Lightweight integration of Kubernetes clusters for ATLAS batch processing

Fernando Barreiro Megino, FaHui Lin
University of Texas at Arlington

Second K8s-HEP Meetup
30 Nov 2020

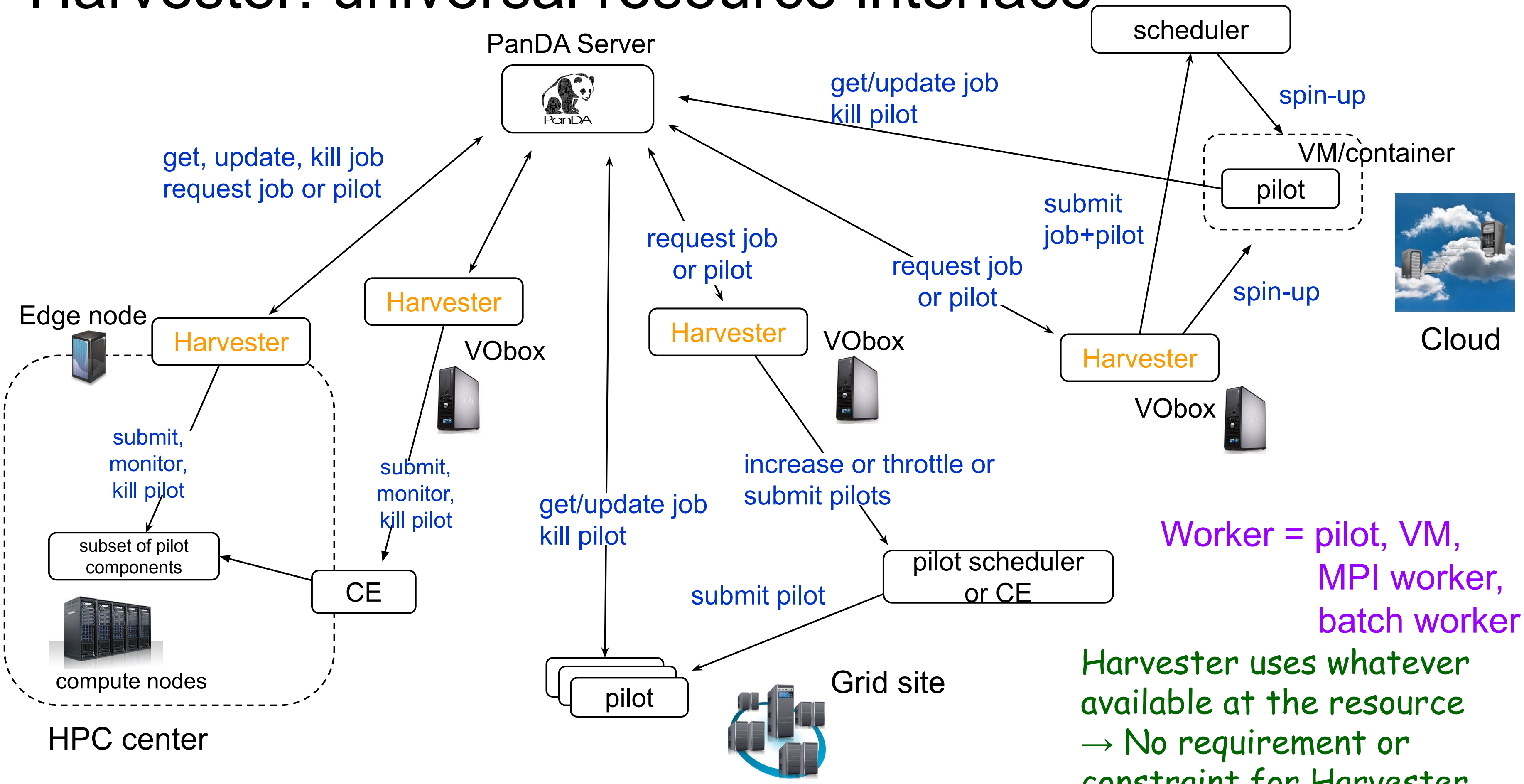
UNIVERSITY OF TEXAS ARLINGTON

PanDA

ATLAS EXPERIMENT

Harvester: universal resource interface

T. Maeno

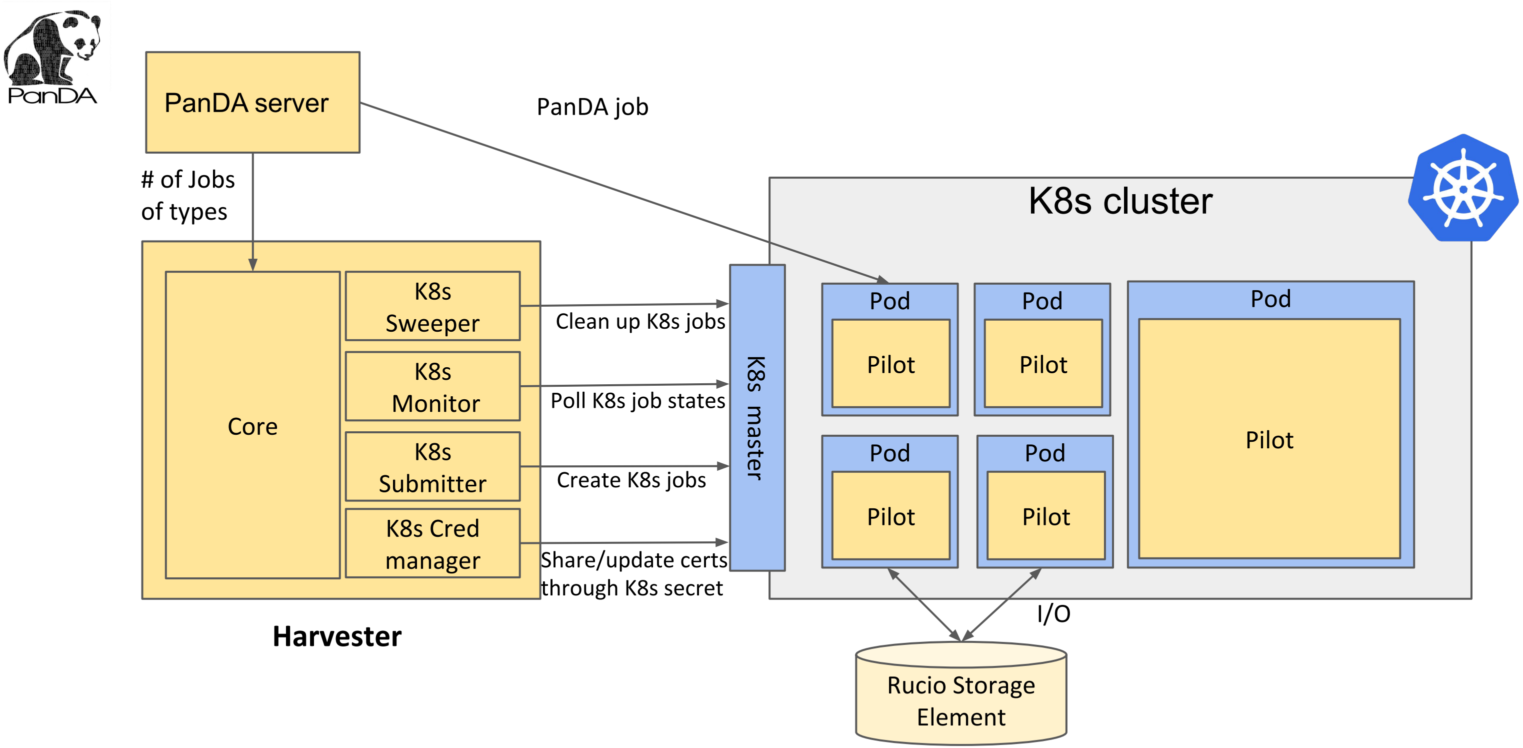


