BlowFish: Dynamic Storage-Performance Tradeoff in Data Stores

Anurag Khandelwal, Rachit Agarwal, Ion Stoica
High-Throughput Data Stores
High-Throughput Data Stores

Key-Value Stores

BigTable

amazon DynamoDB

riakKV

redis

MEMCACHED
High-Throughput Data Stores

Key-Value Stores

- BigTable
- Amazon DynamoDB
- riakKV
- redis
- Memcached

NoSQL Stores

- elastic
- HyperDex
- Cassandra
- Hypertable
- MongoDB
High-Throughput Data Stores

Key-Value Stores
- BigTable
- Amazon DynamoDB
- riakKV
- Redis
- Memcached

NoSQL Stores
- elastic
- HyperDex
- Cassandra
- Hypertable
- MongoDB

High Throughput: Queries/Second
Existing Data Stores
Existing Data Stores
Existing Data Stores

- Storage
  - Uncompressed
- Throughput
  - More cache

High Throughput
Existing Data Stores
Existing Data Stores

![Diagram showing existing data stores with uncompressed and compressed options, indicating improvements in throughput and storage.

Uncompressed: 10x increase in throughput vs. compressed.
Compressed: 10x increase in storage vs. uncompressed.]
Existing Data Stores

![Graph showing existing data stores with points labeled 'Compressed' and 'Uncompressed', and a banner stating 'Unachievable points'.](image-url)
Existing Data Stores

Switching between the two incurs high latency & CPU

Unachievable points
Leads to degraded performance when underlying workload or infrastructure changes.

Existing Data Stores

Switching between the two incurs high latency & CPU.

Unachievable points
A Motivating Example
A Motivating Example

Load across items heavily skewed
A Motivating Example

Load across items **heavily skewed**
A Motivating Example

Load across items **heavily skewed**
A Motivating Example

Load across items heavily skewed

Ideal

Load across items heavily skewed

Load

Compressed

Uncompressed

Ideal

Object

Not Cached
A Motivating Example

Load across items **heavily skewed**
A Motivating Example

Load across items heavily skewed

Ideal

Compressed + Uncompressed

Load across items heavily skewed

Object

Load

UNCOMPRESSED

COMPRESSED

NOT CACHED

Load

UNCOMPRESSED

COMPRESSED
A Motivating Example

Load across items heavily skewed

![Diagram showing load across items]
A Motivating Example

Load across items **heavily skewed**

**Ideal**

**Compressed + Uncompressed**

**Selective Replication:** 
#Replicas $\alpha$ Load

![Diagram showing load across items]

- Not Cached
- Wasted Cache!
A Motivating Example

Load across items heavily skewed

Ideal

Compressed + Uncompressed

Selective Replication: #Replicas $\alpha$ Load
A Motivating Example

Load across items heavily skewed

Ideal

Compressed + Uncompressed

Selective Replication: 
#Replicas \( \alpha \) Load

![Graph showing load distribution and caching efficiency](image-url)
A Motivating Example

Load across items heavily skewed

Ideal

Compressed + Uncompressed

Selective Replication: #Replicas $\alpha$ Load

Load changes over time
A Motivating Example

Load across items heavily skewed

Ideal

Compressed + Uncompressed

Selective Replication: #Replicas \( \alpha \) Load

Load changes over time → Degraded performance
BlowFish

Throughput vs. Storage

- Compressed
- Uncompressed

Diagram shows the relationship between throughput and storage, highlighting the difference between compressed and uncompressed data points.
BlowFish

Smooth Tradeoff Curve

Throughput

Storage

BlowFish

Compressed

Uncompressed
BlowFish

Smooth Tradeoff Curve

Dynamic Navigation
BlowFish

Smooth Tradeoff Curve

Dynamic Navigation

Applications in Several Classical Systems Problems

Throughput vs. Storage Diagram:
- BlowFish
- Compressed
- Uncompressed

Graph showing the tradeoff between throughput and storage for BlowFish applications.
Storage-Performance Tradeoff
Background
Background

