Experiences in Managing the Performance and Reliability of a Large-Scale Genomics Cloud Platform

Michael Hao Tong, Robert L. Grossman, Haryadi S. Gunawi

University of Chicago

Center for Translational Data Science

AT THE UNIVERSITY OF CHICAGO

Introduction

- Bioinformatic pipelines (or workflows) are often long running and can take days or weeks to complete.
- Bioinformatics is an interdisciplinary subject that develops and uses algorithms, methods, software, and systems to help humans explore and understand biological data.
- We share our technical experiences in investigating the performance of long-running bioinformatics pipelines on the Genomic Data Commons (GDC).
- The GDC processes cancer genomics data (several million core-hours of data processing per month) and makes available the data to the public (3.7 PB of data are available today).

Outline

I. **Background**
   1. Bioinformatics Pipeline Platform
   2. GDC and GPAS

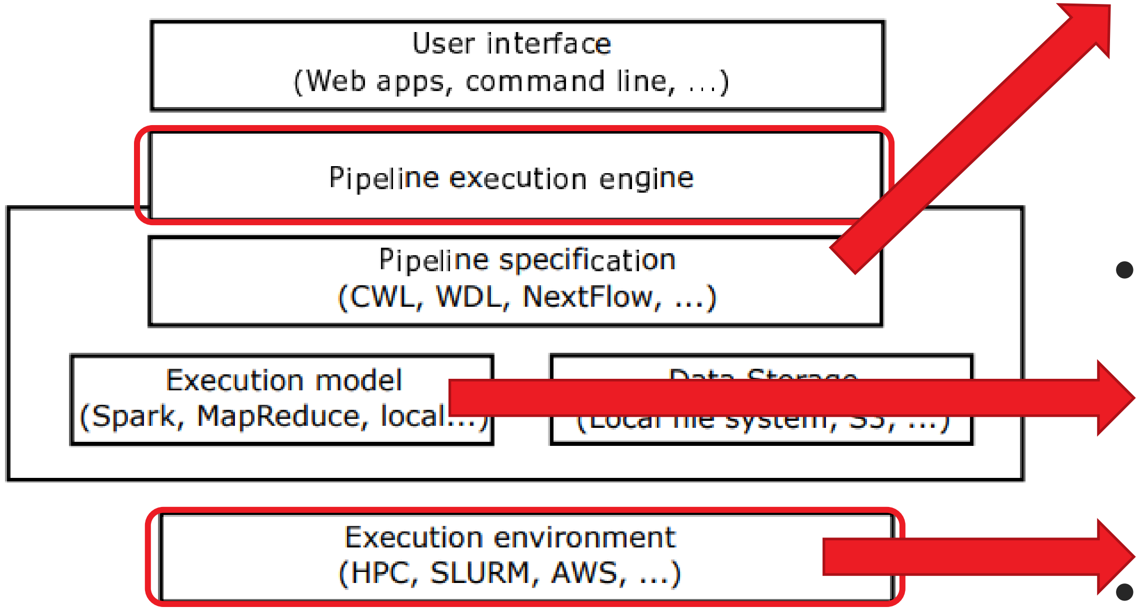
II. Workloads in GPAS

III. Investigate the performance issues

IV. Performance management

V. Conclusion and future challenges

1.1 Abstract of Bioinformatics Pipeline Platform


User interface
(Web apps, command line, ...)
Pipeline execution engine
Pipeline specification
(CWL, WDL, NextFlow, ...)
Execution model
(Spark, MapReduce, local...)
Data Storage
(Local file system, S3, ...)
Execution environment
(HPC, SLURM, AWS, ...)

- Pipeline specification languages (and execution engines)
  - CWL (cwltool)
  - WDL (Cromwell)
  - NextFlow (nextflow)
  - etc.
- Pipeline execution model
  - Simple parallelization ("embarrassingly parallel")
  - MapReduce
  - Spark
- Execution environment
  - HPC cluster, distributed cluster, cloud

1.2 GDC and GPAS

- **GDC (Genomic Data Commons)** co-locates data, storage and computing infrastructure and is designed to store and analyze cancer genomics data and associated clinical and imaging data from NCI and other projects. Data for a single sample is **10 - 100+ GB** in size. The total data volume stored is **7.2 PB,** including both public and internal data.
- **GPAS (GDC Pipeline Automation System)** is the analysis platform supporting the GDC. It is built on a large on-premise cluster that consists of bare-metal nodes and OpenStack/KVM managed VMs. It uses **CWLtool** as the pipeline execution engine.