Worker = pilot, VM, MPI worker, batch worker

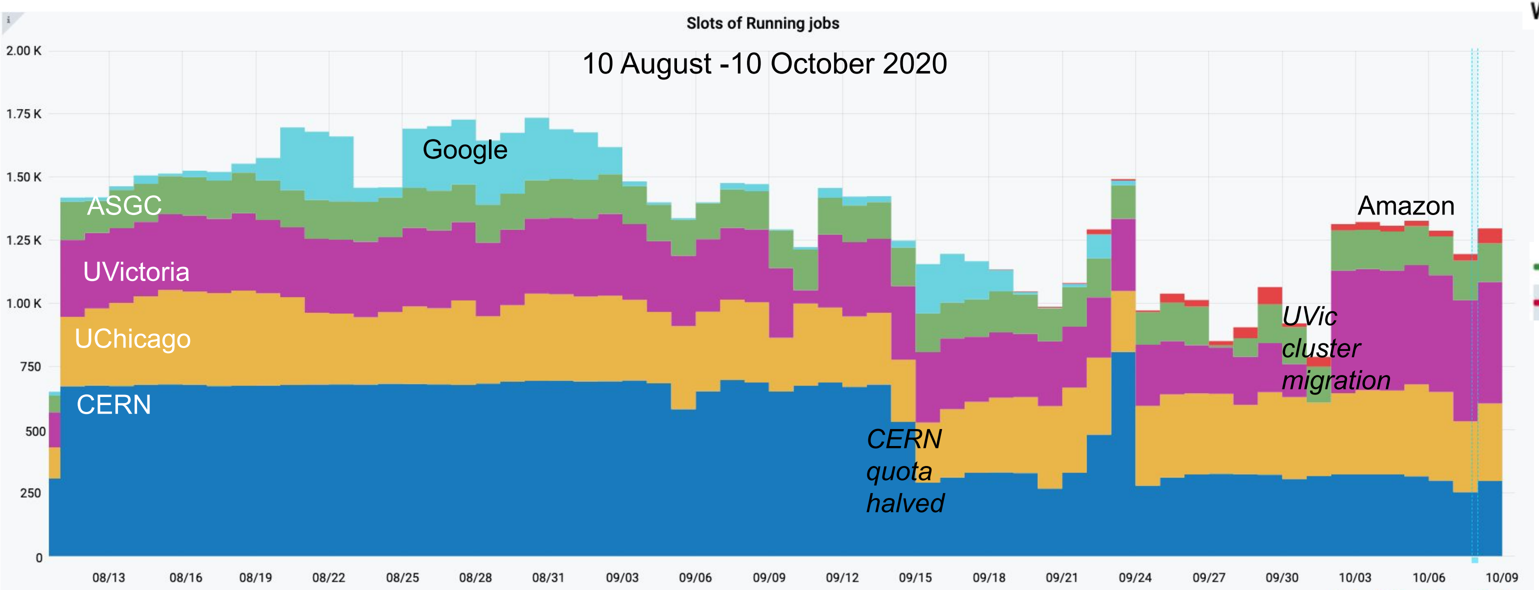
Harvester uses whatever available at the resource
→ No requirement or constraint for Harvester

