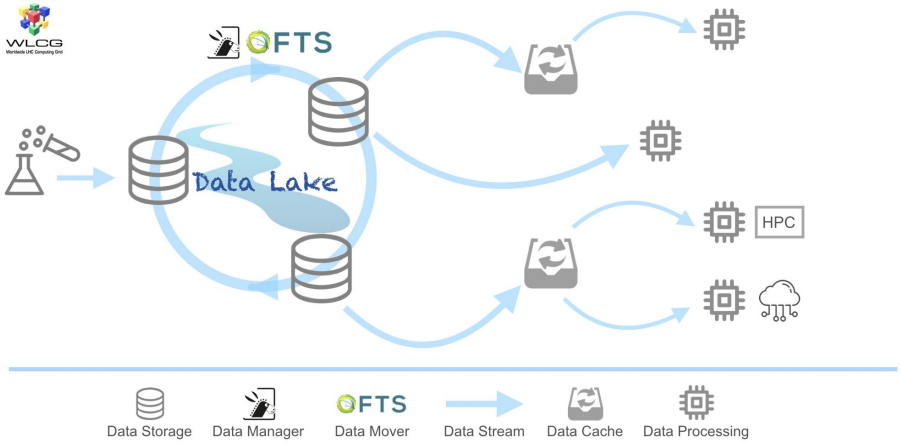
Virtual Placement

for cache usage optimization

Ilija Vukotic
University of Chicago

DOMA Access 2020/04/21

Caches



- Reduce WAN traffic
- Reduce latency / increase CPU eff.
- Cost less to run
  - In terms of person-power
  - In $/TB

Issues:

- They work only if files are accessed multiple times. Cache efficiency expressed as cache-hit-rate. Unlike Netflix, HEP data is not very frequently reused.
- Current job scheduling of jobs “to where the data is” does not work for caches as caches would never get populated.
- We still use multiple protocols to move data around.

Caches - continued

Several ways to deploy them:

- Consider WN local disk as a tiny cache. We had that system (pcache) in use but now had to be reimplemented. A lot of testing needed. Expected cache hit rate 15%. Still valuable as it costs nothing.
- Small site (in terms of CPU) without pledged storage, that is “far” from a large storage site (in terms of distance and/or throughput). Relatively easy to set up. Hard to keep running. Making these sites run only EVGEN is simpler solution.
- Large site/HPC without pledged storage. Can not rely on one (closest) site for all of its data. Need a high cache hit rate to reduce WAN need and still have high CPU eff.

VP - Virtual Placement

1. On registration in RUCIO every dataset gets assigned to N sites in the same cloud.
2. Assignment is done randomly where each sites probability to get the dataset is proportional to fraction of CPUs that the site contributes to ATLAS.
3. Datasets are not actually copied at any of these N sites but only exist in the “lake”.
4. Panda would assign job that needs as an input this dataset to the first site from these 3. In case site is in outage it would get assigned to the second site from the list. Once job is there it would access the data through the cache.

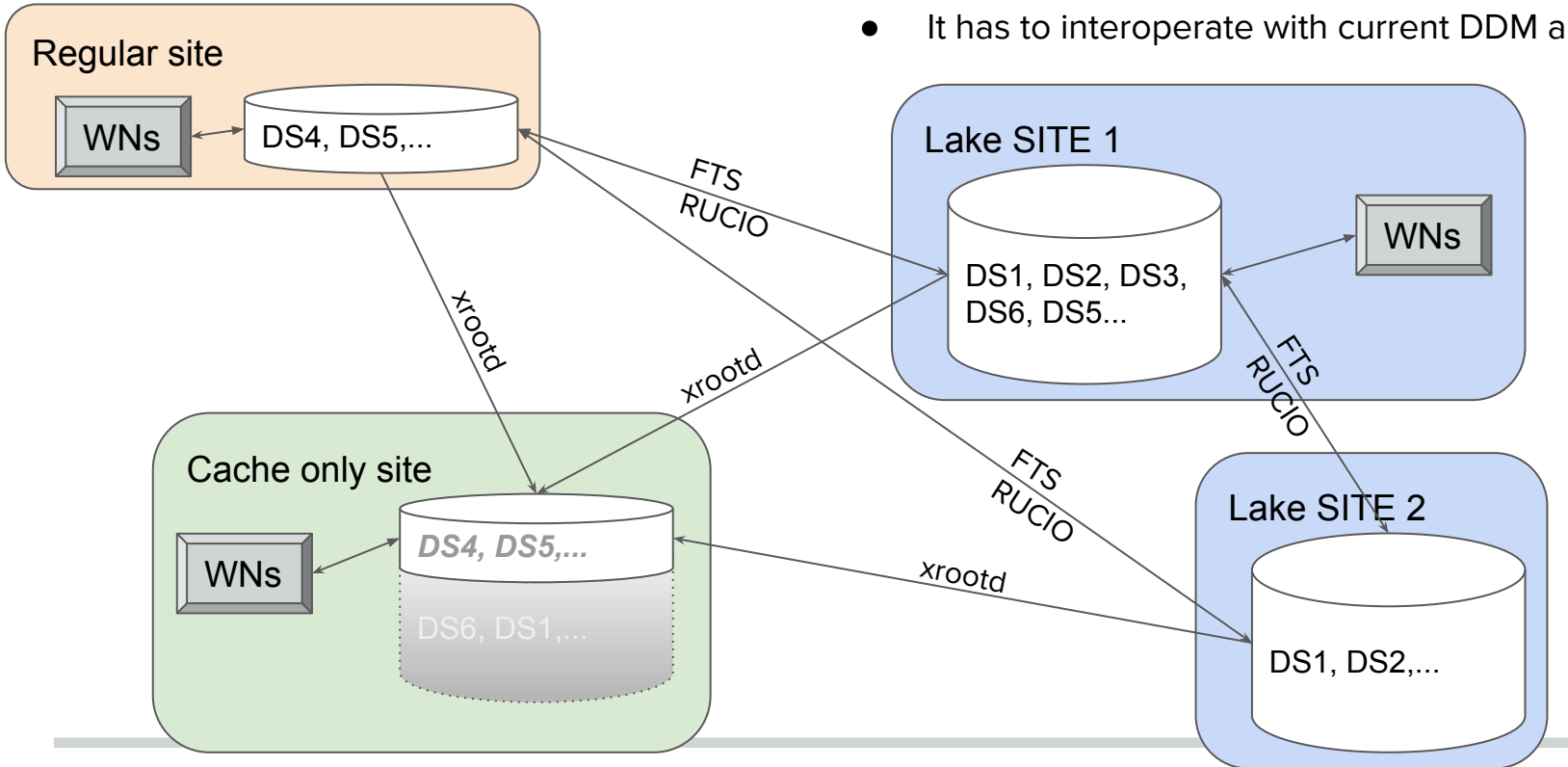
This way we get:

