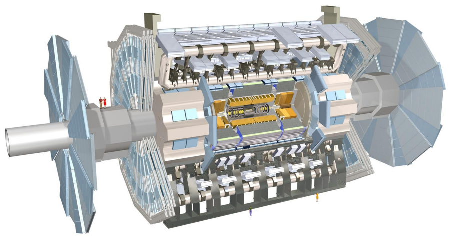
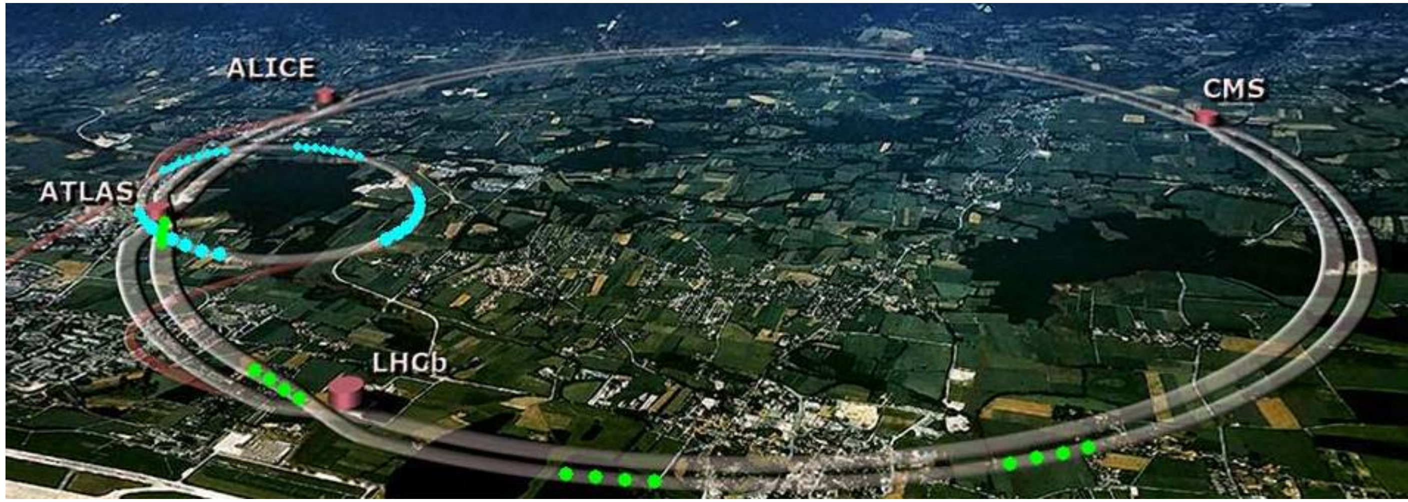
R&D for the expansion of the Tokyo regional analysis center using Google Cloud Platform

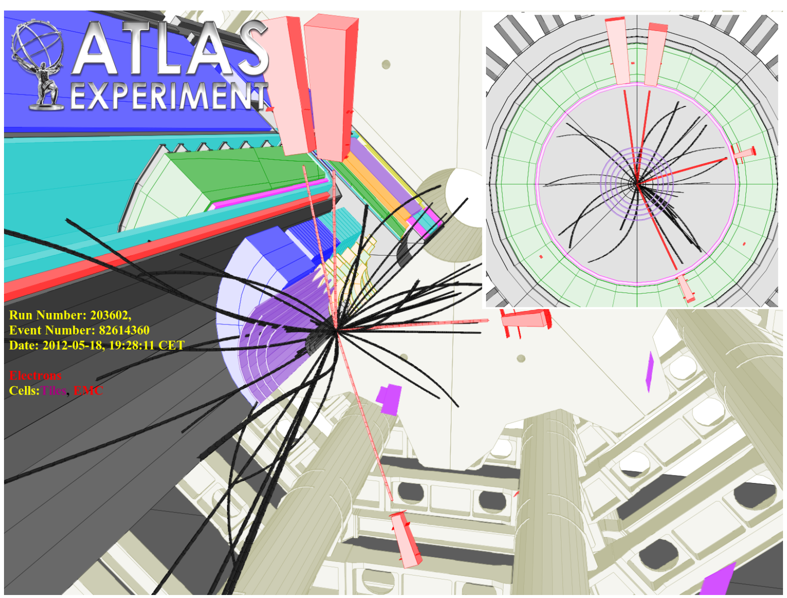
M. Kaneda, J.Tanaka, T. Mashimo,
R. Sawada, T. Kishimoto and N. Matsui
The International Center for Elementary Particle Physics (ICEPP),
The University of Tokyo

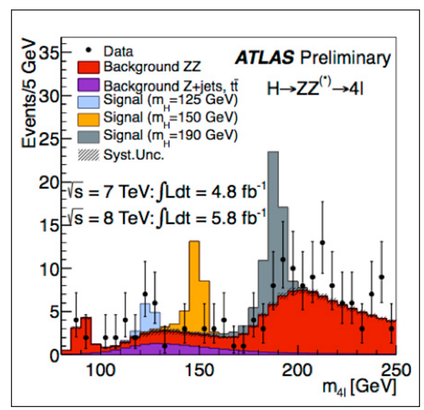
05/Apr/2019, ISGC 2019, Taipei, Taiwan

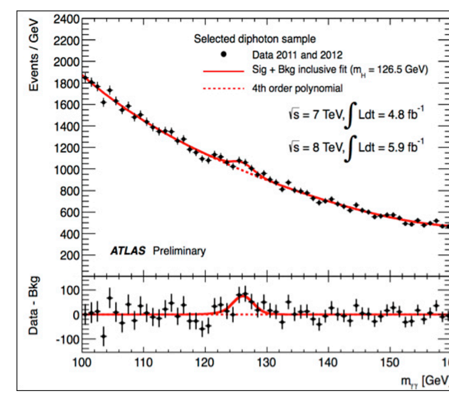
The ATLAS Experiment





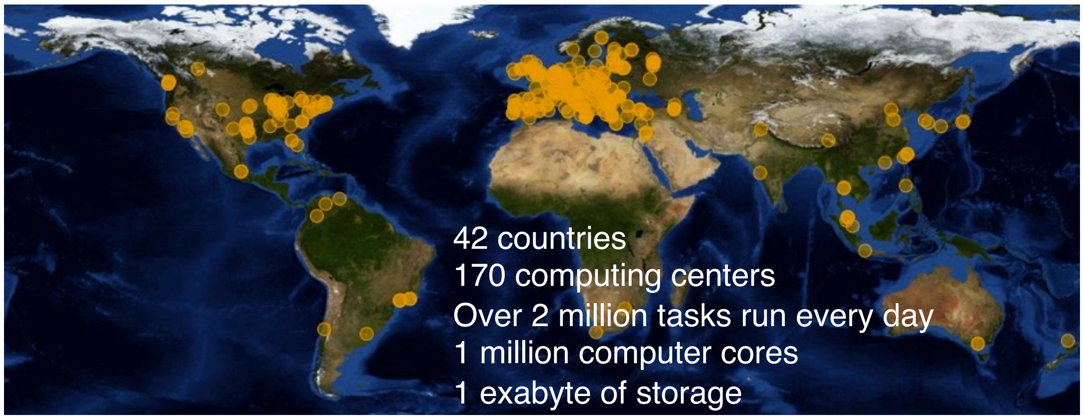


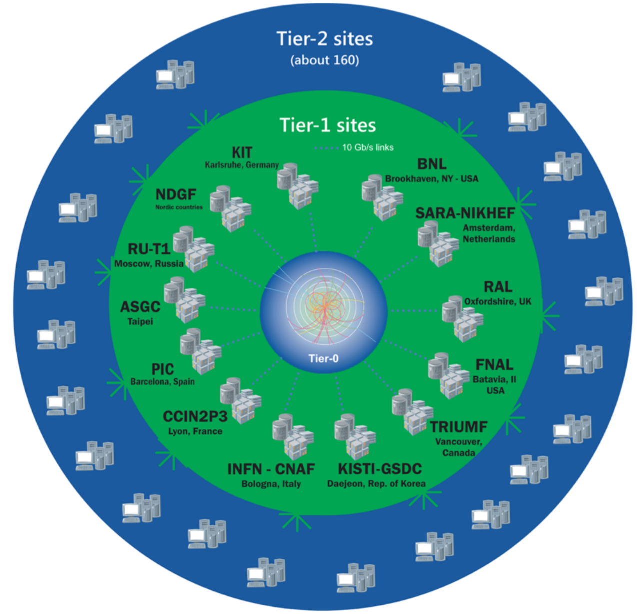




The Higgs Boson Discovery in 2012

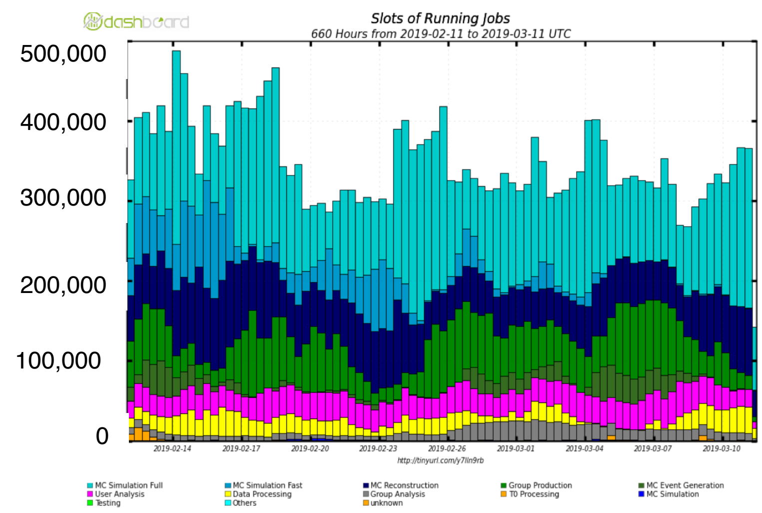
Worldwide LHC Computing Grid (WLCG)





- A global computing collaboration for LHC
  → Tier0 is CERN

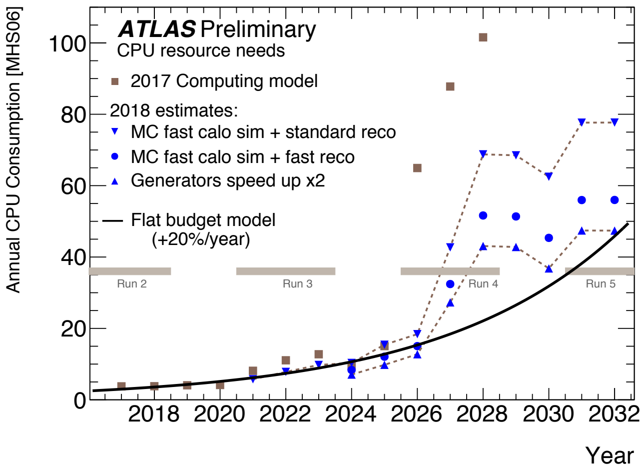
- The Tokyo regional analysis center is one of Tier2 for ATLAS



Number of cores used by ATLAS

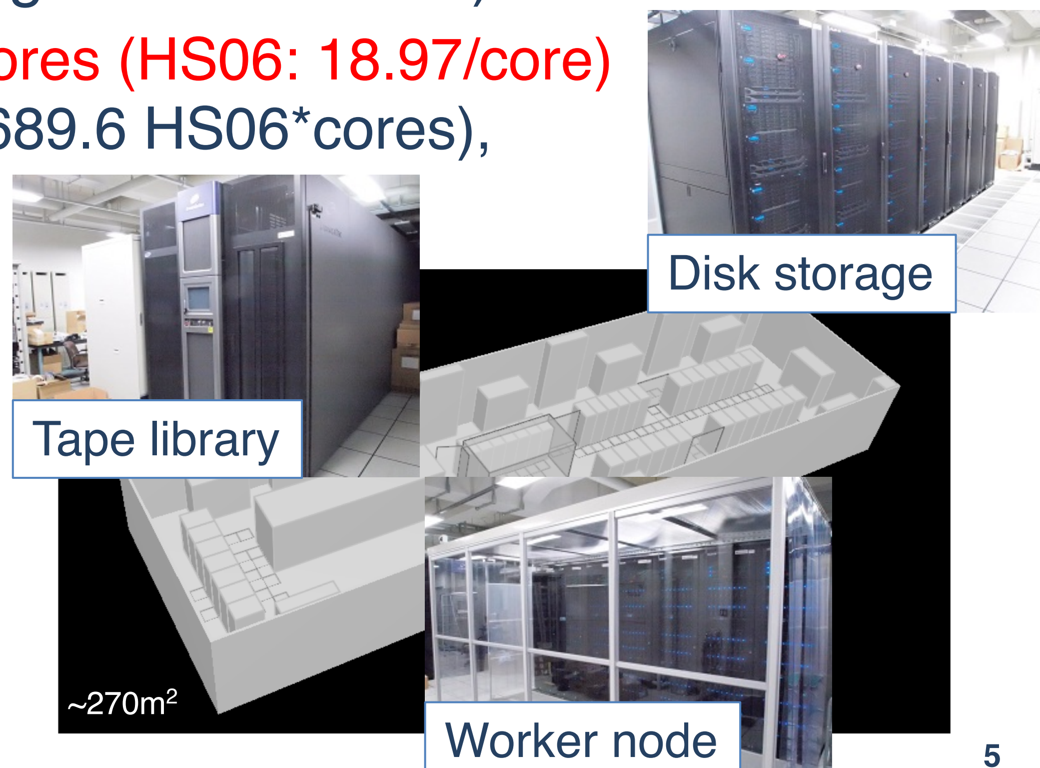
Computing Resources for HEP

- Data amount of HEP experiments becomes larger and larger
  →Computing resource is one of the important piece for experiments
- CERN plans High-Luminosity LHC
  →The peak luminosity: x 5
  →Current system does not have enough scaling power
  →Some new ideas are necessary to use data effectively
    →Software update
    →New devices: GPGPU, FPGA, (QC)
    →New grid structure: Data Cloud
    →External resources: HPC, Commercial cloud



The Tokyo regional analysis center

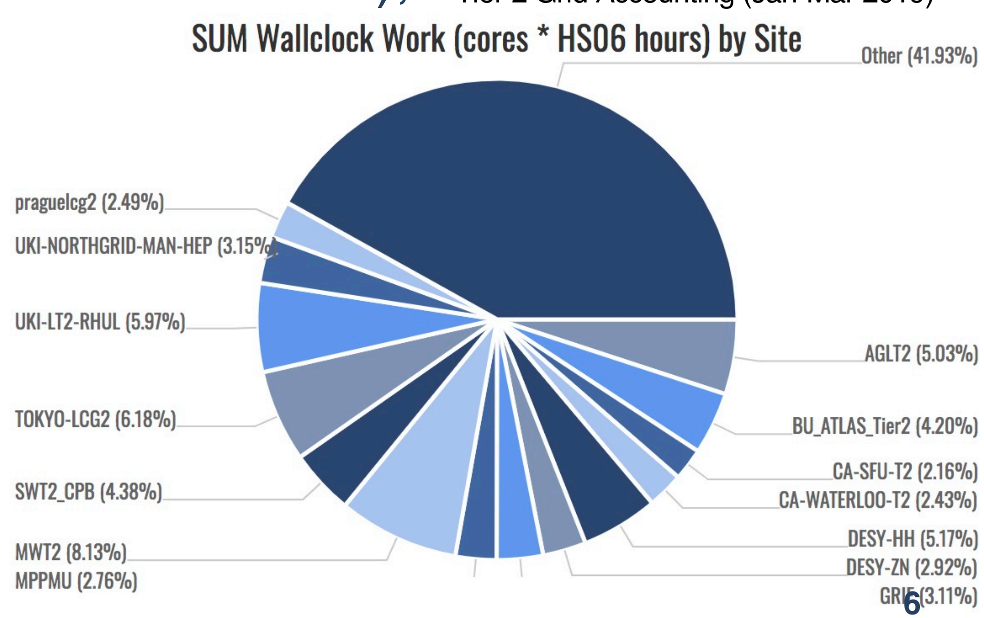
- The computing center at ICEPP, the University of Tokyo
- Supports ATLAS VO as one of the WLCG Tier2 sites
  →Provides local resources to the ATLAS Japan group, too
- All hardware devices are supplied by the three years rental
- Current system (Starting from Jan/2019):
  →Worker node: 10,752cores (HS06: 18.97/core) (7,680 for WLCG, 145689.6 HS06*cores), 3.0GB/core
  →File server: 15,840TB, (10,560TB for WLCG)