Harvester K8s integration - Jobs

https://github.com/HSF/harvester



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

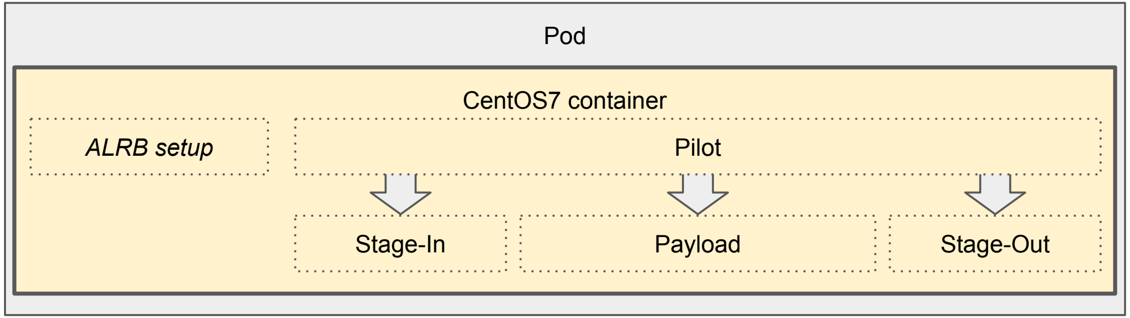
CVMFS: installation methods for K8S

- Directly on the nodes through package manager: most stable solution, but not always possible
- Through DaemonSets and volumes
  - **If you don't set memory and CPU requests/limits, the driver pods will not work properly**
  - CVMFS CSI driver (CSI: Container Storage Interface)
    - 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)
    - **My preferred option when direct installation not possible**
    - CVMFS mount shared through local volume. Much simpler
    - Currently using ATLAS fork at CERN, Google and Amazon PanDA queues
      - Small modifications: liveness probe and blind clean up on start-up
    - Only known issue to me: jobs fail until cache is loaded

CE/batch features

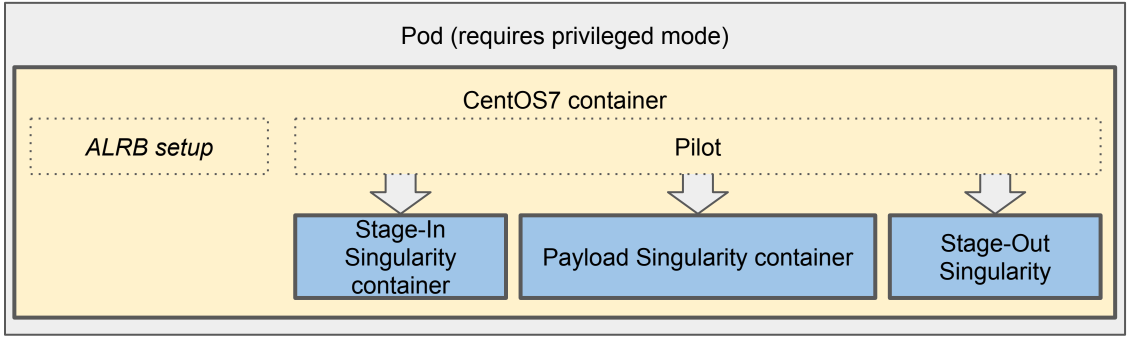
- Authenticating to K8s cluster: manually managed kubeconfig files
- Scheduling: we use default Kubernetes scheduling. K8s job instances submitted with CPU & memory requirements according to our job sizes
    - We use affinity so that single core jobs go to same node and don't spread out
    - We haven't studied in detail Kubernetes scheduling priorities, but in our experience so far we didn't face issues with mixed (single vs multi core) payloads
- APEL accounting: Ryan (UVic) working on it
- Fair shares:
    - So far only ATLAS specific sites/clusters
    - Some Kubernetes projects have implemented them, but it's something that would have to be evaluated
- Traceability: it gets mentioned in every WLCG K8S presentation, but I don't think anyone has looked into it

Native containers, but...