- Very high cache hit rate
- We could use a large fraction of the existing storage as caches
- Reliability
- Adding/removing site would be easily done from a central location
- Less stress on FTS, fewer RUCIO rules (neither needed at xcache-only sites)

VP in practice

DSX - primary copy
DSX - virtual copy fully or partially cached data
DSX - virtual copy - not there at all until needed

- Lake is still not there. We use all the sites in its place.
- It has to interoperate with current DDM and WFMS.



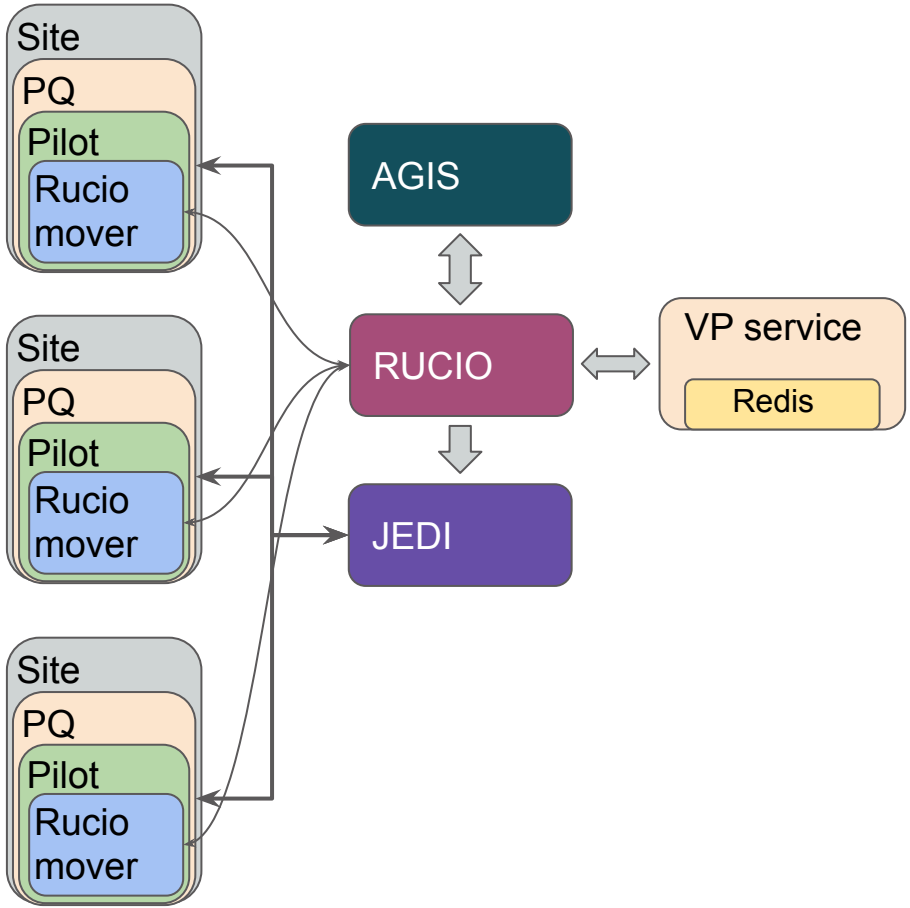
VP service

Parts:

- an engine doing assignments
- REDIS DB to memorize Virtual Placements
- REST API to configure, access placements.

Currently external to RUCIO.

Once proven useful, should become a part of RUCIO.



