ATLAS Compute Sites using native Kubernetes

Fernando Barreiro Megino, FaHui Lin
University of Texas at Arlington

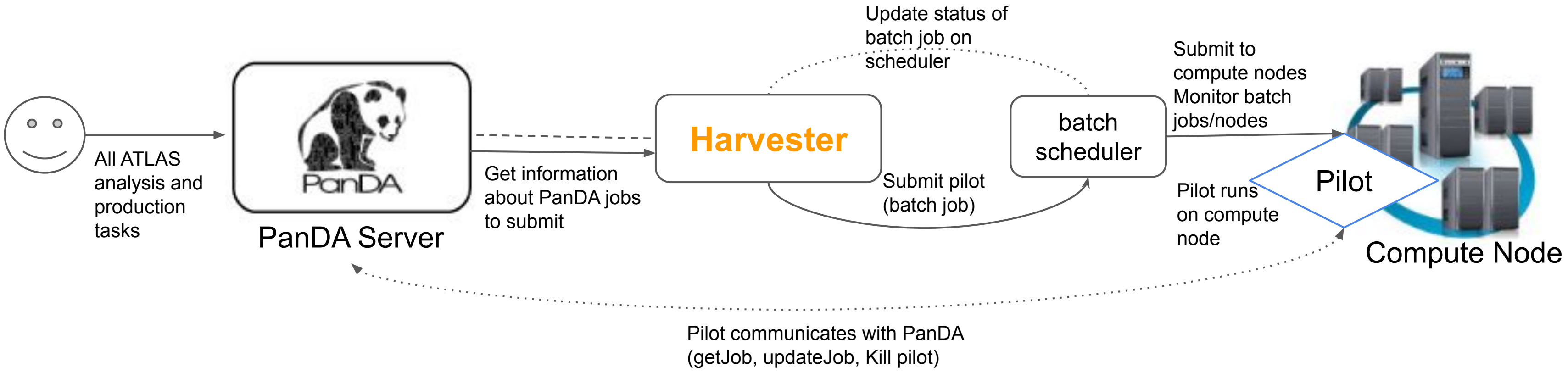
CERN IT Container Service Webinar Series
21 Oct 2020

Outline

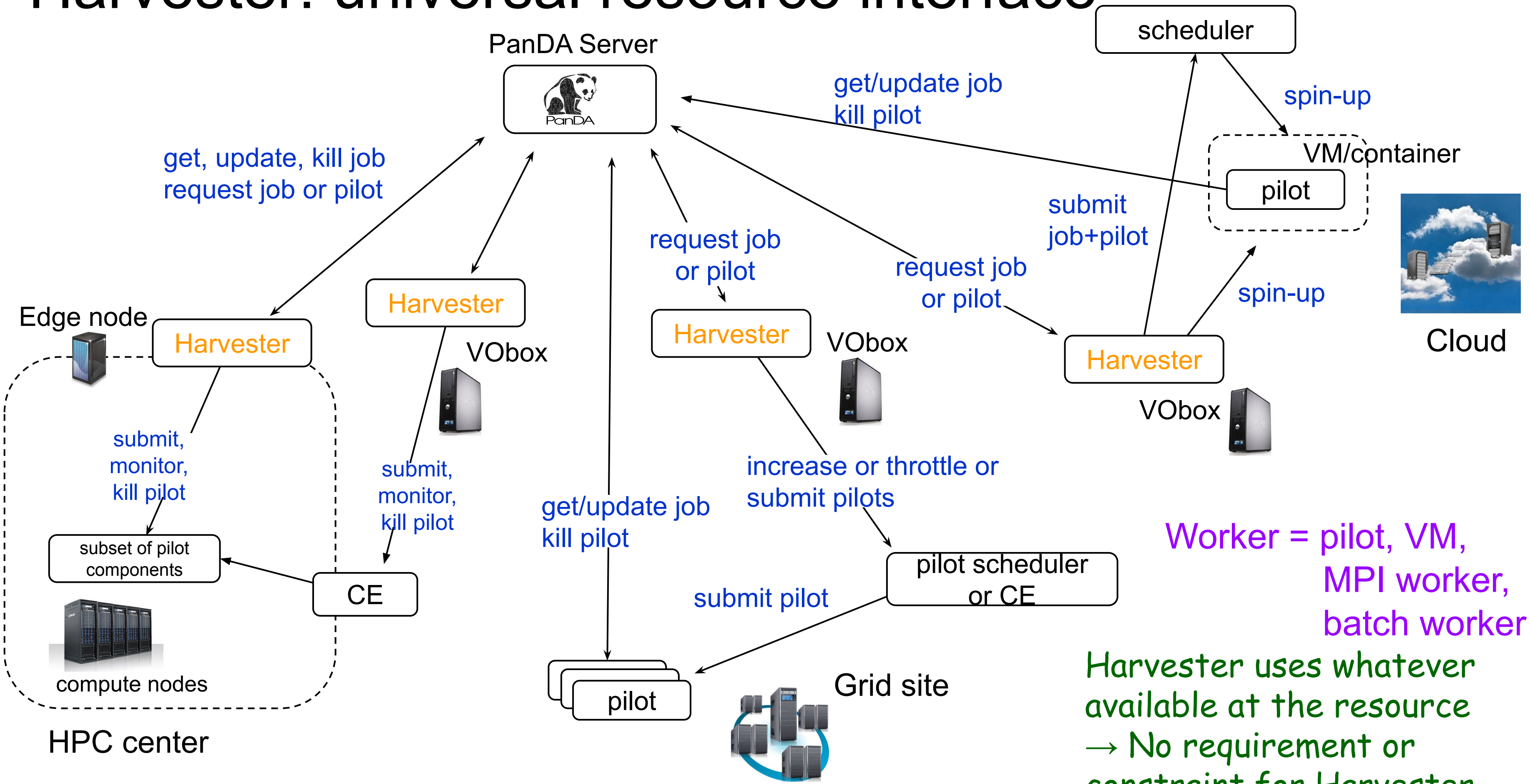
- PanDA server - Harvester - Pilot paradigm
- Motivation for native K8s batch integration
- Implementation details
- CVMFS on K8s clusters
- Resource overview
- Examples on commercial clouds
- Demo