Outline

I. Background

II. Workloads in GPAS

III. Investigate the performance issues

IV. Performance management

V. Conclusion and future challenges

2. Bioinformatics pipelines

- A bioinformatics pipeline consists of multiple tasks in which the outputs of one task are often the inputs of another task.
- GPAS uses a large variety of bioinformatics tools (samtools, BWA, etc.) in the tasks.
- The tools are written by third parties in various languages, showing different performance characteristics in terms of I/O and computation.

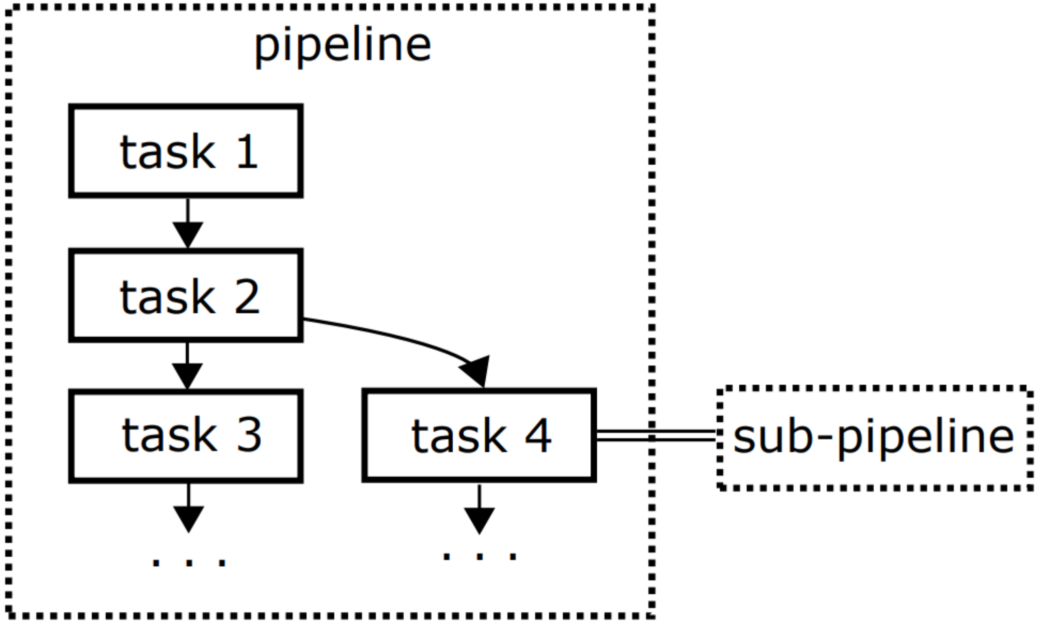
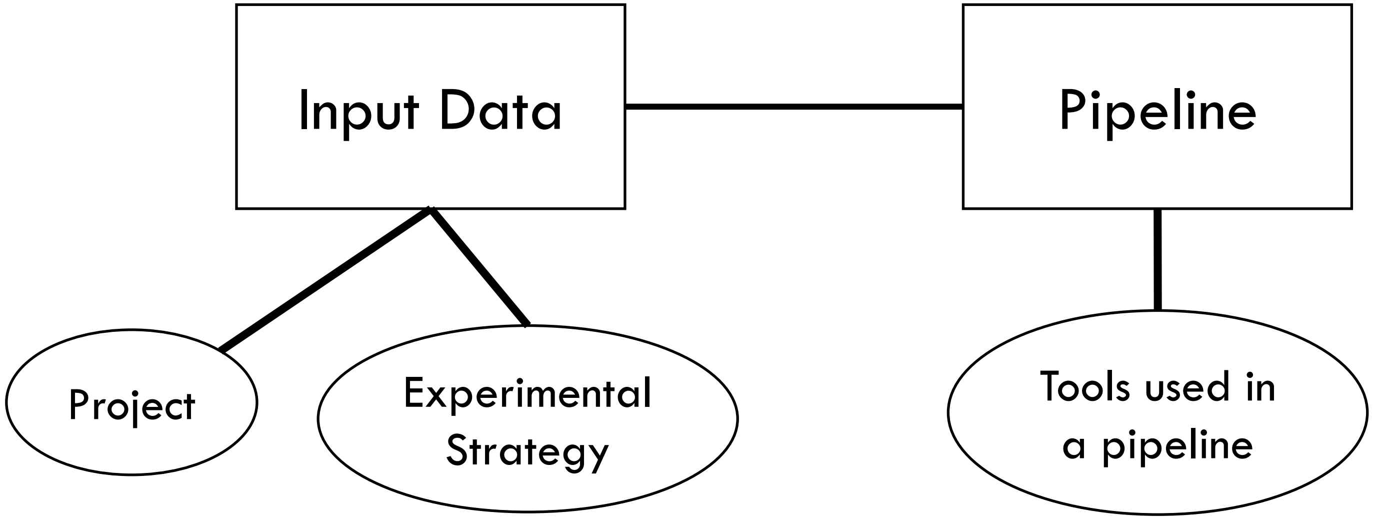


