Fermilab | U.S. DEPARTMENT OF ENERGY | Office of Science

HEP Cloud investigation of Google Compute Engine

Summer student Flavio Giobergia
Supervisor Gabriele Garzoglio
Co-supervisor Steven Timm

September 21, 2016

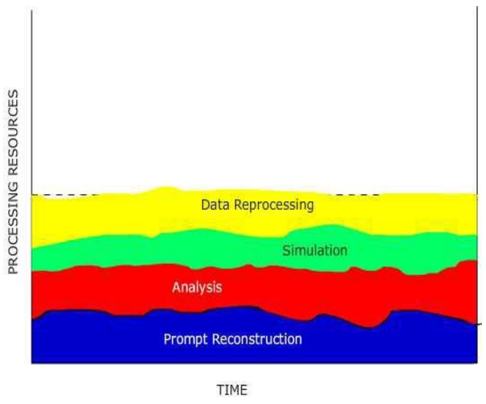
The evolution of HEP computing

- HEP discovered the Higgs utilizing High Throughput Computing (HTC)
- The HEP program is evolving with increased appetite for computing
  - LHC machine and detector upgrades leading to High Luminosity LHC (HL-LHC)
  - The neutrino program is ramping up, culminating with DUNE at the Fermilab Long Baseline Neutrino Facility (LBNF)
  - The muon program at FNAL is about to start
- Increased precision & event complexity, higher luminosity, will push computing needs to **~10x-100x of current HEP capabilities**

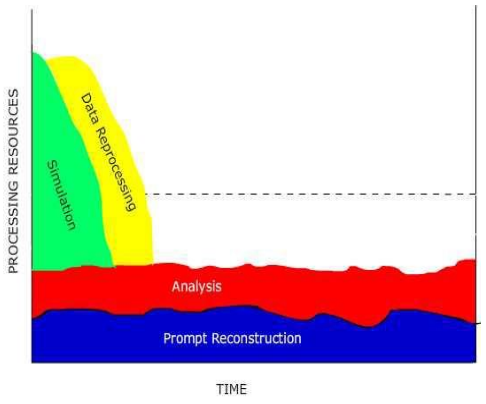
The problem with traditional computing

The demand for computing resources is not constant throughout the year at Fermilab:

- Some periods are **more computationally** intensive
  - Simulations
  - Data processing
  - Analysis
- Others are relatively **calm**



Buying enough hardware for the worst case implies **overprovisioning** the rest of the time!



The solution: Cloud computing

“There is no cloud, it’s just someone else’s computer!”

Anonymous

Cloud computing provides an effective solution to this problem:

- You buy as much computing power as you need, when you need it
- You release it when you are done. After that, you are no longer being charged

No hardware purchase, nor maintenance!

HEP Cloud

A Fermilab project aiming to provide a common interface to access a heterogeneous set of computing resources

- Local clusters
- Grids
- High-performance computers
- Community and **commercial clouds**
  - Amazon Web Services
  - Google Compute Engine

HEP Cloud

A Fermilab project aiming to provide a common interface to access a heterogeneous set of computing resources

- Local clusters
- Grids
- High-performance computers
- Community and **commercial clouds**
  - Amazon Web Services ✔
  - Google Compute Engine

HEP Cloud

A Fermilab project aiming to provide a common interface to access a heterogeneous set of computing resources

- Local clusters
- Grids
- High-performance computers
- Community and **commercial clouds**
  - Amazon Web Services ✔
  - Google Compute Engine ⟵

Summer plans

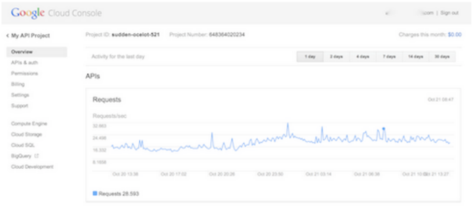
This Summer internship was aimed at investigating the **Google Compute Engine** option, focusing on:

