Kraken

P2P Docker Image Distribution

Uber

Cody Gibb

Evelyn Liu

Yiran Wang

Agenda

- Docker image and docker registry internals
- Common ways to speed up docker pull
- P2P solutions
- How Uber solves scale problem with Kraken
- Q&A

What is a docker image

- Image layers
    - Regular tar.gz files
    - Each layer = one line in Dockerfile
    - One special layer - image config
        - Defines ENV, USER, etc.
- Image manifest
    - Regular json file
    - Contains SHA256 digests of layers
- Image tag
    - Key-value pair, human-readable name to manifest SHA
- You can easily construct a docker image by hand!

What is a docker registry

- A simple web server to store and distribute docker images
- Straightforward REST APIs:

GET	/v2/
GET	/v2/<name>/tags/list
GET, PUT, DELETE	/v2/<name>/manifests/<reference>
GET	/v2/<name>/blobs/<digest>
POST	/v2/<name>/blobs/uploads
GET, PUT, PATCH, DELETE	/v2/<name>/blobs/uploads/<uuid>

What is `docker pull`

- Resolve tag to manifest
- Pull manifest, find layers not available locally
- Pull tar.gz files
- Decompress them

Speed up `docker pull`

- Make your docker images homogeneous
    - Use a common base image
    - Dockerfile template, multi-stage build
    - Use a build tool that supports distributed layer cache
        - Makisu from Uber, Kaniko from Google, BuildKit from Docker/Moby

Scale Docker Registry

- Profile first
- Start with a layer of Nginx caches
  - Ideal for bursty workloads
  - Works better with connection limits
- Nginx was not enough for Uber

Uber’s Workload

- Large images
  - 1G average, 10G is becoming common
- Batch jobs
  - Concurrent docker pull - O(10k)
    - Cannot use HDFS
- Host maintenance
  - Reshuffle large number of unique images - O(1k)
    - Cannot add more Nginx
- Replication across zones - on-prem + cloud zones
  - More expensive and complex as Uber add more zones

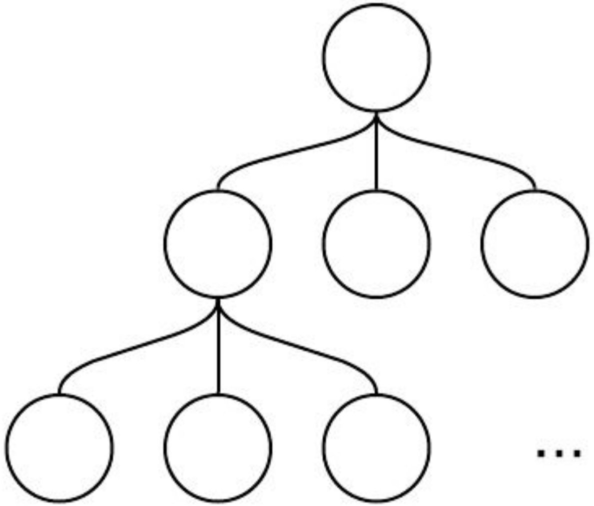
P2P

Design Considerations

- Optimize for data center internal usage
  - We control all peers
- Low maintenance
  - No single point of failure
- Handle bursty load
  - O(10k) of same image
  - O(1k) of unique images
- Care about tail completion time

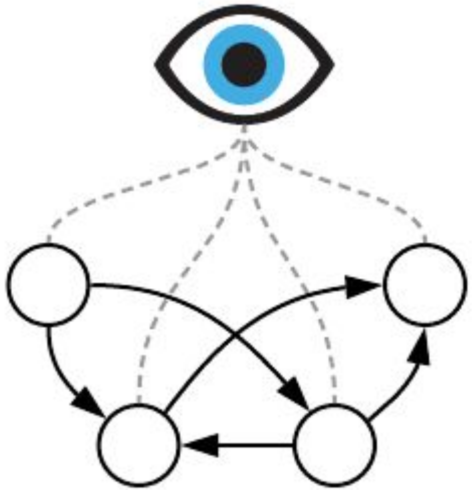
Layered Structure

- Easy to understand
  - Just a tree (or trees)
- Fat trees not optimal for big blobs
  - Speed limited by number of branches
- Hard to maintain topology
- e.g. LAD from Facebook