First integration, being phased out

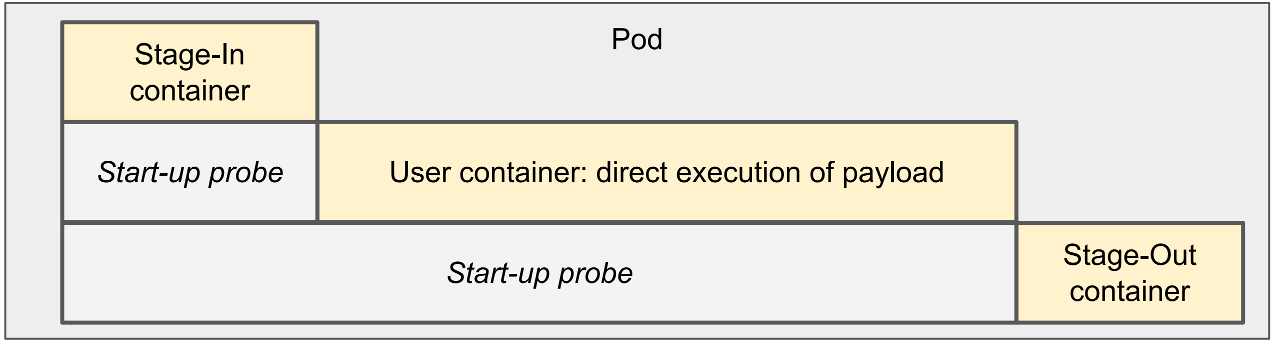
- Push & pull model independent
- Problem when payload not CentOS7 compatible
- Python version mismatches between Stage-In/Out and Payload
- Keeping this model alive requires additional effort and restricts sites



Second integration

- Push & pull model independent
- Each component runs on his favorite container image
- Compatible with Grid, i.e. no extra work
- Nested containers require privileged mode
- Not very elegant

Native containers, but...



Potential future integration

- Requires push model
- Not Grid compatible, i.e. extra work*
- Each component runs on his favorite container image
- Native mode

* initial braindump of work required

- Container synchronization in the pod
- Stage-In/Out containers with Pilot and Rucio client modules
- Absorb pilot/WN components in Harvester or replace directly through Kubernetes features
  - Available in kubernetes: memory, disk and walltime monitoring
  - Management of error codes/messages
  - Generation of execution string (i.e. mimicking ALRB and pilot wrapper)
  - Other pilot features difficult to replace, e.g. looping job monitoring
- Image management:
  - Singularity images to be published in scalable registry
  - Stage-in/out containers to be updated with each pilot/rucio client release