- How it works
    - Launching a new instance
    - Using Scientific Linux
    - Configurations
- How it could be integrated in HEP Cloud
    - Monitoring instances and billing
    - Supporting HTCondor
- How it compares to AWS in terms of performance and prices
    - Benchmarking
    - Pricing options

How it works

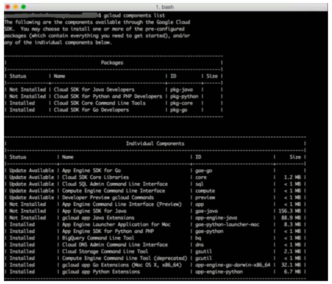
Google Cloud Platform is Google’s cloud solution.

The Infrastructure as a Service (IaaS) component, Google Compute Engine (GCE) enables users to launch virtual machines (VMs) on demand.



VMs (or instances) can be managed using:

- Developers Console
- Command Line Interface (gcloud)
- RESTful API



HTCondor support to GCE

HEP Cloud makes use of HTCondor, a specialized workload management system for compute-intensive job

HTCondor collects jobs from users (in the form of submission files), and sends them to a pool of machines.

Some (limited) support to GCE is provided by HTCondor:

- VMs may be launched, and some customization is possible (only concerning machine type, zone, image and metadata)
- Only user accounts are supported

An update has just been rolled out: everything seems to works!

Scientific Linux

Google already provides a series of vanilla images of different Operating Systems

But HEP Cloud requires images with specific software already installed, to successfully control dispatched instances

Moreover, users expect to find specific tools already available

Scientific Linux is a Fermilab sponsored project, and provides an OS for scientific computing

The Scientific Linux image used for AWS was already available; changing some of the configuration allowed for reuse on GCE

“Basic plumbing”

Launching an instance is only the first step! Everything needs to be set up properly for the machine to work as expected

- Metadata
  - Sent via HTCondor, using **gce_metadata**, **gce_metadata_file**
  - Retrieved from within the VM querying metadata.google.internal
  - Used for passing configurations, scripts, keys and whatnot
- SSH
  - gcloud already takes care of generating SSH keys
  - To get full control over it, third-party SSH keys may be used
    - Hardcoded in the image
    - Sent via metadata
- Firewall rules

Monitoring the Cloud for Fun and Profit

Monitoring the instances' status is of paramount importance, as it provides information about:

- The health of the cloud
- Problems and unexpected behaviours
- Useful statistics
- Billing information

As of today, only three metrics are being monitored:

- Number of instances
- CPU utilization
- Billing

Fifemon

Fifemon is Fermilab's monitoring system for the existing infrastructures.

It uses Graphite for storing numeric time-series data, and Grafana for querying and visualizing time series and metrics

