Revisiting Concurrency in High-Performance NoSQL Databases

Yuvraj Patel¹, Mohit Verma²,
Andrea C. Arpaci-Dusseau¹, Remzi H. Arpaci-Dusseau¹
University of Wisconsin-Madison¹, NVIDIA²
## Technological changes

### Faster storage devices

<table>
<thead>
<tr>
<th></th>
<th>DRAM</th>
<th>NVM</th>
<th>SSD</th>
<th>HDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Latency</td>
<td>100-200 ns</td>
<td>500ns-2us</td>
<td>20-50 us</td>
<td>5-10ms</td>
</tr>
<tr>
<td>Write Bandwidth</td>
<td>20-50 GB/sec</td>
<td>5-10 GB/s per die</td>
<td>200-500 MB/s</td>
<td>50-100 MB/sec</td>
</tr>
</tbody>
</table>

### More processing power available per node

<table>
<thead>
<tr>
<th>Year</th>
<th>2006</th>
<th>2010</th>
<th>2014</th>
<th>2019</th>
</tr>
</thead>
<tbody>
<tr>
<td># of cores</td>
<td>2</td>
<td>8</td>
<td>32-64</td>
<td>128</td>
</tr>
</tbody>
</table>
Concurrency control

• Act of coordinating concurrent accesses

Key to scaling
• Decades worth of effort
  • Optimistic concurrency control
  • Adaptive concurrency control
  • Many other ways.....
Key Question

Concurrency control designed based on older technology
  • Very few cores
  • Hard disk

Is concurrency control a bottleneck?
  • Can current techniques work well with newer technological changes?
  • Is it time to revisit concurrency control?
Answer – Quantitative study

We study performance scaling of 5 popular systems
  • MongoDB, Cassandra, CouchDB, ArangoDB, Oracle NoSQL DB
Measure actual and zero-contention concurrency control (ZCCC) scaling performance on single node

Observation
  • No system scales well
  • Presence of faster storage device does not help
  • Difference between ZCCC and Actual is more than 3X
Answer – Qualitative study

Qualitative study of techniques used by these systems

- Categorize to understand weaknesses
  - Thread Architecture, Batching, Granularity, Partitioning, Scheduling, Low-level efficiency

Observations

- Batching, Thread Architecture – Used by all
- Partitioning, Scheduling not used efficiently for scaling
- Scope for improvement to optimize common cases
Our Solution

We present Xyza – an extension of MongoDB
  • Concentrate on Partitioning, Scheduling & Low-level efficiency
  • Per client and key-space partitioning
  • Two novel scheduling techniques
    • Contention-aware scheduling & Semantic-aware scheduling
    • Optimize common cases using atomics

Performance
  • 2X to 3X faster than MongoDB
  • Xyza single instance performance 0.8X-0.9X that of ZCCC performance
Outline

• Overview
• Concurrency Analysis
• Xyza Design
• Evaluation
• Conclusion
Zero-contention concurrency control (ZCCC)

Performance increases linearly as the workload increases.
Saturates once the cpu/disk/network/memory exhausts.

Throughput (ops/sec) vs Workload Capacity graph

Saturation point
Concurrent application

Today’s concurrent application architecture

- Clients connect to server
- Application – multithreaded
- Multi/Many processing cores
- One or more disks
- Memory
Identify zero-contention concurrency control

For ZCCC,
- Hypothesis - Less contention of resources by perfectly partitioning

How to reduce contention of resources?
- Instantiate multiple instances of the same application
- CPU/disk/memory/network still stressed
Actual vs ZCCC

Why writes?
- Modify global structures frequently
- Hard to scale

Why weakest consistency option?
- Designed to achieve high performance

Workload
- 50M key-value pair inserts, value - 100 bytes

Setup
- 2 socket 8-core hyperthreaded, 1 480 GB SSD, 128 GB RAM, 10Gbps N/w
Results - MongoDB