Builds on Succinct [NSDI’15]
Background

Builds on **Succinct [NSDI’15]**

Succinct stores:
Background

Builds on **Succinct [NSDI’15]**

Succinct stores:
Background

Builds on **Succinct** [NSDI’15]

Succinct stores:

- **Sampled** Array
Background

Builds on Succinct [NSDI’15]

Succinct stores:

SAMPLED ARRAY

SAMPLED VALUES

UNSAMPLED VALUES
Background

Builds on **Succinct [NSDI’15]**

Succinct stores:
Background

Builds on **Succinct** [NSDI’15]

Succinct stores:

- **Sampled Array**
- **Auxiliary Arrays**
Background

Builds on **Succinct** [NSDI’15]

Succinct stores:

- **SAMPLED ARRAY**
- **AUXILIARY ARRAYS**
  - Small
Background

Builds on **Succinct [NSDI’15]**

Succinct stores:

- **Sampled Array**
- **Auxiliary Arrays**
  - Small
  - Compute *unsampled values on the fly*
Background

Builds on Succinct [NSDI’15]

Succinct stores:

- Sampled Array
- Auxiliary Arrays

Sampling Rate proxy for Storage & Performance

- Small
- Compute unsampled values on the fly
Background

Builds on **Succinct** [NSDI’15]

Succinct stores:

<table>
<thead>
<tr>
<th>Sampling Rate (α)</th>
</tr>
</thead>
</table>

**Sampled Array**

- **Sampling Rate proxy for Storage & Performance**
  - Storage ≈ OriginalSize/α
  - Latency ≈ α

- Auxiliary Arrays
  - Small
  - Compute unsampled values on the fly
Layered Sampled Array

Inspired by multi-layered video encoding techniques

<table>
<thead>
<tr>
<th>Original Sampled Array</th>
<th>Rate = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 15 3 0 12 8 14 5</td>
<td></td>
</tr>
</tbody>
</table>
Layered Sampled Array

Inspired by multi-layered video encoding techniques

Original Sampled Array

Rate = 2
Layered Sampled Array

Inspired by multi-layered video encoding techniques

Original Sampled Array

<table>
<thead>
<tr>
<th>Original Sampled Array</th>
<th>Rate = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>15</td>
</tr>
</tbody>
</table>

Rate = 8

<table>
<thead>
<tr>
<th>Rate = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
Layered Sampled Array

Inspired by multi-layered video encoding techniques

Original Sampled Array

Rate = 2

Rate = 8

Rate = 4
Layered Sampled Array

Inspired by multi-layered video encoding techniques

<table>
<thead>
<tr>
<th>ORIGINAL SAMPLED ARRAY</th>
<th>RATE = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 15 3 0 12 8 14 5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATE = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATE = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATE = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATE = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATE = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATE = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
Layered Sampled Array

Inspired by **multi-layered video encoding** techniques

Different combination of layers
Layered Sampled Array

Inspired by *multi-layered video encoding* techniques

<table>
<thead>
<tr>
<th>ORIGINAL SAMPLED ARRAY</th>
<th>RATE = 8</th>
<th>RATE = 4</th>
<th>RATE = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>15</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Different combination of layers → Different points on tradeoff curve
Technical Details
Technical Details
Technical Details
Technical Details

‣ How should partitions share cache on a server?
Technical Details

- How should partitions share cache on a server?
- How should partitions share cache across servers?
Technical Details

- How should partitions share cache on a server?
- How should partitions share cache across servers?
- How should requests be scheduled across replicas?
Technical Details

- How should partitions share cache on a server?
- How should partitions share cache across servers?
- How should requests be scheduled across replicas?

Unified Solution: Back-pressure style scheduling
Technical Details

‣ How should partitions share cache on a server?
‣ How should partitions share cache across servers?
‣ How should requests be scheduled across replicas?