IRL Tests - configuration

XCache Original setup

- MWT2 16 x 12TB (JBODs)
- AGLT2 8 x 8TB (JBODs)
- Prague 2x44TB, 2x37TB, 2x19TB (RAIDs)
- BNL 60TB (NVMe)

VP settings:

- 2k/250k datasets to MWT2 and AGLT2
- 2k/250k to Prague.
- 2k/250k to BNL.

XCache NEW setup

- MWT2 16 x 12TB (JBODs)
- AGLT2 **12** x 8TB (JBODs)
- Prague 2x44TB, 2x37TB, 2x19TB (JBODs)
- LRZ-LMU 20 (JBODs)

VP settings:

- 2k/250k datasets to all

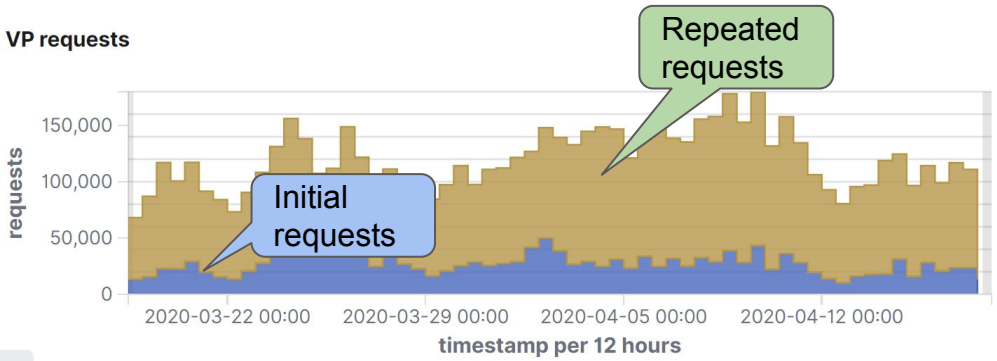
Issues

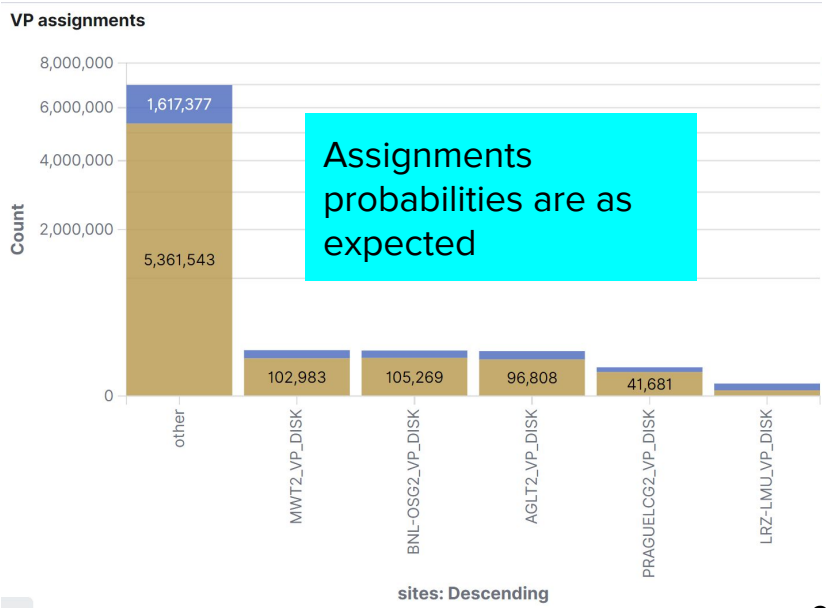
- XCache stability ✔️
- Bad origins ☑
- Which queue can get job that can use VP replicas ✔️
- Copy-to-scratch handling ✔️
- Unavailability of data in origin DDM ✔️
- Sites not having xroot as a primary protocol for WAN reads ☑

Does VP service work? Yes!

VP service has been instrumented so it reports all requests and replies to ES@UChicago.