PanDA, Harvester, Pilot

- PanDA: Production and Distributed Analysis system
  - Data-driven workload management system designed to meet ATLAS production and analysis requirements at LHC scale. All production and users' tasks are submitted to PanDA
- Harvester: A service to interface any compute resource
- Pilot: An execution environment to monitor and execute payload on a compute node
- PanDA server - Harvester - Pilot paradigm:



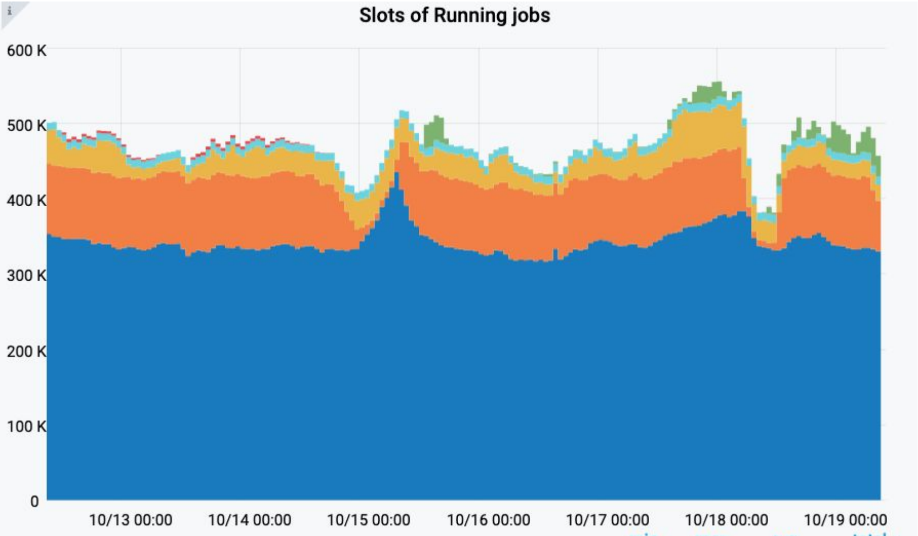
Harvester: universal resource interface



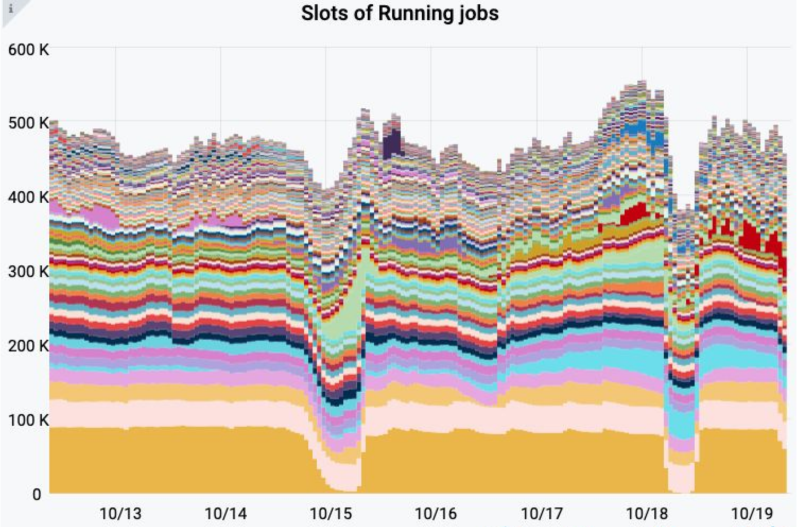
Worker = pilot, VM,
MPI worker,
batch worker