The Tokyo regional analysis center

- The computing center at ICEPP, the University of Tokyo
- Supports ATLAS VO as one of the WLCG Tier2 sites
  →Provides local resources to the ATLAS Japan group, too
- All hardware devices are supplied by the three years rental
- Current system (Starting from Jan/2019):
  →Worker node: 10,752cores (HS06: 18.97/core)
  (7,680 for WLCG, 145689.6 HS06*cores),
  3.0GB/core
  →File server: 15,840TB,
  (10,560TB for WLCG)

Tier 2 Grid Accounting (Jan-Mar 2019)



TOKYO-LCG2 provides 6% of Tier 2

Commercial Cloud

- Google Cloud Platform (GCP)
  →Number of vCPU, Memory are customizable
  →CPU is almost uniform:
    → At TOKYO region, only Intel Broadwell (2.20GHz) or Skylake (2.00GHZ) can be selected (they show almost same performances)
  →Hyper threading on
- Amazon Web Service (AWS)
  →Different types (CPU/Memory) of machines are available
  →Hyper threading on
  →HTCondor supports AWS resource management from 8.8
- Microsoft Azure
  →Different types (CPU/Memory) of machines are available
  →Hyper threading off machines are available

Google Computing Element

- **HT On**
  → All Google Computing Element (GCE) at GCP are HT On
  → TOKYO system is HT off

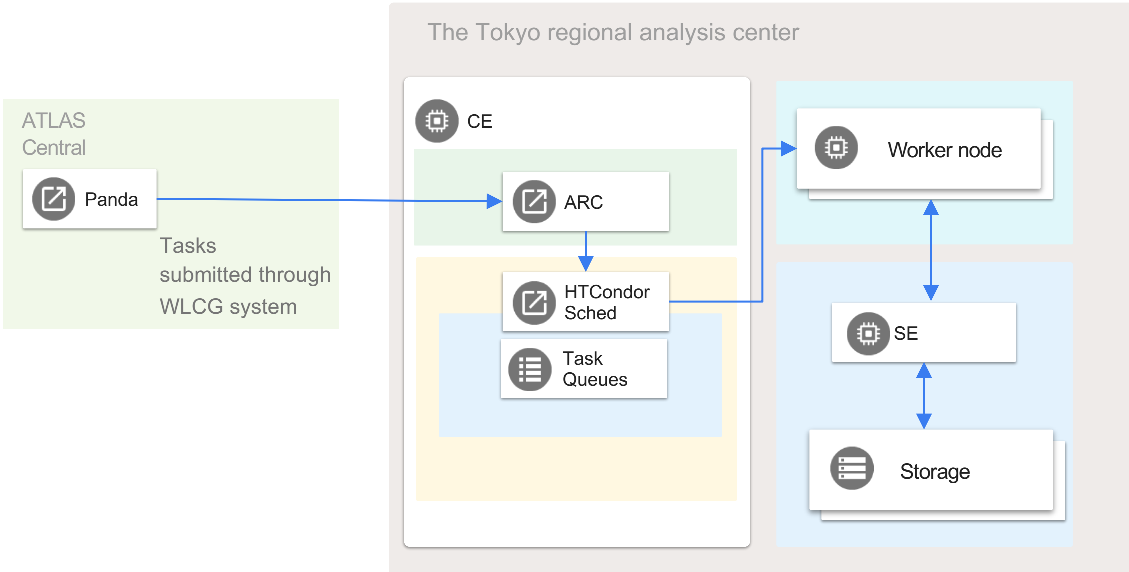
System	Core(vCPU)	CPU	SPECInt/core	HEPSPEC	ATLAS simulation 1000events (hours)
TOKYO system: HT off	32	Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz	46.25	18.97	5.19
TOKYO system: HT on	64	Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz	N/A	11.58	8.64
GCE (Broadwell)	8	Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz	(39.75)	12.31	9.32
GCE (Broadwell)	1	Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz	(39.75)	22.73	N/A
GCE (Skylake)	8	Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz	(43.25)	12.62	9.27

- SPECInt (SPECint_rate2006):
  - Local system: Dell Inc. PowerEdge M640
  - GCE(Google Compute Engine)'s value were taken from Dell system with same corresponding CPU
    - GCE (Broadwell): Dell Inc PowerEdge R630
    - GCE (Skylake): Dell Inc. PowerEdge M640
- ATLAS simulation: Multi process job 8 processes
  - For 32 and 64 core machine, 4 and 8 parallel jobs were run to fill cores, respectively

→ Broadwell and Skylake show similar specs
  → Costs are same. But if instances are restricted to Skylake, instances will be preempted more
  → Better not to restrict CPU generation for preemptible instances
→ GCE spec is ~half of TOKYO system

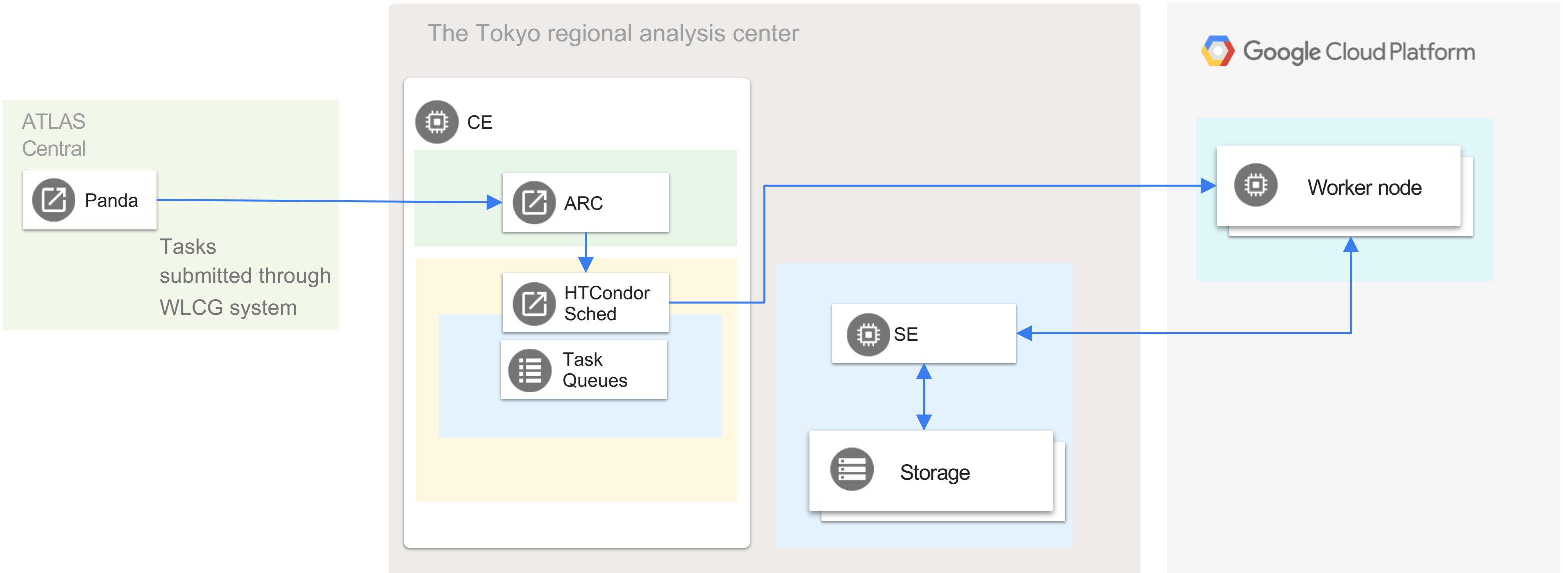
- **Preemptible Instance**
  → Shut down every 24 hours
  → Could be shut down before 24 hours depending on the system condition
  → The cost is ~1/3

Current Our System



- Panda: ATLAS job management system, using WLCG framework
- ARC-CE: Grid front-end
- HTCondor: Job scheduler

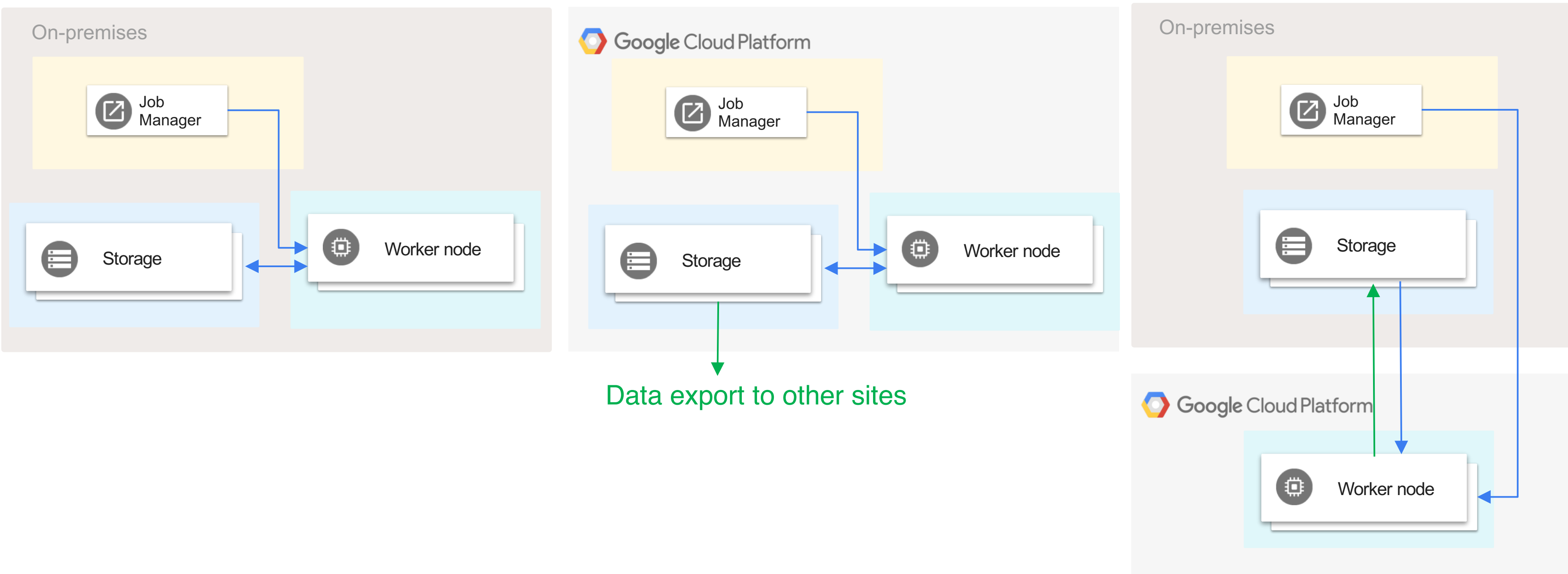
Hybrid System



- Some servers need certifications for WLCG
  →There is a political issue to deploy such servers on cloud
    →No clear discussions have been done for the policy of such a case
- Cost of storage is high
  →Additional cost to extract data
- Only worker nodes (and some supporting servers) were deployed on cloud, and other services are in on-premises
  →Hybrid system

Cost Estimation

Full on-premises system | Full cloud system | Hybrid System



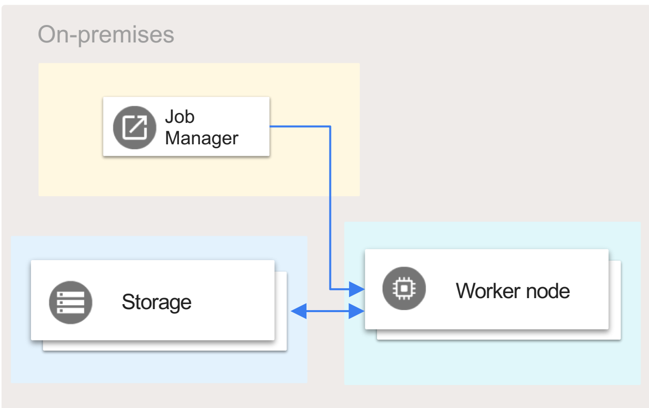
- Estimated with Dell machines
- 10k cores, 3GB/core memory, 35GB/core disk: $5M
- 16PB storage: $1M
- Power cost: $20k/month
  → For 3 years usage: ~$200k/month (+Facility/Infrastructure cost, Hardware Maintenance cost, etc…)

- For GCP, use 20k to have comparable spec
  → Use Preemptible Instance
- 8PB storage which is used at ICEPP for now
- Cost to export data from GCP

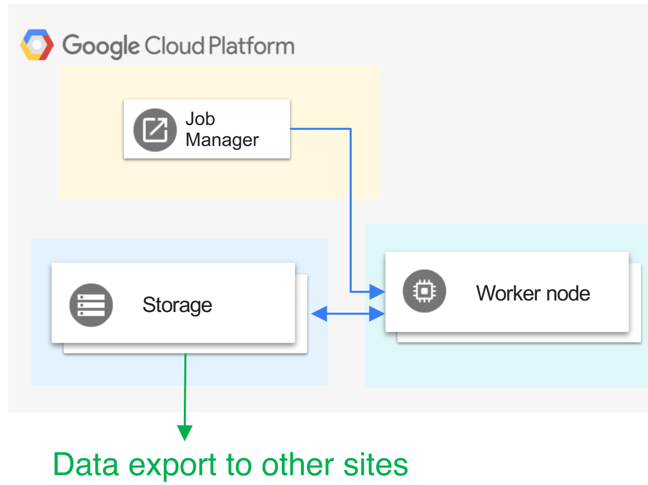
https://cloud.google.com/compute/pricing
https://cloud.google.com/storage/pricing