Figure. Directed graph presentation of a pipeline

2.1 Defining performance in the GPAS



Indicates different data sources

Indicates genome sequencing methods

Defines workloads and data flow

2.1 Defining performance in the GPAS

- Performance is only comparable within the same pipeline, experimental strategy, and the same project.
- Performance is measured by

$$\boxed{\frac{Job\ execution\ time}{Input\ data\ size}\ (\,processing\ rate\colon\ s/GB\,)}$$

Outline

I. Background

II. Workloads in GPAS

III. Investigate the performance issues

1. Tail performance
2. Investigation

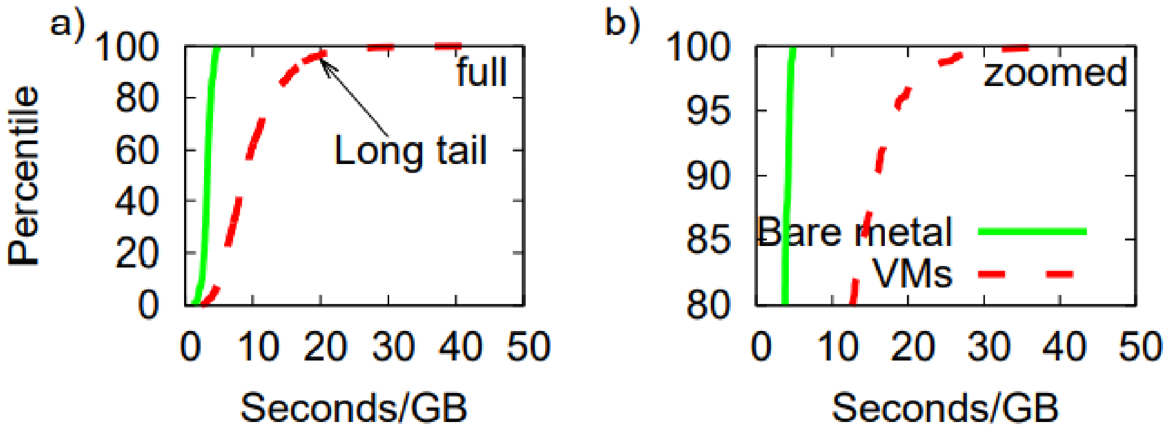
IV. Performance management

V. Conclusion and future challenges

3.1 Tail performance

Variant-Filtration Pindel Pipeline

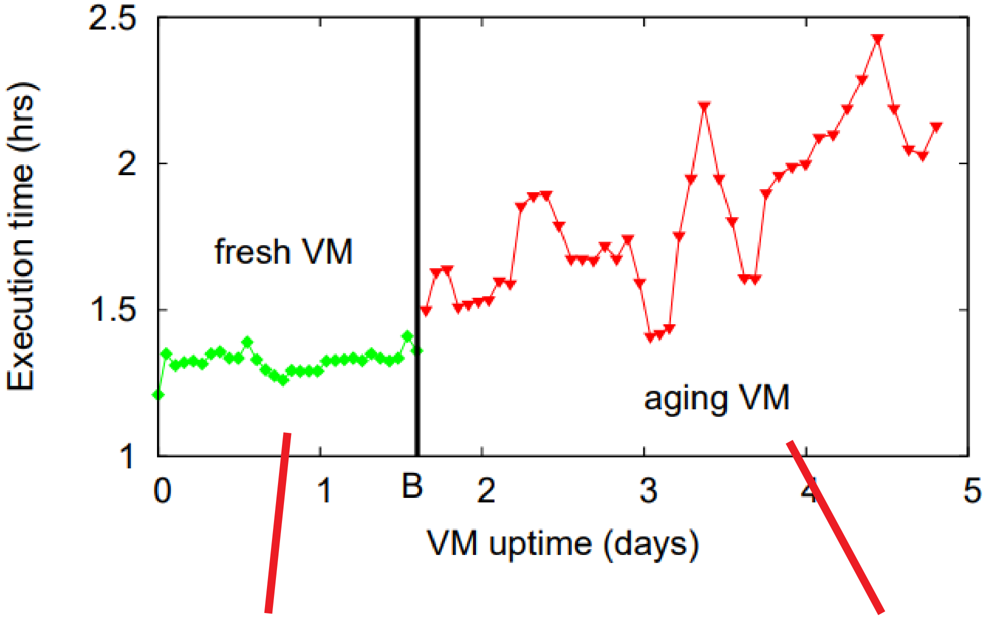