Unified Solution: Back-pressure style scheduling

Cache proportional to load,
Technical Details

- How should partitions share cache on a server?
- How should partitions share cache across servers?
- How should requests be scheduled across replicas?

Unified Solution: Back-pressure style scheduling

Cache proportional to load, without explicit coordination
Dynamic Navigation of tradeoff curve
Layer **Additions & Deletions**
<table>
<thead>
<tr>
<th>RATE = 8</th>
<th>9</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>RATE = 4</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>RATE = 2</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>
Layer Additions & Deletions

Layer Deletions: simple
Layer Additions & Deletions

RATE = 8
It appears there is a blue square in the middle with the number 9.

RATE = 4
There is a green square with the number 3.

RATE = 2
The red squares indicate deletions, but no exact number is visible in the provided image.

Layer Addition:
Layer Additions & Deletions

<table>
<thead>
<tr>
<th>RATE</th>
<th>Layer Addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>[9] 12</td>
</tr>
<tr>
<td>4</td>
<td>[3] 14</td>
</tr>
<tr>
<td>2</td>
<td>[Red] 14 [Red]</td>
</tr>
</tbody>
</table>

Layer Addition:

Unsampled values already computed during query execution
## Layer Additions & Deletions

<table>
<thead>
<tr>
<th>Rate</th>
<th>Layer Additions</th>
<th>Layer Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9, 12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3, 14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15, 8</td>
<td>15, 8</td>
</tr>
</tbody>
</table>

### Layer Addition:

Unsampled values already computed **during query execution**

Layers in LSA populated **opportunistically!!**
Assumptions & Limitations
Assumptions & Limitations

- Functionality close to state-of-the-art NoSQL stores
Assumptions & Limitations

- Functionality close to state-of-the-art NoSQL stores

  - get()
  - put()
  - delete()
  - search()
  - regex()
Assumptions & Limitations

- Functionality close to state-of-the-art NoSQL stores
  - get()
  - put()
  - delete()
  - search()
  - regex()

- Queries do not touch all servers
Assumptions & Limitations

- Functionality close to **state-of-the-art NoSQL stores**
  - get()
  - put()
  - delete()
  - search()
  - regex()

- Queries do not **touch all servers**

  Many systems employ sharding schemes to avoid touching all servers, e.g., [Schism, VLDB’10]
Assumptions & Limitations

- Functionality close to state-of-the-art NoSQL stores
  - get()
  - put()
  - delete()
  - search()
  - regex()

- Queries do not touch all servers
  Many systems employ sharding schemes to avoid touching all servers, e.g., [Schism, VLDB'10]

- System is not network-bottlenecked
Assumptions & Limitations

- Functionality close to state-of-the-art NoSQL stores
  - get()
  - put()
  - delete()
  - search()
  - regex()

- Queries do not touch all servers
  Many systems employ sharding schemes to avoid touching all servers, e.g., [Schism, VLDB’10]

- System is not network-bottlenecked
  [MICA, NSDI’14] → True for most data stores today
Applications
Applications

Look at classical systems problems through a new “lens”
Spatial Skew
Spatial Skew

Load distribution across partitions is heavily skewed.
Spatial Skew

Load distribution across partitions is heavily skewed

Selective Replication

#Replicas $\alpha$ Load
Spatial Skew

Load distribution across partitions is heavily skewed

Selective Replication

#Replicas $\alpha$ Load

BlowFish

Fractionally change storage just enough to meet load
Spatial Skew

Load distribution across partitions is heavily skewed

Selective Replication

#Replicas $\alpha$ Load

BlowFish

Fractionally change storage just enough to meet load

1.5x higher throughput than Selective Replication,
Spatial Skew

Load distribution across partitions is **heavily skewed**

**Selective Replication**

\[ \#\text{Replicas} \propto \text{Load} \]

**BlowFish**

Fractionally change storage just enough to meet load

1.5x higher throughput than Selective Replication, within 10% of optimal
Changes in Spatial Skew
Changes in Spatial Skew

