



Carnegie Mellon University

# Parpulse: I/O Service for Modern OLAP Database System

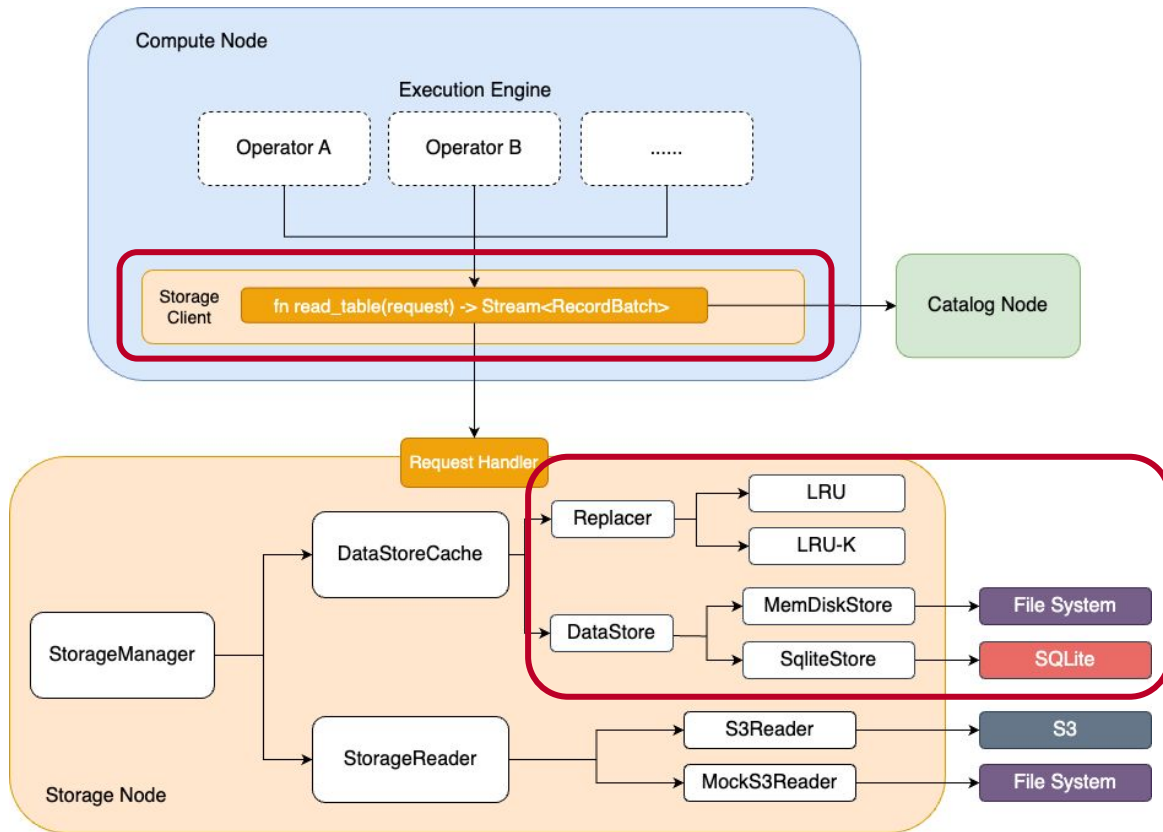
---

Yuanxin Cao, Lan Lou, Kunle Li

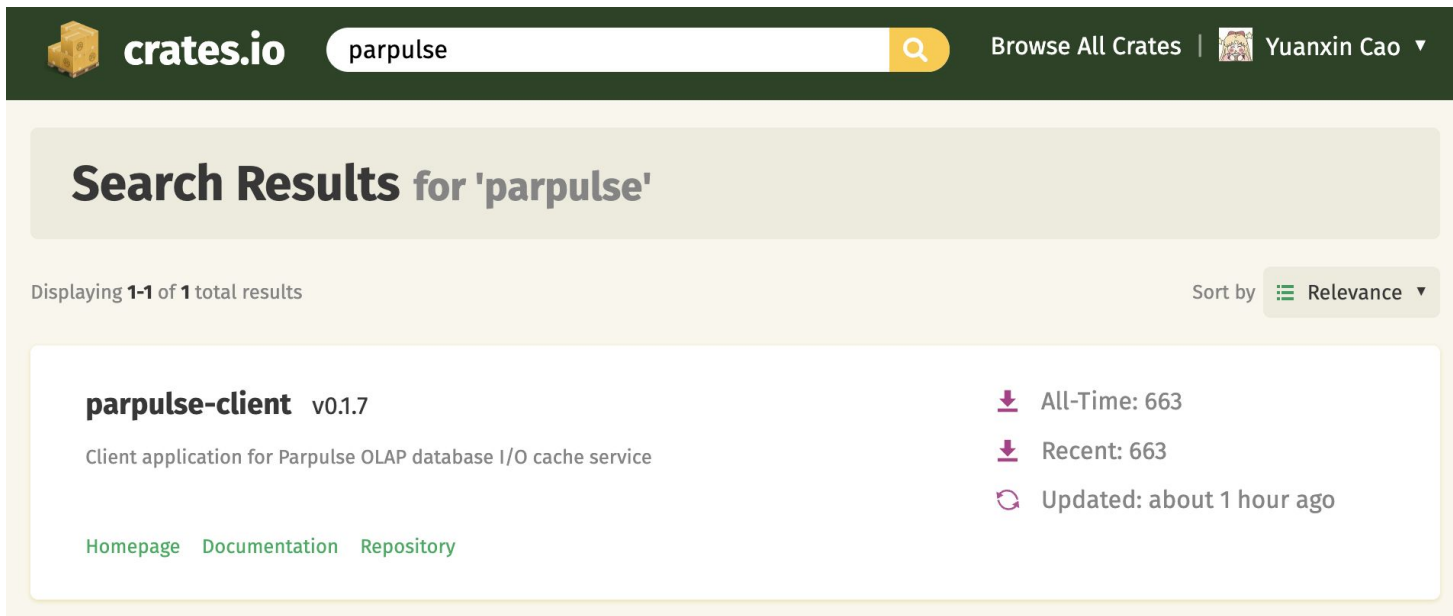


- **75% Goal** – Build a functional I/O Service
  - ✓ Support reading data from the underlying storage (e.g. S3)
  - ✓ Implement a local cache with different cache policies on the Storage Node for fast data retrieval
  - ✓ Send requests from storage client to storage node
- **100% Goal** – Optimization
  - ✓ 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



# Storage Client



The screenshot shows the crates.io search interface. At the top, the crates.io logo and name are on the left, followed by a search bar containing 'parpulse' and a search icon. To the right of the search bar are links for 'Browse All Crates' and a user profile for 'Yuanxin Cao'. Below the search bar, the heading 'Search Results for 'parpulse'' is displayed. Underneath, it says 'Displaying 1-1 of 1 total results' and 'Sort by Relevance'. The search result for 'parpulse-client v0.1.7' is shown, with a description: 'Client application for Parpulse OLAP database I/O cache service'. To the right of the description are three statistics: 'All-Time: 663', 'Recent: 663', and 'Updated: about 1 hour ago'. At the bottom of the result card are links for 'Homepage', 'Documentation', and 'Repository'.

crates.io parpulse Browse All Crates | Yuanxin Cao

## Search Results for 'parpulse'

Displaying 1-1 of 1 total results Sort by Relevance

**parpulse-client** v0.1.7  
Client application for Parpulse OLAP database I/O cache service

All-Time: 663  
Recent: 663  
Updated: about 1 hour ago

[Homepage](#) [Documentation](#) [Repository](#)

