CernVM-FS & Varnish

CernVM-FS & HTTP Caching

CernVM-FS

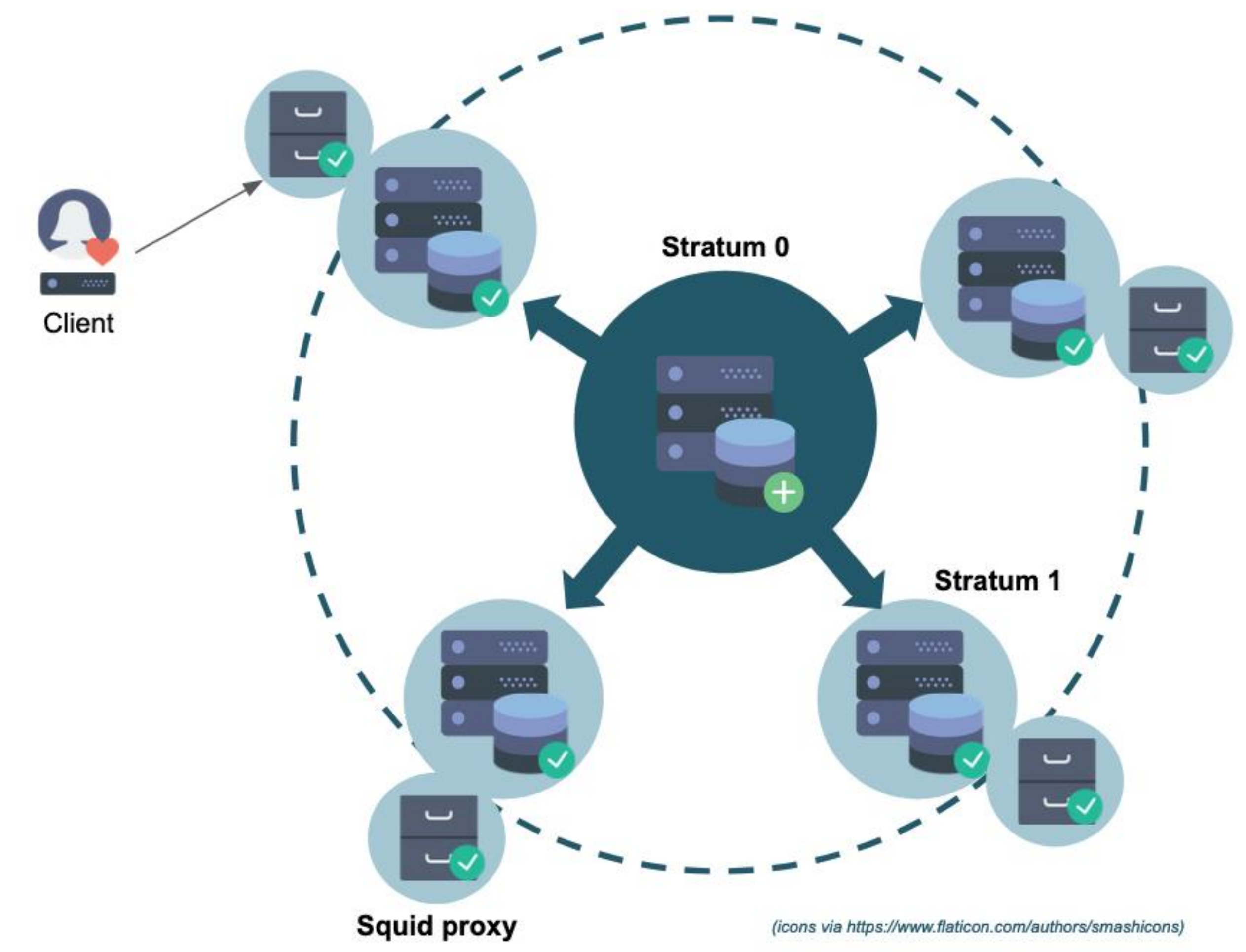
- Clients have read-only access to FS
- Clients get notified of FS changes
- Client = HPC cluster to laptop

Caching

- Reverse Proxy (Squid)

CERN

- Place of birth of the Web
- Culture around “Distributed computing”



VARNISH SOFTWARE

Varnish, born to replace Squid

Squid (1996) one of the forward proxies of reference

- Primary goal is to be feature full
- Designed to supports multiple protocols
- Very versatile, **able to do reverse proxy**
- Designed for single CPU architectures
- Somewhat inefficient virtual memory management
- Squid dev team has low bandwidth to improve it (vulnerabilities stay unfixed for years)

Varnish (2006) **created as a reverse proxy** to optimize infrastructure and reduce costs

- Focused on caching proxy features
- Designed for an optimized handling of HTTP traffic
- Primary goal is to be efficient and secure
- Designed for high parallelism and workload isolation
- Efficient memory management
- Varnish has a community improving it in and out of Varnish Software

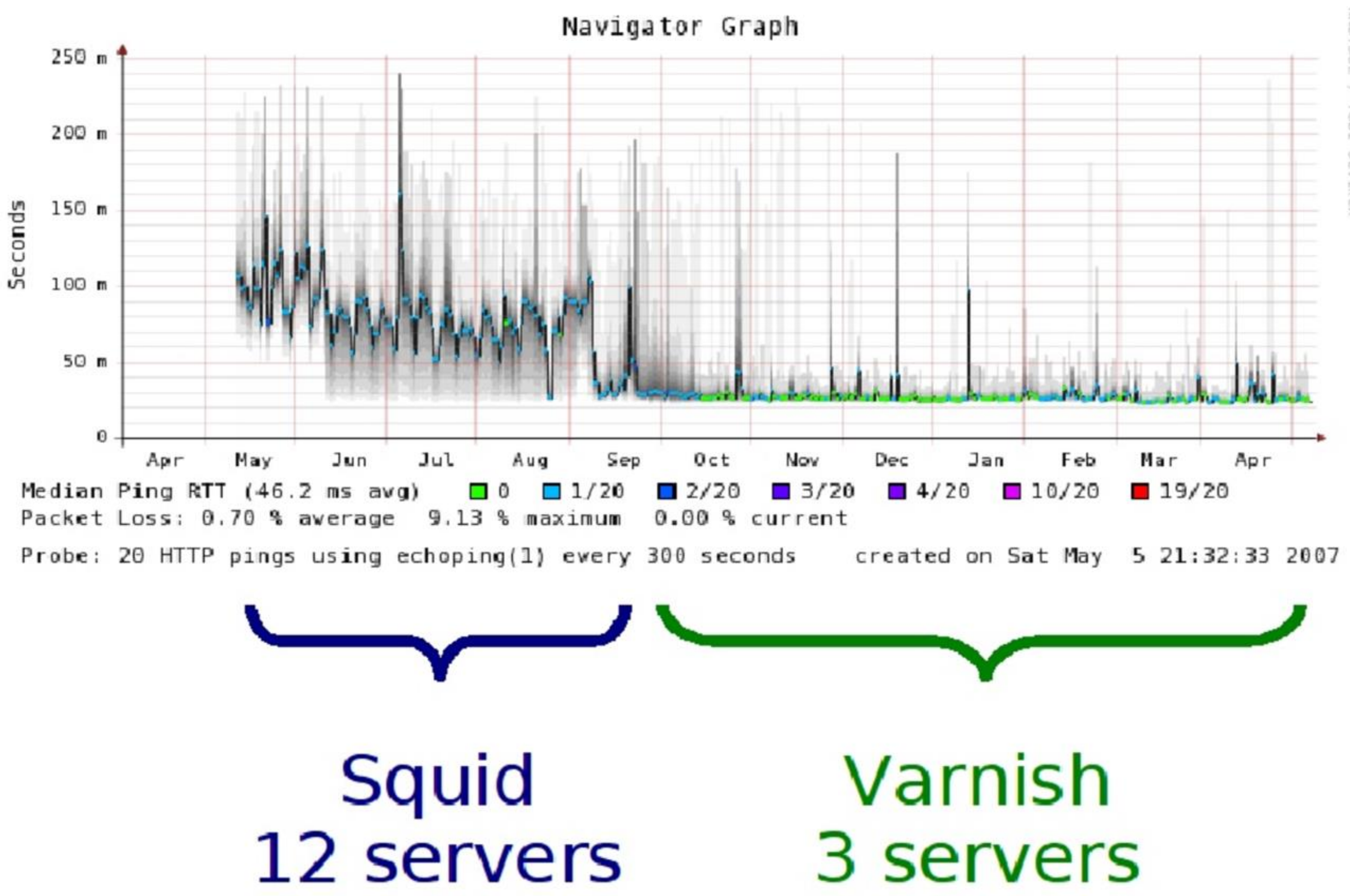
Squid VS Varnish in practice (1)