Study on Facebook Warehouse Cluster

[HotStorage’13]
Changes in Spatial Skew

Study on Facebook Warehouse Cluster [HotStorage’13]  Transient failures → 90% of failures
Changes in Spatial Skew

Study on Facebook Warehouse Cluster [HotStorage’13]

Transient failures $\rightarrow$ 90% of failures
Replica creation delayed by 15 mins
Changes in Spatial Skew

Study on Facebook Warehouse Cluster [HotStorage’13]

Transient failures → 90% of failures
Replica creation delayed by 15 mins

Leads to variation in load over time
Changes in Spatial Skew

Study on Facebook Warehouse Cluster [HotStorage’13]

Transient failures $\rightarrow$ 90% of failures
Replica creation delayed by 15 mins

Leads to variation in load over time
Changes in Spatial Skew

Study on Facebook Warehouse Cluster [HotStorage’13]

Transient failures $\rightarrow$ 90% of failures
Replica creation delayed by 15 mins

Leads to variation in load over time

DATA PARTITIONS

REQUEST QUEUES
Changes in Spatial Skew

Study on Facebook Warehouse Cluster [HotStorage'13]

Transient failures → 90% of failures
Replica creation delayed by 15 mins

Leads to variation in load over time
Changes in Spatial Skew

Replica#1
Replica#2
Replica#3
Changes in Spatial Skew

- Replica #1
- Replica #2
- Replica #3
Changes in Spatial Skew

Graph showing changes in spatial skew over time for Replica #1, #2, and #3.
Changes in Spatial Skew
Changes in Spatial Skew
Changes in Spatial Skew

![Graph showing changes in spatial skew](image)

- **Operations / second**: X-axis [0, 3000] with intervals [0, 600], [600, 1200], [1200, 1800], [1800, 2400], [2400, 3000]. Y-axis [0, 3000] with intervals [0, 600], [600, 1200], [1200, 1800], [1800, 2400], [2400, 3000].
- **Time (mins)**: [0, 120] with intervals [0, 30], [30, 60], [60, 90], [90, 120].

- **Load** and **Throughput** are represented by separate line graphs.

- **Request Queue Size**: X-axis [0, 50K] with intervals [0, 10K], [10K, 20K], [20K, 30K], [30K, 40K], [40K, 50K].
- **Time (mins)**: [0, 120] with intervals [0, 30], [30, 60], [60, 90], [90, 120].

- **Replica#1**, **Replica#2**, **Replica#3** are color-coded with bars indicating load distribution.
Changes in Spatial Skew
Changes in Spatial Skew

Load Throughput

Operations / second

Request Queue Size

Replica #1

Replica #2

Replica #3

TIME (mins)

TIME (mins)
Changes in Spatial Skew
Changes in Spatial Skew

![Graph showing changes in operations per second over time for Replica #1, #2, and #3.](image)

- **Load** (red line)
- **Throughput** (green line)

**Request Queue Size**

- 0K
- 10K
- 20K
- 30K
- 40K
- 50K

**Time (mins)**

- 0
- 30
- 60
- 90
- 120
Changes in Spatial Skew

[Graph showing changes in load and throughput over time]

[Legend showing replica statuses]
Changes in Spatial Skew

Adapts to 3x higher load in < 5 mins
Summary

![Graph showing the relationship between storage and throughput. The graph illustrates an upward curve, indicating that as storage increases, throughput also increases.](image-url)
Summary

Smooth Tradeoff Curve

Throughput

Storage
Summary

![Smooth Tradeoff Curve

Dynamic Navigation](image.png)
Summary

Smooth Tradeoff Curve

Dynamic Navigation

Applications in Several Classical Systems Problems
Summary

Thank You! Questions?

- Smooth Tradeoff Curve
- Dynamic Navigation
- Applications in Several Classical Systems Problems
Backup Slides