Cost Estimation

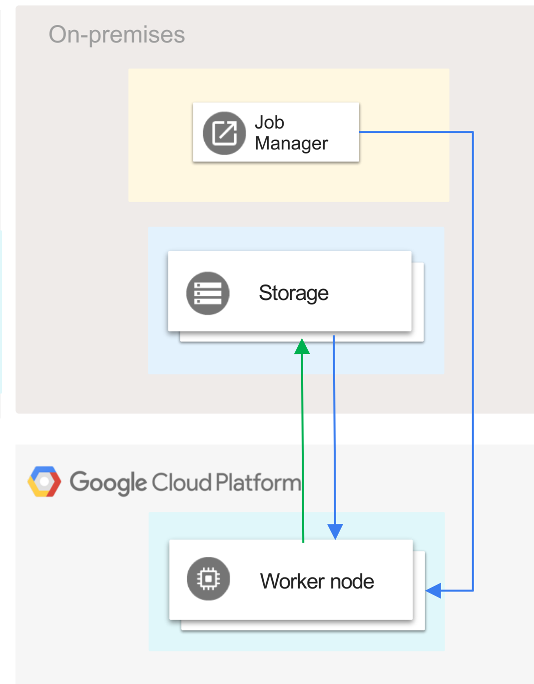
Full on-premises system



Full cloud system



Hybrid System



- Estimated with Dell machines
- 10k cores, 3GB/core memory, 35GB/core disk: $5M
- 16PB storage: $1M
- Power cost: $20k/month

  → For 3 years usage: ~$200k/month (+Facility/Infrastructure cost, Hardware Maintenance cost, etc…)

Resource	Cost/month
vCPU x20k	$130k
3GB x20k	$52k
Local Disk 35GBx20k	$28k
Storage 8PB	$184k
Network Storage to Outside 600 TB	$86k

Total cost: $480k/month

Resource	Cost/month
vCPU x20k	$130k
3GB x20k	$52k
Local Disk 35GBx20k	$28k
Network GCP WN to ICEPP Storage 300 TB	$43k

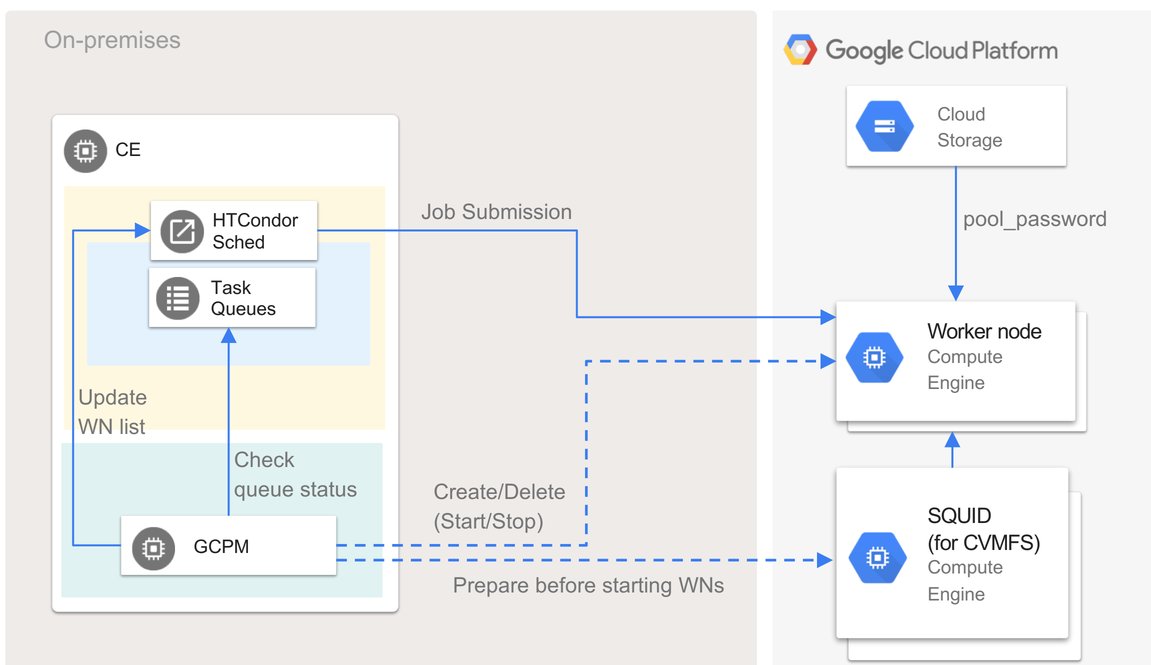
Total cost: $243k/month + on-premises costs (storage $30k/month + others)

Technical Points on HTCndor with GCP

- No swap is prepared as default:
  → No API option is available, need to make swap by a startup script
- Memory must be 256MB x N
- yum-cron is installed and enabled by default
  → Better to disable to manage packages (and for performance)
- Preemptible machine
  → The cost is ~1/3 of the normal instance
  → It is stopped after 24 h running
    → It can be stopped even before 24 h by GCP (depends on total system usage)
    → **Better to run only 1 job for 1 instance**
- Instances are under VPN
  → They don't know own external IP address
  → Use HTCndor Connection Brokering (CCB)
    → ***CCB_ADDRESS = $(COLLECTOR_HOST)***
- Instance's external address is changed every time it is started
  → Static IP address is available, but it needs additional cost
  → To manage worker node instance on GCP, a management tool has been developed:
    → **Google Cloud Platform Condor Pool Manager (GCPM)**

Google Cloud Platform Condor Pool Manager

- https://github.com/mickaneda/gcpm
  → Can be installed by pip:
  → ***$ pip install gcpm***
- Manage GCP resources and HTCondor's worker node list



Google Cloud Platform Condor Pool Manager

- Run on HTCondor head machine

  → Prepare necessary machines before starting worker nodes

  → Create (start) new instance if idle jobs exist

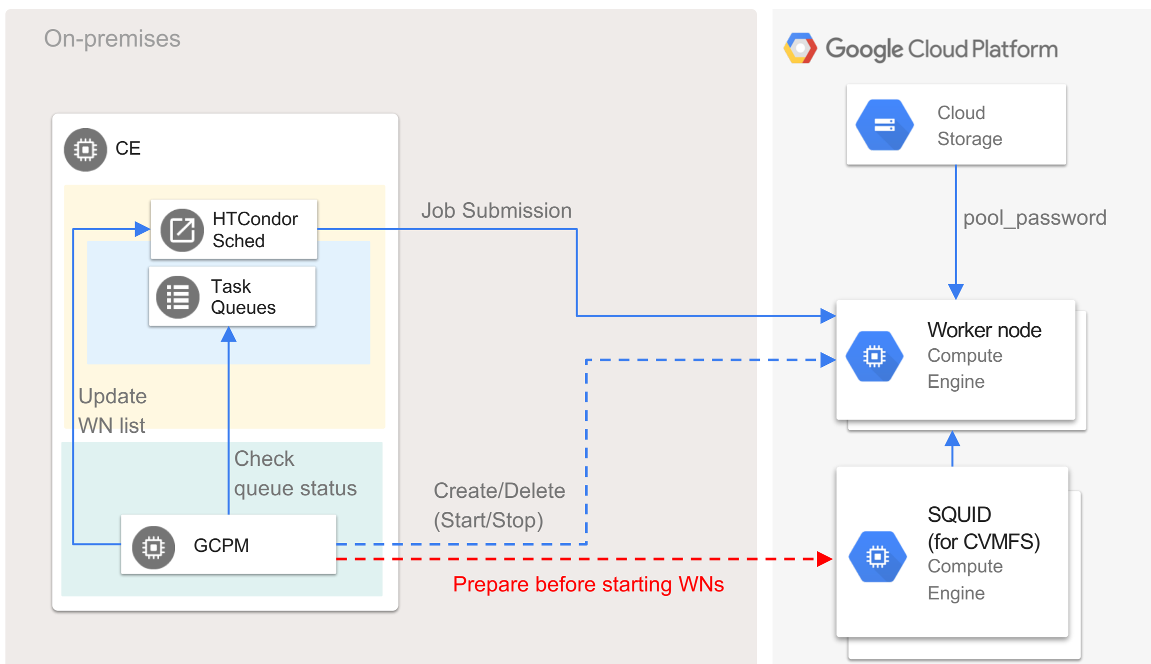
  → Update WN list of HTCondor

  → Job submitted by HTCondor

  → Instance's HTCondor startd will be stopped at 10min after starting

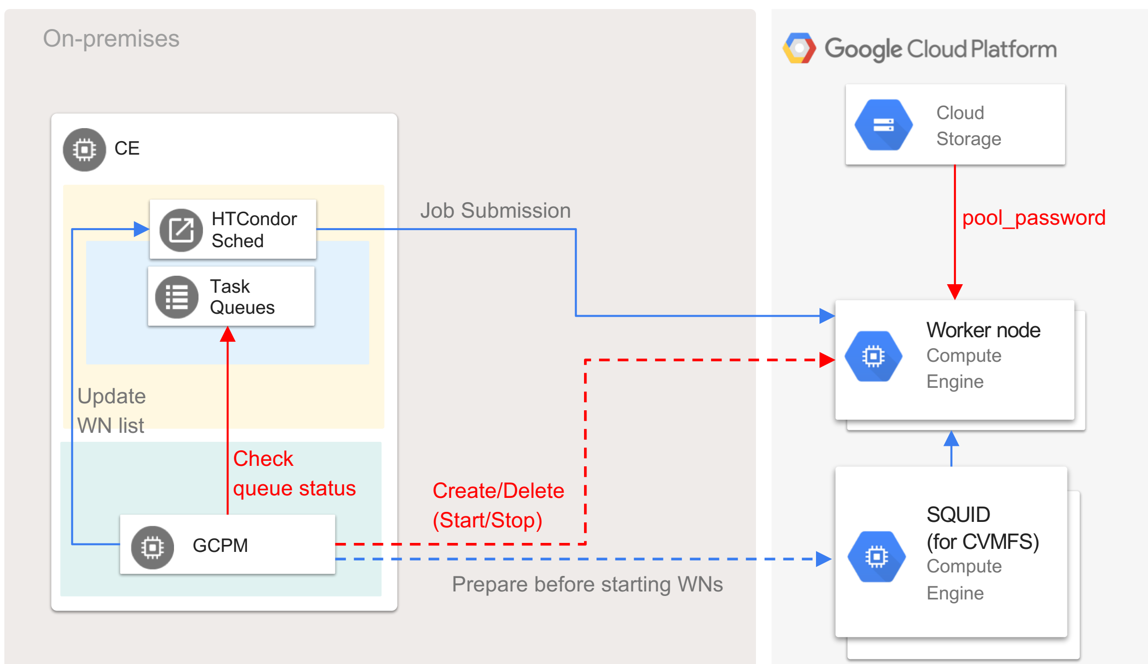
    → ~ only 1 job runs on instance, and it is deleted by GCPM

      → Effective usage of preemptible machine



Google Cloud Platform Condor Pool Manager

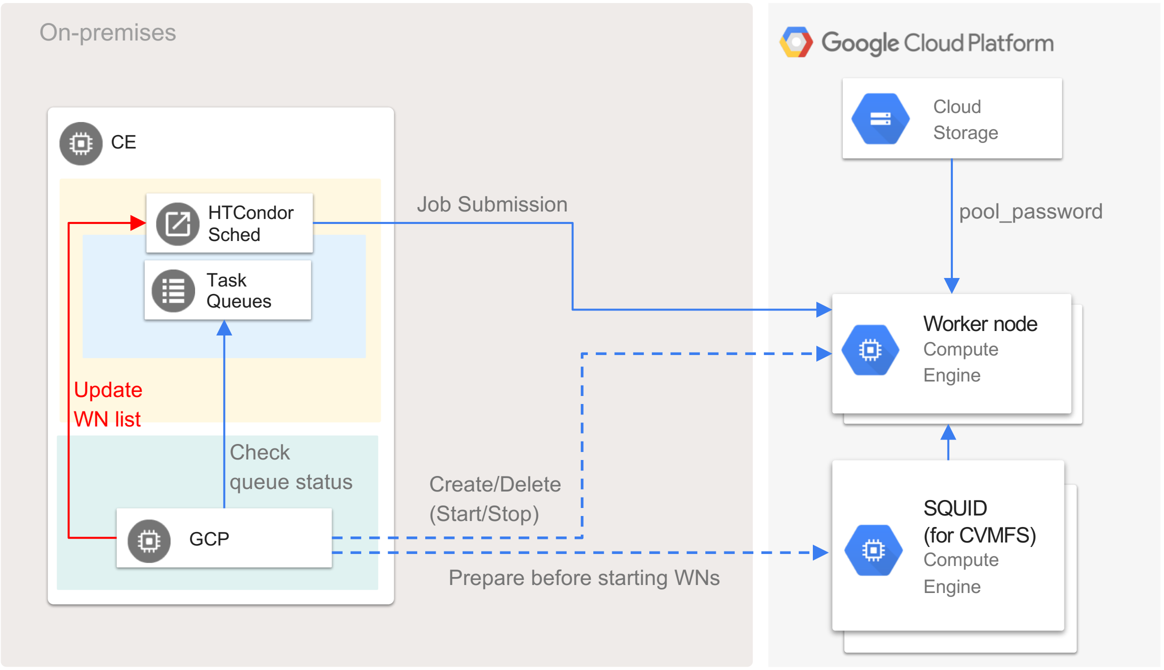
- Run on HTCondor head machine
  - →Prepare necessary machines before starting worker nodes
  - →Create (start) new instance if idle jobs exist
  - →Update WN list of HTCondor
  - →Job submitted by HTCondor
  - →Instance's HTCondor startd will be stopped at 10min after starting
    - → ~ only 1 job runs on instance, and it is deleted by GCPM
      - → Effective usage of preemptible machine



pool_password file for the authentication is taken from storage by startup script

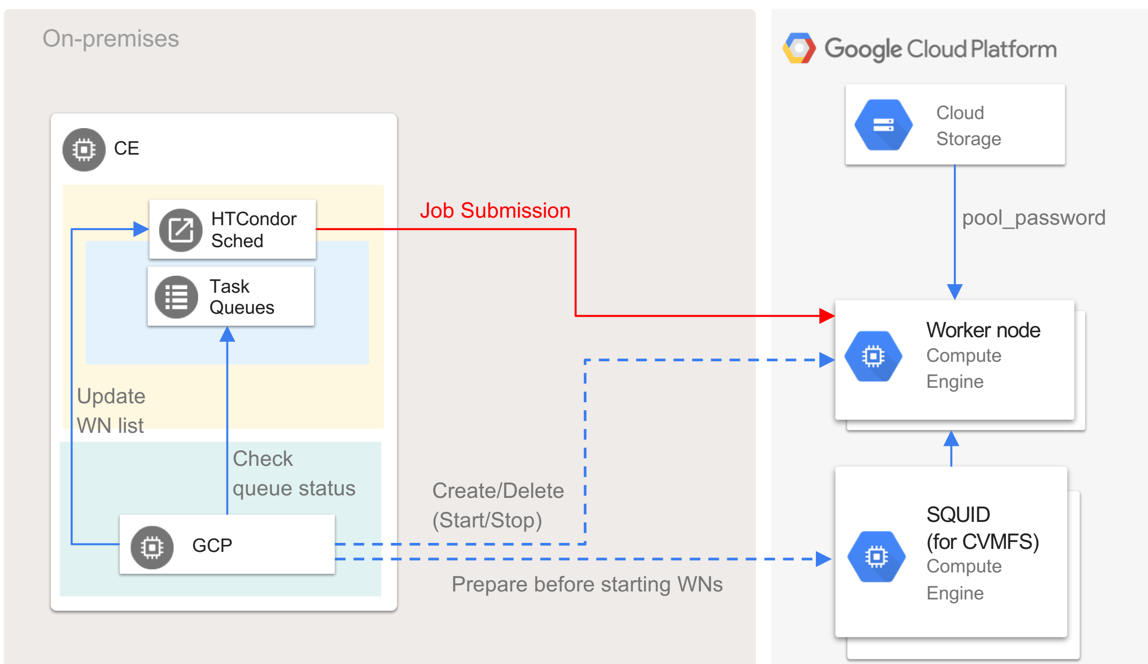
Google Cloud Platform Condor Pool Manager