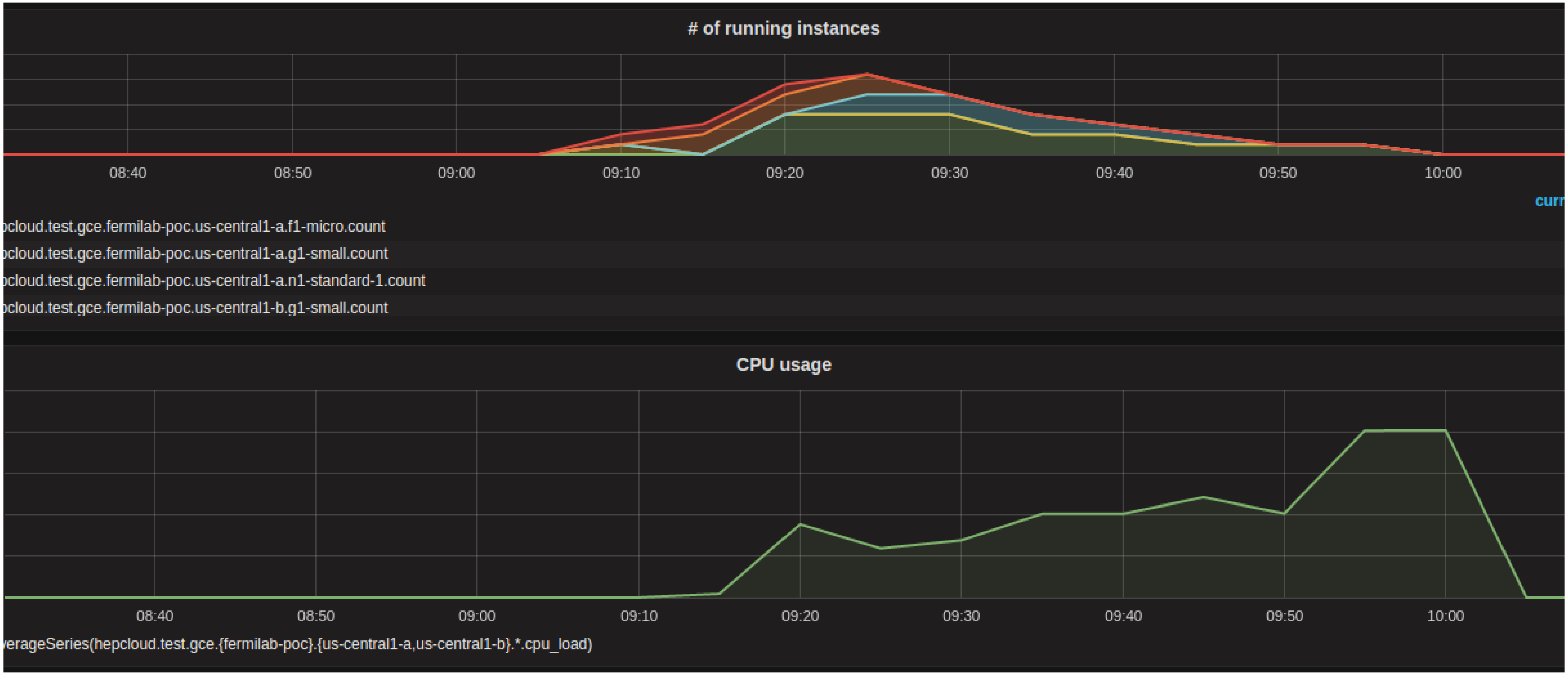
Sending data to Graphite is as easy as

```
echo <path> <value> <timestamp> | nc hostname port
```

(it also supports to Python's pickle format!)

Then, Grafana retrieves the data and provides a WYSIWYG interface for managing dashboards

GCE dashboard on Fifemon



The above is part of the GCE dashboard showing the data collected by the probes

How probes work

Fifemon already provides a specific class (Probe) to extend for implementing a specific probe, and a class for interfacing with Graphite.

The Probe class exposes the `run` method: this consists of an infinite loop that executes every N minutes; collects the required data by calling the `post` method and sends the data using the Graphite class' `send_dict` method

Each probe overrides the `post` method as needed, retrieving the required data and returning it as a dictionary

Fermilab

Instance counting probe

The instance counting probe queries Google's servers to get a list of all instances: the information must be made available for every project, geographical zone, and machine type.

The provided API allows the retrieval of all instances for a given project and zone. Since no selector on the machine type is offered, that part is handled by the probe itself

The metric path uses the following structure:
`namespace.project.zone.machine_type.count`

Where namespace is a uniquely identified path (e.g. `hepcloud.test.gce`), and the other fields are self-explanatory.

CPU utilization probe

The CPU utilization information is only available through Google Stackdriver API, from two different sources

- From within the instance
  - Special software (agent) needs to be installed on each VM
  - Collecting the information requires contacting all the VMs
  - Provides accurate measurements
- From outside the instance
  - Received by the process that is running the VM
  - Collecting and sending data requires no additional operations
  - Measurements not be accurate (based on the external process, which uses some overhead resources)

No machine type information is provided… yet

CPU utilization probe

The CPU utilization information is only available through Google Stackdriver API, from two different sources

- From within the instance
  - Special software (agent) needs to be installed on each VM
  - Collecting the information requires contacting all the VMs
  - Provides accurate measurements

- From outside the instance
  - Received by the process that is running the VM
  - Collecting and sending data requires no additional operations
  - Measurements not be accurate (based on the external process, which uses some overhead resources)

No machine type information is provided… yet

Billing probe

Monitoring the invoices helps with the early detection of behaviours that could result in unnecessary charges.