Harvester uses whatever available at the resource
→ No requirement or constraint for Harvester

ATLAS Grid scale



	min	max	avg	total
GRID	54 K	435 K	341 K	57.632 Mil
cloud (includes CERN P1)	7 K	97 K	83 K	13.985 Mil
hpc	3 K	69 K	31 K	5.223 Mil
cloud_special	2 K	11 K	10 K	1.721 Mil
hpc_special	0	40 K	5 K	776 K
Tier-0	1	4 K	789	133 K



	min	max	avg	total
CERN-P1	307	91.0 K	76.5 K	12.9308 Mil
CERN-T0	6.0 K	38.7 K	35.8 K	6.0556 Mil
MWT2	1.4 K	27.8 K	21.8 K	3.6793 Mil
BNL	3.1 K	20.3 K	18.5 K	3.1240 Mil
UNIBE-LHEP	0	39.4 K	12.5 K	2.1123 Mil
IN2P3-CC	1.9 K	14.5 K	12.2 K	2.0587 Mil

Background for Harvester K8s integration

- ATLAS using a heterogeneous computing infrastructure (Grid, HPC, Cloud) for batch workloads
  - Integration historically done by various teams and in various ways
- PanDA team developed Harvester universal resource interface ~3 years ago, more or less at the same time as first ATLAS-Google PoC phase
  - Looking for lightweight, generic cloud integration
- In the first PoC we implemented direct GCE VM lifecycle management
  - :) It worked
  - :(VM creation overhead
  - :(Specific to GCE
- Harvester team came up with the native K8s integration idea
  - :) It also works and has less overhead
  - :) Generic: available in many major cloud providers and some HEP institutes
  - :) Can also be used to host services: easy to deploy a lightweight/opportunistic site

Advantages and possibilities of K8s integration

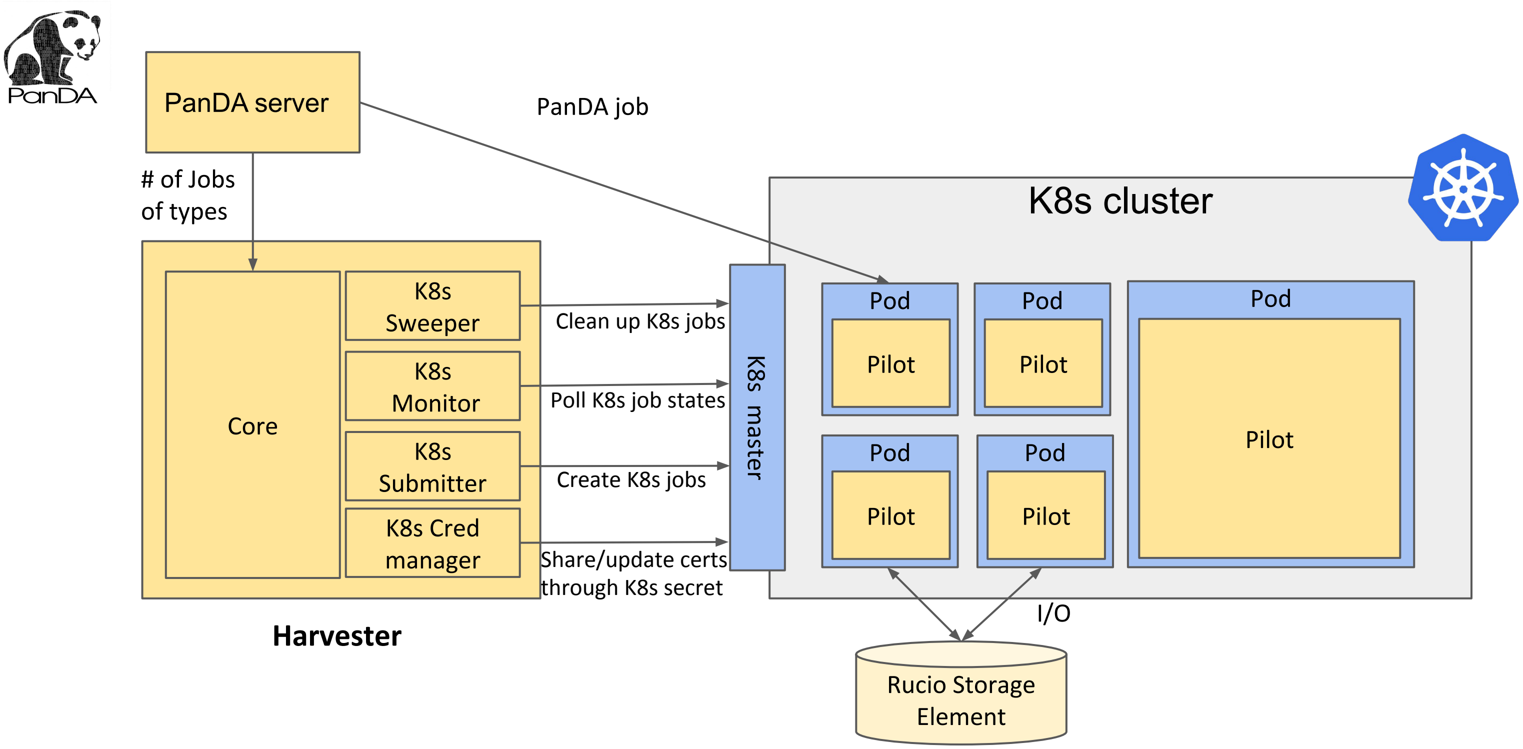
- Native container environment
- In theory standard interface across major cloud providers and WLCG clusters
    - CRITICAL!!!
- Massive infrastructure simplification compared to Grid-batch sites
    - However also losing Grid features/experience
    - Discovering many new behaviours
- Since early 2020 mini Kubernetes-grid with central Harvester growing
    - Couple hundred cores in each site:
        - Academia Sinica (Taiwan)
        - CERN (Switzerland)
        - University of Chicago (US)
        - University of Victoria (Canada)
    - And projects with commercial clouds:
        - Google
        - Amazon
        - Oracle project being set up

