Carnegie Mellon University

Parpulse: I/O Service for Modern OLAP Database System

Yuanxin Cao, Lan Lou, Kunle Li
Current Status

- **75% Goal** – Build a functional I/O Service
  - ✔ Support reading data from the underlying storage (e.g. S3)
  - ✔ Implement a local cache with different cache policies on the Storage Node for fast data retrieval
  - ✔ Send requests from storage client to storage node

- **100% Goal** – Optimization
  - ✔ Add memory cache for small data
  - ✔ Add more parallelism with async and fine-grained lock
    - Handle 2 requests with the same key at the same time efficiently
    - Pull next data and push current data at the same time
  - ✔ Set up an E2E automatic benchmark pipeline

- **125% Goal** – More optimizations......
  - ✔ Optimize the storage reader for high-performance reading (I/O request merging)
  - □ Develop extra features such as prefetching, kernel bypassing for data reading
Architecture

Compute Node

Execution Engine

Operator A

Operator B

......

Storage Client

In read_table(request) -> Stream<RecordBatch>

Catalog Node

Request Handler

DataStoreCache

DataStore

DataStore

Replacer

LRU

LRU-K

MemDiskStore

SQLiteStore

File System

SQLite

S3

File System

S3Reader

MockS3Reader

StorageManager

Storage Reader

Storage Node
Storage Client
Storage Client

trait StorageClient

ParpulseStorageClientImpl (I/O Service Team 1)
IstziioStorageClientImpl (I/O Service Team 2)

Same basic logics:
- Send request to server and get data back (Parquet)
- Decode Parquet -> Arrow
- Stateless!
Mem-Disk Data Store Cache

Fine-grained Lock
Serve Parallel Requests!
Sqlite Data Store Cache

1 Parquet ⇔ 1 Sqlite Blob

Cache Key: row_id
Benchmark

- **Dataset**
  - Self-generated parquet files
    - 20 columns each, all floating point numbers (~5500 rows = 1Mb)
    - Two sizes: 1Mb, 100Mb
  - Trace file
    - csv file

- **Metric**
  - E2E time for client and server

- **Access Pattern**
  - Zipfian

- **Machine**
  - AWS EC2 (ubuntu 22.04, C5.xlarge → 4vCPU, 8GB memory)

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>File Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>365</td>
<td>10</td>
</tr>
<tr>
<td>688</td>
<td>10</td>
</tr>
<tr>
<td>1123</td>
<td>10</td>
</tr>
<tr>
<td>1203</td>
<td>10</td>
</tr>
<tr>
<td>1213</td>
<td>11</td>
</tr>
</tbody>
</table>
Whole process is triggered in GitHub Action!

<table>
<thead>
<tr>
<th>Triggered via push 6 hours ago</th>
<th>Status</th>
<th>Total duration</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>unw9527 pushed e369221</td>
<td>Success</td>
<td>6m 17s</td>
<td>–</td>
</tr>
</tbody>
</table>

benchmark_group_1.yml
on: push

Matrix: Start self-hosted EC2 r...

- 2 jobs completed
  Show all jobs

- Start the Parpulse server
  4m 36s

- Run Benchmark
  4m 43s

- Stop self-hosted EC2 runner
  4s
Discoveries

- Server time (Server E2E time, including polling data from S3, storing it into local cache, return receiver channel) decreases significantly when cache hits.