# Storage Client

trait StorageClient

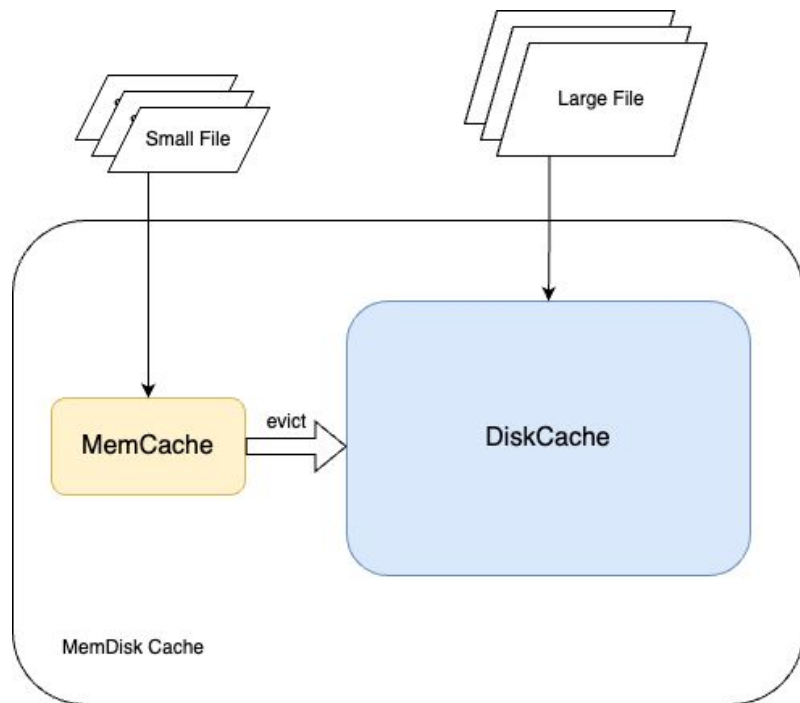
ParpulseStorageClientImpl  
(I/O Service Team 1)

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**

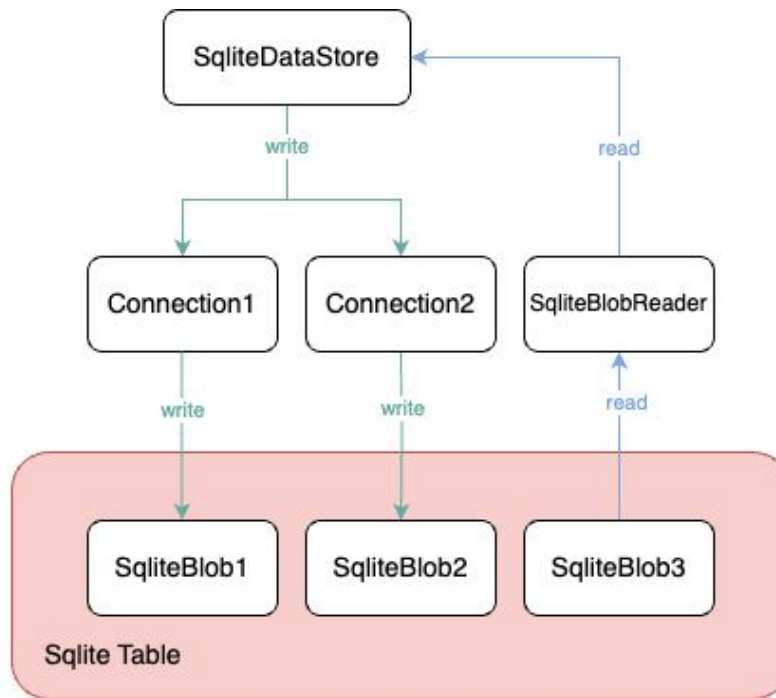
**Serve Parallel Requests!**

# Sqlite Data Store Cache

1 Parquet ⇔ 1 Sqlite Blob

parpulse_cache	
pk	<u>row_id</u>
	content (blob)

Cache Key: row\_id



# Benchmark

- Dataset

```
----- First 5 rows of the parquet file -----
      col 1      col 2      col 3      col 4      col 5      col 6      col 7      ...
0  0.191954  0.481544  0.470787  0.779391  0.218772  0.994886  0.119190  ...
1  0.841532  0.687789  0.691565  0.930984  0.852517  0.107169  0.909794  ...
2  0.606424  0.839445  0.475433  0.842104  0.789378  0.456189  0.564459  ...
3  0.474810  0.274558  0.159864  0.478374  0.891250  0.247201  0.624717  ...
4  0.933724  0.145045  0.506208  0.020810  0.916214  0.142837  0.955672  ...
```

- | Arrival Time | File Index |

- Metric

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

- Access Pattern

- Zipfian

- Machine


- AWS EC2 (ubuntu 22.04, C5.xlarge → 4vCPU, 8GB memoi

1	timestamp	file_index
2	0	11
3	8	10
4	365	10
5	688	10
6	1123	10
7	1203	10
8	1213	11



# Whole process is triggered in GitHub Action!

Triggered via push 6 hours ago

 unw9527 pushed `-o- e369221` `cache1-bench`

Status

**Success**

Total duration

**6m 17s**

Artifacts

–

## benchmark\_group\_1.yml

on: push

Matrix: Start self-hosted EC2 r...