Harvester K8s integration - Job

- Harvester submits K8s Jobs (job controller) as workloads on K8s cluster
  - "*A Job creates one or more Pods and ensures that a specified number of them successfully terminate*" (official doc)
  - "*As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete*" (official doc)
- One K8s Job <=> one batch job
  - Harvester submits jobs
  - Each job runs one pod. Pilot runs in the pod
  - Harvester monitors jobs and pods
  - After jobs finish, Harvester deletes them
- K8s job retry mechanism is **not** used
  - If container fails, then pod will fail and job will fail
    (.spec.backoffLimit = 0 and .spec.template.spec.restartPolicy = "Never")
  - We manage retries on PanDA side

Harvester K8s integration - Jobs

https://github.com/HSF/harvester



Harvester K8s integration - Jobs

https://github.com/HSF/harvester

```
kind: Job
...
  backoffLimit: 0
...
    restartPolicy: Never
    containers:
      - args:
      - -c
      - cd; wget
https://raw.githubusercontent.com/HSF/harvester/mast
er/pandaharvester/harvestercloud/pilots_starter.py;
chmod 755 pilots_starter.py;
./pilots_starter.py || true
      command:
      - /usr/bin/bash
      env:
      - name: computingSite
        value: $computingSite
      - name: pandaQueueName
        value: $pandaQueueName
      - name: proxySecretPath
        value: /proxy/x509up_u25606_prod
      ...
      image: atlasadc/atlas-grid-centos7
```

```
resources:
limits:
        cpu: "8"
requests:
        cpu: 7200m
        memory: 12G
...
volumeMounts:
- mountPath: /cvmfs/atlas.cern.ch
  name: atlas
...
- mountPath: /proxy
name: proxy-secret
...
volumes:
- name: atlas
persistentVolumeClaim:
claimName: cvmfs-config-atlas
readOnly: true
...
- name: proxy-secret
secret:
defaultMode: 420
secretName: proxy-secret
```

Harvester K8s integration - Pod Affinity

- Two resource types of ATLAS job:
  - **SCORE** (1 core) vs **MCORE** (usually 8 cores = whole node, sometimes 4 cores or else)
  - Each pod has label about resource type (# of pods of either type is according to ATLAS jobs)
- K8s spreads out pods across nodes by default
  - May cause inefficient situation: Each node only runs 1 or 2 SCORE pods. The node still has plenty of empty slots but MCORE pod cannot fit in the node and there may not be enough SCORE pods to fill the node
- We set pod affinity policies to fill the slots more efficiently
  - SCORE and MCORE have anti-affinity against each other
  - SCORE has affinity to SCORE itself
- Thus SCORE pods tend to gather on the same nodes

```
labels:
    controller-uid: a59104f5-b8e1-4666-8abc-7e407bbe8ebb
    job-name: grid-job-2035575
    pq: CERN-EXTENSION_KUBERNETES
    prodSourceLabel: managed
    resourceType: MCORE
```

```
affinity:
    podAntiAffinity:
preferredDuringSchedulingIgnoredDuringExecution:
    - podAffinityTerm:
    labelSelector:
        matchExpressions:
        - key: resourceType
        operator: In
        values:
        - SCORE
    topologyKey: kubernetes.io/hostname
    weight: 100
```