Throughput (Kops/sec) vs. No. of Clients

- Disk b/w ranges 26-350 MB/s
- CPU utilization ranges 6-90%
- Near zero contention helps in scaling performance

ZCCC
Results - MongoDB

Throughput (Kops/sec)

- **Single Instance**
  - Disk b/w ranges 26-350 MB/s
  - CPU utilization ranges 6-90%

- **ZCCC**
  - CPU utilization increases even though throughput is saturated
  - Near zero contention helps in scaling performance
  - Disk b/w ranges 25-130 MB/s
  - CPU utilization ranges 6-85%
Results – Cassandra

Throughput (Kops/sec) vs No. of Clients for Single Instance and ZCCC (2 instances).
Results – Oracle NoSQL DB

Throughput (Kops/sec)

No. of Clients

- Single Instance
- ZCCC (10 instances)
Results – CouchDB

![Graph showing throughput vs. number of clients for Single Instance and ZCCC.]

- **Throughput (Kops/sec)**
- **No. of Clients**
- **Single Instance**
- **ZCCC**
Results – ArangoDB

Throughput (Kops/sec) vs No. of Clients

- **Single Instance**
- **ZCCC**
Categorization

Study 5 systems qualitatively

- Analyze code, design and architecture documents
- Categorize the techniques - Understand the weaknesses

6 categories

- Thread architecture, Batching, Granularity, Partitioning, Scheduling, Low-level efficiency (Read the paper)

Observation

- Partitioning and Scheduling not used efficiently
- Scope to improve common cases performance
Analysis - Summary

Quantitative study shows
• Resource competition kills actual performance
• ZCCC performance scales as hypothesized
• Storage and Network not a bottleneck

Qualitative study shows
• Partitioning & Scheduling not used efficiently
• There is a need to optimize for common cases

Systems designed for slow-storage media than faster devices
Outline

- Overview
- Concurrency Analysis
- Xyza Design
- Evaluation
- Conclusion
Xyza Design

Xyza – extension of MongoDB

Why MongoDB
  • Certain design ideas useful for Xyza
  • Scope for improvements
  • Popular and heavily used

Xyza design concentrated on partitioning, scheduling and low-level efficiency
MongoDB architecture

MongoDB’s concurrency architecture diagram

Clients → Per Client thread

CPU 1 → Global Shared Data

V (Vector) → J (Journal)

Key-Space (per collection) → Cache → Disk

Checkpointing, Logging thread

7/10/2018
MongoDB architecture problems

MongoDB’s concurrency architecture diagram

- Clients
- Per Client thread
- CPU 1
- CPU 2
- Global Shared Data
  - V: Vector
  - J: Journal
- Key-Space (per collection)
- Cache
- Disk
- Checkpointing, Logging thread
Xyza - Partitioning

- Take advantage of per client thread

Xyza's concurrency architecture diagram

- Per Client thread
- Exclusive access
- Per client data:
  - V1
  - V2
  - J1
  - J2

- Key-Space
- Cache

- Vector
- Journal

- B-tree (per collection)

- Checkpointing, Logging thread
- Disk

Clients

CPU 1

CPU 2

29/10/2018
Xyza - Partitioning

- Take advantage of per client thread

- Key-space partitioning facilitates execution of non-overlapping operations

Xyza’s concurrency architecture diagram:

- Per Client thread
- CPU 1
- CPU 2

Per client data:

- V1
- V2

- J1
- J2

- K1
- K2

- Cache

- Vector
- Journal

- B-tree (per collection)

- Disk

Checkpointing, Logging thread
Xyza - Partitioning

- Take advantage of per client thread
- Key-space partitioning facilitates execution of non-overlapping operations

Xyza’s concurrency architecture diagram

Per Client thread

- Exclusive access
- Thread Pinning

Per client data

- Vector
- Journal

B-tree (per collection)