⇒ effort to implement this needs to be justified and have an important use case

Cloud sites

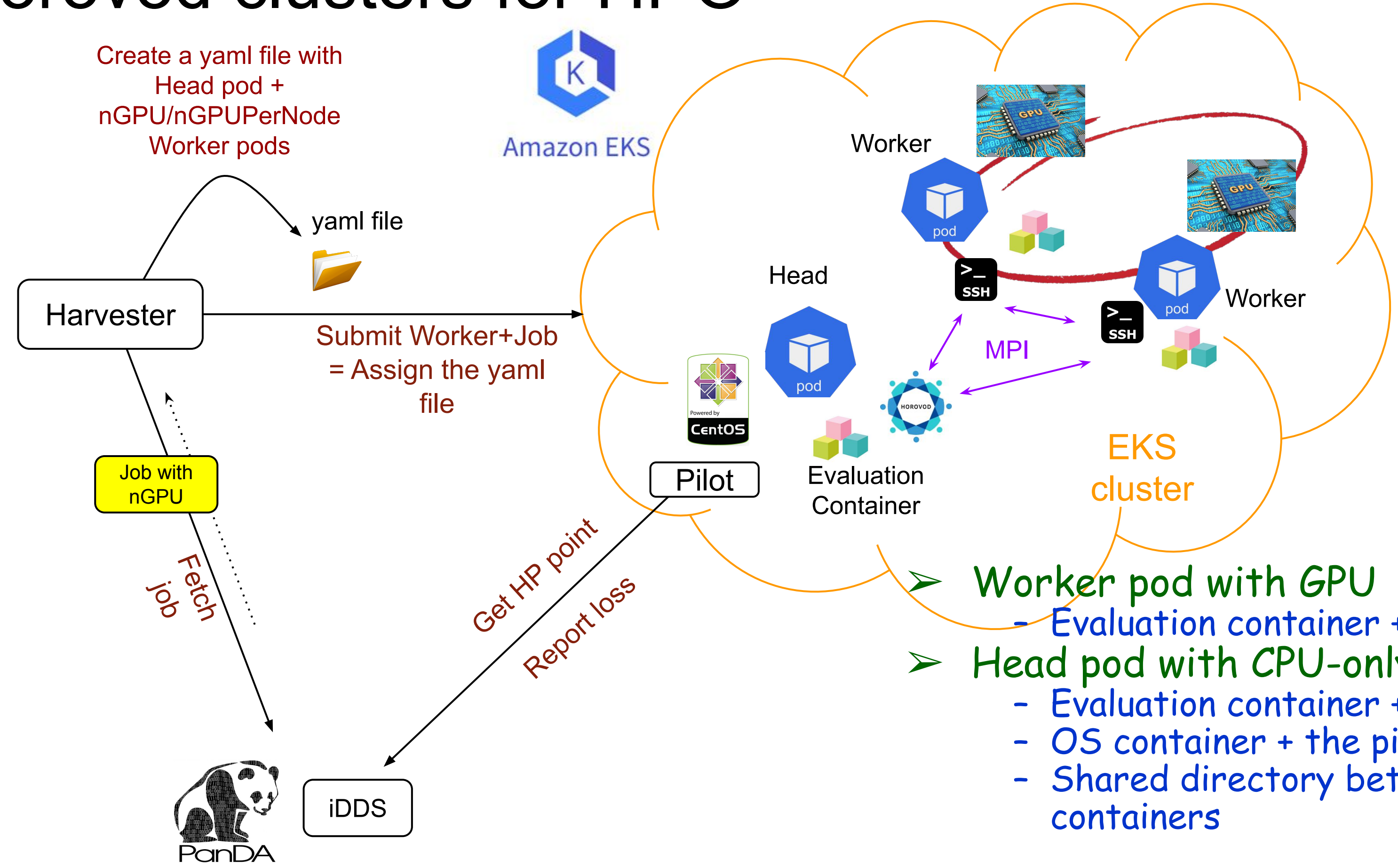
- Done in collaboration with Rucio team: lightweight cloud site with compute and storage
- Google
  - Evaluated jobs
    - MC Simulation with remote RSE
    - User analysis with local RSE
  - Very easy to setup clusters and additional features, e.g. preemptible VMs and service accounts
  - Preemptible VMs can only last 24h, but ~70% cheaper
    - Limits duration of acceptable payloads and increases failure rate to ~15%
  - I/O demanding jobs require higher-end VMs with local $$D
  - Rucio SEs on GCS functional, e.g. 3rd party copy, download, upload
  - Issue with direct I/O from GCS (file corruption errors)
  - This model was evaluated by LSST and they successfully ran a pipeline (S. Padolski)

Cloud sites

- Amazon
  - Evaluated jobs
    - MC Simulation with remote RSE
    - User Analysis with local RSE ongoing
  - AWS setup more complicated, in particular setup of Spot instances and auto scaling
    - Assuming your Spot bid is good enough, the instances can run indefinitely
  - OS for nodes with old systemd, mounting volumes to pods starts failing after a while
  - Rucio team ironing out last details to complete integration
  - Direct I/O worked on preliminary tests with S3
- Oracle cloud
  - Evaluated jobs: HC on trial account
  - Easy setup, but service accounts have to be created directly on Kubernetes cluster
  - Available VM sizes in Zurich not ideal for ATLAS payloads
  - Potential project from UOslo
- General remark: egress cost represent very significant fraction

Horovod clusters for HPO

T. Maeno



- ➢ Worker pod with GPU
  - Evaluation container + sshd
- ➢ Head pod with CPU-only
  - Evaluation container + horovodrun
  - OS container + the pilot
  - Shared directory between two containers

Conclusions

- Straightforward, standard integration of major cloud providers
- Lightweight, industry standard model for smaller Grid sites
  - But some CE functionalities need to be replaced
- Scale of our exercises has been hundreds to few thousand cores per cluster
  - Mostly limited by availability of resources
  - No stress at current scale
- Potential for advanced features: User Analysis facilities, machine learning clusters, etc.