Google provides a daily invoice as a CSV file in Google Storage. Getting an update every 24 hours is not ideal (Amazon provides the same information 4 times as often)

The probe fetches the CSV files, parses them, filters by date, and aggregates data in a few categories

Gabriele Garzoglio worked on the bulk of the code (as he already wrote the AWS probe), with some minor contributions by myself (fixed some bugs, implemented the aggregation part)

Benchmarking results

Benchmarking is needed to assess how well a given machine can perform certain jobs, providing a metric for comparison with other resources

AWS has already been benchmarked using ttbar (from CMS) and hepspec06 (from SPEC2006). These tests run operations that are similar in nature to the ones that will be effectively used in production

In order to compare AWS to GCE, the same benchmark tests have been run on these GCE machines

Machine type	# cores	Memory (GB)
n1-standard-1	1	3.75
n1-standard-2	2	7.5
n1-standard-4	4	15
n1-standard-8	8	30
n1-standard-16	16	60

Benchmarking results: Google Compute Engine

Machine type	# cores	Speed (GHz)	Price ($/h)	ttbar/s per core	ttbar/s	ttbar/s per $/h	HS06 per core	HS06	HS06 per $/h
n1-standard-1	1	2.3	0.038	0.03048	0.03048	0.802	23.67	23.67	623
n1-standard-2	2	2.3	0.076	0.02024	0.04048	0.5326	16.92	33.85	445
n1-standard-4	4	2.3	0.152	0.01973	0.07892	0.5192	18.5	74.01	487
n1-standard-8	8	2.3	0.304	0.02075	0.16598	0.546	13.95	111.63	367
n1-standard-16	16	2.3	0.608	0.01947	0.31157	0.5125	12.56	200.97	331

Benchmarking results: Amazon Web Services

Machine type	# cores	Speed (GHz)	Price ($/h)	ttbar/s per core	ttbar/s	ttbar/s per $/h	HS06 per core	HS06	HS06 per $/h
m4.xlarge	4	2.4	0.252	0.0201	0.0806	0.32	16.1	64.5	256
m4.2xlage	8	2.4	0.504	0.0191	0.153	0.304	15.1	121	240
m4.4xlarge	16	2.4	1.008	0.0198	0.317	0.315	13.5	217	215
c4.xlarge	4	2.9	0.22	0.0228	0.091	0.415	17.5	69.9	318
c4.2xlage	8	2.9	0.441	0.0226	0.181	0.41	16.5	132	300
c4.4xlarge	16	2.9	0.882	0.0205	0.327	0.371	14.8	237	268

Benchmarking results: AWS vs GCE

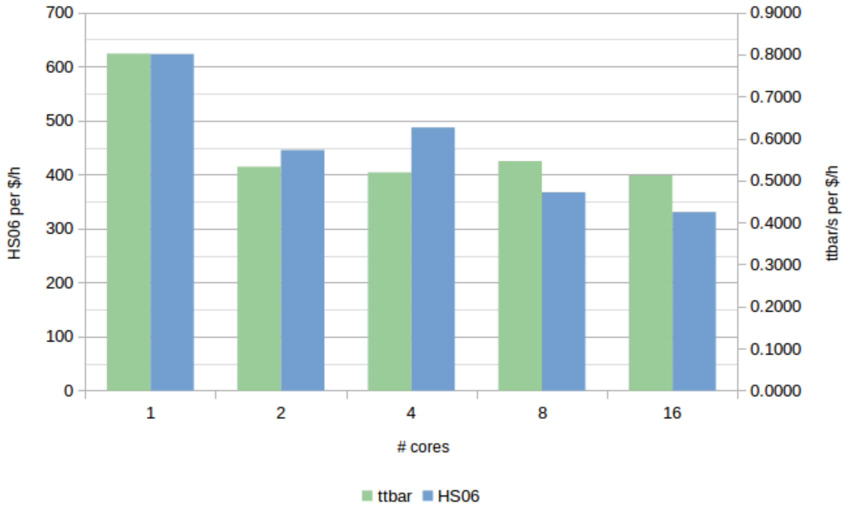
- Amazon results date back to 2015; some of them may not be up to date: although still supported, some of the machine types are no longer listed in the offering