VG Multimedia (2006)

- From: 12 Squid instances
  ~100% CPU usage
  ~150ms average latency

- To: 3 Varnish servers
  ~10% CPU usage (**-90%**)
  ~30ms average latency (**-80%**)

First deployment of Varnish Cache in production



On Virtual memory management design by Poul-Henning Kamp, Varnish Architect
"You're Doing It Wrong. Think you've mastered the art of server performance? Think again."
https://queue.acm.org/detail.cfm?id=1814327

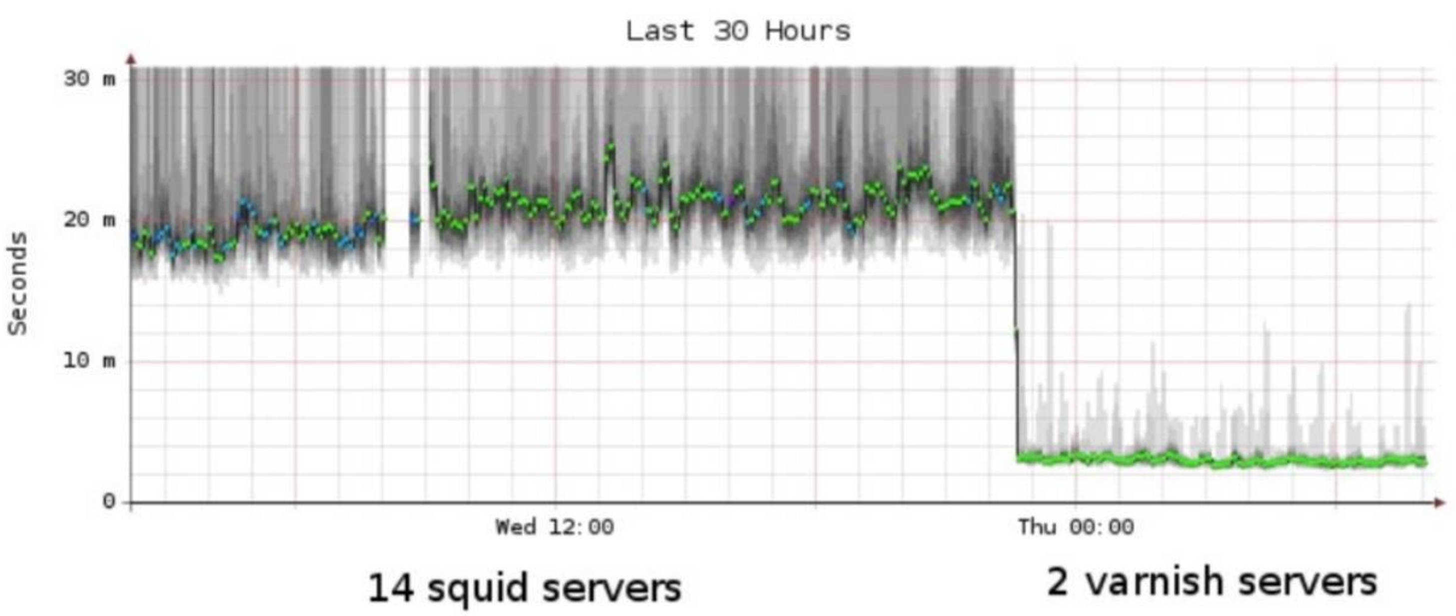
Squid VS Varnish in practice (2)

Aller Internett (2011)

- 20ms -> 4ms average (**-80%**)
- 99th percentile improved
- Smoothes reads
- Reliable streaming

Reduced average latency and average CPU usage is very significant

Reducing outliers is critical



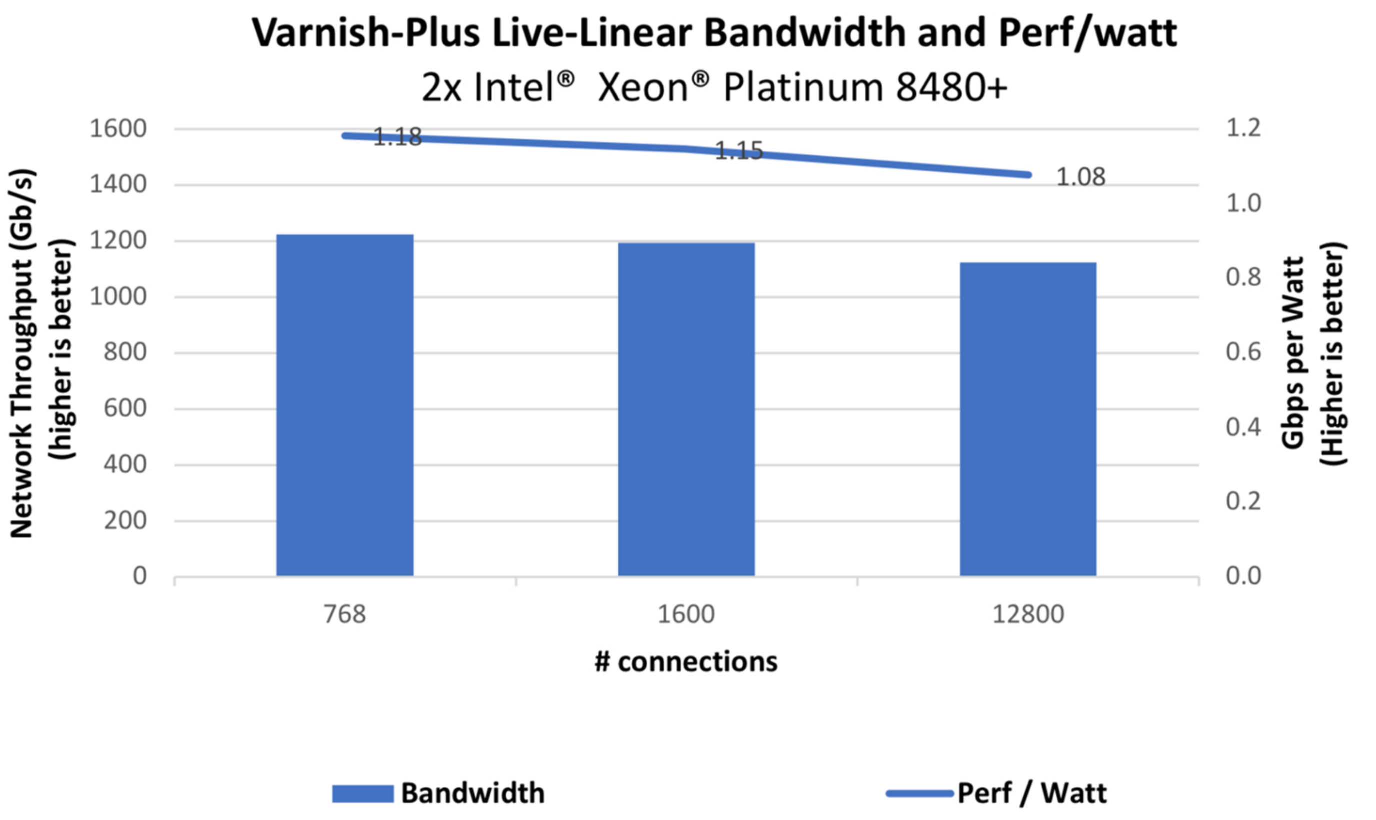
Article about the move from Squid to Varnish (Norwegian only)
“Mye raskere digi.no med friprog-rakett”
https://www.digi.no/artikler/mye-raskere-digi-no-med-friprog-rakett/284924

Varnish Enterprise built for performance

Intel (2024)