✓ 2 jobs completed

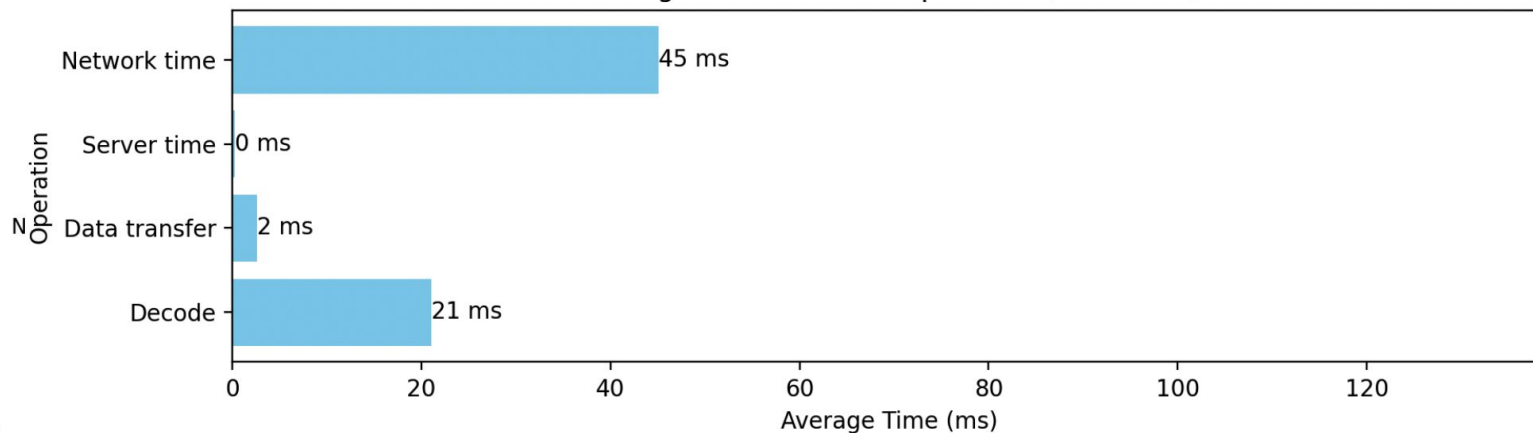
Show all jobs

✓ Start the Parpulse ser... 4m 36s

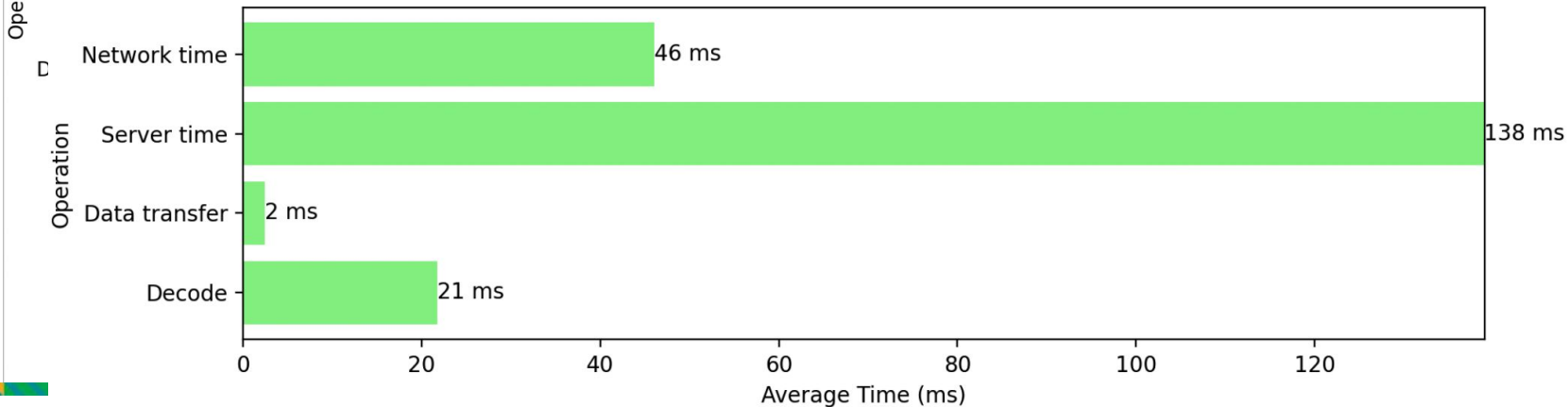
✓ Run Benchmark 4m 43s

✓ Stop self-hosted EC2 runner 4s

