CloudJump: Optimizing Cloud Database For Cloud Storage

Zongzhi Chen, Xinjun Yang, Feifei Li, Xuntao Cheng, Qingda Hu, Zheyu Miao, Rongbiao Xie, Xiaofei Wu, Kang Wang, Zhao Song, Haiqing Sun, Zechao Zhuang, Yuming Yang, Jie Xu, Liang Yin, Wenchao Zhou, Sheng Wang
1 Background and Motivation
2 Design Considerations
3 Case Study: PolarDB
4 Case Study: RocksDB
Background and Motivation

**Cloud-native database**
- Massive amounts of data
- Elasticity
- High availability and durability
- High performance
- Serverless, pay-as-you-go

**Cloud-storage**
- Large storage capability
- Data persistence
- High availability
- High aggregated I/O bandwidth
- On-demand pricing
- Reduce maintenance costs

Target: Can we build a “more” cloud-native database through migrating an on-premise database kernel onto the cloud using a cloud storage?
Experience from our online service

- Slow SQL with cloud storage
- Low bandwidth utilization
- Bad log performance when flushing dirty pages

Micro-benchmark

- High I/O latency and bandwidth

These hinder cloud storage from becoming an performance-satisfied service for cloud-native databases
Background and Motivation

- Architecture differences in on-premise and on-cloud-storage database

Challenges:
- Local accesses v.s. remote accesses
- Local bandwidth v.s. aggregated bandwidth
- Consistency among multiple database nodes
- I/O isolation
- Big table further worsen the performance
## Background and Motivation

### Challenges

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Design knobs</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote accesses</td>
<td>WAL</td>
<td>Slow serial logging</td>
</tr>
<tr>
<td></td>
<td>Log replay</td>
<td>Applying logs to multiple pages</td>
</tr>
<tr>
<td></td>
<td>Data read</td>
<td>Loading dependent remote pages</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Synchronization</td>
<td>Blocking updates while writing pages</td>
</tr>
<tr>
<td>Aggregated bandwidth</td>
<td>Data write</td>
<td>Low bandwidth (accessing a small single page)</td>
</tr>
<tr>
<td></td>
<td>Data read</td>
<td></td>
</tr>
<tr>
<td>Consistency among nodes</td>
<td>Page cache</td>
<td>With cache: high consistency overhead; Without cache: amplified I/O with no buffers</td>
</tr>
<tr>
<td>I/O isolation</td>
<td>I/O scheduling</td>
<td>Concurrent and extensive log and data I/Os cause unpredictable performance</td>
</tr>
</tbody>
</table>

### Problems

<table>
<thead>
<tr>
<th>Update-in-place</th>
<th>Append-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote accesses</td>
<td>WAL</td>
</tr>
<tr>
<td></td>
<td>Log replay</td>
</tr>
<tr>
<td></td>
<td>Data read</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Synchronization</td>
</tr>
<tr>
<td>Aggregated bandwidth</td>
<td>Data write</td>
</tr>
<tr>
<td></td>
<td>Data read</td>
</tr>
<tr>
<td>Consistency among nodes</td>
<td>Page cache</td>
</tr>
<tr>
<td>I/O isolation</td>
<td>I/O scheduling</td>
</tr>
</tbody>
</table>
1. Background and Motivation
2. Design Considerations
3. Case Study: PolarDB
4. Case Study: RocksDB
Design consideration: optimize on cloud storage

- **Thread-level Parallelism**
  
  *eg.* Adopt multiple logging and data I/O threads, use asynchronous I/O models to fully scatter data across multiple storage nodes

- **Task-level Parallelism**
  
  *eg.* Partitioned log on page-space and written in parallel to multiple tasks. Concurrent Recovery based on partition.

- **Reduce remote read and Prefetching**
  
  *eg.* Prefetching potentially achieves larger performance gains on the cloud storage compared with those on local SSDs,
Seven design consideration: optimize on cloud storage

- **Fine-grained Locking and Lock-free Data Structures**
  *eg.* To minimize the chances of contention during prolonged I/O.