- Varnish Enterprise testing
- 1.2 Tbps Data Rate,
- 1.18 Gbps/Watt Efficiency

- World’s Fastest Delivery
- High & Stable Throughput
- Energy efficiency



Intel White Paper testing Varnish Enterprise

“Varnish Enterprise Shows Up To 1.2 Tbps Data Rate, Up To 1.18 Gbps/Watt Efficiency”[3]

https://cdrdv2-public.intel.com/788717/Varnish_23_WP_091523_REV02.pdf

Varnish Enterprise & Cache

Varnish Cache

Open source project: https://varnish-cache.org/

- Finely grained caching policy
  - Differentiate easily mutable and immutable data
  - Manage precise lifecycle in cache
- In-memory and on-disk storage available
- Advanced load-balancing capabilities
- HTTPS client support through a TLS proxy (such as Hitch)

Varnish Enterprise

Commercial product from Varnish Software: https://www.varnish-software.com/ - Built for high-performance workloads

- **Massive Storage Engine (MSE4)**
  - Store objects persistently on local disk(s)
  - Runtime disk management inc fault tolerance
  - Hybrid storage for indexes (NVMes) and data (HDDs)
  - Policy based data placement and metering
  - Automatically adjust cache size according to memory consumption
- **Slicer** allows subdivision of payloads
  - Efficiently request and serve fragments of objects instead of whole objects
- **Native TLS** support both for serving and backend requests

Varnish Cache easy setup

Varnish Cache available on most distributions

- Distributions have specific versions available

Ubuntu 22.04	Varnish Cache 6.6.1
Debian 12	Varnish Cache 7.7.1
RHEL8 / Alma Linux 8 / Rocky Linux 8	Varnish Cache 6.0.13

- Backward compatibility not guaranteed between Varnish Cache versions

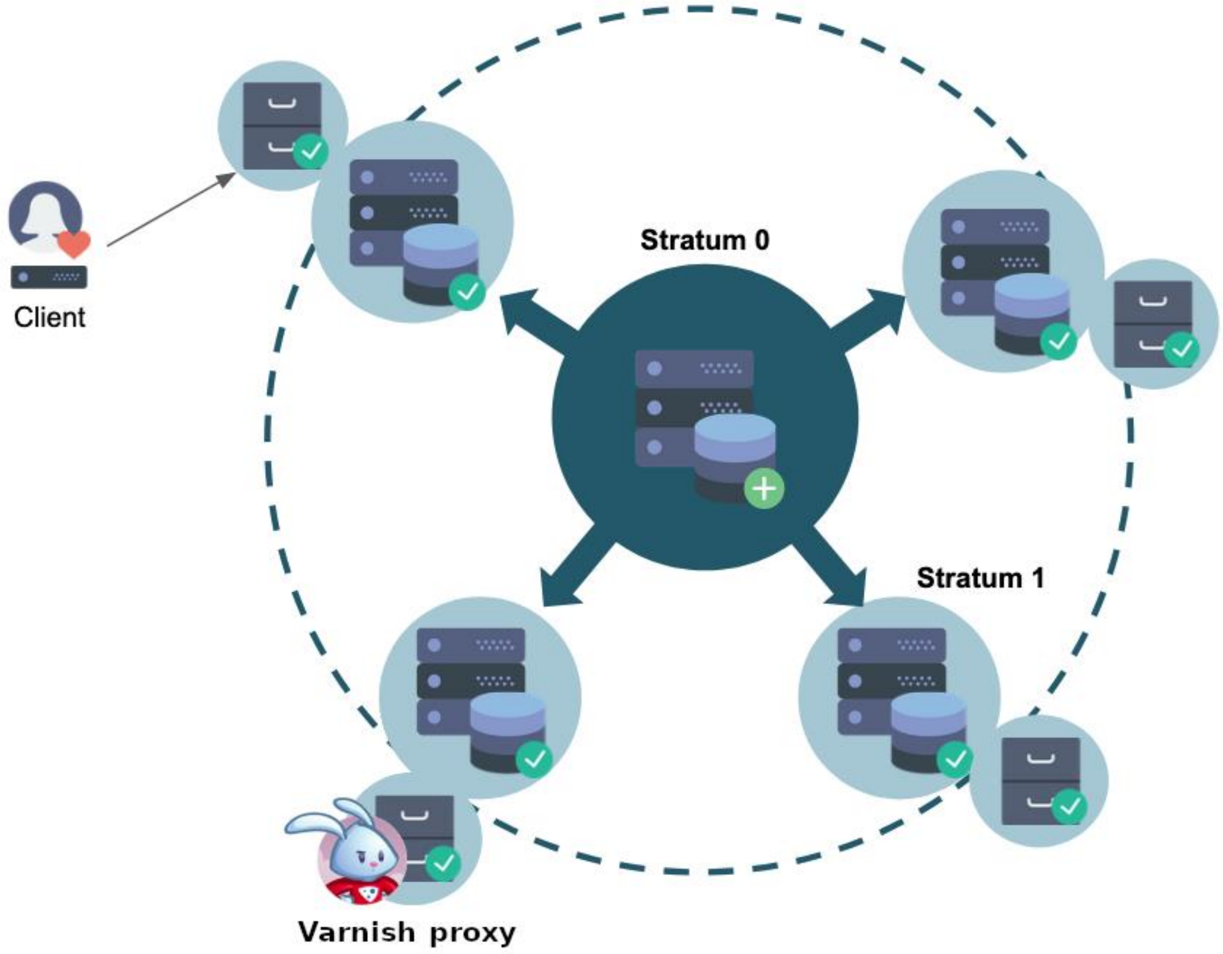
Varnish Software provides a set of packages:

- Repositories available for all major distributions
- Docker images available

Varnish provides very extensive and flexible configuration options through the Varnish Configuration Language aka VCL

CernVM-FS + Varnish
For HTTP Caching

Join our hands-on breakout session on Wednesday morning and get your own reverse proxy caching system up and running!



Large scale data processing with CVMFS

CVMFS Workshop, September 2024

Matt Harvey

HPC Production Engineer

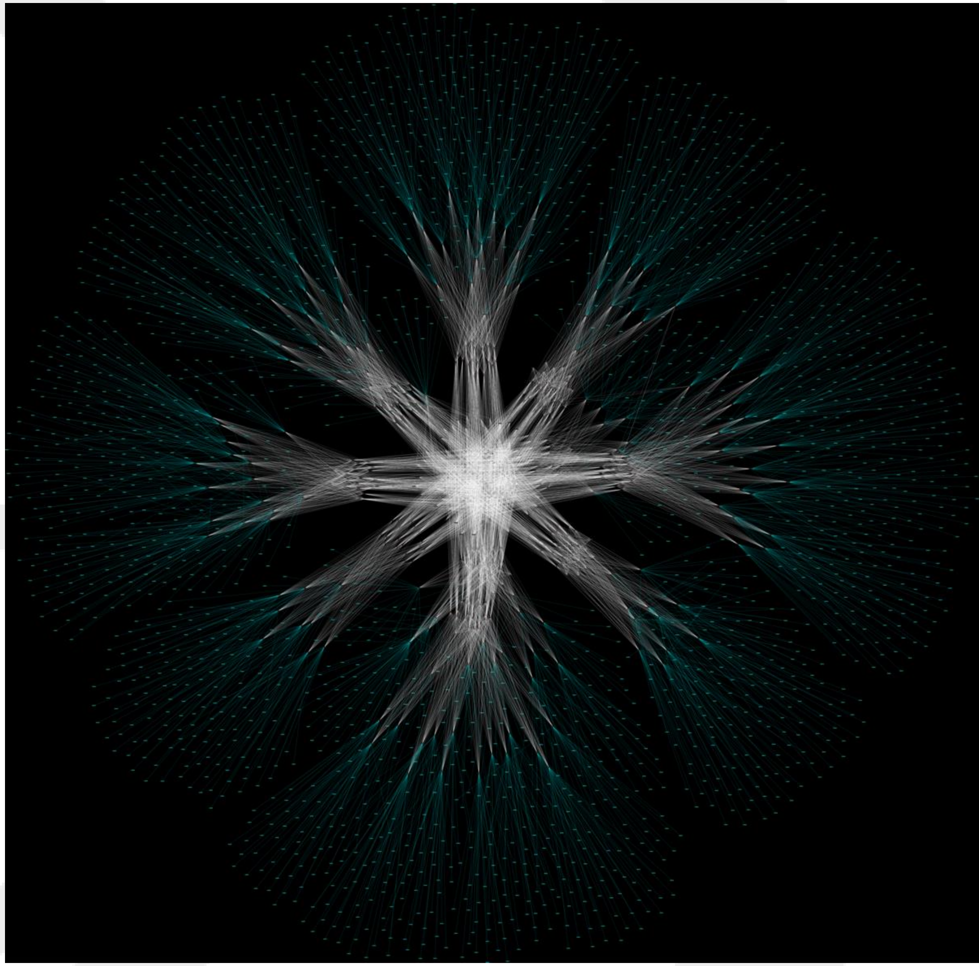
JUMP TRADING

- Privately-owned proprietary trading firm
- Applying cutting-edge research to global financial markets
- World-wide operations
  - offices across US, EU, Asia, Pacific