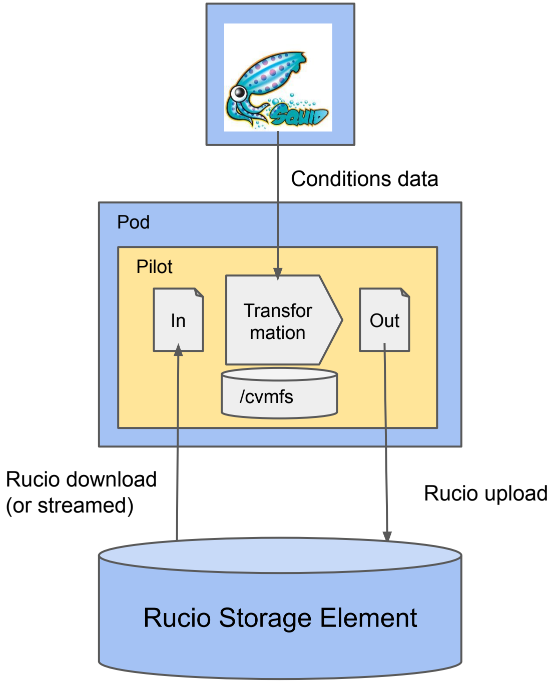
Harvester K8s integration - Pod Affinity

- Kubernetes site CERN-EXTENSION_KUBERNETES with 320 slots
- Slots are almost kept full during SCORE and MCORE transition

CVMFS & Squid setup on K8S clusters

- **CVMFS**: read-only hierarchically distributed read-only file-system
  - **ATLAS relies on CVMFS to distribute its Software on all resources (Grid, HPC, Cloud)**
  - Installed through daemonset + k8s volumes

- **Frontier Squid**: access to ATLAS run conditions database and local CVMFS cache through squid cache
  - Installed on dedicated VM or as part of the K8s cluster