- Run on HTCondor head machine
  - → Prepare necessary machines before starting worker nodes
  - → Create (start) new instance if idle jobs exist
  - → Update WN list of HTCondor
  - → Job submitted by HTCondor
  - → Instance's HTCondor startd will be stopped at 10min after starting
    - → ~ only 1 job runs on instance, and it is deleted by GCPM
      - → Effective usage of preemptible machine



Google Cloud Platform Condor Pool Manager

- Run on HTCondor head machine
  - → Prepare necessary machines before starting worker nodes
  - → Create (start) new instance if idle jobs exist
  - → Update WN list of HTCondor
  - → Job submitted by HTCondor
  - → Instance's HTCondor startd will be stopped at 10min after starting
    - → ~ only 1 job runs on instance, and it is deleted by GCPM
      - → Effective usage of preemptible machine



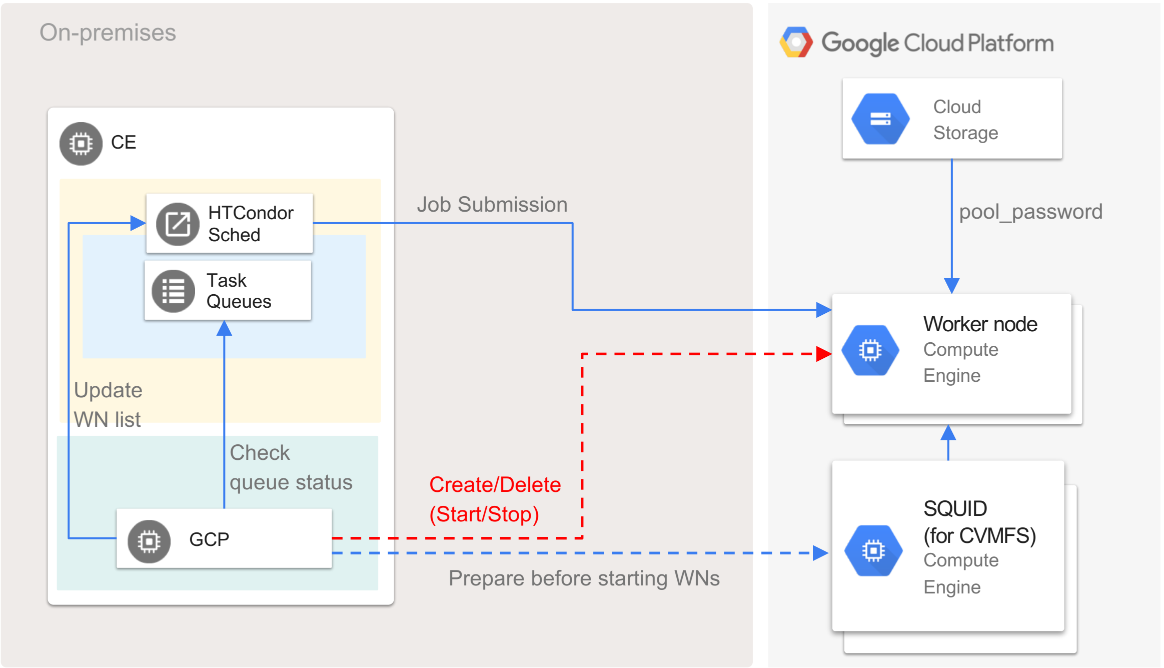
Google Cloud Platform Condor Pool Manager

- Set to execute `*condor_off -peacefull –startd*` after 10min by the startup script for GCE instance
- When a job finished, the instance is removed from `*condor_status*` list
- Then GCPM deletes (sotps) the instance

→ Instance's HTCondor startd will be stopped at 10min after starting

→ ~ only 1 job runs on instance, and it is deleted by GCPM

→ Effective usage of preemptible machine



ARC CE Hacking

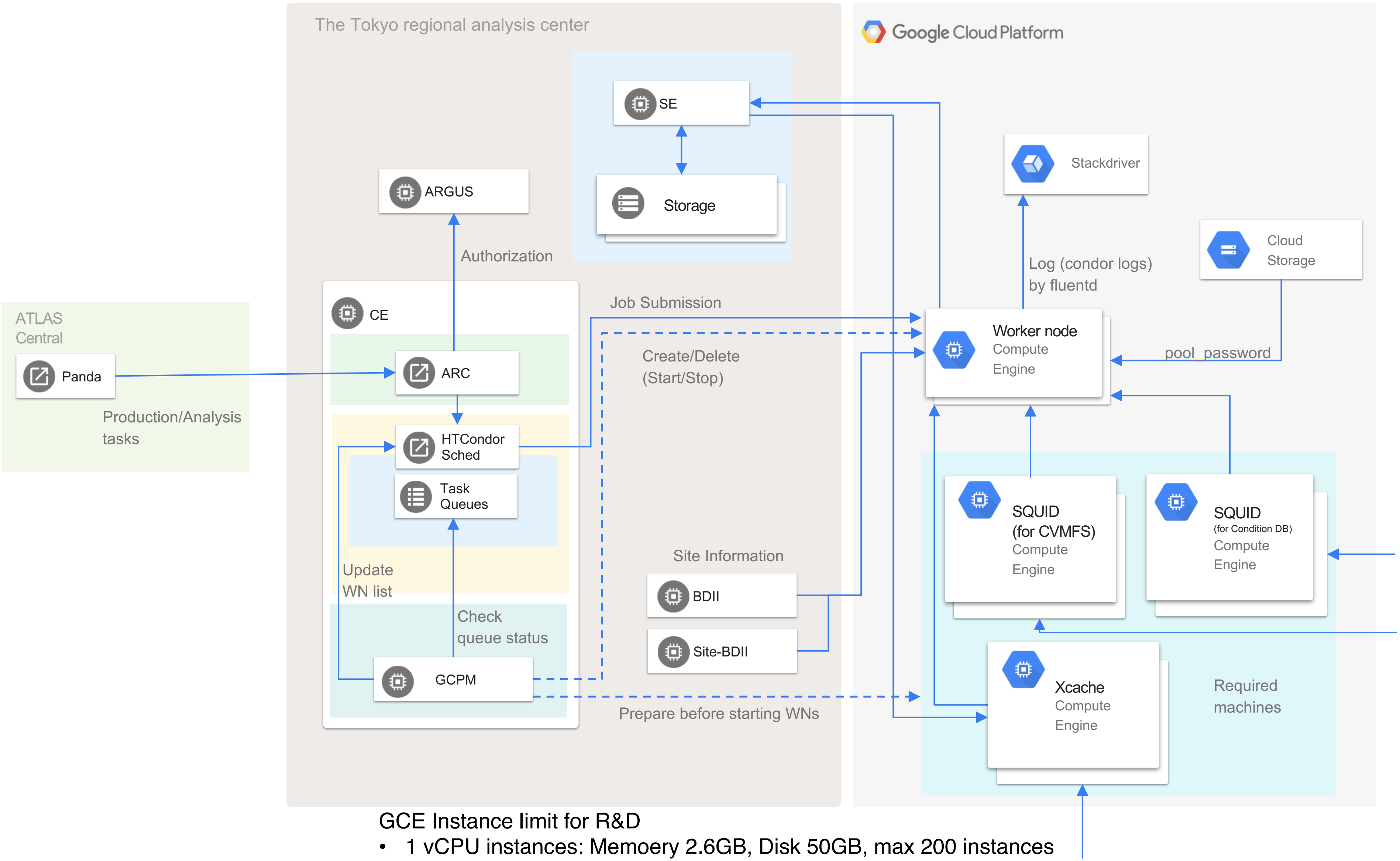
- ARC checks a number of available slots before submitting jobs
  - → If a job specifies a number of CPUs and there are not enough slots, job submission fails
  - → GCP pool has no slot at the start, jobs cannot be submitted
  - → Hack /usr/share/arc/Condor.pm to return non-zero cpus if it is zero

```
#
# returns the total number of nodes in the cluster
#
sub condor_cluster_totalcpus() {
    # List all machines in the pool. Create a hash specifying the
TotalCpus
    # for each machine.
    my %machines;
    $machines{$$_{machine}} = $$_{totalcpus} for @allnodedata;

    my $totalcpus = 0;
    for (keys %machines) {
        $totalcpus += $machines{$_};
    }

    # Give non-zero cpus for dynamic pool
    $totalcpus ||= 100;
    return $totalcpus;
}
```

System for R&D

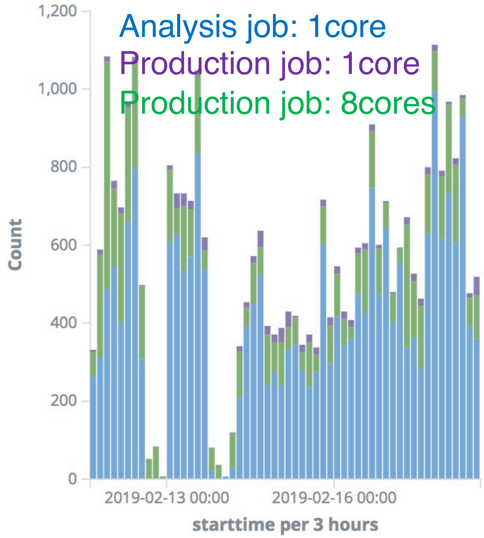


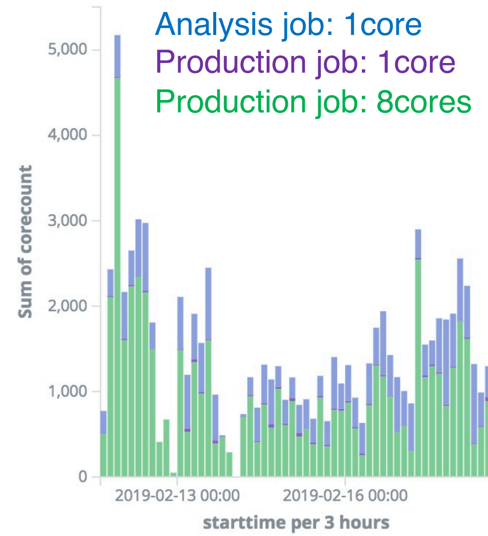
GCE Instance limit for R&D

- 1 vCPU instances: Memoery 2.6GB, Disk 50GB, max 200 instances
- 8 vCPU instances: Memory 19.2GB, Disk 150GB, max 100 instances
  → Total vCPU max: 1000

Jobs Running on GCP

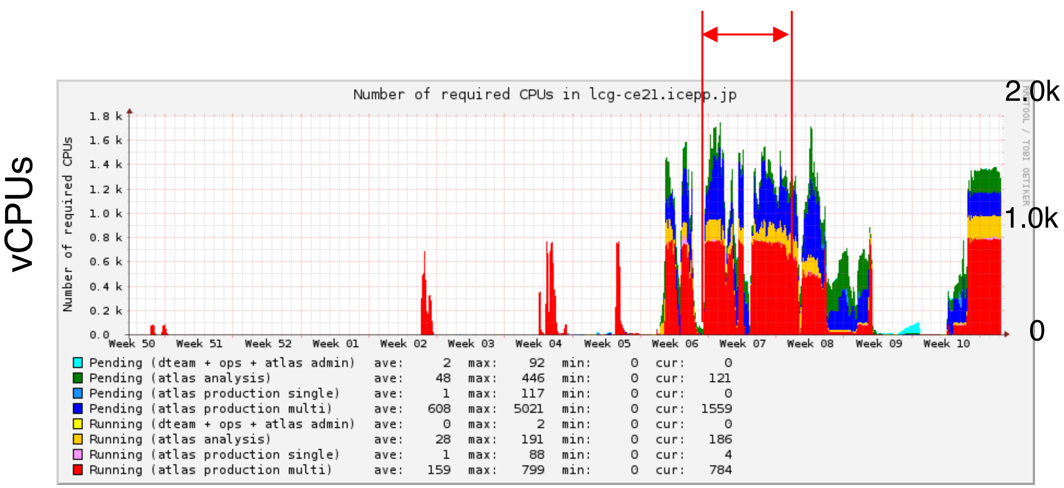
Monitors of job starting time





Number of jobs

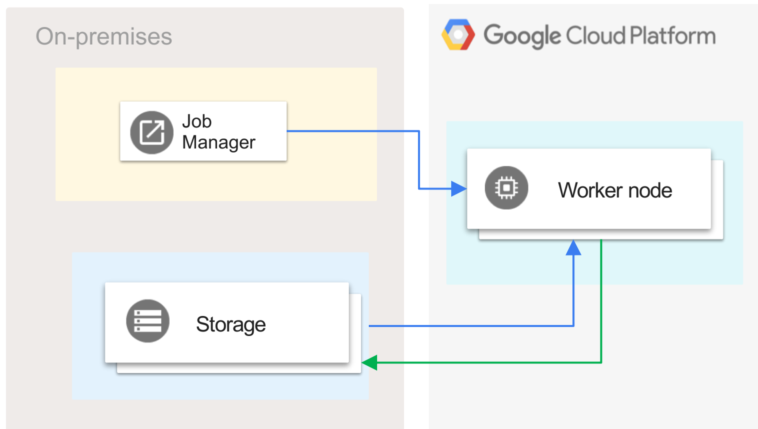
Number of vCPUs



Analysis job: 1core idle
Production job: 8cores idle
Analysis job: 1core running
Production job: 8cores running

HTCondor status monitor

1 Day Real Cost



Hybrid system: 1k cores, 2.4GB/core memory

→ Cost for month (x30), with 20k cores (x20): ~$240k + on-premises costs

1 Day Real Cost (13/Feb)

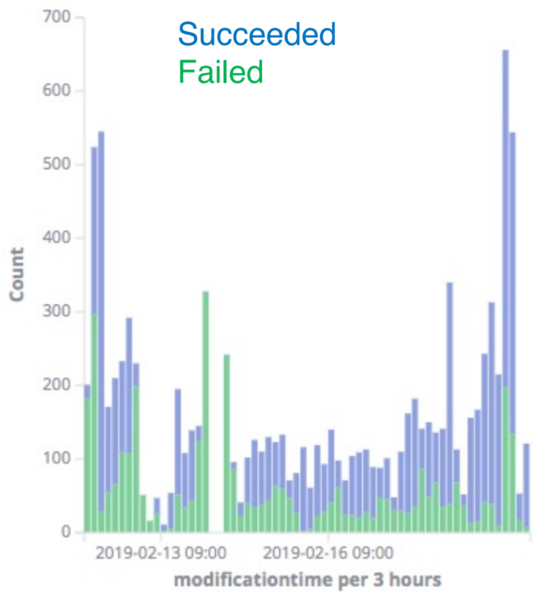
	Usage	Cost/day	x30x20
vCPU (vCPU*hours)	20046	$177	$106k
Memory (GB*hours)	47581	$56	$34k
Disk (GB*hours)	644898	$50	$30k
Network (GB)	559	$78	$47k
Other services		$30	$18k
Total		$391	$236k

vCPU: 1vCPU instances max 200, 8 vCPUs instances max 100
Memory: 2.4 GB/vCPU
Disk: 50GB for 1vCPU instance, 150 GB for 8 vCPUs instance