HPC at Jump

Jump's Research Environment (HPC / "The Grid")

- The platform where we develop and optimize trading strategies
- Technologically competitive with some of the largest publicly known research systems in the world
  - Thousands of servers
  - Hundreds of petabytes of storage
  - Fast network interconnects
  - Keeps growing: more hardware every year
- Sophisticated data-intensive and compute-intensive research workflows

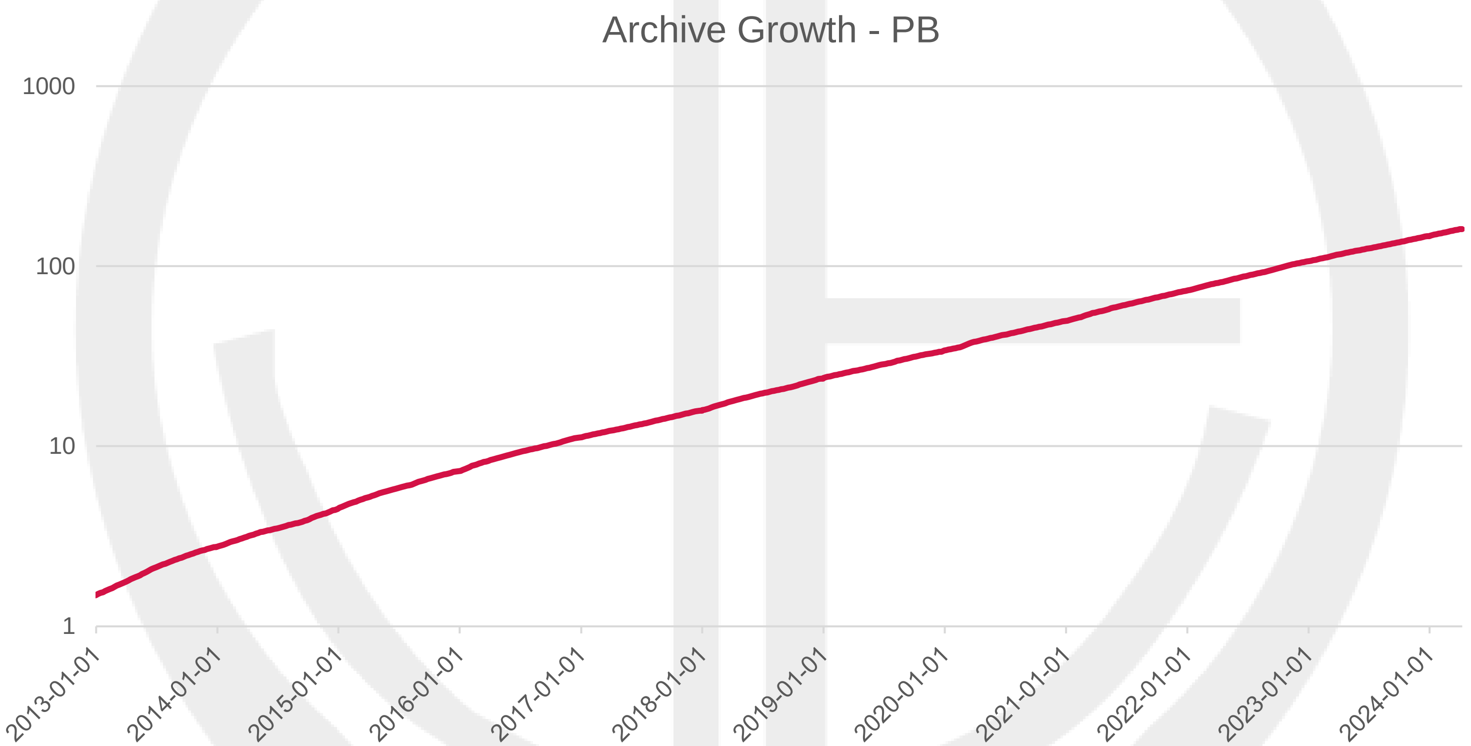


Fabric logical diagram
Image Credit: Olli-Pekka Lehto

Data Archive

- Realtime-updated repository time-series market data
- Contains raw data and derivative products for end-users

Ten Years of Archive Growth



Doubling ~2 years, currently growing at ~1.5PB/week
substantial day-on-day volatility

Data Archive Requirements

- Able to run existing work-loads unmodified
  - POSIX filesystem presentation
- Decoupled from HPC fabric / filesystems
  - Accessible outside of HPC environment, across clusters
- Able to accommodate >100x growth in capability
  - capacity, bandwidth
- Be temperature-aware
- Non-requirements:
  - Read-write mounts on compute nodes
  - Concurrent writes on the same file
  - Global consistency and file locking

The Tiered Archive

- Cloud object storage
  - Unbounded scalability
  - Globally accessible

- CVMFS presentation on client machines
  - POSIX presentation

The Tiered Archive

- Client
  - Latency O(1-10ms),
  - Bandwidth O(100Gbps)/node 1TB/s tot
  - much data reuse

The Tiered Archive

- Client
  - Latency O(1-10ms),
  - Bandwidth O(100Gbps)/node 1TB/s tot
  - extensive data reuse
- Cloud object storage
  - Latency O(0.1-1s)
  - Bandwidth O(100Gbps) tot
  - $/access

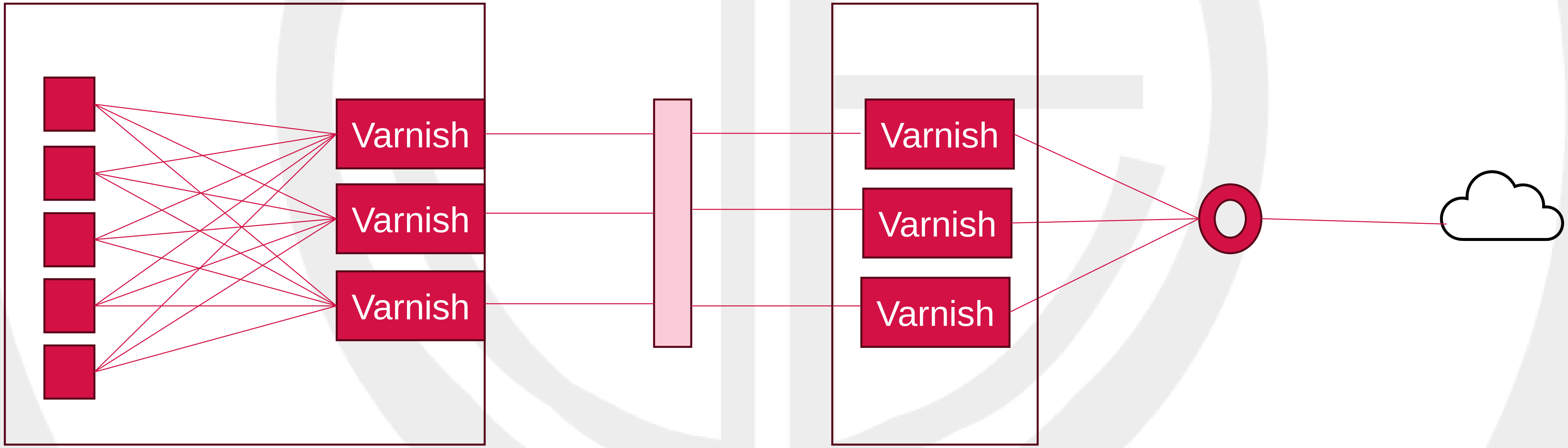
The Tiered Archive

- Client
  - Latency O(1-10ms),
  - Bandwidth O(100Gbps)/node 1TB/s tot
  - extensive data reuse
- Cloud object storage
  - Latency O(0.1-1s)
  - Bandwidth O(100Gbps) tot
  - $/access