Cache

Checkpointing, Logging thread

Clients

CPU 1

CPU 2
Xyza – Contention-aware scheduling

Xyza’s concurrency architecture diagram

- Locks/partition as resource
- Exclusive access per key-space partition
- Execute operations that will not contend for same resources
- Overlapping operations wait

Pinned Per Client thread
CPU 1

Partitioned Data
V1 | V2
J1 | J2

Vector Journal

B-tree (per collection)
KS1 | KS2

Cache

Checkpointing, Logging thread

Disk

Scheduling - form of primitive synchronization
Xyza scheduling example – No conflict

Different partition access

Clients

Pinned Per Client thread
CPU 1

V1         V2
J1          J2
KS1      KS2
Cache

CPU 2

Partitioned Data

Vector

Journal

B-tree (per collection)

Disk

Checkpointing, Logging thread

Key:1 Value:1

Different partition access
Xyza scheduling example – No conflict
Xyza scheduling example - Conflict

Key: 1
Value: 1

Same partition access

Clients

Pinned Per Client thread
CPU 1

CPU 2

Partitioned Data

Vector

Journal

B-tree (per collection)

Cache

Disk

Checkpointing, Logging thread

Partitioned Data

V1 V2

J1 J2

KS1 KS2

Scheduling

Scheduling

Scheduling

Scheduling

Scheduling

7/10/2018

31
Xyza scheduling example - Conflict

Clients

Key:1
Value:1

Key:2
Value:2

Second operation waits for the first one to complete

Pinned Per Client thread
CPU 1

CPU 2

Same partition access

Partitioned Data

Vector

Journal

Checkpointing, Logging thread

B-tree (per collection)

Disk

Cache

Key:1
Value:1

Key:2
Value:2

Second operation waits for the first one to complete.
Xyza - Semantic-aware scheduling

What can be done when the operation waits for its turn?

Semantic-aware scheduling is the answer (Read the paper)
Xyza – Low-level efficiency

MongoDB
- Use atomics internally
- Lock manager for multiple-granularity locking

Xyza
- Lock manager replaced
  - Simple wait-signal mechanism
- Optimize common case
  - Mutex locks heavy compared to atomics
  - Bypass mutex - Instead use atomics
Outline

• Overview
• Concurrency Analysis
• Xyza Design
• Evaluation
• Conclusion
Evaluation

Throughput (Kops/sec) vs. No. of Clients

- Single Instance
- ZCCC MongoDB
- Xyza

Partitioning and efficient scheduling helps reduce contention

2X faster

*Same workload & setup as earlier

Disk b/w ranges 25-280 MB/s
CPU utilization ranges 6-78%
Evaluation – contention reduction

Throughput (Kops/sec) vs. No. of Clients

- ZCCC MongoDB

7/10/2018 37
Evaluation – contention reduction

![Throughput vs. No. of Clients graph]

- ZCCC MongoDB
- Single Instance MongoDB
Evaluation – contention reduction

Throughput (Kops/sec) vs No. of Clients

- Single Instance Mongo_PCDB
- ZCCC MongoDB
- Single Instance MongoDB

Lock manager stressed even when contention reduced

*PCDB – Per Client Database
Evaluation – contention reduction

Throughput (Kops/sec) vs. No. of Clients for different databases:
- Single Instance Mongo_PCDB
- ZCCC MongoDB
- Xyza_PCDB

Key observations:
- Less resource contention – key to scaling
- Optimizing common case helps scaling
- 0.8-0.9X of ZCCC performance

*PCDB – Per Client Database
Outline

• Overview
• Concurrency Analysis
• Xyza Design
• Evaluation
• Conclusion
Conclusion

Today’s NoSQL DB’s do not scale well despite the presence of faster storage medium and high processing capacity.

We present Xyza that scales well and comes close to ZCCC performance.

Time to revisit concurrency and consider aggressive concurrency mechanisms.

The proposed techniques can be applied in many other systems.

....WXYZAB....
Thank you