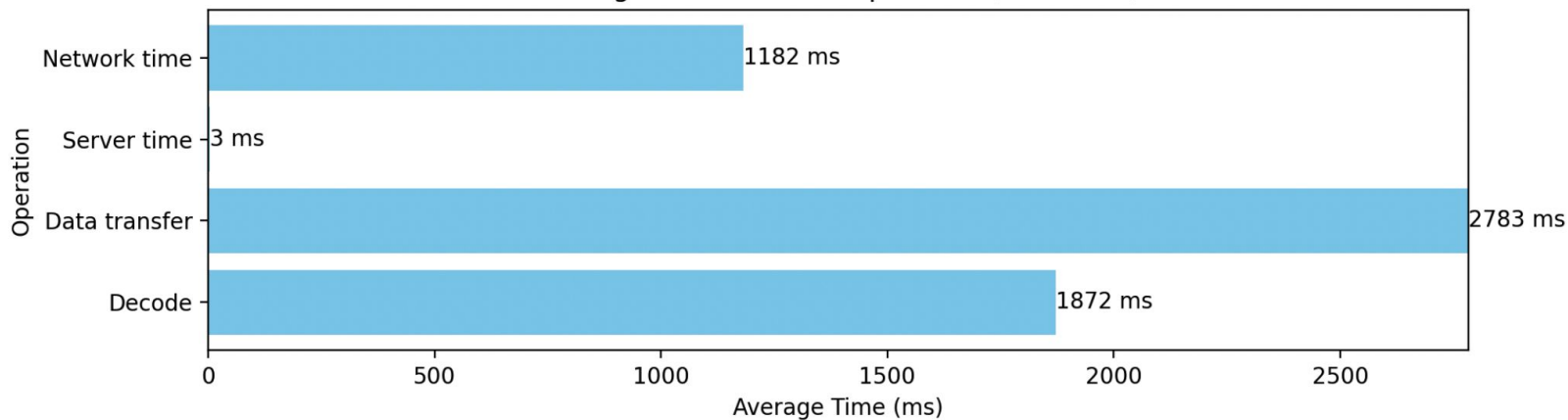
Average Time for Each Operation ( $\bar{C}$  Cache Hit)



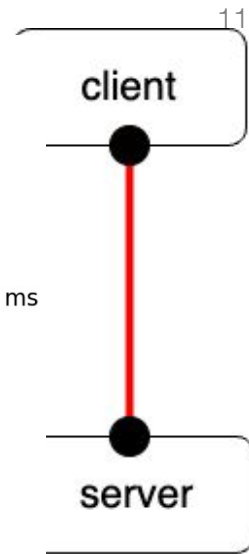
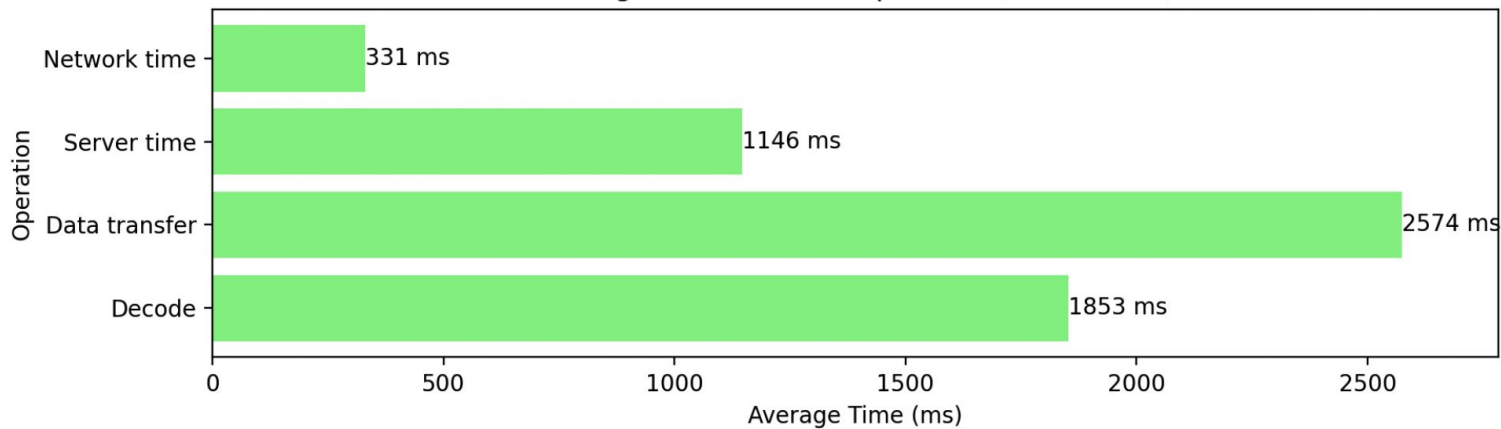
Average Time for Each Operation (Cache Miss)