- Jobs running on bare metal nodes have stable performance, showing in the green line which is almost vertical.
- Jobs running on VMs have worse performance and exhibit **a long tail**



3.2 Investigation

Long running VarScan2 experiments

- VarScan2: variant calling tool, Java
- In the test:
  - VM with SSD storage restarts at the beginning
  - Each VarScan2 task runs with 8 CPUs
  - 5 VarScan2 tasks run in parallel for each experiment (a point in the graph)
  - Between each experiment, VM idles for 30s
- Prior to point B (~1.6 days of VM uptime), execution time for each experiment is short and stable.
- Beyond point B, experiments become slower and slower.



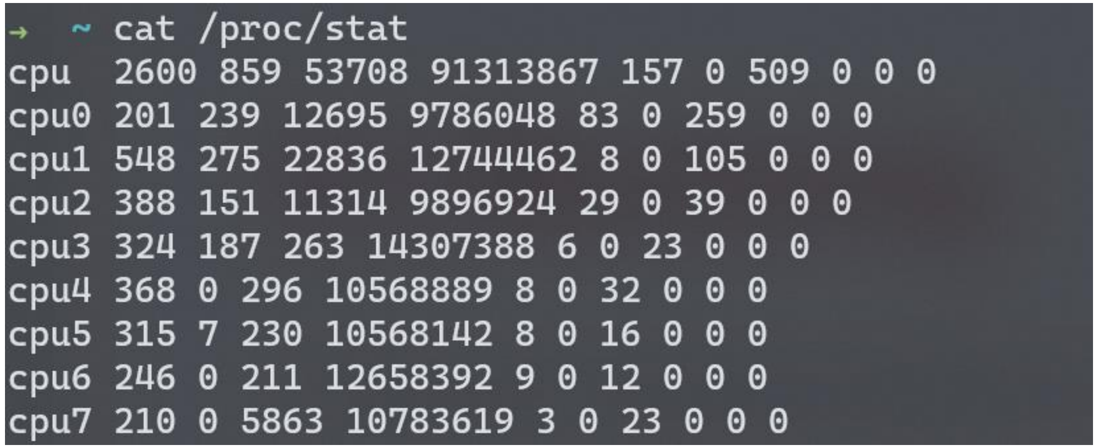
VM in fresh state (Fresh VM)

VM in aging state (Aging VM)