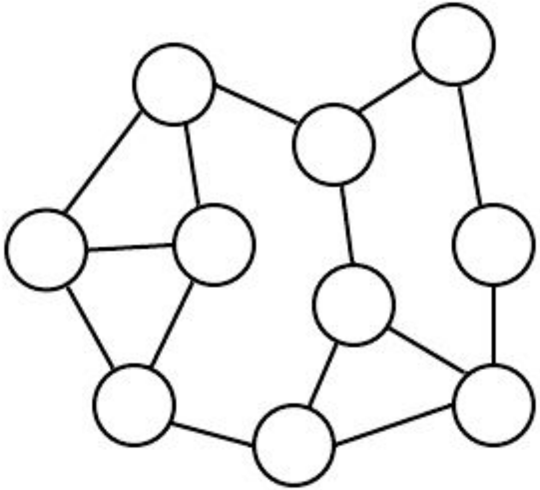
Central Overseer

- Divide blob into small data chunks (4MB)
- One central component makes all decisions
  - Schedules P2P transmission of each data chunk for each node
- Optimal in theory, hard to implement
  - Need to support very high QPS
  - Hard to handle node failure and slowdown
- e.g. Dragonfly from Alibaba



Random Graph

- A central component makes connection decisions
- Nodes make transfer decisions
- "*Random regular graph*"
    - Good connectivity, small diameter
    - Jellyfish, NSDI 2012
- Performant
    - ≈ 80% of max speed in simulation with only one seeder
- Resilient to failures
- Our pick



Kraken

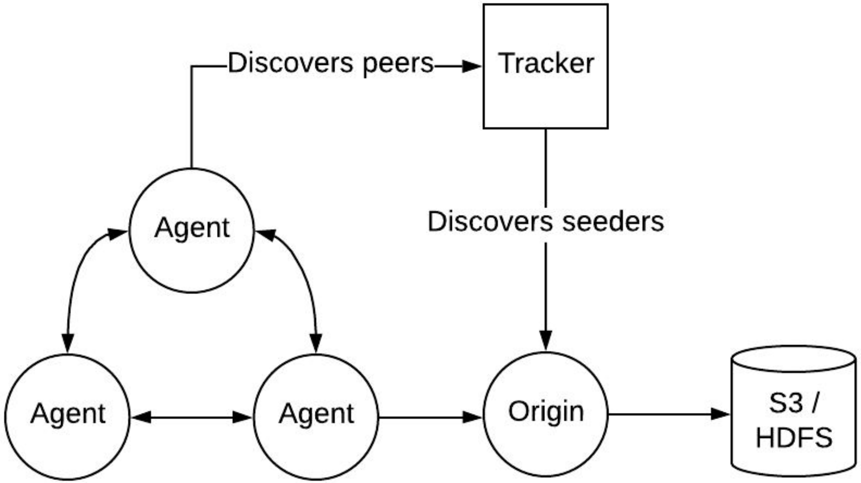
Glossary

- Torrent
    - File broken into multiple *pieces*, typically 4MB each
    - Pieces transferred independently
- Peer
    - Participant in torrent network
    - Connected peers transfer pieces between each other
    - Peer with 100% of pieces is a *seeder*