- ~3.5 Hz requests
- A lot of repeated requests in avg. 7.3 times per DS





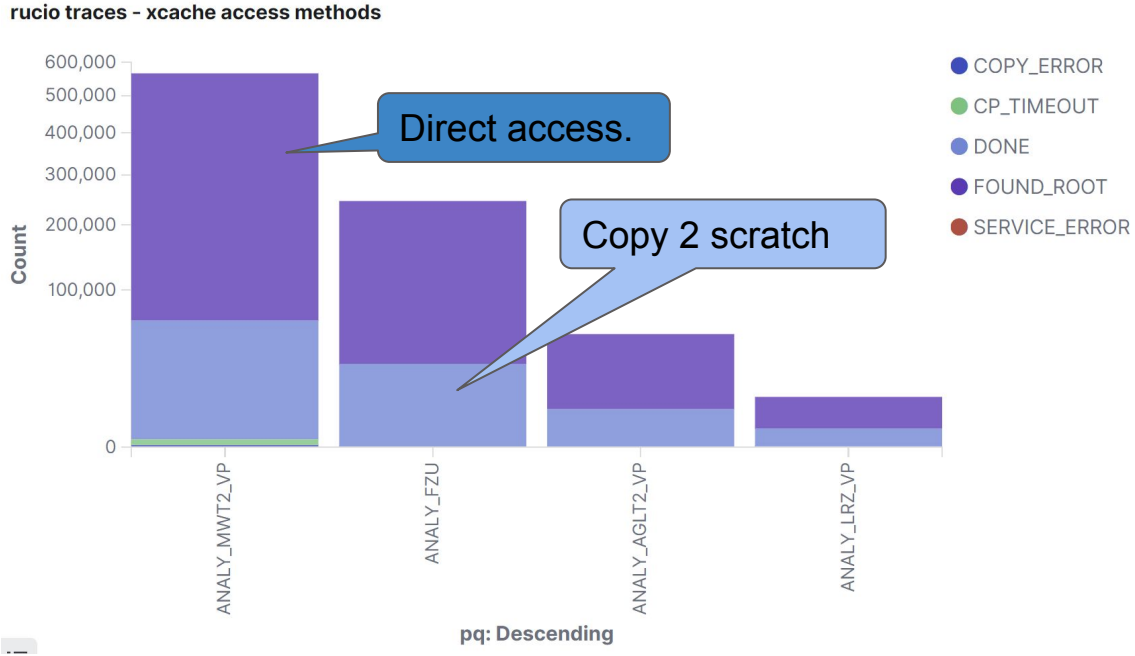
Does scheduling for VP works?

We collect rucio traces.

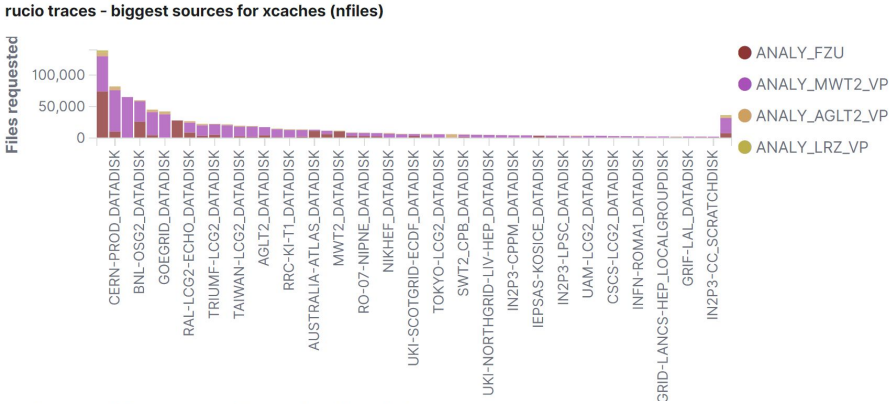
Look for paths (url:root*root\:*).

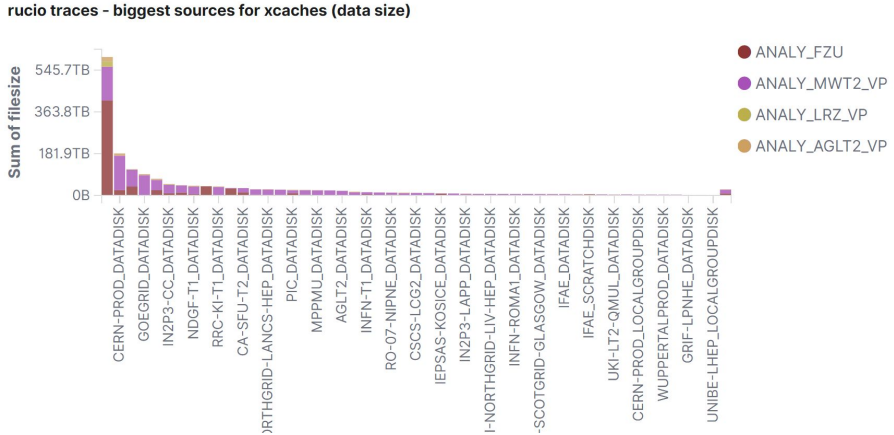
Week ago we discovered bug that made only one file access per job visible.

These numbers are very under reported, but ratios are correct.