- Amazon already offers some solid discount to Fermilab: the prices considered, though, are those from their website

Benchmarking results: Machine types compared

- The 1 core instance outperformed the rest of the machines, when value for money is considered

- The hourly rate doubles as the number of cores doubles; but the amount of work done grows slower. There has been a similar effect on the local FermiCloud machines (using same OS and hypervisor)
  There is a number of reasons behind it:
  - OS policies
  - Virtualization
  - Processor architecture



Mu2e case study: overview

The Mu2e experiment has been chosen as alpha user for the Google Compute Engine integration.

Simulations required for verifying the updated design of the Cosmic Ray Veto system.

Cosmic rays can interact with the detector, possibly leading to a fake conversion signal: in order to achieve the experiment's designed sensitivity, the CRV needs to cut most of these events.

Mu2e case study: simulation profile

The simulation requires running 500,000 to 2 million jobs.

Each job:

- Takes about 4 hours to complete (previous OSG experience)
- Produces a 100-200 MB output
- Requires 1 core, 2 GB of memory, and 9 GB of storage

Two options for returning data back to Fermilab:

- Sending each output back as soon as it is ready
  - No storage costs, but Google-Fermilab link is slow!
- Store outputs on Google Storage, and fetch them in bursts
  - Google-Google link is faster, but data needs to be stored!

Mu2e case study: results

	500,000 jobs		2,000,000 jobs	
Machine type	**1st scenario (USD)**	**2nd scenario (USD)**	**1st scenario (USD)**	**2nd scenario (USD)**
custom 1 core, 2 GB	88,107	88,172	352,430	352,820
n1-standard-1	98,259	98,324	393,038	393,428
n1-standard-2	98,259	98,324	393,038	393,428
n1-standard-4	98,259	98,324	393,038	393,428
n1-standard-8	98,259	98,324	393,038	393,428
n1-standard-16	98,259	98,324	393,038	393,428

Cost for the execution of N = 500K, 2M jobs in different scenarios (using approximations – consult report for more details)

- Cost does not scale with the machine type: underlying assumption that machines improve linearly with the number of cores (more complex model could account for benchmark results)
- The total number of cores provided does not impact the cost, only the total execution time (but cranking up the cores could result in the link and the storage becoming bottlenecks)

Mu2e case study: it gets cheaper…

The instance running time accounts for more than 90% of the total cost (the rest being network and storage charges)

Google, though, offers some discounts for:

- **Sustained use**: if the machine is used for a significant fraction of the month, Google will refund part of the expenses (e.g. 50% usage: 10% off; 100% usage: 30% off)

- **Preemptible VMs**: Google sells spare computing power for a fraction of the cost (roughly 28%). Machines using these resource are called preemptible. They run normally, until Google needs that power, at which point, they are shut down.
  - Pros: way cheaper; some guarantees on minimum exec. time
  - Cons: Amazon is cheaper still; there's a cap on maximum exec. time

Summing up

- The core functionalities of GCE are supported

- The HTCondor update provides further support to GCE (successfully tested this morning!)

- There are some limitations with Google’s offering:
  - Nothing that can’t be overcome
  - But it would be nice…

- Google instances have proven to be as good as the Amazon ones

- Mu2e will hopefully benefit from the Google Compute Engine integration