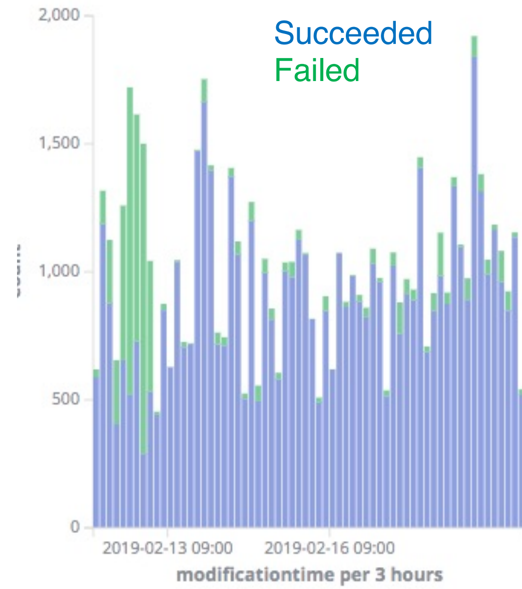
Cost Estimation

Resource	Cost/month
vCPU x20k	$130k
3GB x20k	$42k
Local Disk 35GBx20k	$28k
Network GCP WN to ICEPP Storage 300 TB	$43k
Total	$243k

Failure Rate (Production Jobs)



GCP Worker Nodes
(Production Job)

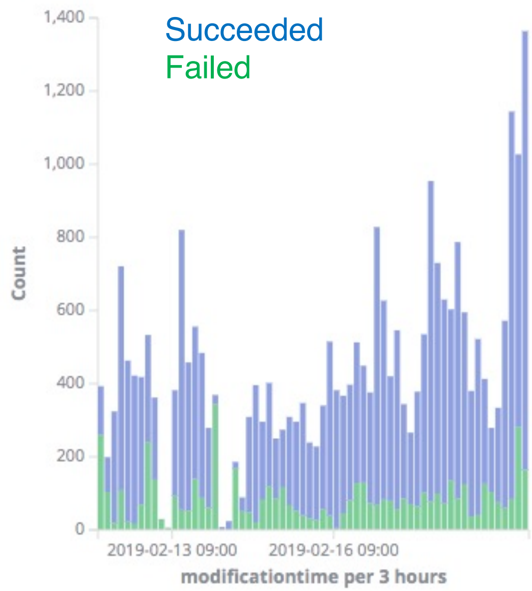


ICEPP Worker Nodes
(Production Job)

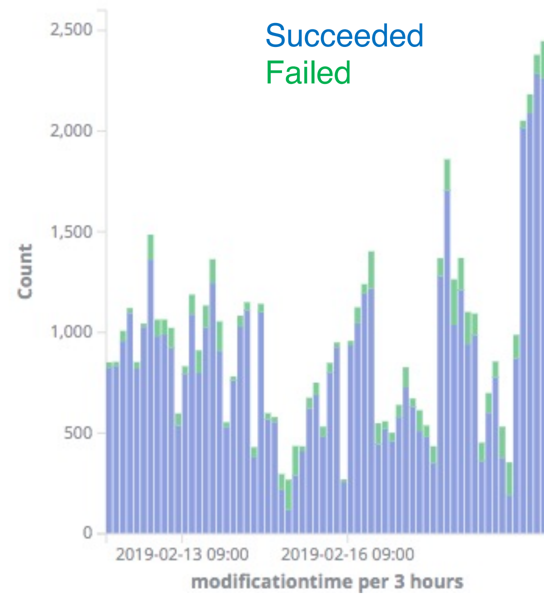
Job Type	Error rate
GCP Production (Preemptible)	35%
GCP Production (Non-Preemptible)	6%
Local Production	11%

Mainly 8 core jobs, long jobs (~10 hours/job)

Failure Rate (Analysis Jobs)



GCP Worker Nodes
(Analysis Job)



ICEPP Worker Nodes
(Analysis Job)

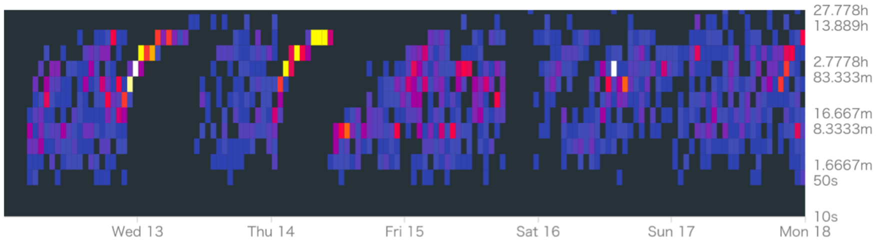
Job Type	Error rate
GCP Analysis (Preemptible)	19%
GCP Analysis (Non-Preemptible)	14%
Local Analysis	8%

Only 1 core job, shorter jobs

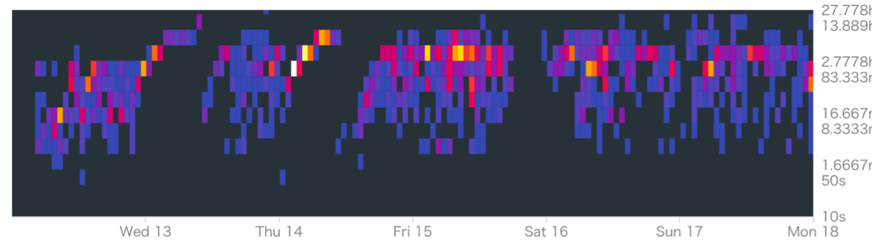
Preemption

1 core instances	8 core instances

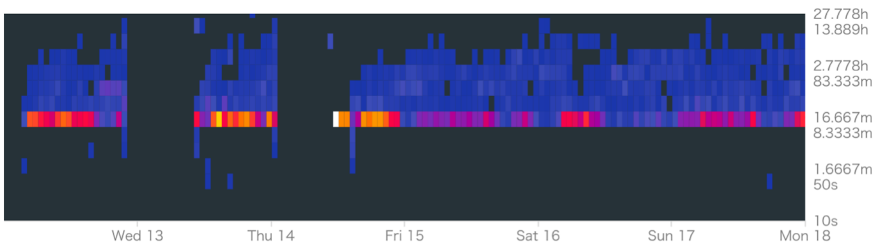
Uptime: 1 core, Preempted



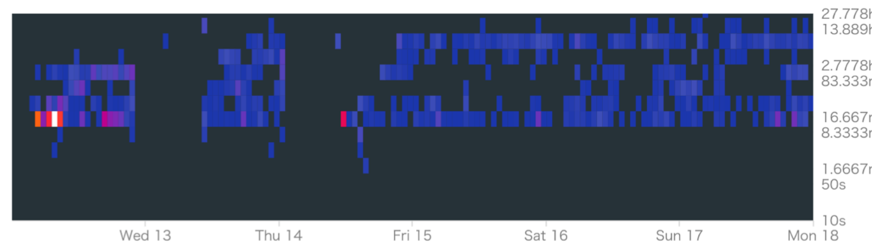
Uptime: 8 cores, Preempted



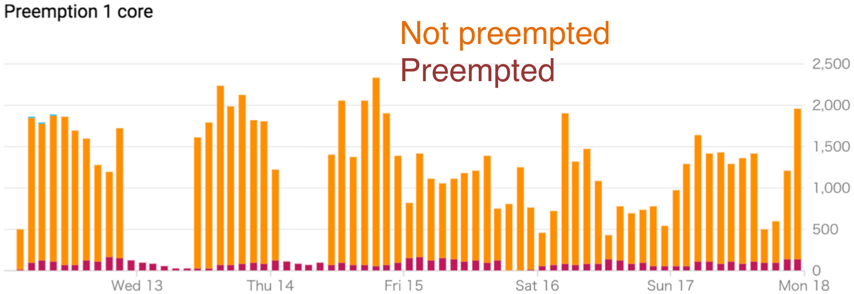
Uptime: 1 core, Not preempted



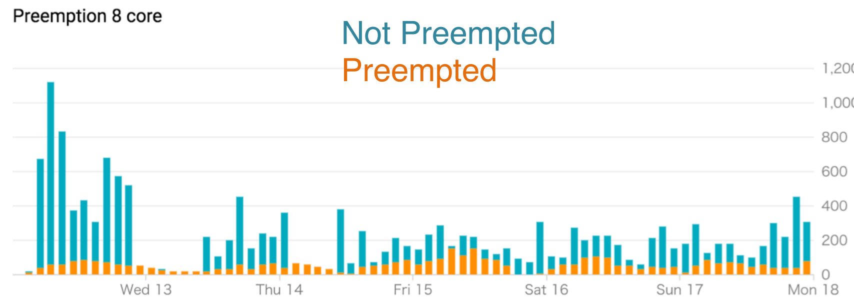
Uptime: 8 cores, Not preempted



Preemption 1 core

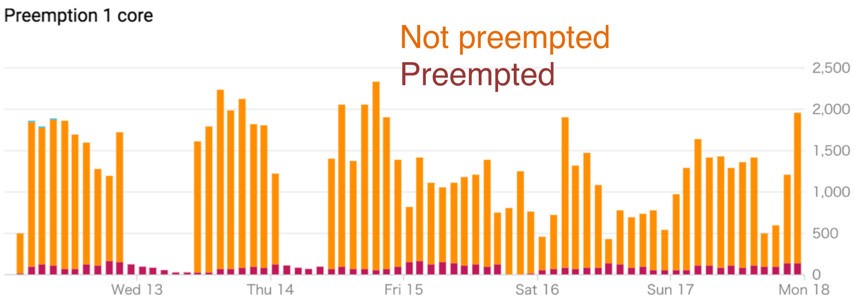


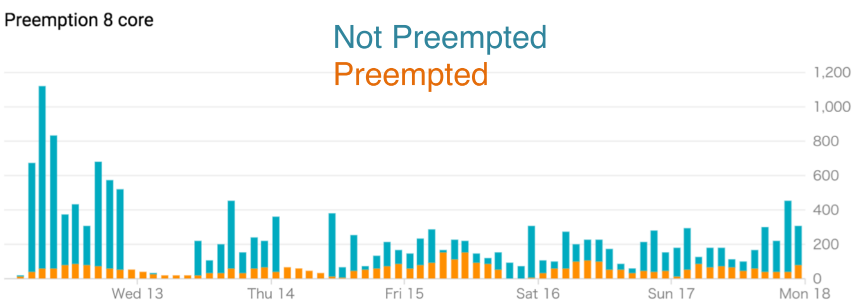
Preemption 8 core



Preemption v.s. Failure jobs

- 5~30 % instances were shut down by Preemption
  →Made failure jobs
- Typically shut down around 3~10 hours
  →Some instances were shutdown before 1 hours running
- More preemptions in 8 core jobs (production: reco/sim) because job running times are longer





Summary

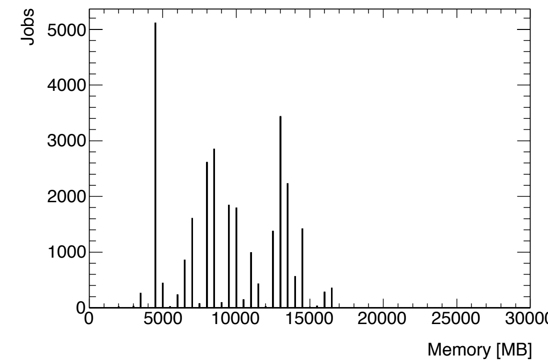
- The cost of GCP is reasonable
  →Same order compared with on-premises, if preemptible instances are used

- Hybrid system with GCPM works on the ATLAS Production System in WLCG
  →HTCondor+GCPM can work for small clusters, too, in which CPUs are always not fully used
    →You need to pay only for what you used

- Failure due to preemptible instances
  →10~30% higher rates compared to jobs in the local nodes
    →The cost performance is still better to use preemptible instances
  →Shorter jobs are less affected

Plan

- Dynamic Memory Assignment



Required memory for 8 core jobs

- Use AWS, Azure, IBM

  →HTCondor natively supports
  AWS worker nodes (condor_annex)

- Non-CPU pools

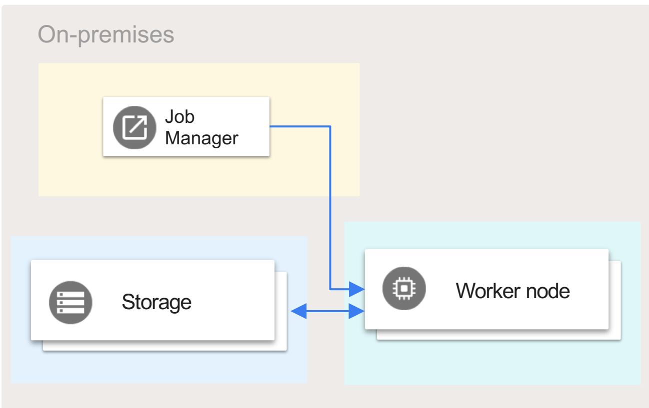
  →GPU, FPGA(AWS),
  TPU (GCP), Inferenita (AWS), Brainwave(Azure)