Requires effective caching between cloud and consumer

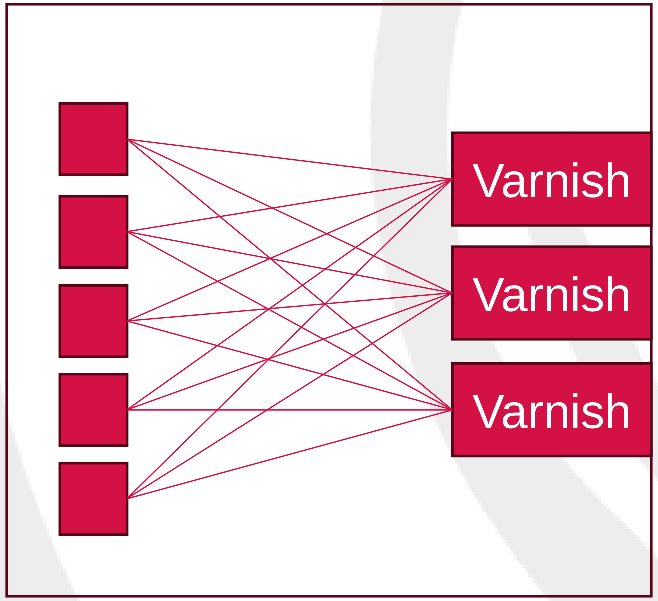
Cache hierarchy

- Uses several tiers of Varnish HTTP caches



Cache hierarchy

- HPC: nodes and ‘edge’ varnish servers all connected to IB
- Performance biased



High performance NVME storage
Infiniband and Ethernet networking
30+ GiB/s per server

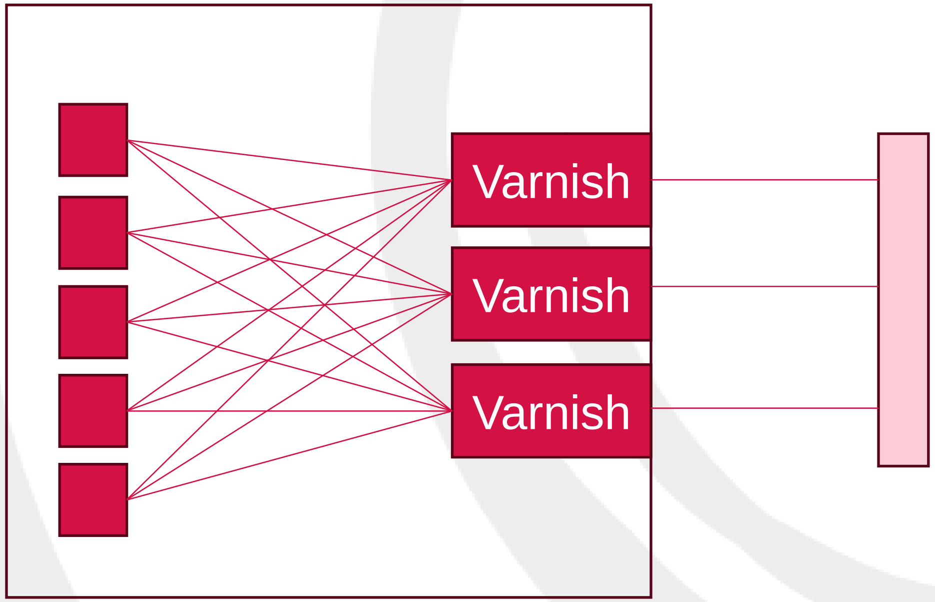
1-2MiB requests – byte-ranged

Scales horizontally

No CVMFS client data cache!

Cache hierarchy

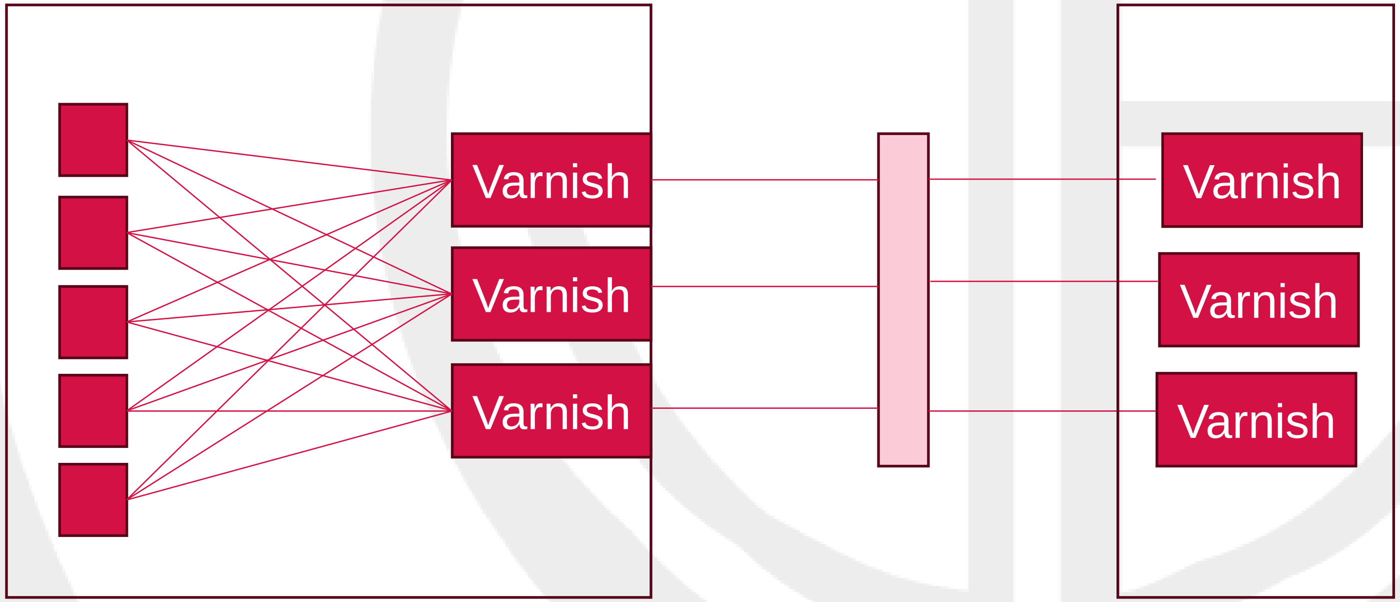
- Edge varnish servers connected to data-centre ethernet