Kraken Architecture

Components

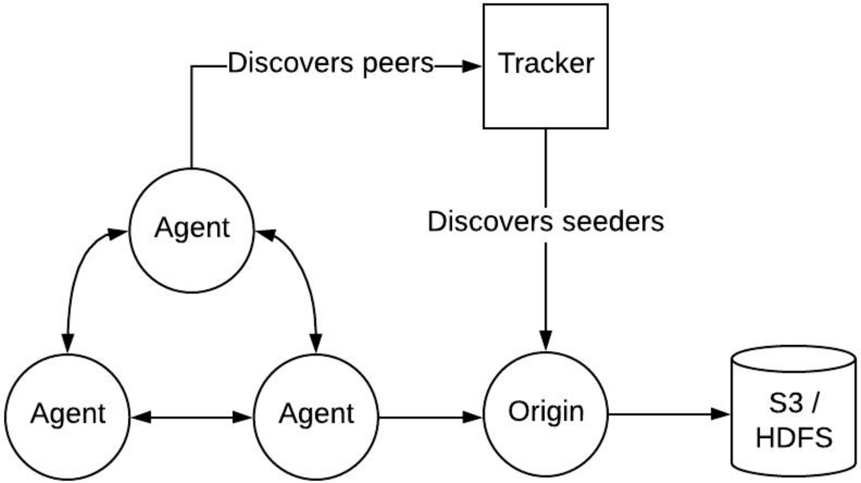
- Agent
  - A peer on every host
  - Implements Docker registry interface
- Origin
  - Dedicated seeders
  - Pluggable storage backend (e.g. S3)
  - Self-healing hash ring
- Tracker
  - Tracks peers and seeders in-memory
  - Self-healing hash ring



Kraken Architecture

Key Features

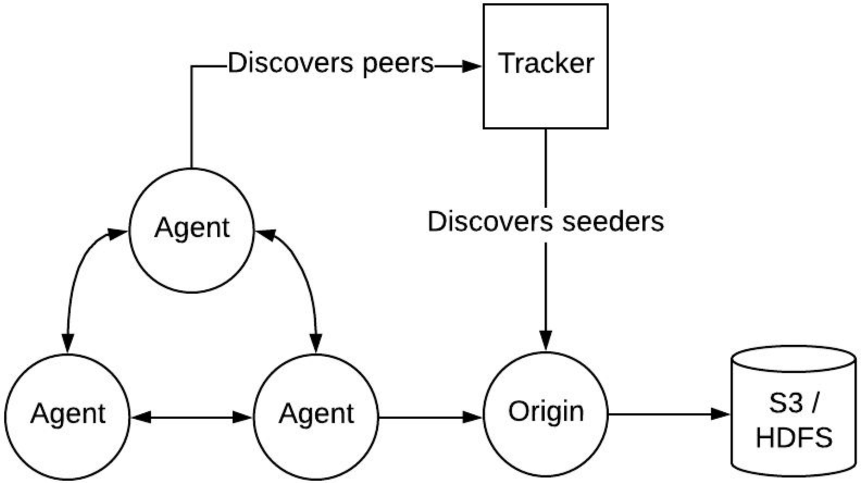
- Just a caching / distribution layer
  - Should be able to suffer total data loss and recover
  - All blobs have TTL
- Minimal dependency set
  - DNS
- Content-addressable blobs
  - Blob identifier is hash of blob content
  - Immutable
  - Disadvantage: not user friendly



Kraken Architecture

Peer Discovery

- Tracker returns 50 random peers, sorted by preference
  - Completed agents (highest)
  - Origins
  - In-progress agents (lowest)
- Agent iterates through the 50 until it has 10 connections