3.2 Investigation

CPU utilization monitoring during *sysbench fileio* benchmark

CPU utilization is calculated from /proc/stat



```
→  ~ cat /proc/stat
cpu  2600 859 53708 91313867 157 0 509 0 0 0
cpu0 201 239 12695 9786048 83 0 259 0 0 0
cpu1 548 275 22836 12744462 8 0 105 0 0 0
cpu2 388 151 11314 9896924 29 0 39 0 0 0
cpu3 324 187 263 14307388 6 0 23 0 0 0
cpu4 368 0 296 10568889 8 0 32 0 0 0
cpu5 315 7 230 10568142 8 0 16 0 0 0
cpu6 246 0 211 12658392 9 0 12 0 0 0
cpu7 210 0 5863 10783619 3 0 23 0 0 0
```

user,nice,system,idle,iowait,irq,softirq,steal,guest

- CPU slices (in unit of 10ms) since system boot are recorded in /proc/stat

$$\mathbf{UtilRaw} = \frac{User+Sys}{t}\%$$

$$\mathbf{UtilDrv}\ (derived) = \frac{User + Sys}{all} \times 100\%$$

- These two are usually equivalent because in value

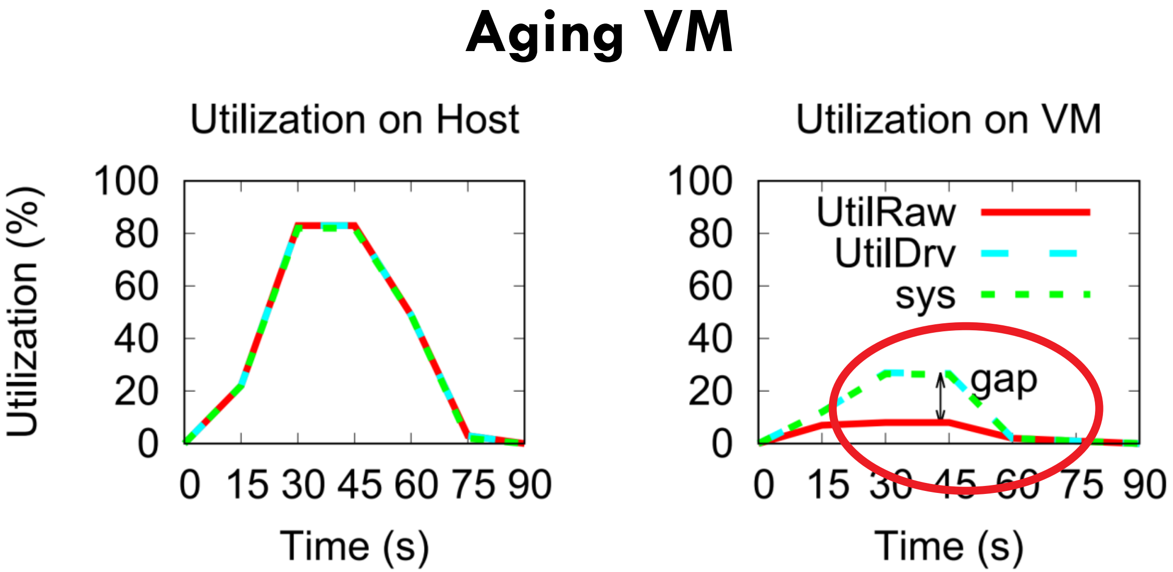
$$all = slices\_per\_second \times t$$

Usually 100

3.2 Investigation

CPU utilization monitoring during *sysbench fileio* benchmark

- Various experiments were conducted but there is no difference in benchmarking results comparing FreshVM and AgingVM except for *fileio*
- A gap is presented between different calculations of CPU utilization.
- **Less times slices are used in the VM**