VP sources

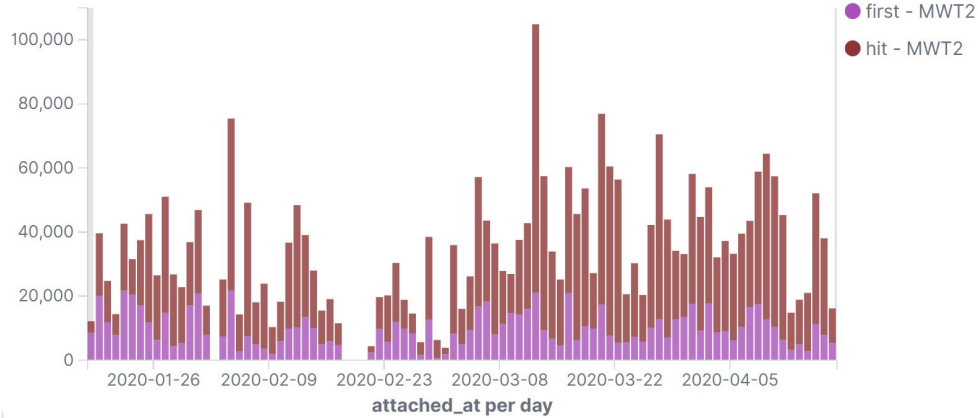


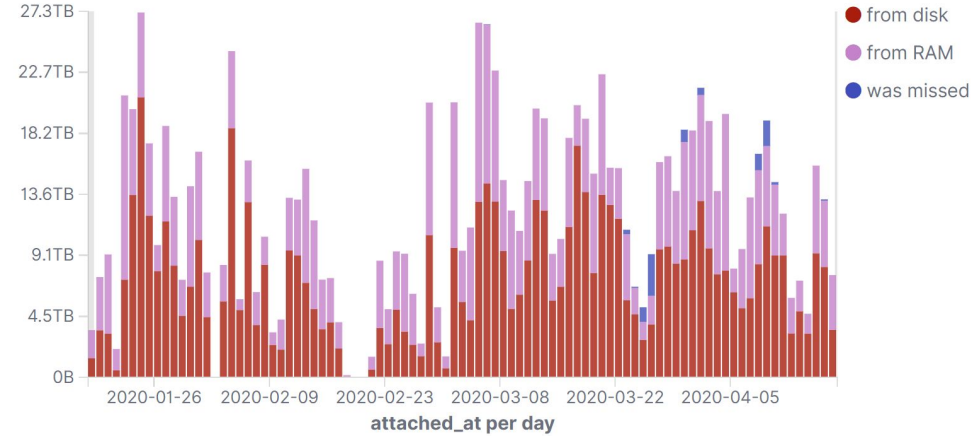


Sources are mainly large sites

Need serious development to assure best copy (closest) is returned.

XCache reports





MWT2 in last 3 months

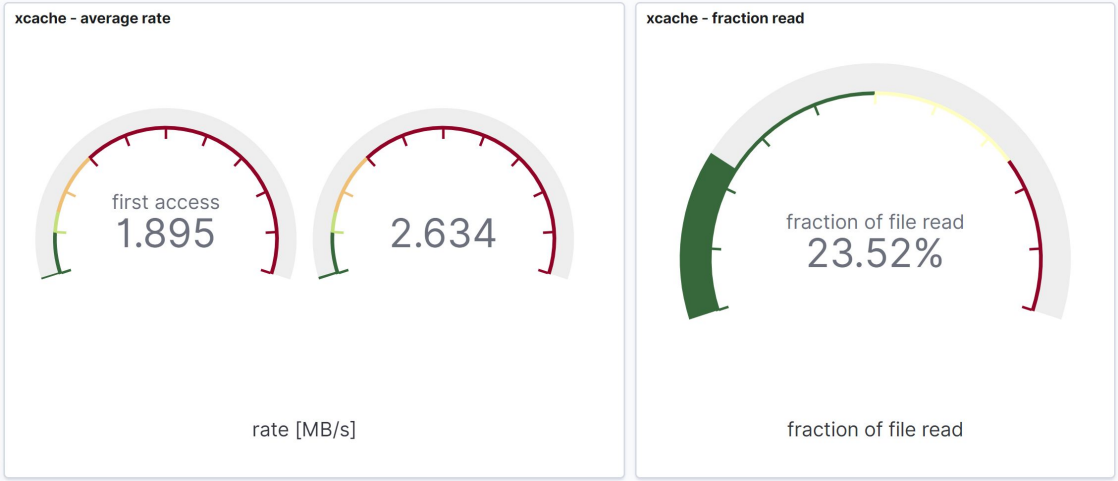
Had 3 full cleanups.

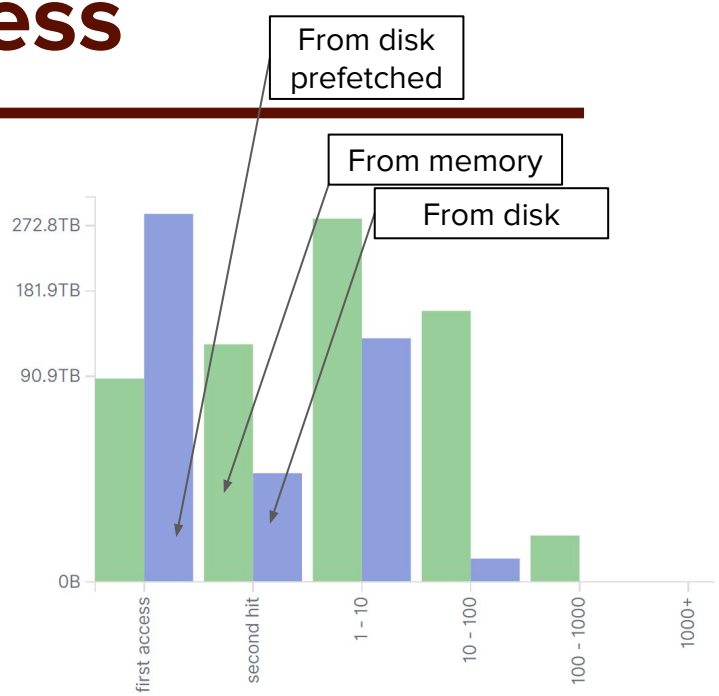
71.4% cache hit probability.

65.5% data delivered in following accesses.

59.6% data delivered from xcache disk.