  →HPC

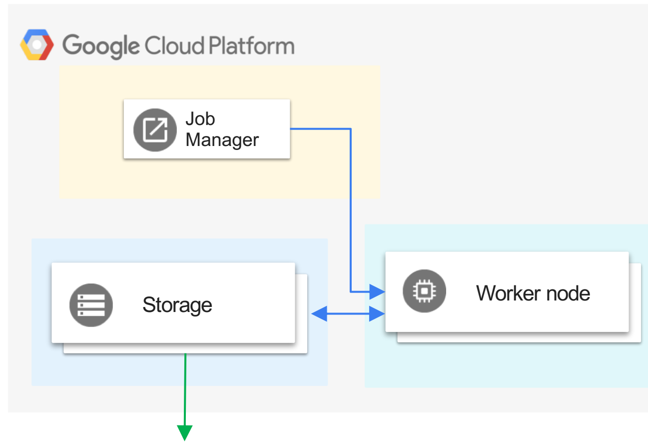
Backup

Cost Estimation

Full on-premises system

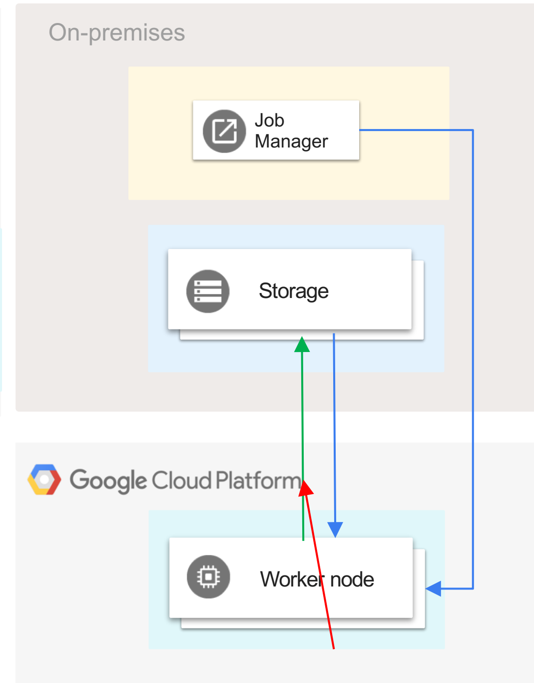


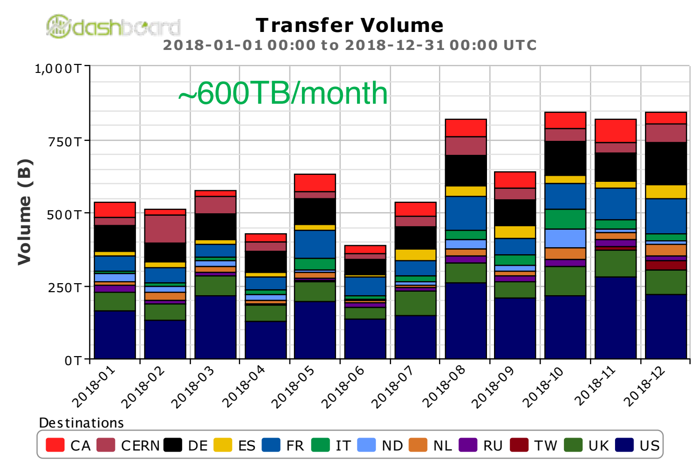
Full on-premises system



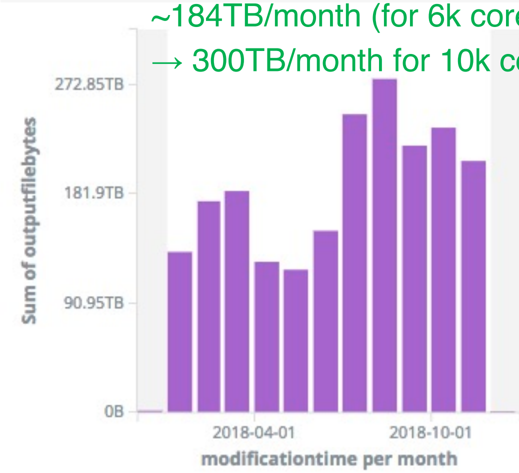
Data export to other sites

Hybrid System





~184TB/month (for 6k core)
→ 300TB/month for 10k core

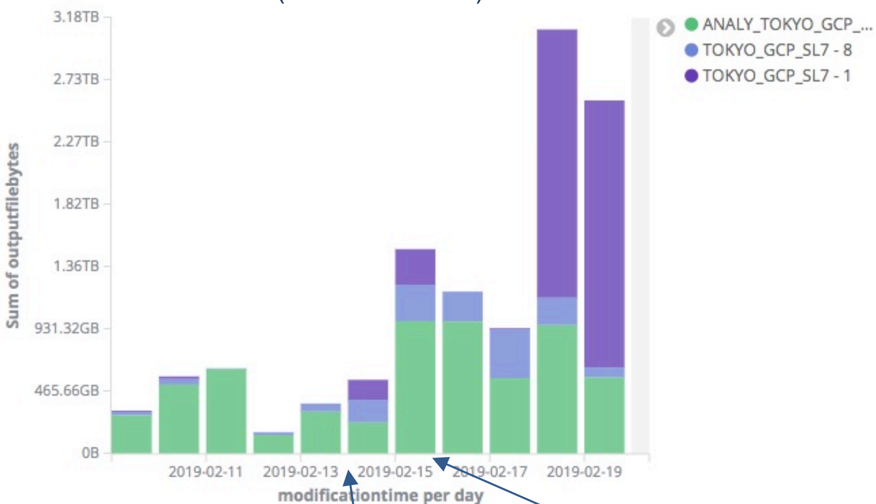


- Estimated with Dell machines
- 10k cores, 3GB/core memory, 35GB/core disk: $5M
- 16PB storage: $1M
- Power cost: $20k/month

→ For 3 years usage: ~$200k/month (+Facility/Infrastructure cost, Hardware Maintenance cost, etc…)

Output file size fluctuation

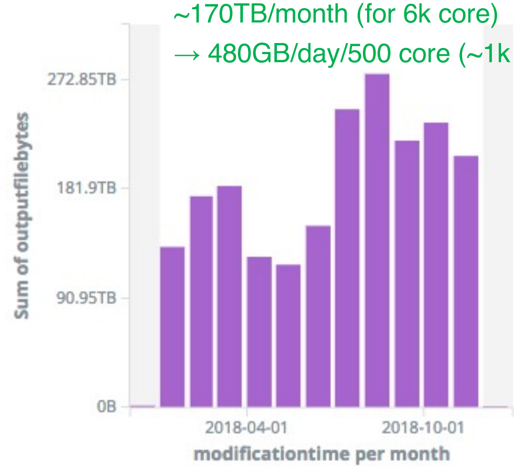
GCP R&D (10/Feb~20/Feb)



TOKYO site (last year)

~170TB/month (for 6k core)

→ 480GB/day/500 core (~1k core@GCP)



1 Day Real Cost (13/Feb)

	Usage	Cost/day	x30x20
vCPU (vCPU*hours)	20046	$177	$106k
Memory (GB*hours)	47581	$56	$34k
Disk (GB*hours)	644898	$50	$30k
Network (GB)	559	$78	$47k
Other services		$30	$18k
Total		$391	$236k

1 Day Real Cost (15/Feb)

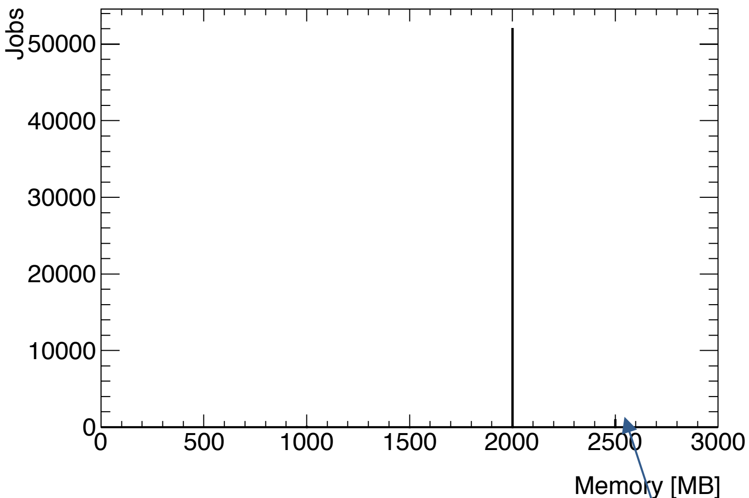
	Usage	Cost/day	x30x20
vCPU (vCPU*hours)	21974	$194	$116k
Memory (GB*hours)	52014	$61	$37k
Disk (GB*hours)	569081	$44	$26k
Network (GB)	1713	$239	$143k
Other services		$28	$17k
Total		$566	$340k

Sum of output file size shows large fluctuation

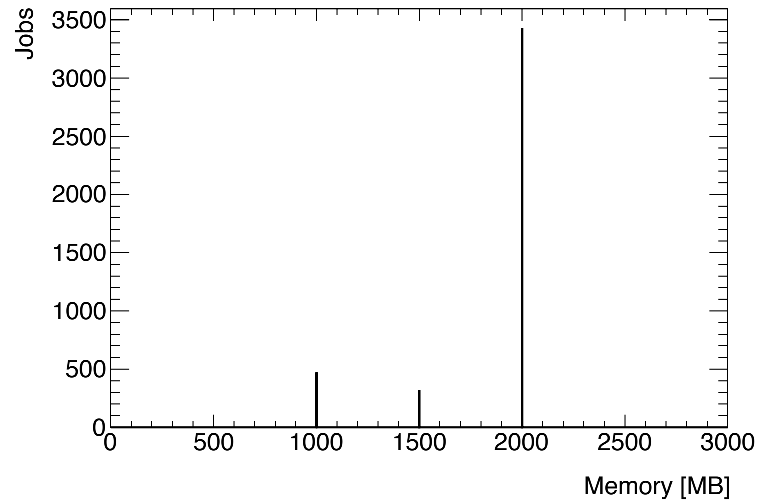
→ Sometimes it becomes ~x5

Required Memory

Extracted from jobs to TOKYO_SL7 and ANALY_TOKYO_SL7
TOKYO site has MCORE_HIMEM queue

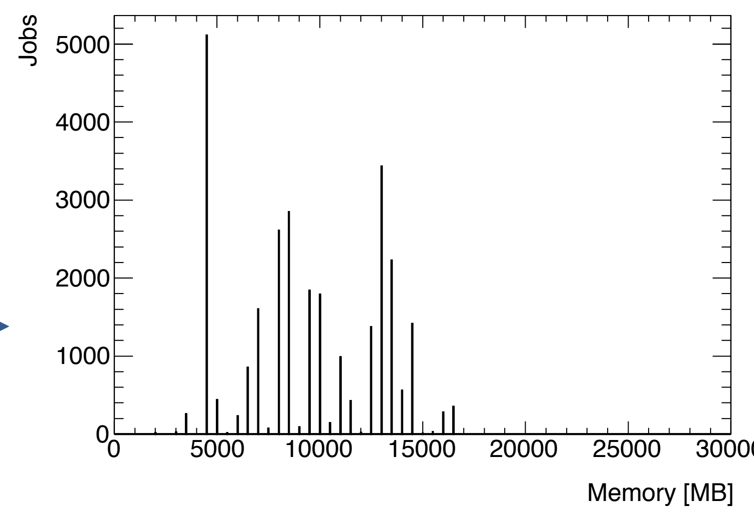


Analysis 1 core



Production 1 core

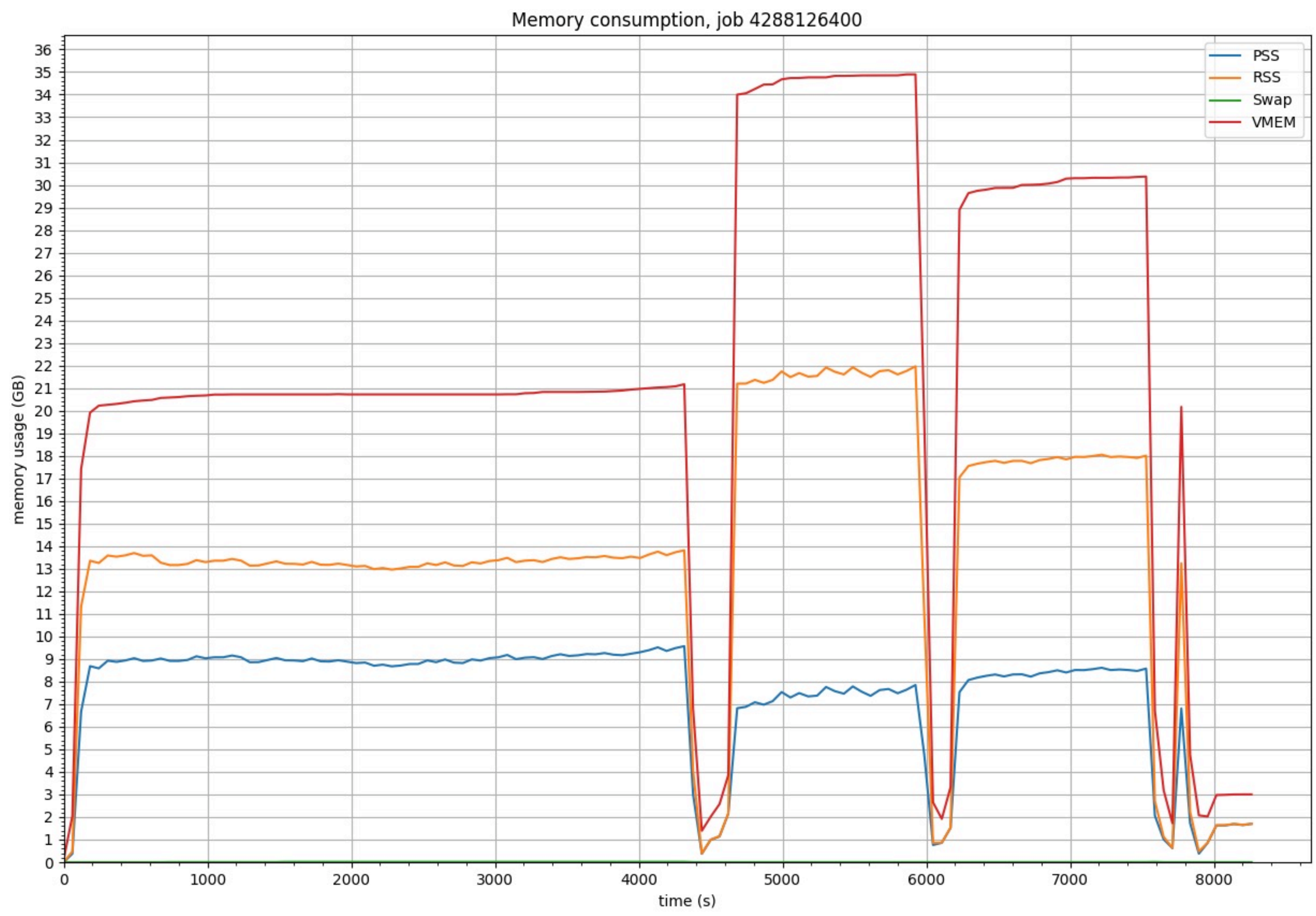
Some 2.5GB memory jobs



Production 8 cores

This includes MCORE_HIMEM queue jobs
Multicore jobs requires smaller memory
Even MCORE_HIMEM jobs requires less than 2GB/core?