Connected to data-centre ethernet @ 100Gbps

Cache hierarchy

- Core varnish caches connected to data-centre ethernet
- Capacity biased



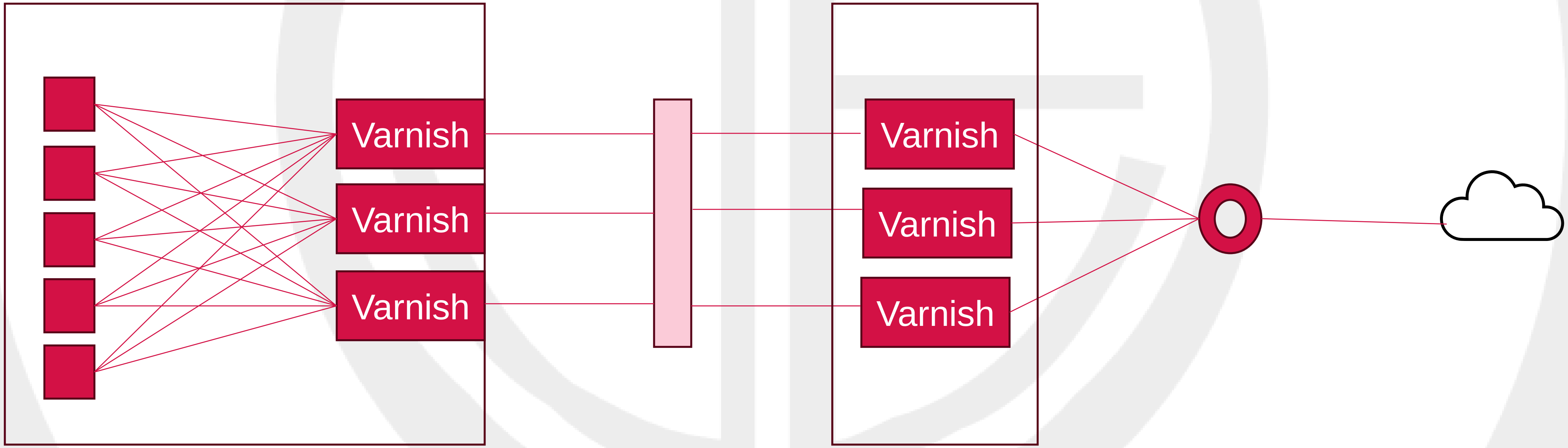
SSD and HDD storage
Ethernet networking

6MiB requests SSD
24MiB requests to HDD

Scales horizontally

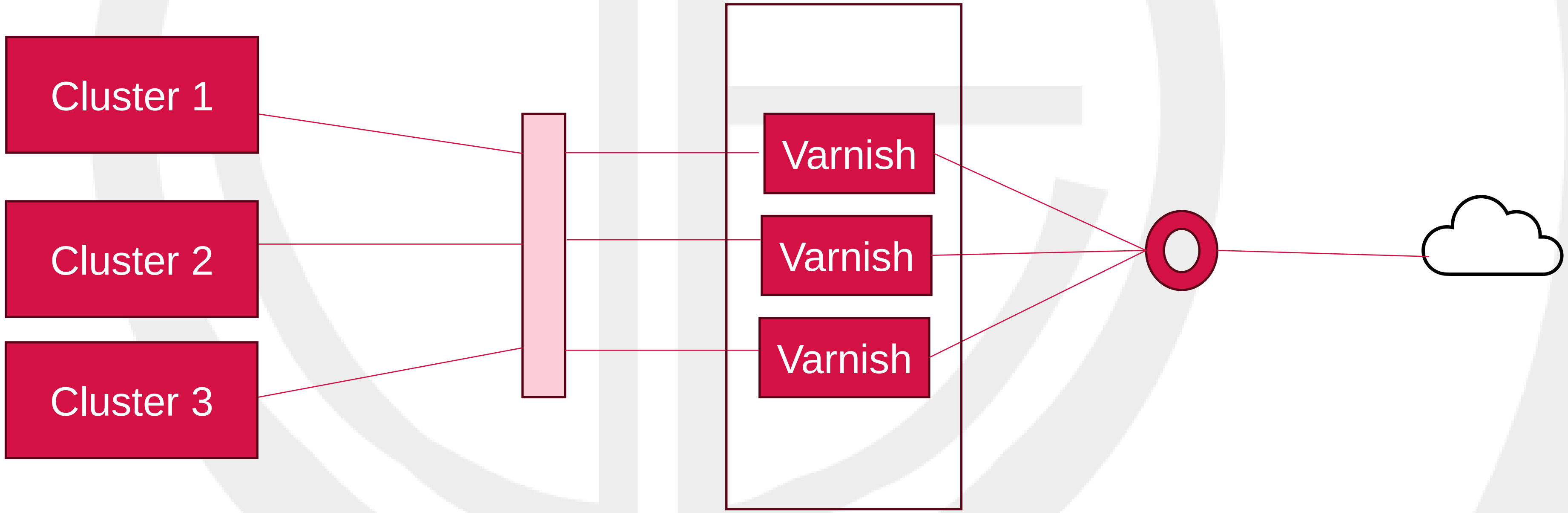
Cache hierarchy

- Core varnish servers connected to circuit link to cloud



Cache hierarchy

- Multiple clusters can use the core varnish tier
- Each cluster with its own set of edge caches



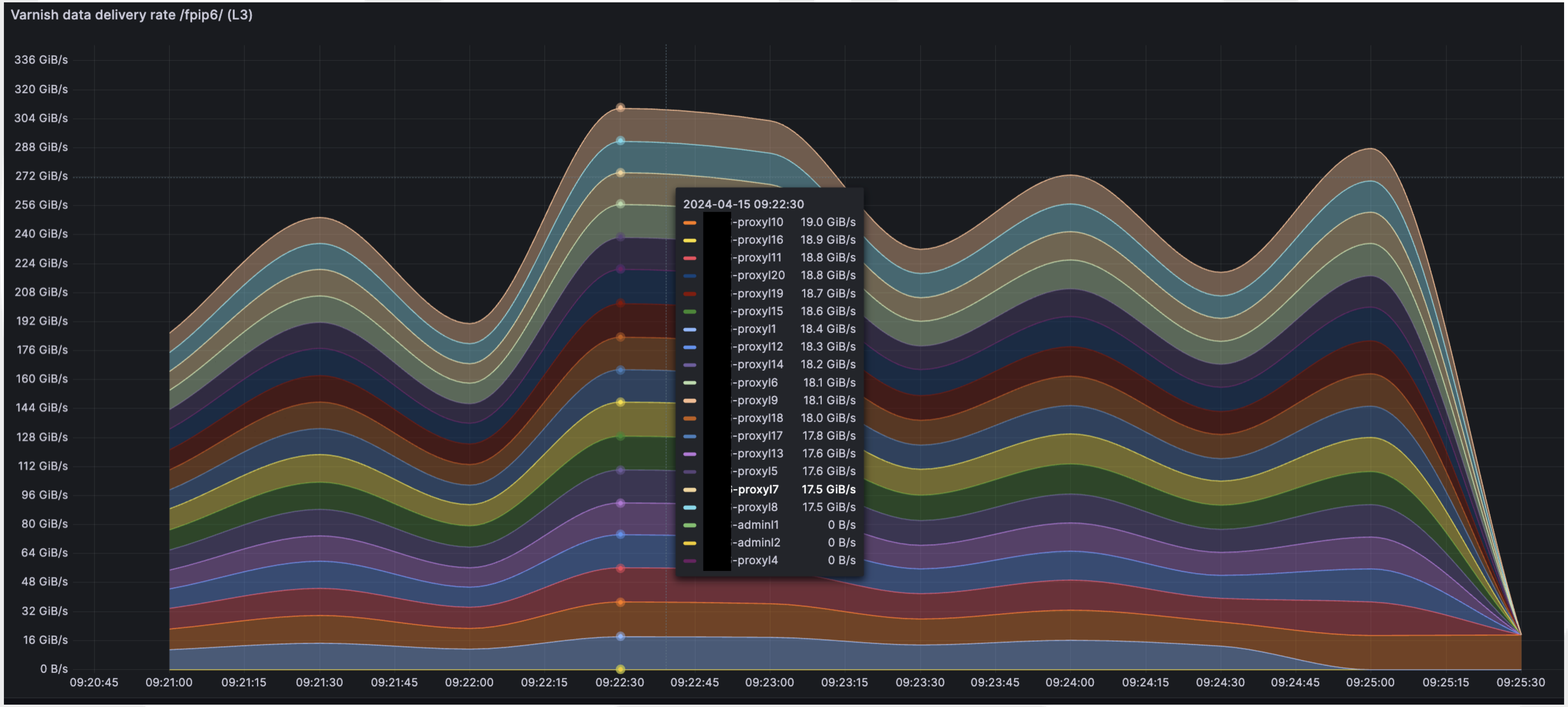
Sharding over Caches

- Shard to avoid duplication of objects across edge caches
- Rendezvous hashing:
  - For each cache instance:
    - hash key(URL + chunk offset + cache name) -> cache instance
    - Order hashes, try cache instance in corresponding order

Sharding over Caches

- Shard to avoid duplication of objects across edge caches
- Rendezvous hashing:
  - For each cache instance:
    - hash key(URL + chunk offset + cache name) -> cache instance
    - Order hashes, try cache instance in corresponding order
  - Allows graceful failover
    - no thundering herd, cascade failure
    - Traffic spread over remaining N-1 caches
    - invalidates ~1/N of each cache’s contents

Sharding over caches



Varnish features

Varnish Configuration Language (VCL)

- Programmable handling of all requests
- Fast – compiled
- Use to:
    - Route requests appropriate caches and cloud buckets
    - Prefetch data on a cache miss
    - Sign cloud requests

Varnish features

Varnish Configuration Language (VCL)

- Programmable handling of all requests
- Fast – compiled
- Use to:
    - Route requests appropriate caches and cloud buckets
    - Prefetch data on a cache miss
    - Sign cloud requests