Elapsed: 0.01s

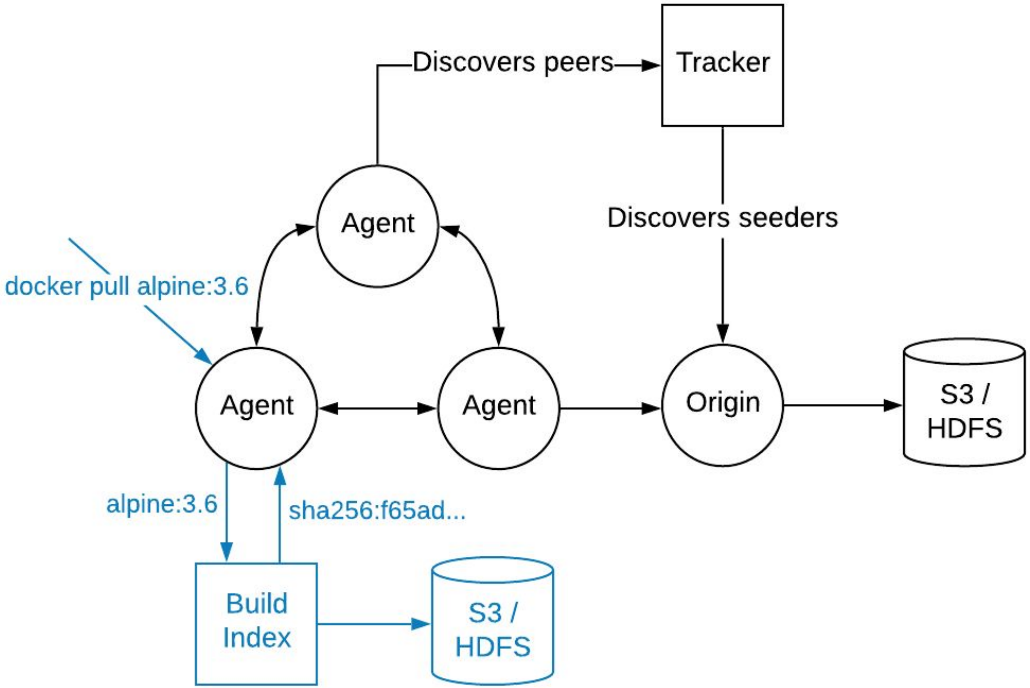
Num peers: 59

Blue	Origin
Grey	Agent
Yellow	Agent (downloading)
Green	Agent (completed)

Kraken Architecture

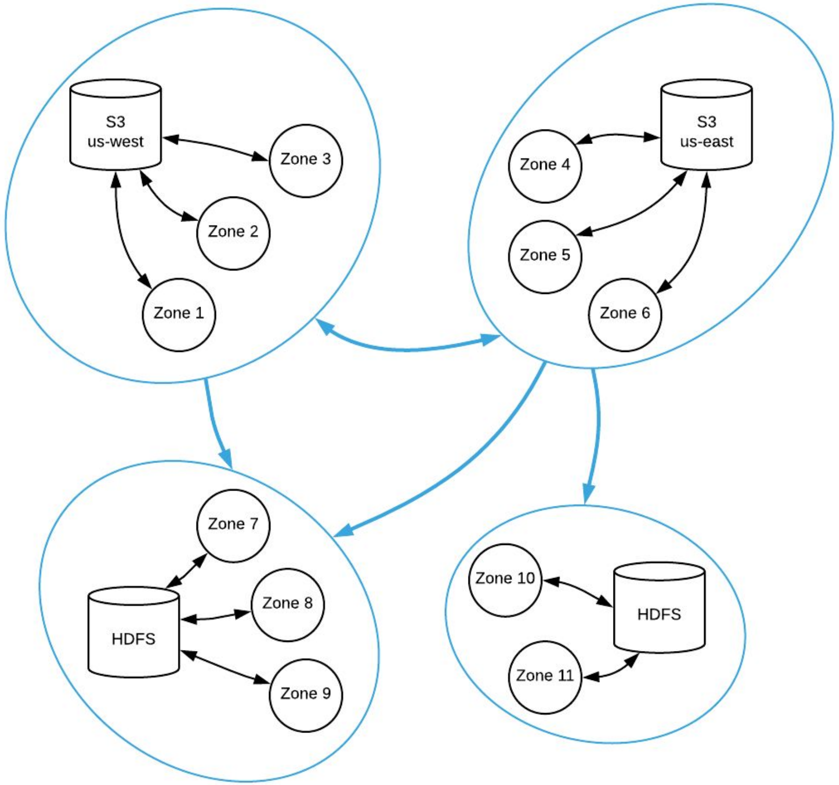
Build Index

- Mapping of tag to manifest SHA256 digest
- No consistency guarantees
  - Client must use unique tags
- Pluggable storage
- Powers image replication between clusters
  - Simple queue with retry