MWT2 - rate and sparseness





In average ~170k files in cache.
Fill factor of the files in cache is ~72%.
Part of the jobs do copy2scratch so these have ~100% fill.

Subsequent accesses add more data to cache.

Cache comparisons

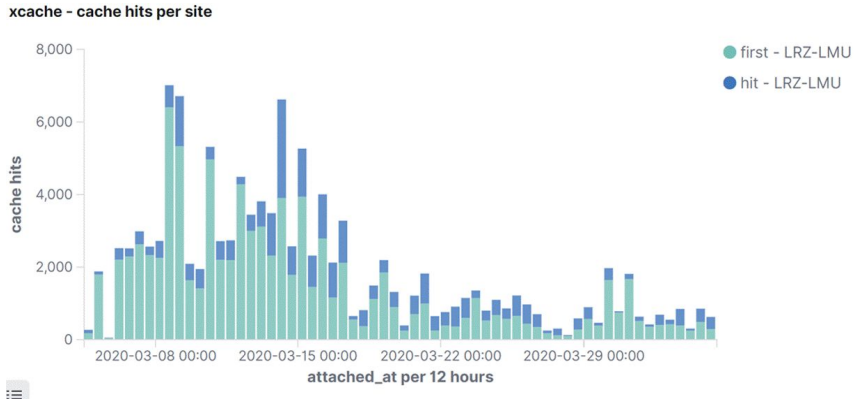
last 30 days

Site	access	files accessed	Delivered from disk	Delivered from RAM	Avg. rate [MB/s]	Avg. read sparseness	Cache hit rate
MWT2	first	295,076	28.4TB	105.9TB	1.858	33.23%	76.5%
	following	960,577	211.7TB	55.7TB	2.218	14.13%	
praguelcg2	first	65,443	2.9TB	18.3TB	2.677	32.87%	81.3%
	following	284,989	92.8TB	18.5TB	3.192	13.34%	
AGLT2	first	91,280	4.8TB	25.8TB	4.456	63.06%	41.0%
	following	63,418	9.1TB	9.4TB	4.944	16.84%	
UKI-SOUTHGRID-BHAM-HEP	first	13,977	7TB	16.9TB	46.609	81.69%	30.6%
	following	6,151	8.5TB	697.4GB	73.398	111.92%	
UKI-SOUTHGRID-CAM-HEP	first	0	0B	0B			100.0%
	following	614	2.5TB	0B	100.964	99.35%	
RU-LAKE	first	2	2.5MB	305.5MB	4.278	8.71%	99.5%
	following	376	954.9GB	0B	469.513	131.92%	
BNL-ATLAS	first	6	47.6MB	73.9MB	0.092	9.12%	25.0%
	following	2	32.2MB	101.9KB	0.207	3.70%	

- VP caches can be very efficient.
- AGLT2 cache shows bad situation: small disk, low number of jobs using it, copy2scratch.
- Non-VP caches show low utilization and low efficiency.

LRZ-LMU xcache

- Big site with well supported xcache
- Worked in direct mode until 7 days ago.
- Now starting from scratch in VP mode.
- Before
  - ~30% cache hit rate (files & data)



- Now
  - 50% file cache hit rate
  - 75% data delivered from cache

Future

- XCache developments
  - Update to version that supports CRC once it’s ready
  - Fix for ROOT TChain:Add
- Origin fixes - constant load.
- VP
  - Find and understand all the things that change load on the caches.
  - A large site served only via xcache
  - Deploy in front of an HPC
- Far future
  - Multi node xcache support
  - Moving VPservice into Rucio
  - **Adaptive caching instead of LRU currently used**

Adaptive caching

- By “pinning” datasets to caches (VP) we solve most of the low cache hit rate problem. That can get us to ~80% cache hit rate with the Least Recently Used cleanup model (LRU).
- If we could gain 5% by changing caching model, that would reduce WAN traffic by 25%, which is a lot.
- All kinds of schemes proposed, some even tried.
- Most popular idea is “we know what we are using most”.
  - We don’t really know, up to now all assumptions proved wrong.
  - What is popular changes while hard coded rules tend to stick.
  - Would require continual effort on analysis and re-tunning. Impossible to do for all the sites/panda queues.
- Naive approaches trying to detect “popular” datasets failed.

Why do we need it now?

- For now, we really don’t… first we need to:
  - get xcaches integrated in regular operations
  - gain operational experience
  - characterize performance and effects on job eff, wan throughput etc.
- But we need to start making it now:
  - RL is not something you do in a week
  - Training takes time
  - Integrating it into xcache would take time
- Can be useful for other DDM operations eg. select files to move to tape.

Reinforcement learning

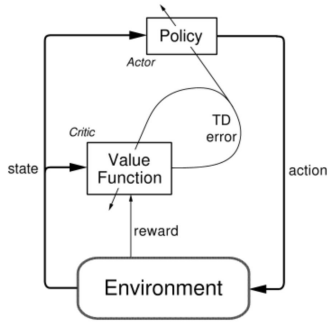
An **actor** gets trained **once** or **online**, by an **environment** that gives a **reward** for “good” **actions**.

Used for everything from Chess, Go, to Hide & seek.