MCORE_HIMEM Job Example



https://bigpanda.cern.ch/job?pandaid=4288126400

Payment

- A payment to GCP is done by Credit Card (or through Bank in some countries) as a late payment

  →Our institute system doesn't allow such a payment

  →Only pre-payed, fixed price
- To make such a payment, we use a payment agency

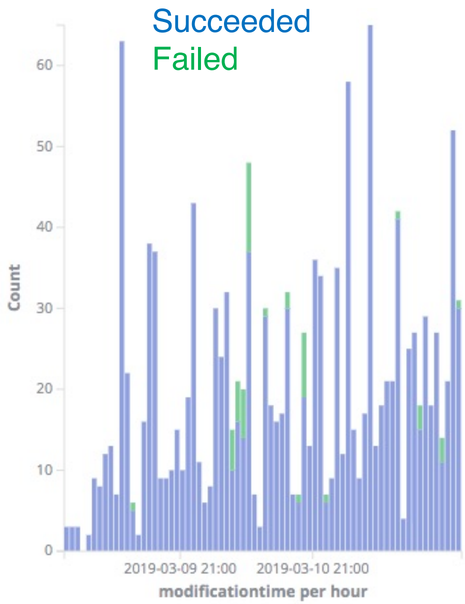
  →There are a lot of agencies in Japan, maybe in other countries, too
- There are some differences in the prices between the (our) agency and GCP direct payment

  →No Sustained Use Discount (up to 60% discount) neither Committed Use Discounts (50%~70%) is applied

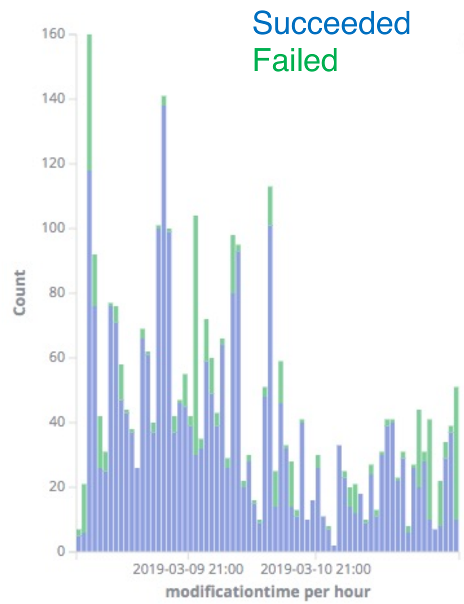
  →Preemptible discount (~1/3 cost) is same as direct

  →Original discount (~a few %?) is applied

Failure Rate (Non-Preemptible)

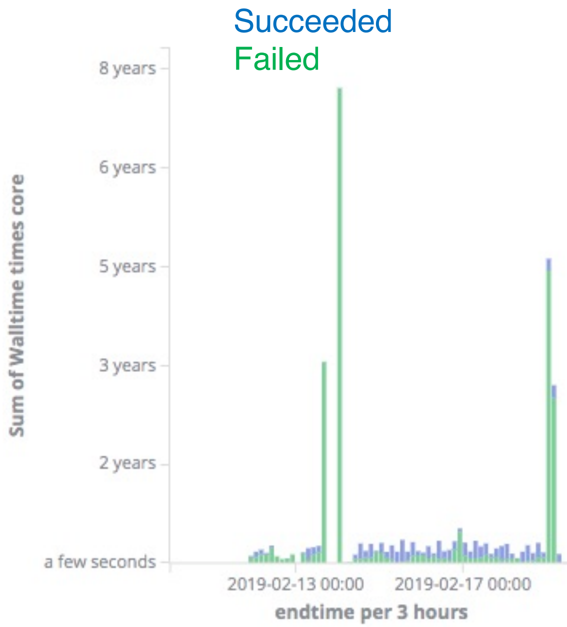


GCP Worker Nodes
(Production Job)

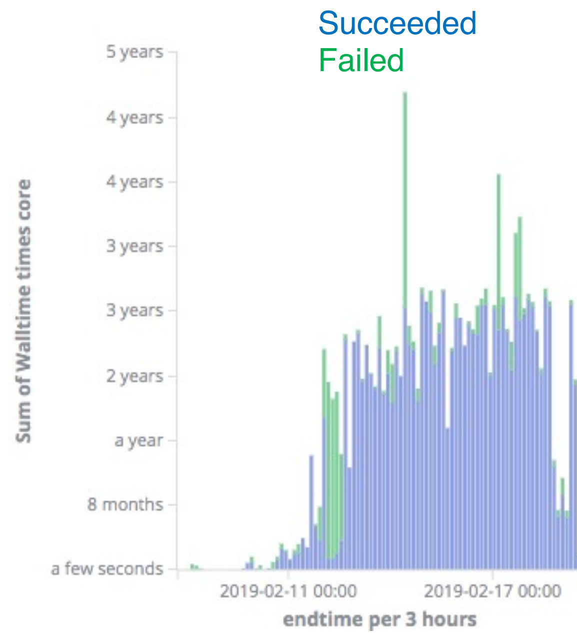


GCP Worker Nodes
(Analysis Job)

Failure Rate (Wall time)

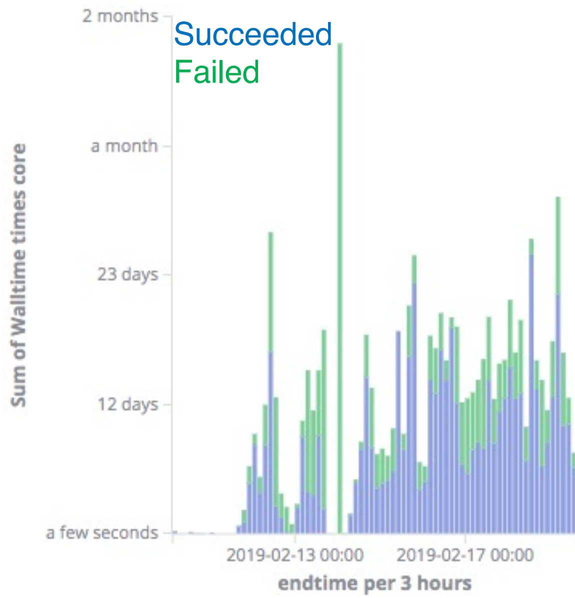


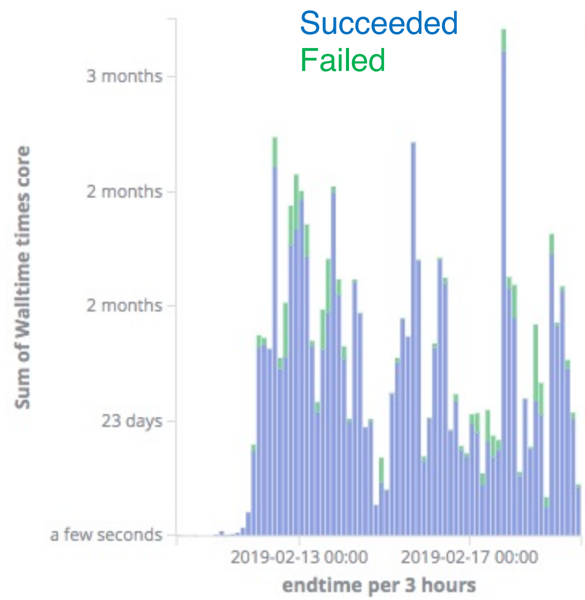
GCP Worker Nodes
(Production Job)



ICEPP Worker Nodes
(Production Job)

Failure Rate (Wall time)



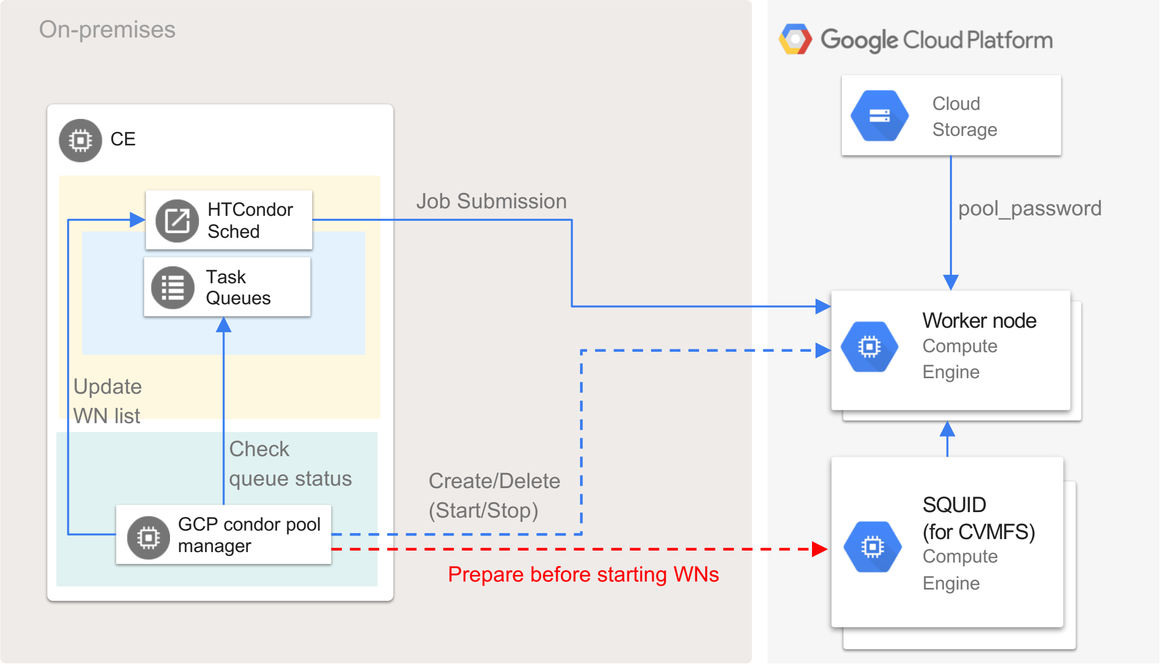


GCP Worker Nodes
(Analysis Job)

ICEPP Worker Nodes
(Analysis Job)

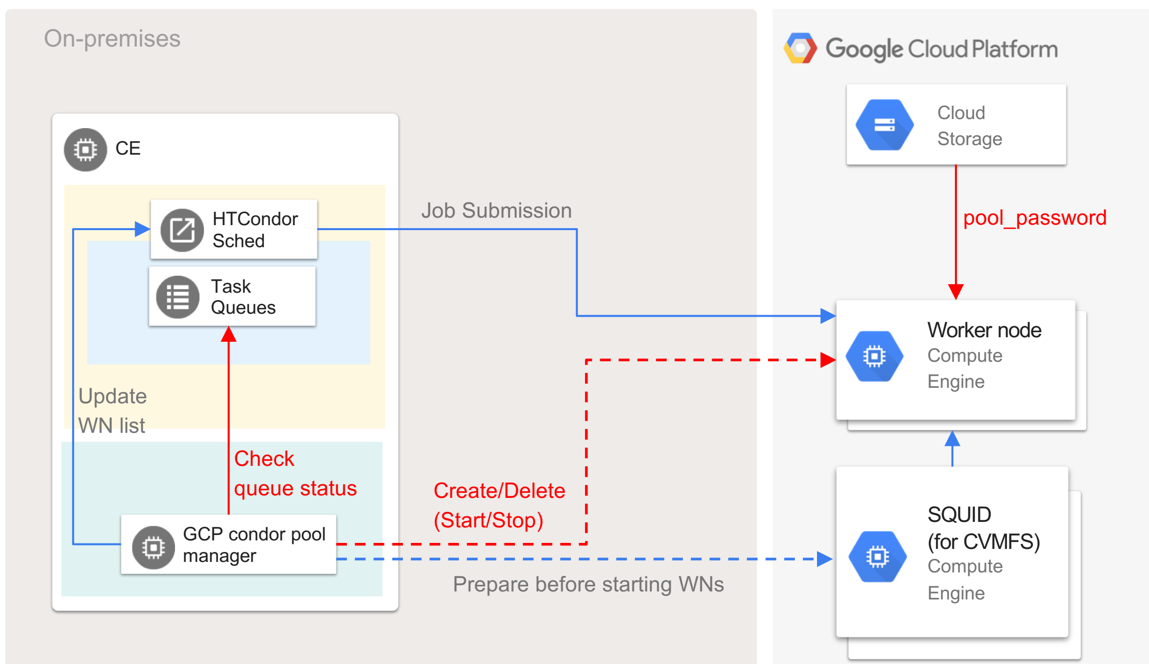
Google Cloud Platform Condor Pool Manager

- Run on HTCondor head machine
  - → Prepare necessary machines before starting worker nodes
  - → Create (start) new instance if idle jobs exist
  - → Update WN list of HTCondor
  - → Job submitted by HTCondor
  - → Instance's HTCondor startd will be stopped at 10min after starting
    - → ~ only 1 job runs on instance, and it is deleted by GCPM
      - → Effective usage of preemptible machine



Google Cloud Platform Condor Pool Manager

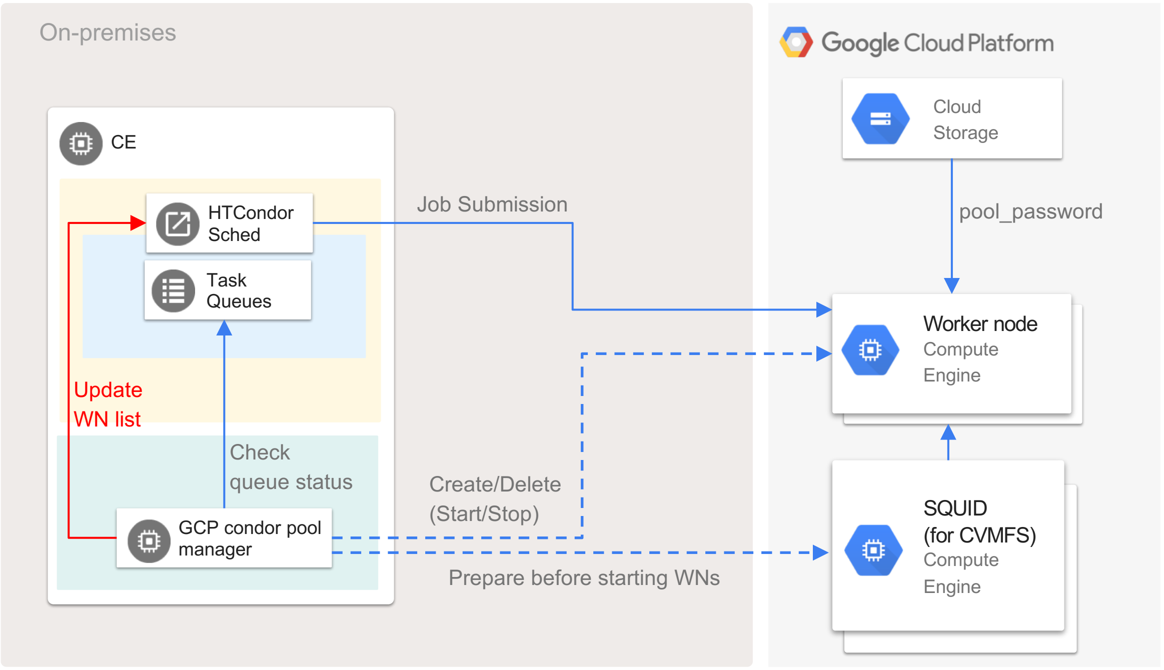
- Run on HTCondor head machine
  - → Prepare necessary machines before starting worker nodes
  - → Create (start) new instance if idle jobs exist
  - → Update WN list of HTCondor
  - → Job submitted by HTCondor
  - → Instance's HTCondor startd will be stopped at 10min after starting
    - → ~ only 1 job runs on instance, and it is deleted by GCPM
      - → Effective usage of preemptible machine



pool_password file for the authentication is taken from storage by startup script

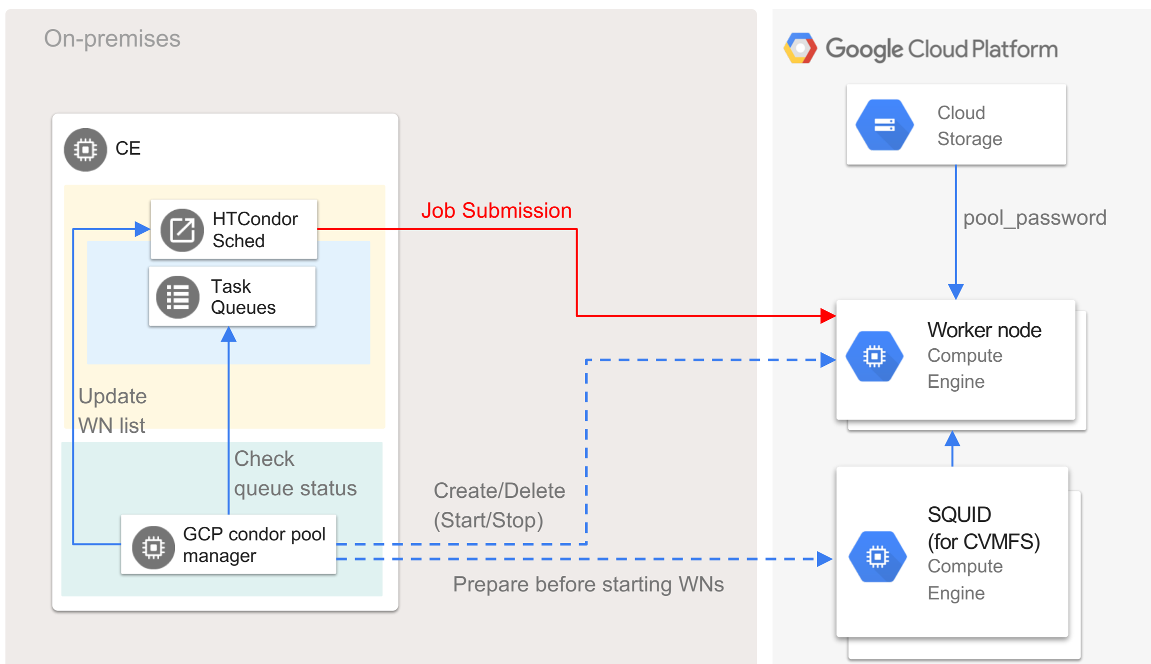
Google Cloud Platform Condor Pool Manager