Backup

Also see https://indico.cern.ch/event/950884/

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

```
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
```

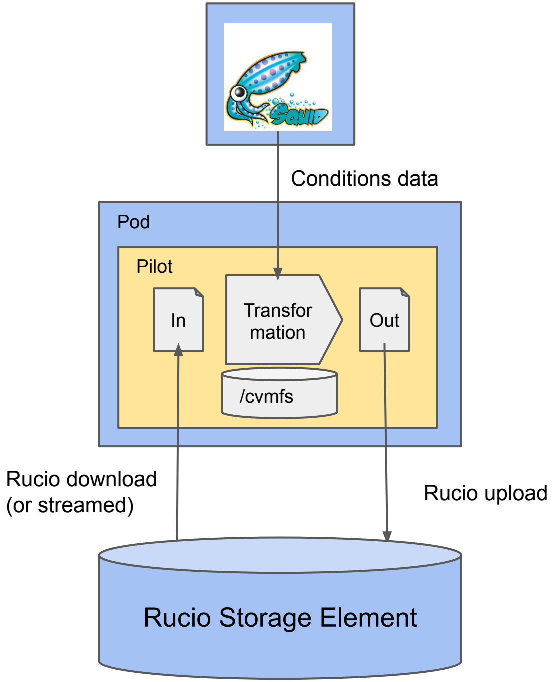
```
affinity:
    podAntiAffinity:

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

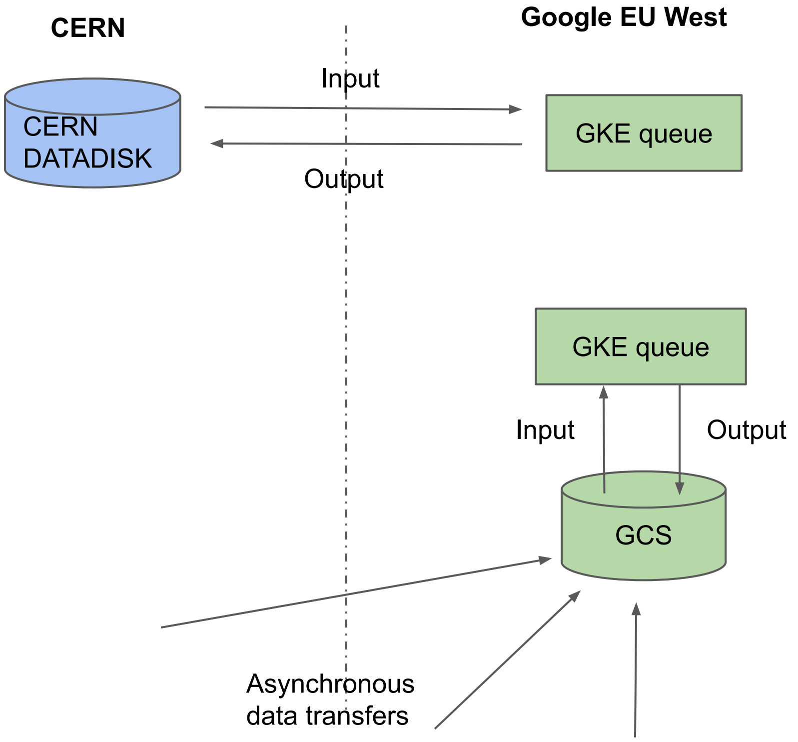


CVMFS drivers: importance of CPU/mem requirements

- 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

