Hello!

CS 744: OWL

Shivaram Venkataraman
Fall 2022
Project checkin – feedback
GPU availability

Midterm 2: Dec 6th in class

Poster presentation: Dec 13th
Final report: Dec 20th

CloudLab limited

Google Cloud
Recipes for Marius Tools? Ask us!
NEW DATA, HARDWARE MODELS
CONTENT DISTRIBUTION

What is content?
- Docker containers
- AI models
- Search indexes

How is this workload different?
- Read heavy, very few updates
- Skew / hot content → launch 100,000 workers → all read the same binary at same time
- Hot spots which lead to slowdown / failures
CHALLENGES / GOALS

Goals

- Minimize requests to external storage
- Latency of content fetch -> minimize
- Availability -> minimize failed requests

Challenges

- Load spikes hot content -> spike in time
- Different policies for contents -> configurable / flexible
- Manageability -> Debugging, Updating
PRIOR SOLUTIONS

Centralized solution

Hierarchical caching
- Need a lot of resources
- Quotas, Traffic bursts
- Millions of clients

BitTorrent
- Scalable, decentralized
- Stale peer data,
- Lack of global picture

Manageability
Errors
OWL

Data Source

Tracker

Tracker

Tracker

Metadata of where data is cached

FileSystem

Database

Decentralized data plane

Centralized control plane

Which peers cache what data

Which peers fetch data from where

Processes that want to fetch data

Peer

Peer

Peer
OWL DESIGN

Peers, Superpeers
Trackers
Ephemeral Distribution Trees
Tracker Sharding
Fault Tolerance
Peers, SuperPeer

What is a Peer?
- Simple API, functionality
- Ask Tracker where to fetch
- Cache in memory / local disk

SuperPeer
- Standalone process (no client)
- More resources for caching
**TRACKERS**

Centralized state for large number of peers
Peers register with a random tracker

What is state? → Two Data structures
Chunk → Peers caching it
Peer → List of chunks cached

Soft-state (similar to GFS)

Tracker

- millions of peers in a data center
- chunk → which peers
- Chunk1, chunks...
- Peer2 → Chunk1
- Bottleneck peers connect
- Single Tracker - Fault Tolerance
- Memory for state
- (5 MB) free
DISTRIBUTION TREES

To fetch data

Peer sends `get_data(range)` → Chunks

For a chunk, `getSource(chunk)` →

Peer/Super Peer/External Storage

Peer directly reads from source.

Trackers

Build ephemeral distribution tree for a chunk

Stream data from peers

Locality based
POLICIES IN TRACKERS

Selection Policy
Which peer should we fetch from

→ locality based, load balancing

Caching Policy
Which blocks should be stored in memory

→ Cache eviction
   LRU as default, least rare chunk is evicted.

Buckets to control configuration across applications
TRACKER SHARDING

 Millions of peers, tracker bottlenecks
Partition peers across trackers
  - peers pick a random trackers
Challenge?
  - We need to share knowledge of chunks cached by peers in other tracker!
  - Periodically exchange list of chunks cached
VIRTUAL SUPERPEERs

What data should a peer store?
   So far: Data already fetched a peer
   Can we use a peer for caching other data?

Partition cache space into peer / virtual superpeer
   Use spare memory on the machine

Tracker-only concept!
SUMMARY

Problem: Content distribution is challenging
Approach: Decentralized data-plane, centralized control plane

Features
- Ephemeral data distribution trees
- Policies on tracker for selection, caching
- Sharding trackers for scalability, fault tolerance
DISCUSSION

https://forms.gle/cbAyPYsVGqdcaZyx9
What is one disadvantage of the design used in Owl? Construct one scenario to highlight how this disadvantage might affect a client.

- Client needs to sacrifice memory/disk space → affects performance.
- If you use no space on clients, caching is only on super peers, which is similar to hierarchical caching.
- Peer churn could be high → if app is short lived.
- Client network bandwidth is used by Owl (esp. for hot data).
- Shard peers → tracker only looks at peers within shard → sacrifice locality.
- Very small files → latency added by RPCs to tracker.
Next steps:

- TPU Paper
- Midterm 2, Dec 6th
- Poster session, Dec 13th