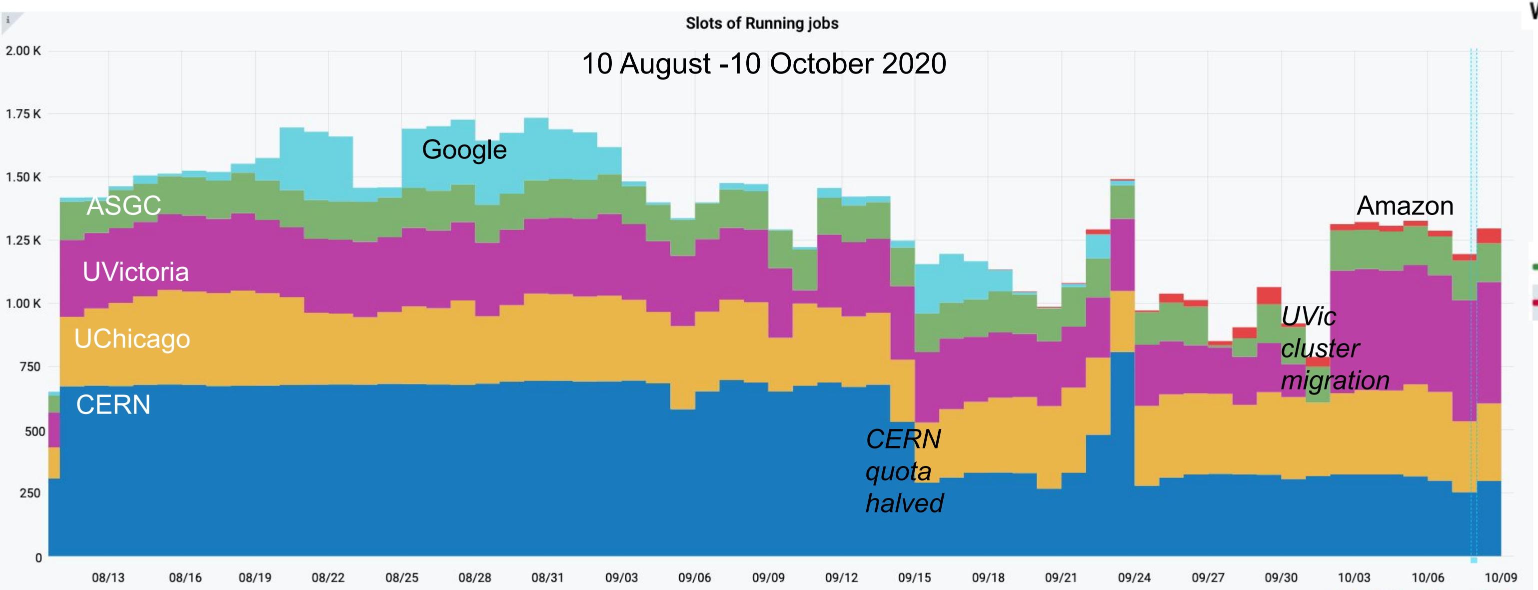
CVMFS: installation methods for K8S

- Directly on the nodes through package manager: most stable solution, but not always possible
- Through DaemonSets and volumes
  - DaemonSet: ensures all nodes run a copy of a pod
  - CVMFS CSI driver
    - CSI: Container Storage Interface
      - Standard to expose storage volumes to the containers
    - Implemented by CERN IT and used initially in some of our clusters
    - Golang implementation of required methods
    - Complicated and some issues e.g. on restart
  - CVMFS PRP driver
    - PRP: Pacific Research Platform
    - CVMFS mount shared through local volume. Much simpler
    - Currently using ATLAS fork at CERN, Google and Amazon PanDA queues
    - My preferred option when direct installation not possible

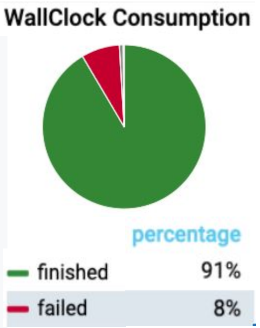
CVMFS drivers: importance of CPU/mem requirements

- Our K8S nodes typically fully exploited: jobs submitted with “burstable” QoS
- Drivers installed at CERN Openstack clusters typically have no requirements
- No CPU and memory requirements for driver pods means “best effort” QoS (i.e. lowest priority)
    - **No memory requirement**: causes CVMFS driver pod to be killed first when OOM
    - **No CPU requirements:** causes CVMFS driver to be throttled, i.e. gets absolutely no CPU cycles when node is packed with jobs
    - **Both end up with an extremely unstable cluster and unacceptable failure rates**
- Requesting small amount of CPU and memory solves situation

Current ATLAS K8S queues

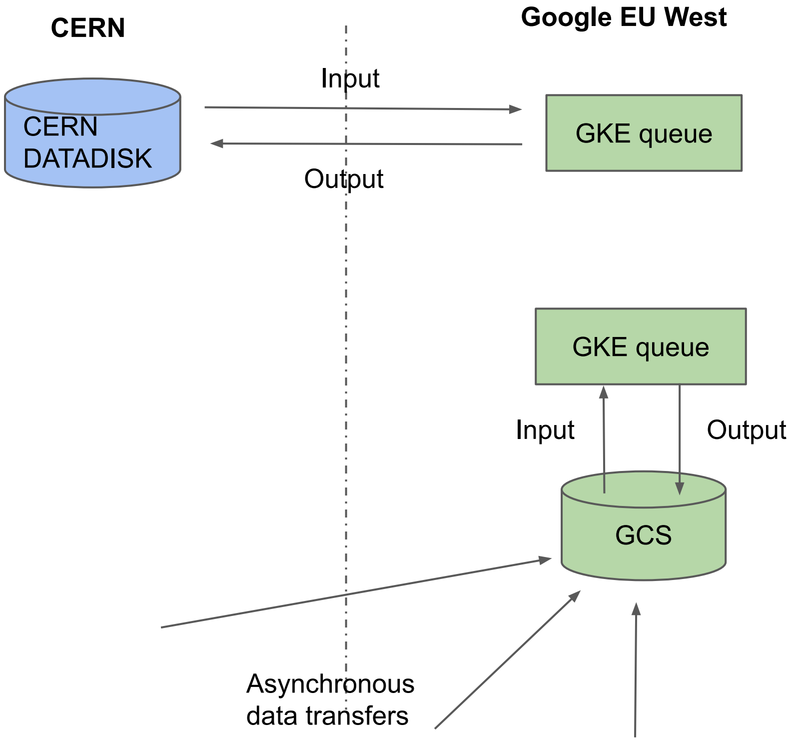


	min	max	avg	total
CERN-EXTENSION_KUBERNETES	156	808	527	32.132 K
MWT2_K8S_UCORE	123	377	306	18.643 K
CA-VICTORIA-K8S-T2	0	485	288	17.543 K
Taiwan-LCG2_K8S_UCORE	8	163	144	8.804 K
GKE	0	270	68	4.130 K
FRESNO-EKS-REMOTE	0	69	9	523
FRESNO-EKS-S3	0	0	0	0



