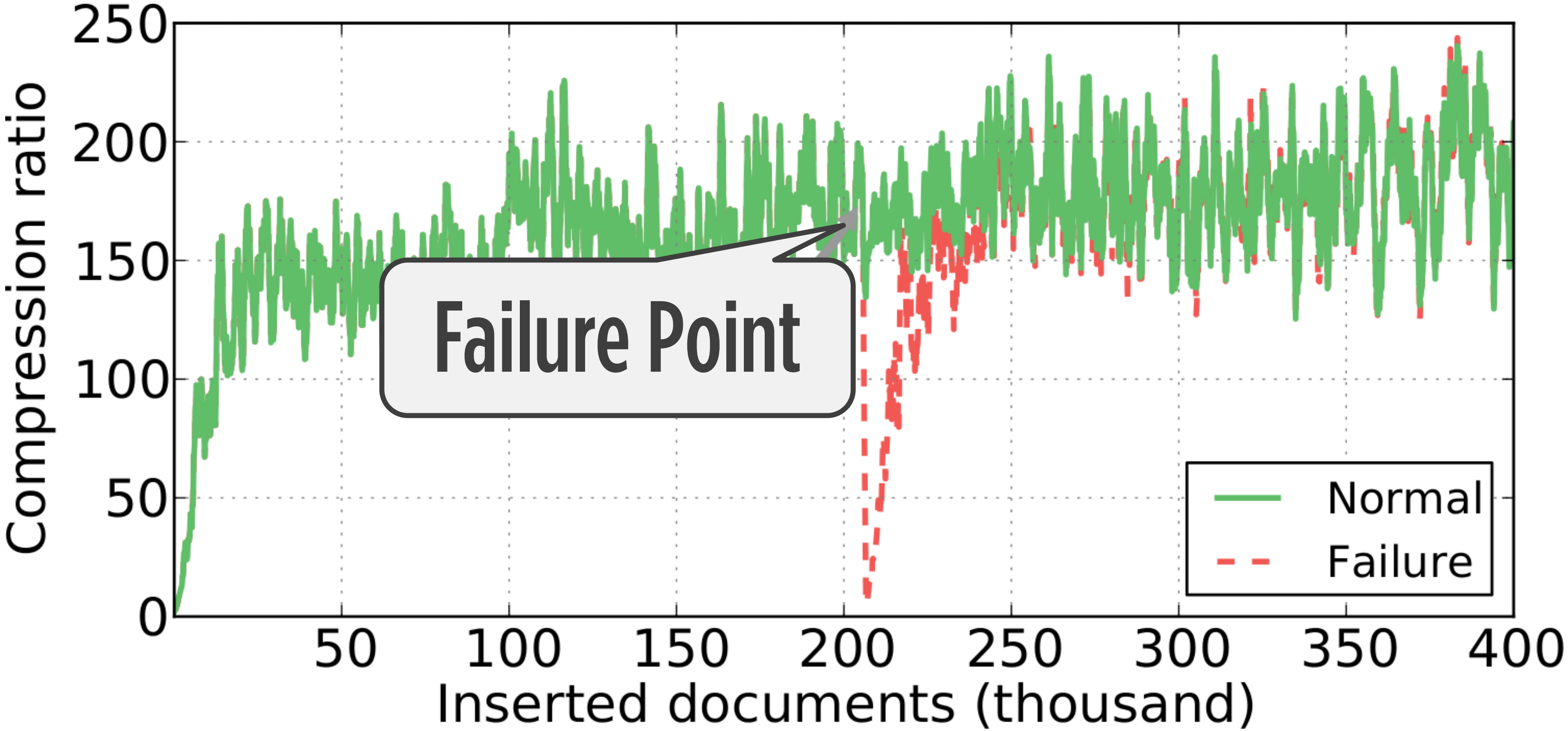


Dedup + Sharding



20GB sampled Wikipedia dataset

Failure Recovery



20GB sampled Wikipedia dataset.

Conclusion

- Similarity-based deduplication for replicated document databases.
- **sDedup** for MongoDB (v2.7)
  - *Much greater data reduction than traditional dedup*
  - *Up to 38x compression ratio for Wikipedia*
  - *Resource-efficient design for inline deduplication with negligible performance overhead*

What's Next?

- Port code to MongoDB v3.1
- Integrating **sDedup** into WiredTiger storage manager.
- Need to test with more workloads.

- Try not to get anyone pregnant.

WiredTiger vs. sDedup

	Compression Ratio
Snappy	1.6x
zLib	3.0x
sDedup (no compress)	38.4x
sDedup + Snappy	60.8x
sDedup + zLib	114.5x

20GB sampled Wikipedia dataset.

END

@andy_pavlo