Specially useful for situations where not all info available and multiple actors influence the system simultaneously, thus requiring cooperation (eg. multi-level cache actors, Rod and Ivan).

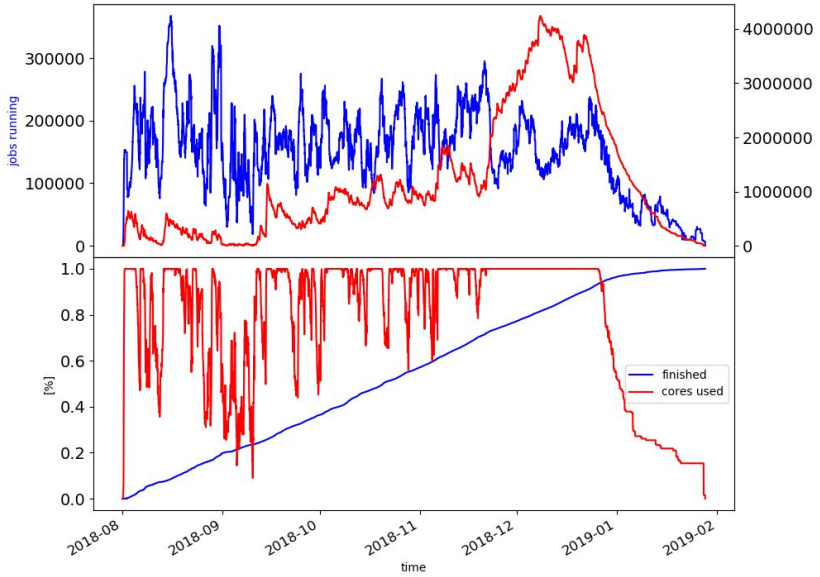
Plan

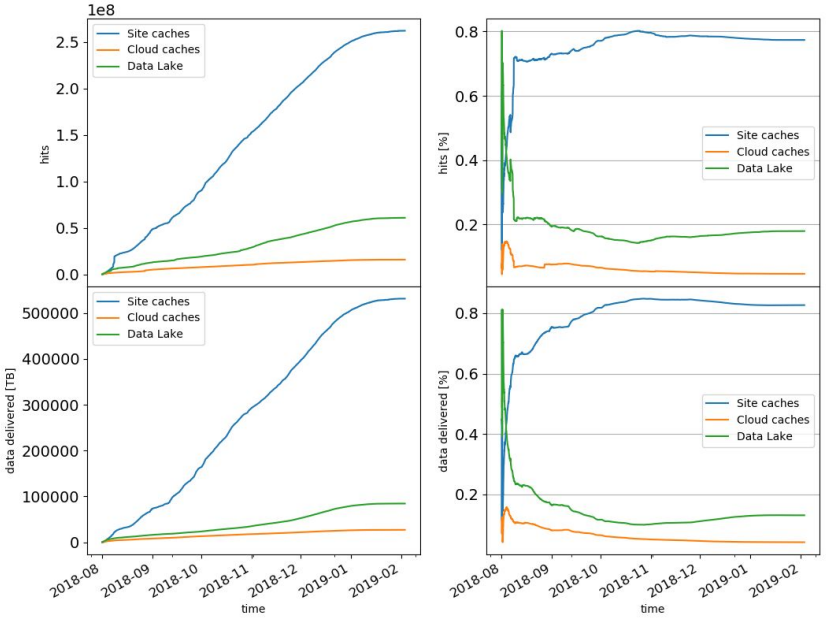
- Get data - already in ES, extract it. ✔
- Preprocess data (tokenization of filenames, dataset names too). ✔
- Create environment - basing it on OpenAI gym environment. Two environments:
  - discrete action (cache/not cache) ✔
  - Continuous action (predicting probability that a file is already in cache/ should be cached)
- Train different actors
  - Deep-Q network (DQN) or Dueling DQN for discrete action env. ✔
  - Actor-Critic model for continuous action env
- Compare them with LRU



Appendix

VP - expectations





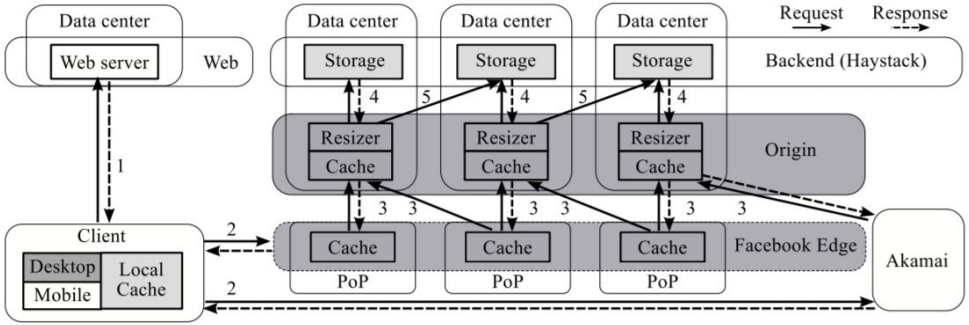
Resources would be fully used. TTC would be the same as now. Cache would deliver 80% of data. Throughput at caches would be reasonable.