US ATLAS - Google project

Tested various configurations and payloads during extensive periods, but at low scale



Stage 1: Simulation with storage at CERN

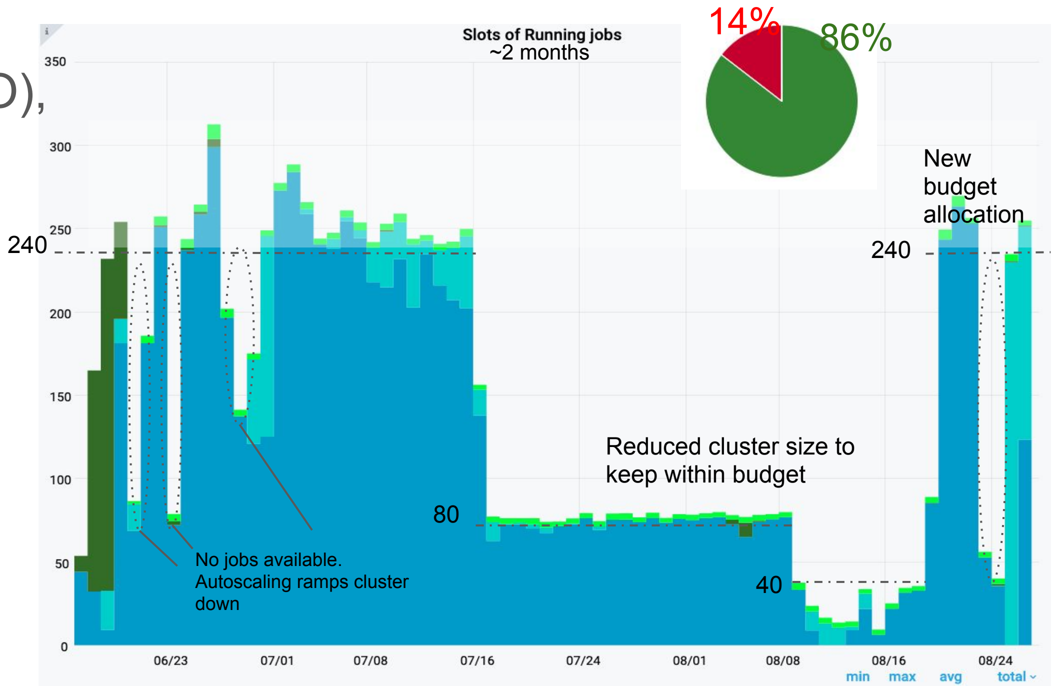
- **Very light I/O jobs**
- GKE setup and evaluation

Stage 2: End-user analysis with storage at Google

- **I/O heavy** jobs from volunteer analysis user
- Storage at Google possible thanks to Rucio/FTS/middleware integration
- VM/node tuning

Stage 1: GKE simulation cluster with CERN storage

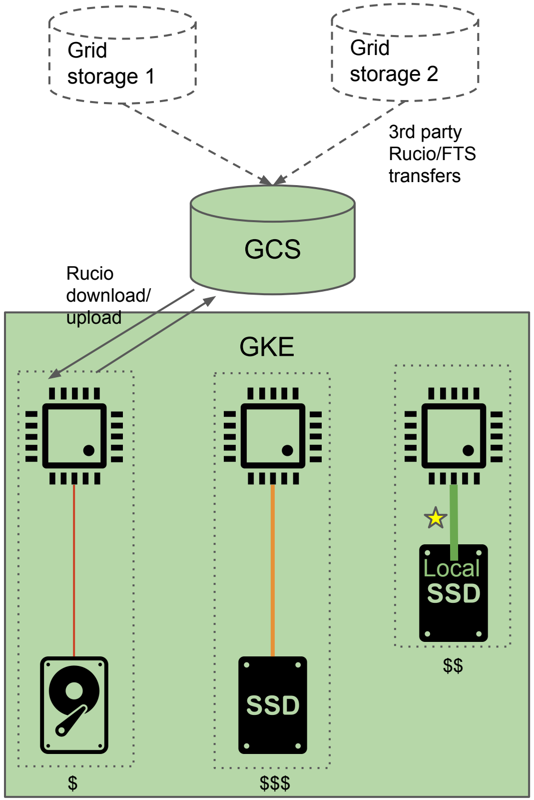
- Limited to Simulation jobs (low I/O), since storage at CERN
- **Preemptible** nodes
  - Causing most of the failures
  - Limiting job duration to <5 hours
  - Attractive deal: big cost reduction, slightly higher failure rate
- **Autoscaled** cluster
  - Cluster ramps down and lowers the cost when no jobs queued
- Costs with remote storage: ~2kUSD/month for 150 cores including egress to CERN



	min	max	avg	total
MC Simulation Fast	0	299	121	8.7968 K
MC Simulation Full	0	230	14	1.0217 K
MC Event Generation	0	199	6	426
Testing	0	9	3	251