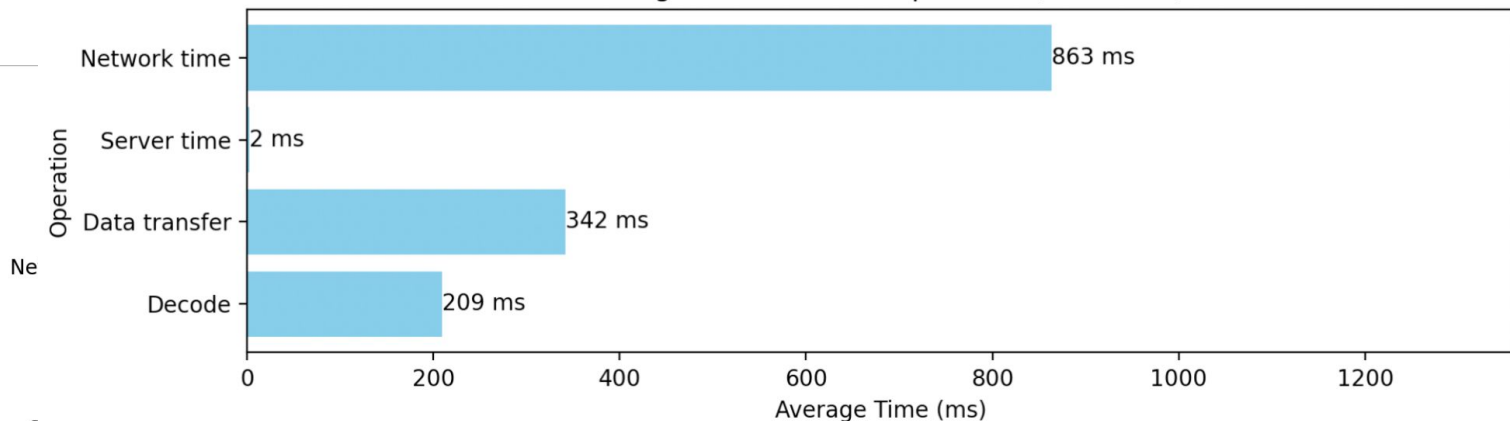
Average Time for Each Operation (Cache Hit)



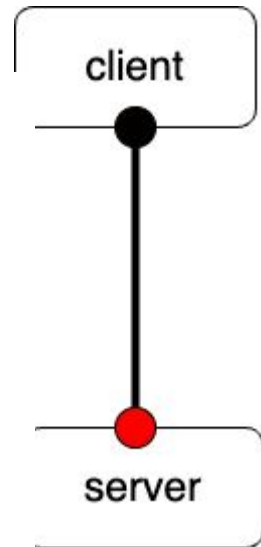
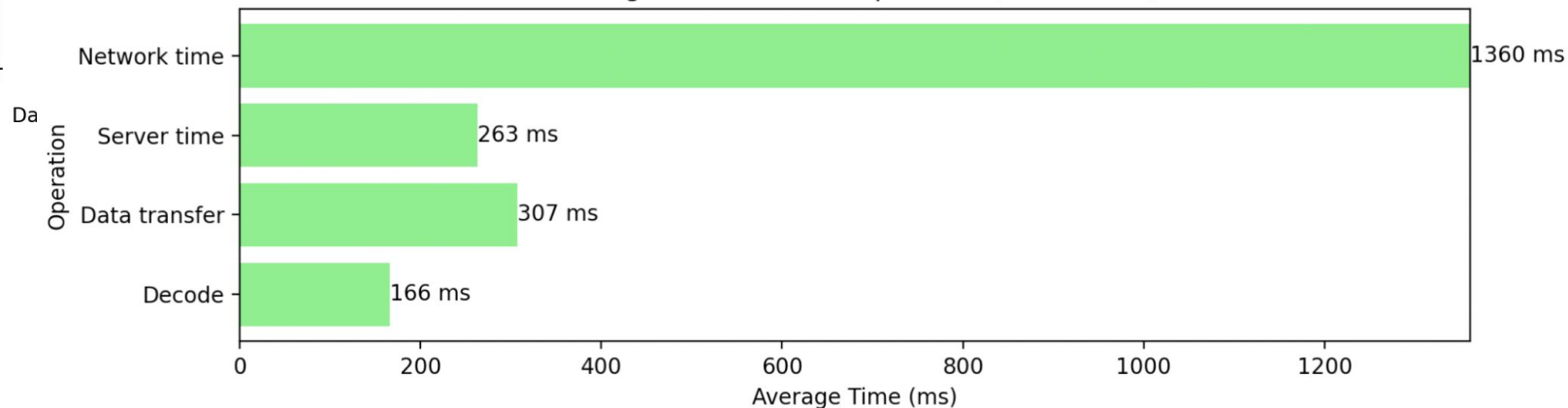
Average Time for Each Operation (Cache Miss)