Slicer

- Divide up requests into partial requests on aligned byte ranges
- Manages partial objects in the MSE cache to minimize prefetching
- Coalesces reply handling for concurrent fetches an object

Varnish features

Varnish Configuration Language (VCL)

- Programmable handling of all requests
- Fast – compiled
- Use to:
    - Route requests appropriate caches and cloud buckets
    - Prefetch data on a cache miss
    - Sign cloud requests

Slicer

- Divide up requests into partial requests on aligned byte ranges
- Manages partial objects in the MSE cache to minimize refetching
- Coalesces reply handling for concurrent fetches an object

- MSE
    - High performance storage engine that manages the in-memory and on-disk caches
    - Fault tolerance – disks can be brought in and out of service without stopping varnish

Optimising the first read

- The first read hits a cold cache hierarchy
  - Read back from cloud storage
  - High latency
  - Cost
- Prefill the core varnish tier
  - Separate request – can’t efficiently do write-through of arbitrary PUTs
  - Prefilling logic in VCL

Avoiding cache invalidation

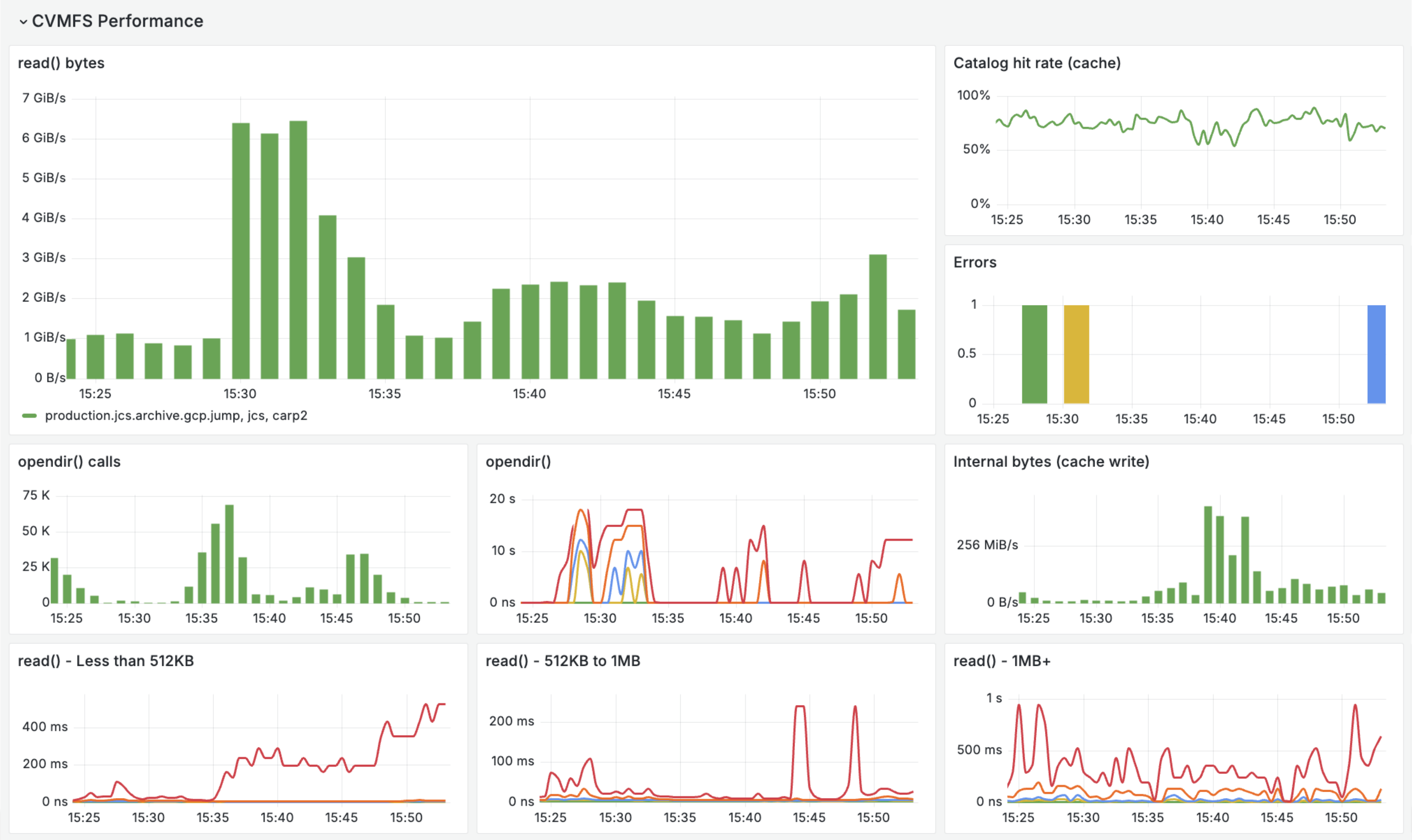
- CVMFS has direct mapping from filepath -> object key (external data)
- Changing a file requires cache invalidation
  - Expensive, difficult, race-prone
- Solution:
  - Object key = `path/.filename.<content shasum>`
  - Add file `path/filename` to CVMFS as:
    - File: `path/.filename.<content shasum>`
    - Symlink: `path/filename -> .filename.<content shasum>`
- Changing a file -> new dot file, flipped symlink (atomic)

Observability

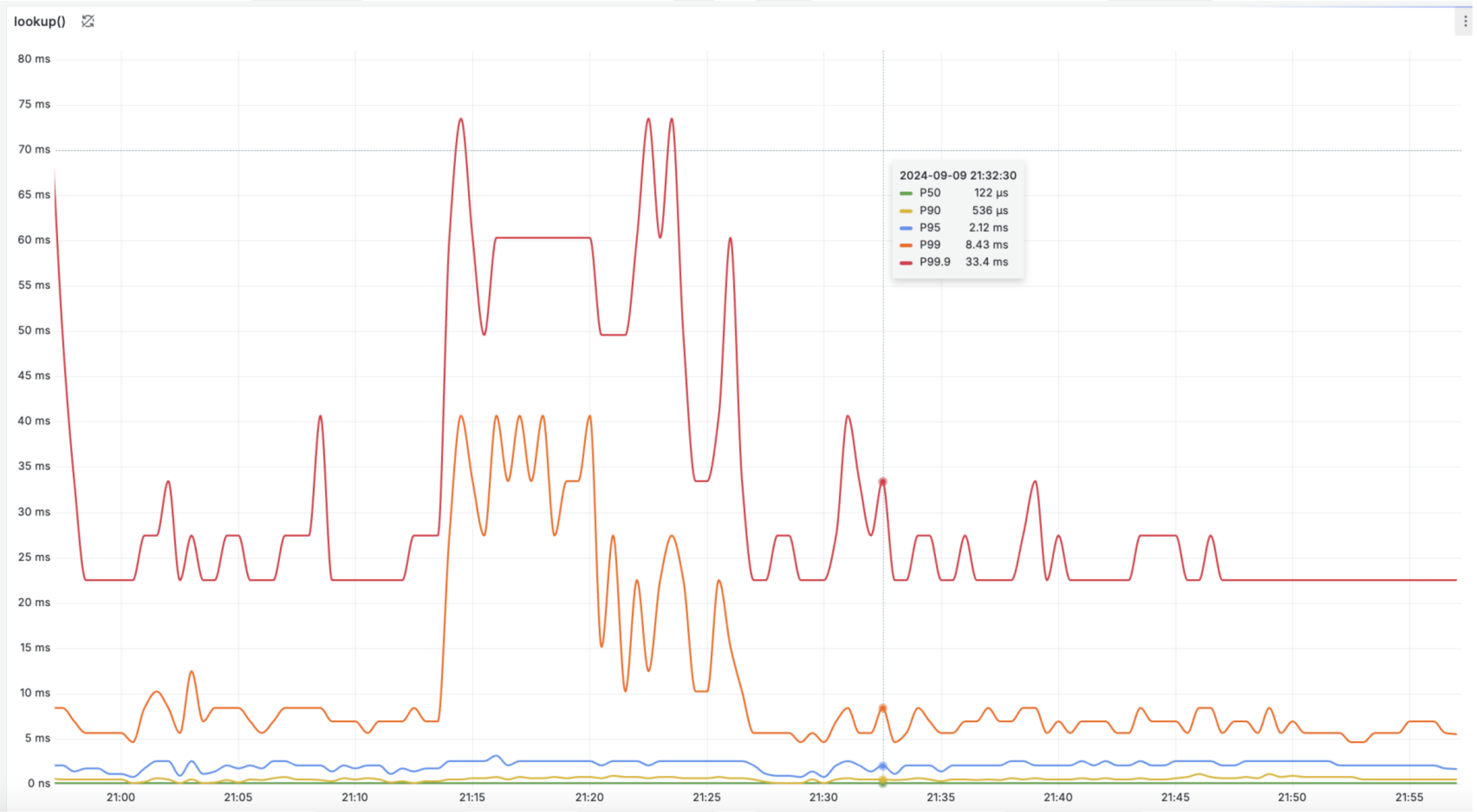
Observability

- End-to-end view of system performance
- Quicky detect and isolate defective services
- Tag all HTTP requests made by CVMFS
  - Add custom metadata – user, batch job id, etc
- Ingest Varnish logs into time-series database
  - Hard: O(10^4) requests/sec per server
- Export all CVMFS statistics and latencies