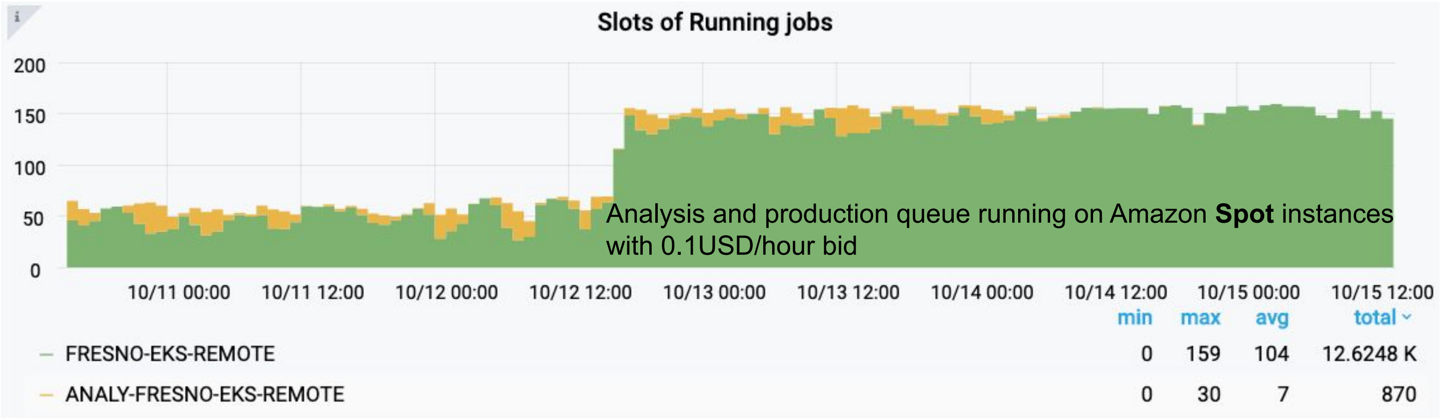
Stage 2: GKE User Analysis and GCS storage

- First ATLAS attempt to run a site (compute + storage) fully in the cloud
- Volunteer user analyzing 1TB dataset
  - 2.5 to 12.5 (=5 x 2.5) GB of input per job
- Side-condition: All input files need to be downloaded within 10 min (signed URL lifetime)
- Google throttles throughput to resources to balance usage across tenants
  - Found bottleneck in CPU→disk throughput on lower end VMs
  - To improve you can upgrade storage type or over-allocate disks
  - Jobs required VMs with local SSD (~50% more expensive)
- Preemptible nodes confuses end users



Other commercial cloud projects

- More recently we started running K8s clusters at Amazon (Fresno State grant) and Oracle (Univ. of Oslo contract, setup in progress)
    - Rucio team also working with davix team to sort out issues for transfers to S3
- Basic compute integration is straightforward and no code changes required
- Effort mostly spent understanding different setups between cloud providers (network details, usage of Spot instances, setting up autoscaling, service accounts)



Demo

- Show how simple it is to setup a PanDA queue in a commercial cloud with Kubernetes
- Google setup
  - Create GKE cluster: http://console.cloud.google.com/
  - Show service account: roles and permissions
- PanDA queue
  - Show AGIS (ATLAS central configuration service) queue configuration
  - See if there are ‘activated’ jobs in PanDA monitoring
- Connect the GKE cluster to central Harvester instance
  - Download Kubeconfig via gcloud CLI
  - Install CVMFS
  - Add queue to Harvester configuration
  - See secrets, submitted pods and PanDA jobs changing status to ‘running’
  - See GCE monitoring
- Compare to a running cluster in AWS

Conclusions

- Straightforward, standard integration of major cloud providers
  - PanDA queues can be deployed quickly
  - This used to take weeks and custom code. No heavy middleware components
- Useful model for smaller Grid sites, but CE functionalities get lost
- It requires some experience, but we are starting to get it right
  - CVMFS setup still the weakest link, e.g. while cache gets warm
  - In ideal world we would use self contained SW images, but model needs to stay compatible to mainstream Grid
- Scale of our exercises has been hundreds to few thousand cores per cluster
  - Mostly limited by availability of resources
  - At these levels the system is quite relaxed
  - Interested to see which scale we could reach
- Slight worry on dockerhub policy changes next month
- Still potential for advanced features: User Analysis facilities, machine learning clusters, etc.

Questions?