Getting the most out of XCache

MANIAC LAB

Ilija Vukotic
UChicago

27-31 March 2023, IJS, Ljubljana

ATLAS EXPERIMENT

ATLAS cache usage and requirements

Two main use cases:

- Virtual Placement
  - Basically a CDN made in such a way that minimal changes were needed in existing components (Panda, Pilot, Rucio...)
- ServiceX
  - Single, performant and reliable XCache is sufficient.

Our requirements

- Subfile caching
- Persistified caching
- Performance
  - Thousands of concurrent clients per xcache node
  - 40+ Gbps per node
- Detailed monitoring
- Long term support
- Free

Motivations

Now that XCache network is working and we have experience with VP, we can step back and see what it took and how it could be made better.

The difference between a single node cache and a CDN caching can’t be overestimated.

We looked at possibilities for HTTP caching, tested and evaluated three options: Squid, Varnish, Nginx and Apache Traffic Server. This gave us experience of how modern CDNs are made, what features are important.

Fishes

Everyone gives you free caching as long as you are a small fish. Big fish features are paid for.

Squid - no multithreading so no big fish is using it. Also 3-6x less performant.

Varnish - a big fish gets: cache persistence, range requests, ssl, centralized configuration and monitoring tools, Massive Storage Engine…

NGINX - a big fish gets: active health checks, cache purging api, PLUS api for dynamic reconfiguration, JWT, SSO, build in dashboards.

My wishlist

Disk failure handling

Docker image, docker compose, k8s deployment, helm chart

No burn-in test

Liveness / heartbeats

Configuration reload at runtime

Client/origin/path management at config/runtime

JWT, SSO

Multiorigin, multiprotocol accesses

Modern monitoring

Self configuring

CDN support (Centralized configuration, monitoring, path handling, SSO).

Reasons I

Disk failure handling

More often than not, we run on unreliable hardware. This was the main reason we stopped running at Prague. With 80+ 1TB drives, failures were too frequent. It is rather basic requirement and should be possible to do easily and inexpensively (check each disk once per minute, write file to it/read it out). This is offered by every http caching proxy.

Docker image, docker compose, k8s deployment, helm chart*

If your product is a well rounded, easily configurable thing, there should not be a need for ATLAS/CMS/XXX images. We would use the official image and just add our configmap. It would make things much more repeatable, testable, stable. You control base OS, mallocs, limits, etc...

You could provide basic configurations for several important use cases.

The best thing is you don't have to start from scratch. Both me and OSG did most of the work.

*** Yes, I understand that there are still people that refuse to use docker, singularity, k8s, helm,... luddites, vi and pine users. But the majority of the world is moving.**

Reasons II

No burn-in test

External testing is made more complicated by cache burn-in. One needs to invent new files in order to test external accessibility.

My testing/central testing would be much simpler if a certain kind of file would be expunged from cache(if it is there) every 60 seconds. You could also organize hosting of a few of these small, world accessible files on a http/root endpoint.

Liveness / heartbeats

Liveness is a rather complicated thing. It can mean:

- Something is listening on a port
- Service is responsive (not overloaded)
- Service actually does what is supposed to do.

I need to know all of the above. Should be both push/pull. Heartbeats need to send basic data about instance state. It should be REST JSON.

Reasons III

Configuration reload

It would decrease downtimes, simplify management, speed up debugging, your could easily change log level, etc...

Client/origin/path management

It happens that an origin server gets messed up, then the client is left to wait indefinitely. I need to be able to ban that origin (with error returned or simply redirecting to it).

Some clients are not welcome to use my xcache so need to ban them too.

I found some paths gain nothing from caching. Would like to be able to immediately redirect these to origin.

All of this at runtime.

Reasons IV

JWT

This is self explanatory.

X509 is dead.

And I am not asking for NFT based authentication. Or Web3 wallet auth :)

Multi-origin, multi-protocol support

While paths like root://cache.node//root://origin/path/file are easily readable, it would be great if we could have more flexibility and support multiple origins and allow for multiple protocols. Something like:

root://cache.node:1094//?ml=compressedmetalink

Reasons V

Modern monitoring

It is a bit annoying that I need to run a UDP2TCP decoder to get a reliable monitoring. Others are doing much more complex monitoring chains.

Push or pull, Prometheus, ELK or any other way, I don’t care as long as results are trustable and user doesn’t need to run extra services.

Self routing cluster

Some use cases would benefit from a self routing cluster.

If I deploy on k8s, I want a HOA (Horizontal Autoscaling) cluster based on load/latency/number of connections.

But requests must be hashed in an optimal way and redirected to the right node in the cluster.

It can be accomplished with an NGINX reverse proxy but it would be nicer if done in xcache itself.

Reasons VI

CDN

- Active external testing
- Central everything
  - Configuration - maybe from a GitHub repo, similar to OSG topology
  - Integrated central monitoring out of the box
  - Multi level cache support
  - Detecting and working around issues with network/origins/congestions.
  - Optimal caching path calculation service
    - For a given client and origin file calculates optimal caching path.
    - For a given origin file calculates optimal xcache endpoint.
  - SSO

Conclusion

Some of these suggestions would be great addition even without going all the way.

Most of it was already done by different people but in disparate ways.

We could make all of these as a single separate service (just don’t make it cmsd) so core code is not touched.

If you go all the way in, you could even sell it. Make most of it free and make CDN part pay-for but cheap (for the start). And free for academic users.