Average Time for Each Operation (Cache Hit)



Average Time for Each Operation (Cache Miss)



## 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

# Benchmark

Trace	Average time (ms)	Network Time (ms)	Server Time (ms)	Data Transfer Time (ms)	Decode Time (ms)	Result Link
1m	105.569877	45	34	2	21	<a href="#">link</a>
100m	5913.843632	714	631	2668	1862	<a href="#">link</a>
Parallel	1958.889723	1168	162	320	182	<a href="#">link</a> *
Serial	765.985548	40	244	63	409	<a href="#">link</a>

**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-s2-4-cache-benchmark/actions/workflows/benchmark\\_group\\_1.yml](https://github.com/cmu-db/15721-s2-4-cache-benchmark/actions/workflows/benchmark_group_1.yml)

<p>✓ <b>9 caches run 1m</b> benchmark-group-1 #78: Commit <a href="#">8c63217</a> pushed by xx01cyx</p>	cache1-bench
<p>✓ <b>9 caches run 100m</b> benchmark-group-1 #77: Commit <a href="#">bbe756a</a> pushed by xx01cyx</p>	cache1-bench
<p>✓ <b>9 caches run parallel</b> benchmark-group-1 #76: Commit <a href="#">625ab78</a> pushed by xx01cyx</p>	cache1-bench
<p>✓ <b>9 caches run 100m</b> benchmark-group-1 #75: Commit <a href="#">7bb1ff5</a> pushed by xx01cyx</p>	cache1-bench
<p>✓ <b>9 caches run serial</b> benchmark-group-1 #74: Commit <a href="#">38e452f</a> pushed by xx01cyx</p>	cache1-bench
<p>✓ <b>9 caches run parallel</b> benchmark-group-1 #73: Commit <a href="#">f11e165</a> pushed by xx01cyx</p>	cache1-bench
<p>✓ <b>9 caches run 100m</b> benchmark-group-1 #72: Commit <a href="#">50e8b6b</a> pushed by xx01cyx</p>	cache1-bench
<p>✓ <b>9 caches run 1m</b> benchmark-group-1 #71: Commit <a href="#">2aa1774</a> pushed by xx01cyx</p>	cache1-bench

## Single 100M Request

(screenshot from **serial** trace)

timestamp	file_index
10000	10

**All requests are cache miss!**

(timestamp unit: ms, arrive time for each request)

## Parallel Multiple 100M Request

	timestamp	file_index
1	0	11
2	250	11
3	255	12
4	265	13
5	278	14
6	500	14
7	510	15

Server Time: 1.663519s



Avg Server Time: 2.053836s

Fanout cache num = 1 [link](#)

Avg Server Time: 1.795651s

Fanout cache num = 3 [link](#)



## 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**



## 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**

**Server Time**

# Fine-grained Lock + Unlock Disk Manager

write

read

evict

- **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>

```
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

[cmu-db](#) / [15721-s24-cache1](#) / [main](#)

**Coverage** [Flags](#) [Commits](#) [Pulls](#)

## Branch Context

main 

**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...



Carnegie Mellon University

Thank You!

---