- Run on HTCondor head machine
  - →Prepare necessary machines before starting worker nodes
  - →Create (start) new instance if idle jobs exist
  - →Update WN list of HTCondor
  - →Job submitted by HTCondor
  - →Instance's HTCondor startd will be stopped at 10min after starting
    - → ~ only 1 job runs on instance, and it is deleted by GCPM
      - → Effective usage of preemptible machine



Google Cloud Platform Condor Pool Manager

- Run on HTCondor head machine
  - → Prepare necessary machines before starting worker nodes
  - → Create (start) new instance if idle jobs exist
  - → Update WN list of HTCondor
  - → Job submitted by HTCondor
  - → Instance's HTCondor startd will be stopped at 10min after starting
    - → ~ only 1 job runs on instance, and it is deleted by GCPM
      - → Effective usage of preemptible machine



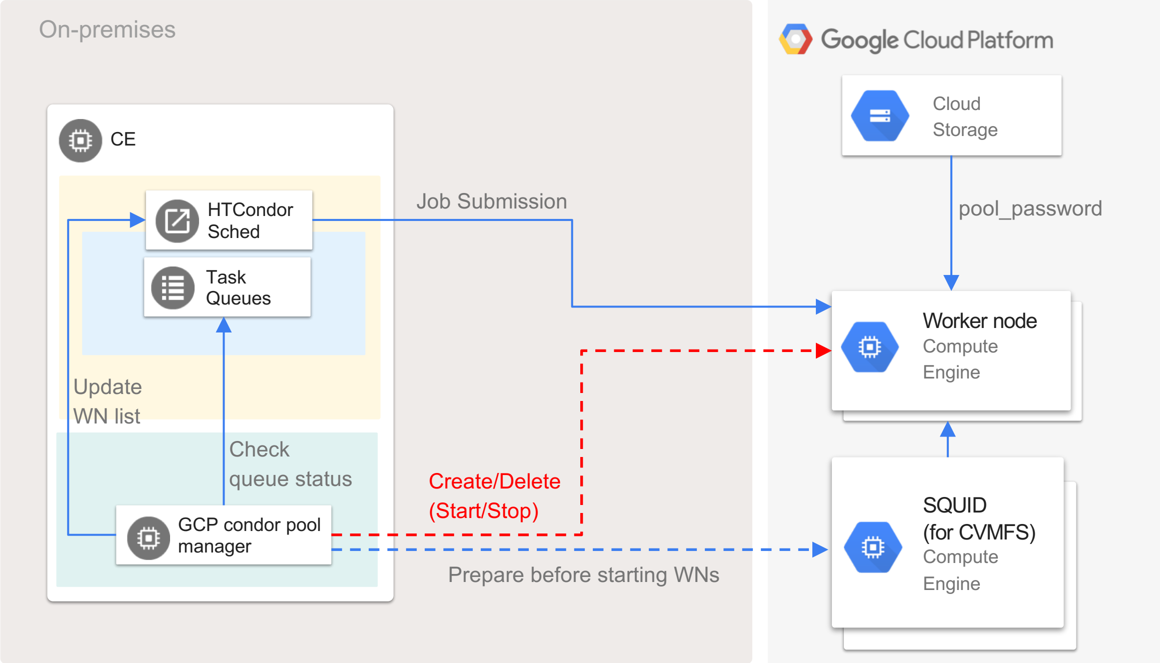
Google Cloud Platform Condor Pool Manager

- Set to execute `*condor_off -peacefull –startd*` after 10min by the startup script for GCE instance
- When a job finished, the instance is removed from `*condor_status*` list
- Then GCPM deletes (sotps) the instance

→ Instance's HTCondor startd will be stopped at 10min after starting

→ ~ only 1 job runs on instance, and it is deleted by GCPM

→ Effective usage of preemptible machine



Other Features of GCPM

- Configuration files:
  →YAML format
- Machine options are fully customizable
- Can handle instances with different number of cores
- Max core in total, max instances for each number of cores
- Management of other than GCE worker nodes
  → Static worker nodes
  → Required machines
  → Working as an orchestration tool
- Test account
- Preemptible or not
- Reuse instances or not
- Pool_password file management
- Puppet files are available for
  →GCPM set
  →Example worker node/head node for GCPM
  →Example frontier squid proxy server at GCP

Evicted Jobs

- Some of failed jobs' Panda log show an error like:
  - →already running elsewhere - aborting
- At condor logs, these jobs were evicted and resubmitted:
  - →ShadowLog:
    12/14/18 11:03:22 (10186.0) (2279157): Job 10186.0 is being evicted from gcp-wn-8core-0008.c.grid-test-204503.internal
- Panda system can not manage such a case
  - →Because the first connection was not closed correctly and is remained
- Such eviction happens in our local worker nodes
  - →But very small rate (< 1%)
- It could be connectability between Head node and WN node
  - →Preemption can make it
  - →Some non-preempted instance also showed it
    - →Could be failing to extract preempted flag?

Develop Environment for Python-GCPM

Package manager: Poetry
CLI: made with python-fire
License: Apache 2.0

Local Machine

Secret files are encrypted by git-crypt
travis encrypt-file is also used for travis job
(service account file for gcp, etc…)

GitHub

Tests by pytest
On Ubuntu Xenial
For python 2.7, 3.5, 3.6, 3.7

pytest-cov result
(in gh-pages branch)

Travis CI

The Python Package Index (PyPI)
($ pip install gcpm)

Develop Environment for Python-GCPM

Package manager: Poetry
CLI: made with python-fire
License: Apache 2.0

Package manager: Poetry

Directory Structure

```
gcpm
 |--|-- pyproject.toml
    |-- src
    |    |-- gcpm
    |    |    |-- __init__.py
    |    |    |-- __main__.py
    |    |    |-- __version__.py
    |    |    |-- cli.py
    |    |    |-- core.py
    |-- tests
    |    |-- __init__.py
    |    |-- conftest.py
    |    |-- data
    |    |    |-- gcpm.yml
    |    |    |-- service_account.json
    |    |-- test_cli.py
```

- Package initialization
- Management of package dependencies
- Build & Publish to PyPi
- Automatic virtualenv management
- Easy to make CLI tool

```
$ poetry init                   # Initialize package
$ poetry add fire               # Add fire to dependencies
$ poetry run gcpm version       # Run gcpm in virtualenv
$ poetry run pytest             # Run pytest in virtualenv
$ poetry publish –build         # Build and publsh to PyPi
```

The Python Package Index (PyPI)
($ pip install gcpm)

Develop Environment for Python-GCPM

Package manager: Poetry
CLI: made with python-fire
License: Apache 2.0

CLI: made with python-fire

```python
from .core import Gcpm
import fire


class CliObject(object):

    def __init__(self, config=""):
        self.config = config

    def version(self):
        Gcpm.version()

    def run(self):
        Gcpm(config=self.config).run()


def cli():
    fire.Fire(CliObject)

if __name__ == "__main__":
    cli()
```

- Library for automatically generating CLI from absolutely any Python object

```
$ gcpm version
gcpm: 0.2.0
$ gcpm --config /path/to/config run
Starting gcpm
…
```

The Python Package Index (PyPI)
($ pip install gcpm)

Develop Environment for Python-GCPM

python powered

Package manager: Poetry
CLI: made with python-fire
License: Apache 2.0

Secret files are encrypted by git-crypt
travis encrypt-file is also used for travis job
(service account file for gcp, etc…)

GitHub

t
nial
, 3.5, 3.6, 3.7

pytest-cov result
(in gh-pages branch)

Source code at: GitHub

mickaneda / gcpm

<> Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights

HTCondor pool manager for Google Cloud Platform.

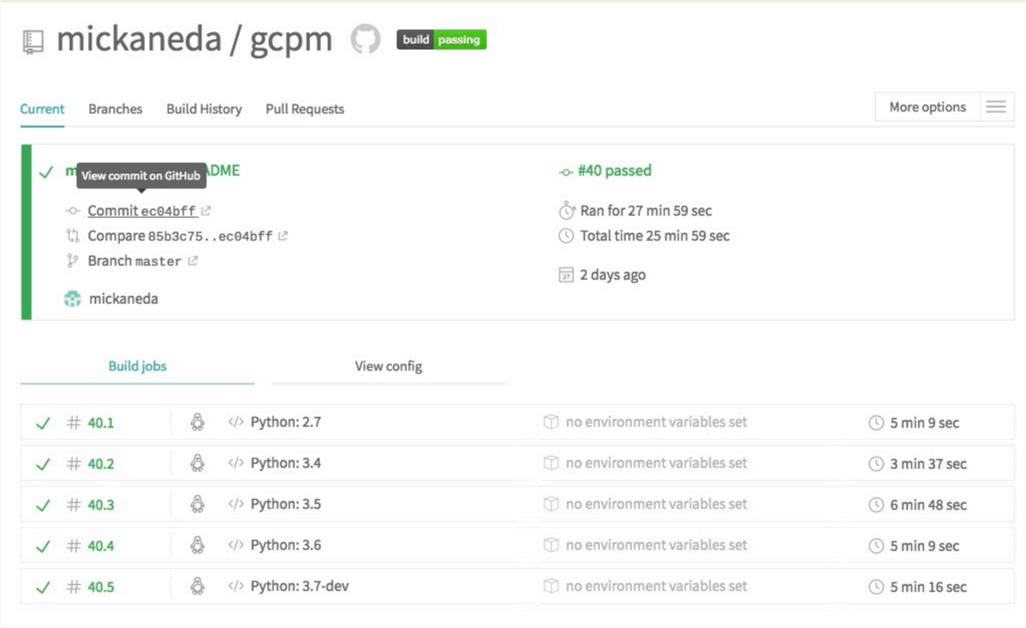
- Open source
    - License: Apache 2.0
- Automatic test/build on Travis CI at push

Travis CI

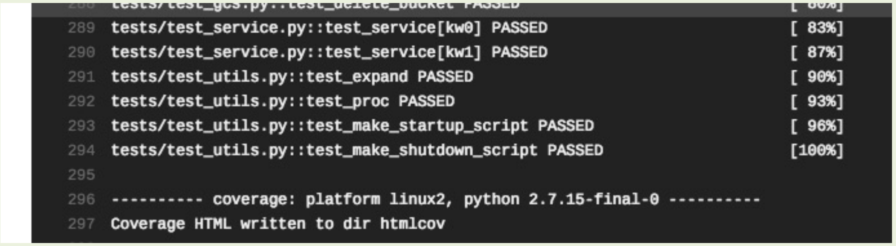
Develop Environment for Python-GCPM

Package manager: Poetry
ire

Test/Build on Travis CI



- Run pytest for every push
- Tested with python2.7, 3.4, 3.5, 3.6 and 3.7-dev
- Build & publish to PyPi after test on Tag may be useful (not implemented)



t
vis job

GitHub

t
nial
, 3.5, 3.6, 3.7

pytest-cov result
(in gh-pages branch)

Travis CI

Develop Environment for Python-GCPM

Package manager: Poetry

ire

pytest-cov result in gh-pages branch

- Test coverage is measured by pytest-cov
- There result is published in gh-pages of gcpm repository at GitHub

Google Cloud Platform Condor Pool Manager (GCPM)

build passing (Coverage report)

HTCondor pool manager for Google Cloud Platform.

Coverage report: 69%

Module ↓	statements	missing	excluded	coverage
src/gcpm/__init__.py	3	0	0	100%
src/gcpm/__main__.py	6	3	0	50%
src/gcpm/__version__.py	1	0	0	100%
src/gcpm/cli.py	26	3	0	88%
src/gcpm/condor.py	90	65	0	28%
src/gcpm/core.py	457	163	0	64%
src/gcpm/files.py	30	8	0	73%
src/gcpm/gce.py	139	17	0	88%
src/gcpm/gcs.py	41	1	0	98%
src/gcpm/service.py	43	3	0	93%
src/gcpm/utils.py	27	1	0	96%
Total	**863**	**264**	**0**	**69%**

coverage.py v4.5.2, created at 2019-01-20 17:16

t

vis job

GitHub

t

nial

, 3.5, 3.6, 3.7

pytest-cov result
(in gh-pages branch)

Travis CI

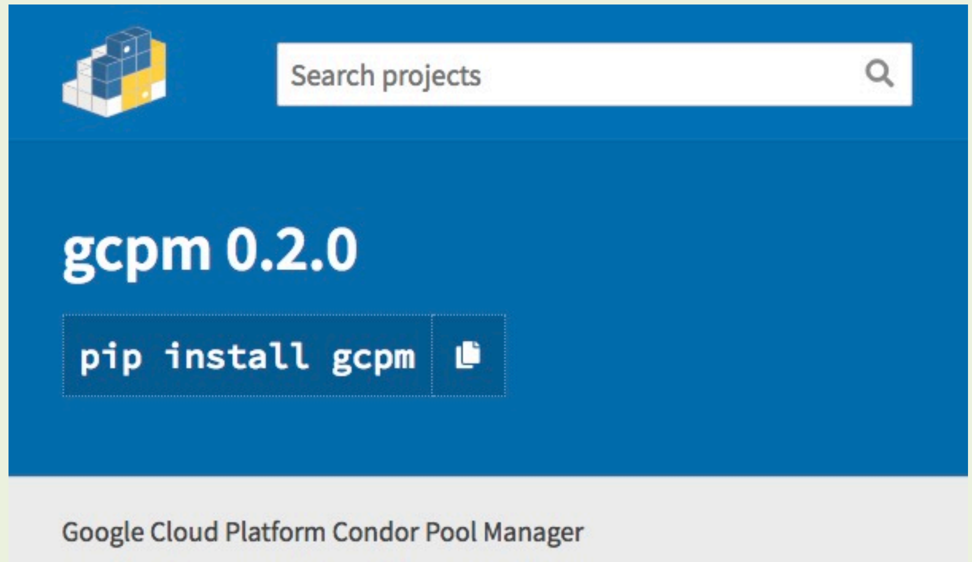
Develop Environment for Python-GCPM

Local Machine

Secı
travi
(serv

ıch)

Published on the Python Package Index (PyPI)




```
$ pip install gcpm
```

Travis CI

The Python Package Index (PyPI)
($ pip install gcpm)