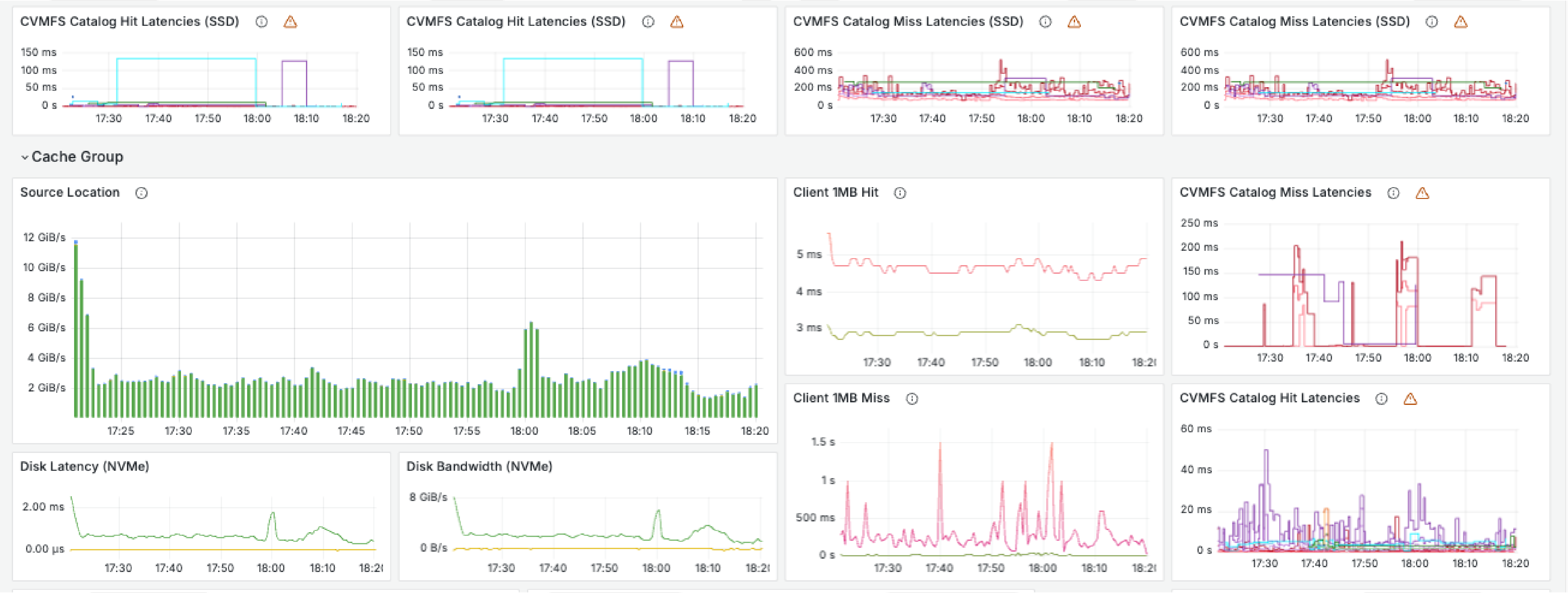
Observability – Client Performance



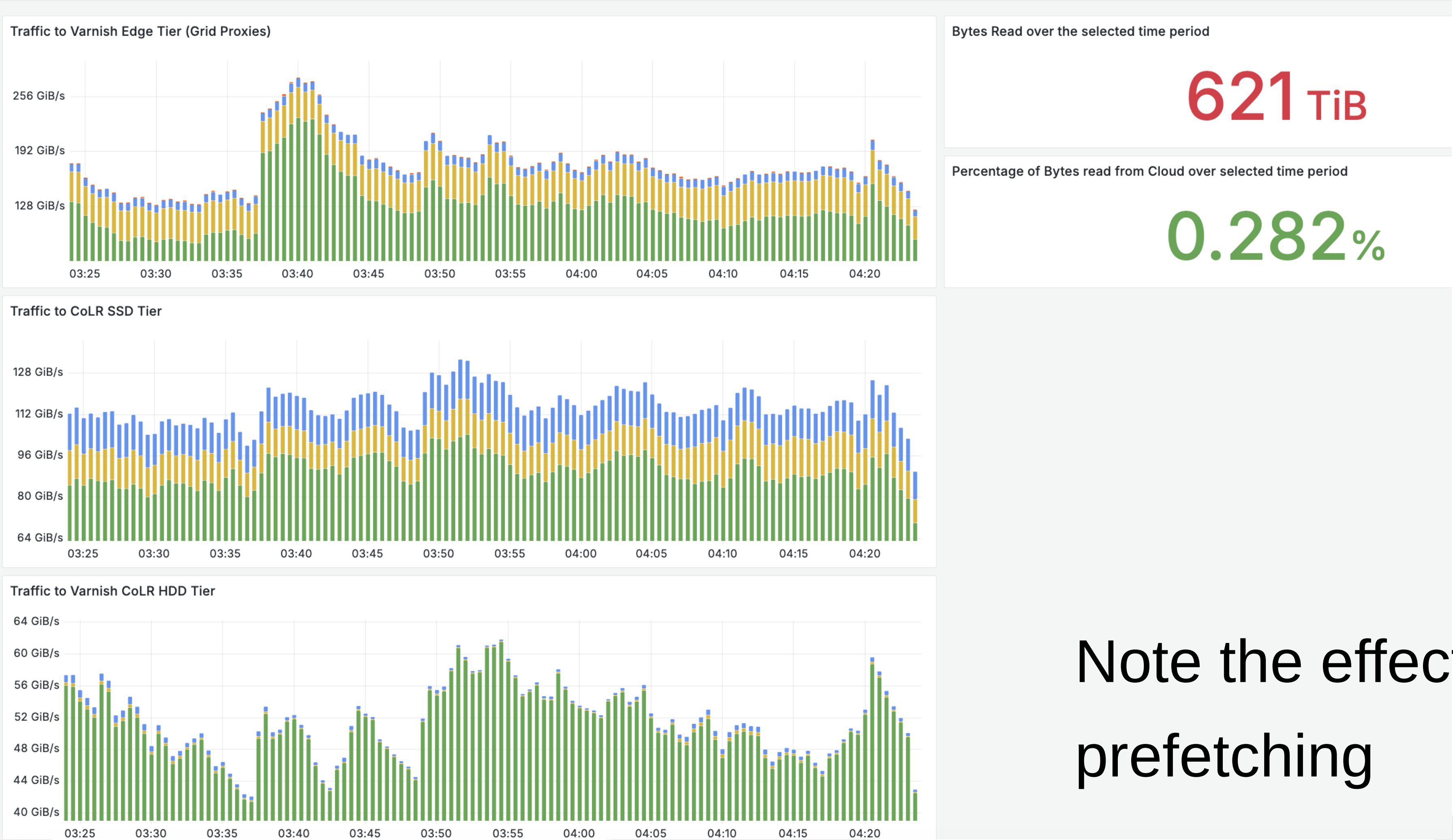
Observability – Client Performance



Observability – Cache instance performance



Observability – end-to-end performance



Note the effect of prefetching

Thank you!

Questions?

mharvey@jumptrading.com