- **Scattering among Distributed Nodes**
  *eg.* Distribute large log I/Os to different storage nodes to make full use of the aggregated bandwidth.

- **Bypassing Caches**
  *eg.* Avoid the coherence issue and optimize I/O formats on database layer.

- **Scheduling Prioritized I/O Tasks**
  *eg.* Marking and scheduling priorities for different I/Os on database layer.
1. Background and Motivation
2. Design Considerations
3. Case Study: PolarDB
4. Case Study: RocksDB
Case Study: PolarDB

- B-tree based storage engine
- Multiple computation nodes
Case Study: PolarDB

Scattered & Partitioned Global Log

- High WAL I/O Latency
- Sequential WAL I/O
- Low bandwidth utilization

- Log buffer partition, Parallelized writing
- Asynchronous multi-task threads, high bandwidth utilization
- Scattered I/O with high distributed writing performance
Case Study: PolarDB

Scattered & Partitioned Global Log

Log Buffer 1
Log Buffer 2
Log Buffer n

LPLSNs  LBLSNs  BRLs

Log Buffer - Partition 1

A large log I/O request

Slicing

4KB aligned

16KB sliced

Parallel I/Os

ChunkServer1  ChunkServer2

File1 Part 1  File1 Part 2

Direct I/Os via libpfs

Async Writers

Persistant logs
Fully buffered logs
Pending logs
Unused buffer

LinkBuf

GPLSN

mtr1-3  mlog_p1  mlog_p2

mtr2-1

mlog_pn

Parallel I/Os

File1 Part 1  File1 Part 2

Direct I/Os via libpfs

Async Writers

Persistant logs
Fully buffered logs
Pending logs
Unused buffer

LinkBuf

GPLSN

mtr1-3  mlog_p1  mlog_p2

mtr2-1

mlog_pn
Case Study: PolarDB

Parallel Recovery

✓ Multi-task concurrent recovery / log application based on Log Partition
Case Study: PolarDB

Fast Validating

- Chunk 0
- Chunk 1
- Chunk 2

Centralized - backup Metadata

Avoid scanning all files during file verification

✓ Reduce the access to remote storage during startup

Logical Prefetch

Primary index

Secondary index

Asynchronous read

Trigger read-ahead

✓ Utilize aggregate bandwidth to reduce read delay
Case Study: PolarDB

Lock-optimized B-tree Index

1. to split
2. new node
3. Other read operation

- Remove redundant locks for operations (e.g., SMO) to improve the concurrency of memory and I/O operations.

Shadow Page

- Lock-free for write
- Optimize the long locking time during Page I/Os, to improve operation concurrency.

Flusher

Data File

Chunk
I/O Alignment & Scheduler

- For the direct I/O as bypassing the Cache of distributed file system
  - ✓ Align the optimal I/O offset & length to accelerate the direct I/O
  - ✓ Remove invalid I/O merge and perform random write
  - ✓ Adopt multi-asynchronous I/O task queue, fully utilize the advantage of high bandwidth

- For the long remote access and low I/O isolation
  - ✓ Adopt I/O priority scheduling: prioritizes critical I/Os to eliminate low isolation effects
Case Study: PolarDB

Experimental Results

Figure 10: Total performance evaluation of PolarDB.

Figure 11: Performance Breakdown.

Figure 12: WAL acceleration breakdown.
1. Background and Motivation
2. Design Considerations
3. Case Study: PolarDB
4. Case Study: RocksDB
Case Study: RocksDB
Case Study: RocksDB

Port corresponding optimizations to RocksDB:

- Scattered & Partitioned Global Log
- Scheduled Multi-queue Scatter I/O
- Direct I/O Alignment

Achieve expected performance gains

Figure 15: RocksDB Performance.
Thanks !