Global Replication in a Hybrid Cloud Environment

- Kraken cluster in each zone
- Zones in each region share a storage backend



Security

- Mutual TLS for all communications with central components
    - End to end security
- P2P traffic doesn’t go through TLS (yet)
    - No need to worry about data integrity

Results

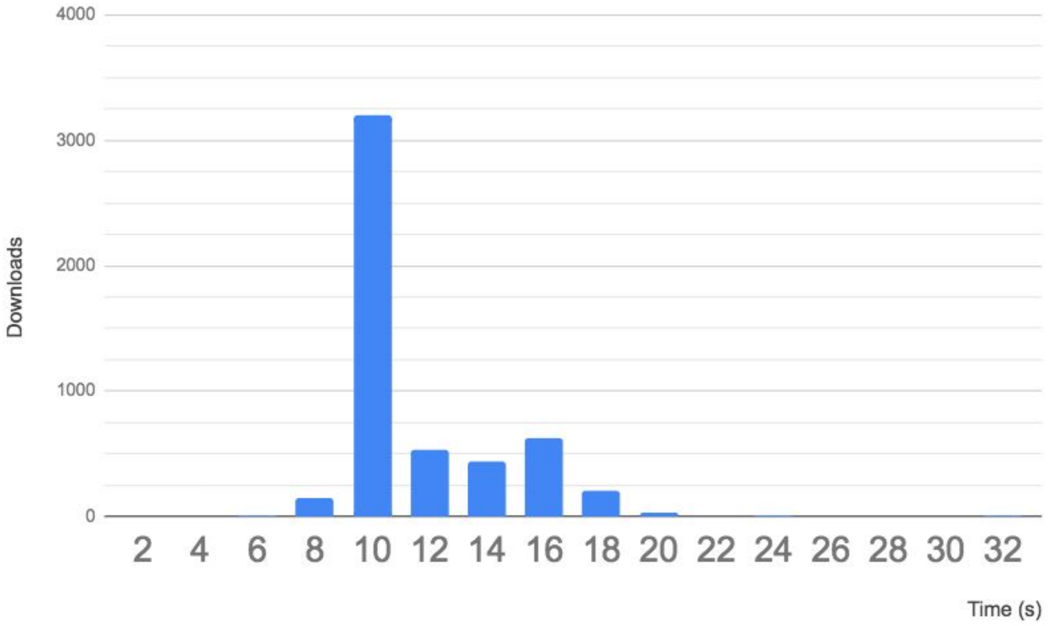
Performance in Test

Setup

- 3G image with 2 layers
- 2600 hosts (5200 downloads)
- 0.3GB/s speed limit
- Theoretical max 10s

Result

- P50 10s
- P99 18s
- Max 32sec
  - Outlier (bad host?)



Performance in Production

Blobs distributed per day in busiest zone:

- 1mil 0-100MB blobs
- 600k 100MB-1G blobs
- 100k 1G+ blobs

Peak

- ***20k 100MB-1G blobs within 30 sec***
    - With old setup, this would've caused outage

Optimizations

- Low connection limit
  - Less overhead
- Aggressively disconnect unneeded connections
  - Rebalance network
- Pipelining
  - Maintain a request queue of size n for each connection
  - Less idle time between peers
- Endgame mode
  - For the last few pieces, request from all connected neighbors

Unsuccessful Optimizations

- Prefer peers on the same rack
  - Likely to form a disjoint graph
  - Not needed for over-provisioned network
- Reject incoming connection based on number of mutual connections
  - Haven't seen issues caused by graph density problems

Takeaways

- Solution not specific to Docker
  - Integrations with other storage systems
- P2P solutions can work within data centers
- Randomization works
- Get something working first before optimization
  - Hard to predict how P2P works without experimentation

Future Plan

- Performance for massive number of tiny files
- Observability
- Tighter integration with other registry implementations
- Tighter integration with Kubernetes

github.com/uber/kraken