US ATLAS - Google project

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



Stage 1: Simulation with storage at CERN

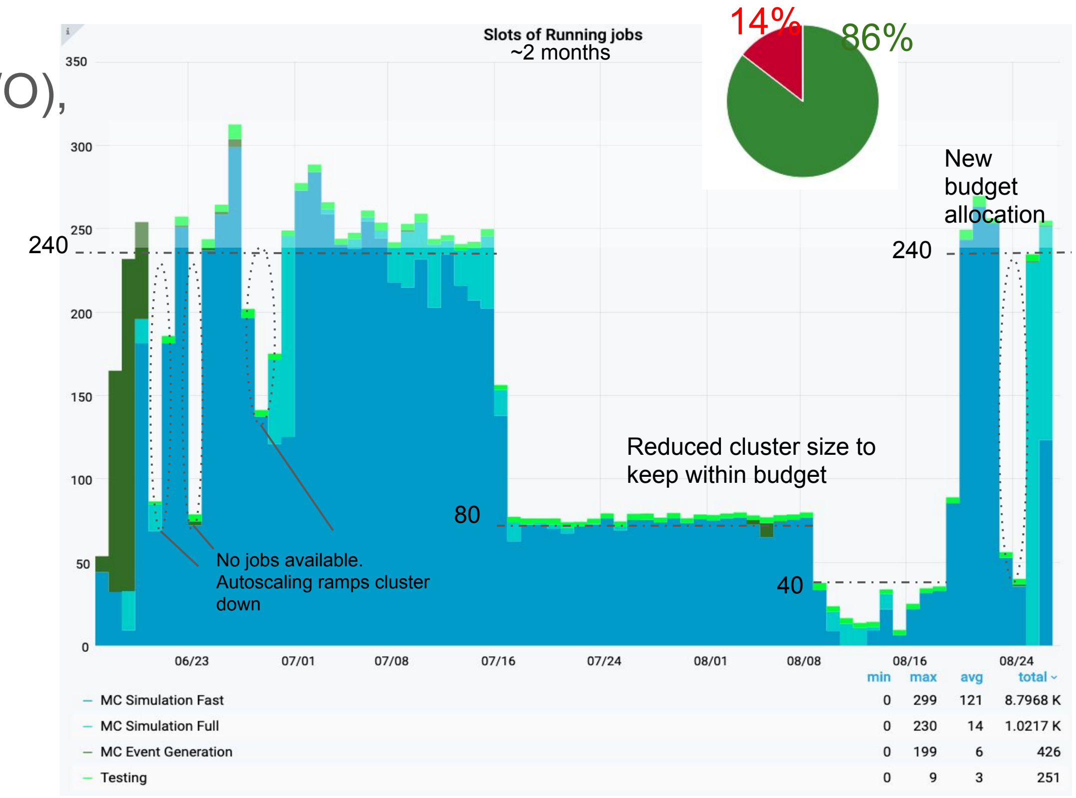
- **Very light I/O jobs**
- GKE setup and evaluation

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

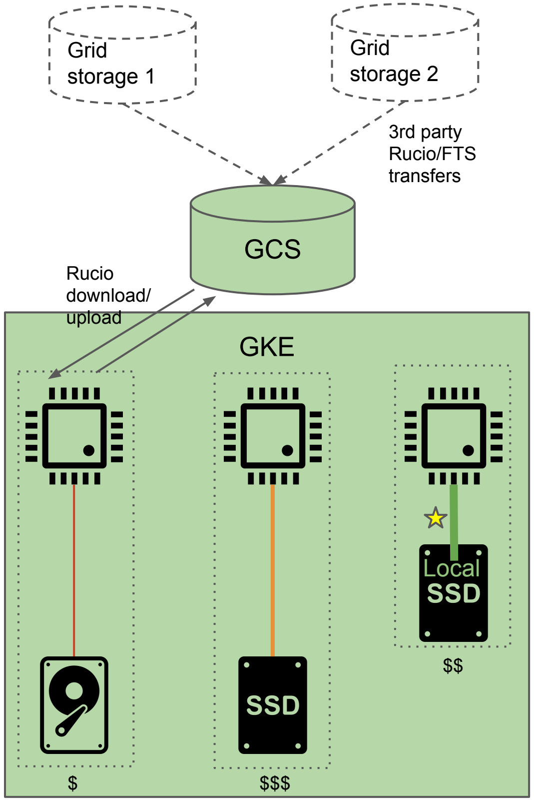
Stage 1: GKE simulation cluster with CERN storage

- 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



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)