- **Bottleneck**
  - Big data: data transfer
  - Small data: HTTP setup

- When there are too many requests at the same time
  - Data transfer time will be long → B/W not enough
  - If TOO MANY → *Wait time will be long* → server cannot handle, requests stuck in client
<table>
<thead>
<tr>
<th>Trace</th>
<th>Average time (ms)</th>
<th>Network Time (ms)</th>
<th>Server Time (ms)</th>
<th>Data Transfer Time (ms)</th>
<th>Decode Time (ms)</th>
<th>Result Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m</td>
<td>105.569877</td>
<td>45</td>
<td>34</td>
<td>2</td>
<td>21</td>
<td>link</td>
</tr>
<tr>
<td>100m</td>
<td>5913.843632</td>
<td>714</td>
<td>631</td>
<td>2668</td>
<td>1862</td>
<td>link</td>
</tr>
<tr>
<td>Parallel</td>
<td>1958.889723</td>
<td>1168</td>
<td>162</td>
<td>320</td>
<td>182</td>
<td>link*</td>
</tr>
<tr>
<td>Serial</td>
<td>765.985548</td>
<td>40</td>
<td>244</td>
<td>63</td>
<td>409</td>
<td>link</td>
</tr>
</tbody>
</table>

**100m trace for multiple runs:** 5.924315929s, 6.683559418s, 6.004198074s (6-7s)

* From benchmark result, it is parallel test, but we wrongly set the commit message : ( 
Benchmark

- Where we get our result:
  https://github.com/cmu-db/15721-s24-cache-benchmark/actions/workflows/benchmark_group_1.yml
Single 100M Request

(screenshot from serial trace)

<table>
<thead>
<tr>
<th>timestamp</th>
<th>file_index</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>10</td>
</tr>
</tbody>
</table>

All requests are cache miss!
(timestamp unit: ms, arrive time for each request)

Server Time: 1.663519s

Parallel Multiple 100M Request

<table>
<thead>
<tr>
<th></th>
<th>timestamp</th>
<th>file_index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>265</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>278</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>510</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Avg Server Time: 2.053836s
Fanout cache num = 1

Avg Server Time: 1.795651s
Fanout cache num = 3
Memory Cache

- > 10 mb (large file) -> disk
- <= 10 mb (small file) -> memory
- don’t need to send extra S3 request to get size
- eviction -> write to disk cache

Basic Server Logic

- First get from cache -> hit?
- If hit, read & return
- If miss, ...
  - put data to cache
    - poll from s3
    - write data to cache
  - get data from cache again
  - return receiver channel

Disk Cache
Fine-grained Lock + Unlock Disk Manager

- **Write & Write**: Complete Status for keys + notify waiters
  - “status_of_key”: hashmap with completed/uncompleted status for each key
  - when requests come into put_data, see *uncompleted*, sleep to be notified
  - see *nothing*, insert incompleted, put_data, then notify all waiters

- **Get & Put Atomicity**: Get -> Put (but data in cache) -> Get (but data evicted)
  - status_of_keys also record all the keys in mem_replacer + disk_replacer
  - when requests come into put_data, see *completed*, pin data, directly return
Fine-grained Lock + Unlock Disk Manager

write  read  evict

- **Evict & Read**: pin & unpin data in replacer
  - Pin data when using (transfer to network, between put & get)

- **Write & Evict**: correctly update “status_of_keys”
  - Mem evict: lock “status_of_keys”
  - Disk evict: remove from “status_of_keys”

- **Write & Read**: First write to cache, then write to replacer
  - no need to lock replacer when writing data to mem/disk
  - if putting to replacer fails, then clean the mem/disk
  - “optimistic put”
Other Optimizations

- Fanout Cache (Benchmark is set to 9)
  
  https://grantjenks.com/docs/diskcache/tutorial.html#fanoutcache

```python
index = self._hash(key) % self._count
cache = self._caches[index]
try:
    # lock `cache`
    return cache.add(key, value, expire, read, tag, retry)
```

- Write current data to disk and poll next data from S3 at the same time
# Code Coverage Report

```text
| cmu-db / 15721-s24-cache1 / `main
<table>
<thead>
<tr>
<th>Coverage</th>
<th>Flags</th>
<th>Commits</th>
<th>Pulls</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>main</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## `main` Branch Context

**Source:** latest commit b4e2bc8

**Coverage on branch:** 85.64%

3078 of 3594 lines covered
Future work

- Predicate pushdown to storage node
- Kernel bypass when reading data (io_uring)
- Eliminate disk I/O on Storage Client
- Network improvement...
Thank You!