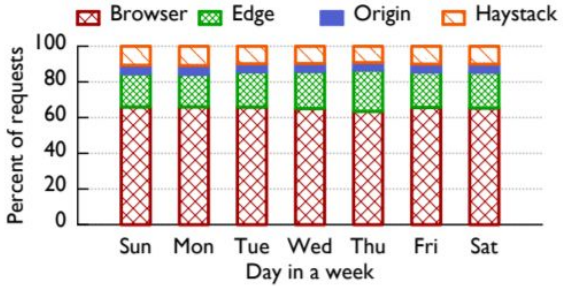
VP to two sites of same cloud
One Data Lake (has all the data)
Each cloud has XCache (100TB/2k cores)
Each site has XCache (100TB/1k cores)

FB way

Their problem is much bigger: much more data, popularity changing much faster, it has to deliver data in seconds, geographical distribution of producers/consumers much more spread.

Solution: network of 3 layers of caches. Each cache in each layer is redundant. On each file access caches in each layer repopulated (if needed).





https://scontent-ort2-2.xx.fbcdn.net/v/t1.0-9/89280660_3432061820144419_3933167136544915456_n.jpg?_nc_cat=104&_nc_sid=110474&_nc_oc=AQlduvrj3RFheT3n1eJs-Ed6Oh9JjuWDFKhEtG27H4HGP-YiR7QeUrvuGCZ053QxPJq&_nc_ht=scontent-ort2-2.xx&oh=c73fbdec0bff0383228d42cdc8eff2ab&oe=5E8E4F18

Path to the file encodes full chain of caches so no searching for a file is done at any place.

Doing it FB way

We have no infrastructure for any of it:

- No single origin / Data Lake
  - Have to use all of the current storages that have xroot protocol enabled.
- No control over cache deployments
  - Using SLATE to get most of the control needed
- Rucio does not know anything about caches
  - Adding a separate service to provide functionality needed (VP)
- No multi-level caches
  - Still too early for that.

Has to play nice with the current system.

Political headwinds...

Implicit assumptions:

- XCache delivers stability and performance required.
- XCache is centrally managed.
- Sites keep their xrootd endpoints up and running.

What is it actually ?

VP service is a Node.js server deployed in k8s at CERN.

It has a bunch of functions but the most important is:

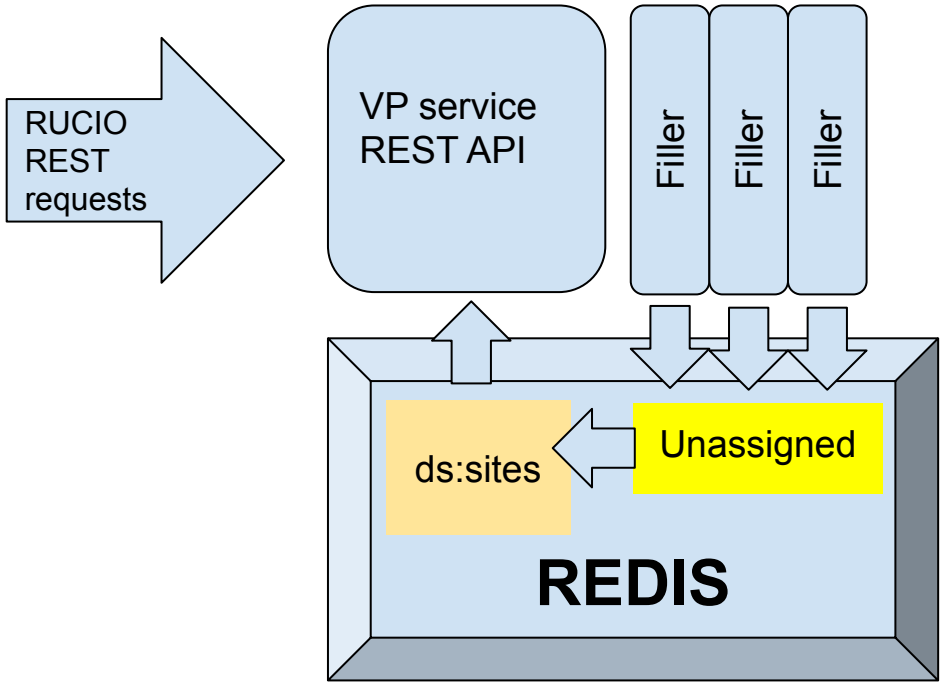
```
curl -XGET vpservice.cern.ch/ds/3/<scope:name>
```

Whatever you give it as scope and name it will return something like:

- [‘other’], - means no VP replicas.
- [DDM1], - only one VP replica.
- [DDM1, DDM2, DDM3] - two “spare” replicas will be in the same cloud.

What it does?

- It connects to its own Redis DB and checks if there is a key <scope:name>.
- If it does, it returns corresponding value.
- If it does not:
  - it throws random number to decide if and where to place a virtual replica
  - puts that decision in Redis DB



How a job knows where to get data?

Pilot uses **Rucio mover** to get data.

Rucio mover checks all the replicas and if a replica is remote and site has xrootd internal proxy defined, it constructs correct path which looks like:

```
root://xcache.xxx.org//root://origin//path/file
```

It works correctly in both copy2scratch and direct access modes.