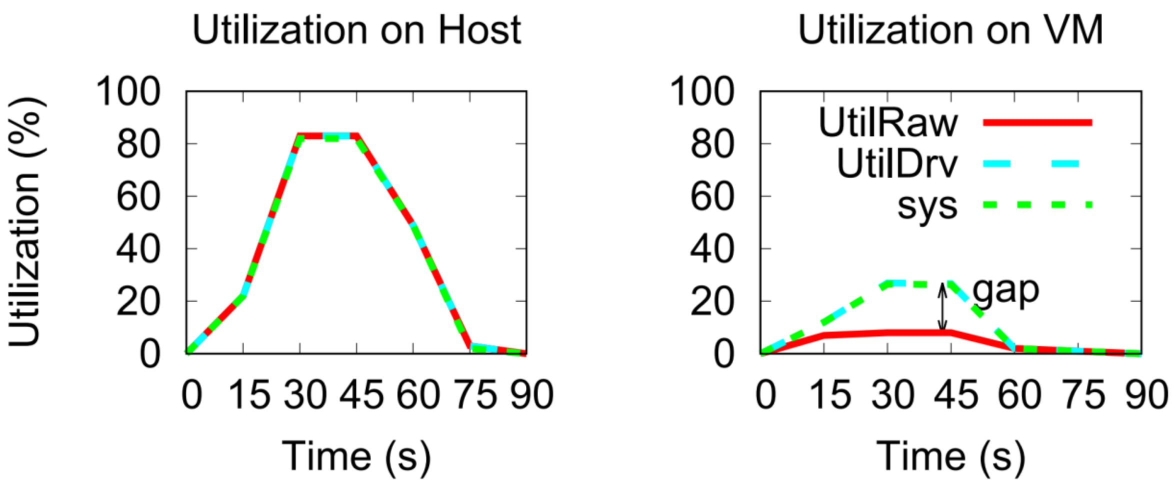


3.2 Investigation

CPU utilization monitoring during *sysbench fileio* benchmark

- System CPU is high on the host and almost equal to overall CPU utilization, although it is I/O bound workload
- CPU utilization on host is 82%, while only 27% on the VM. **The VM hypervisor works intensively on the host.**
- **This VM is the only tenant on the host.**

Aging VM



3.2 Investigation

Read latencies in applications on aged VM

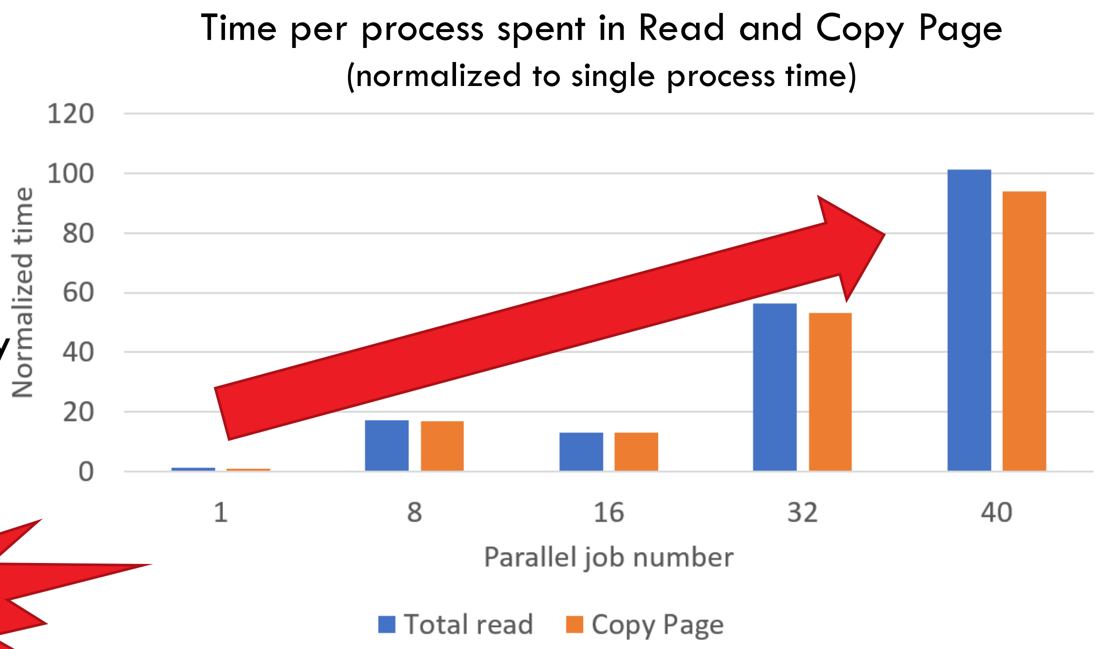
Benchmarking: read operation is affected most in aged VM

A read roughly goes through 5 steps in kernel:

- Step 1 Check page cache
- Step 2 Synchronized read
- Step 3 Send asynchronized read
- Step 4 Wait for page
- **Step 5 Copy page to**

I/O latency

Memory Ops


Time per process spent in Read and Copy Page (normalized to single process time)

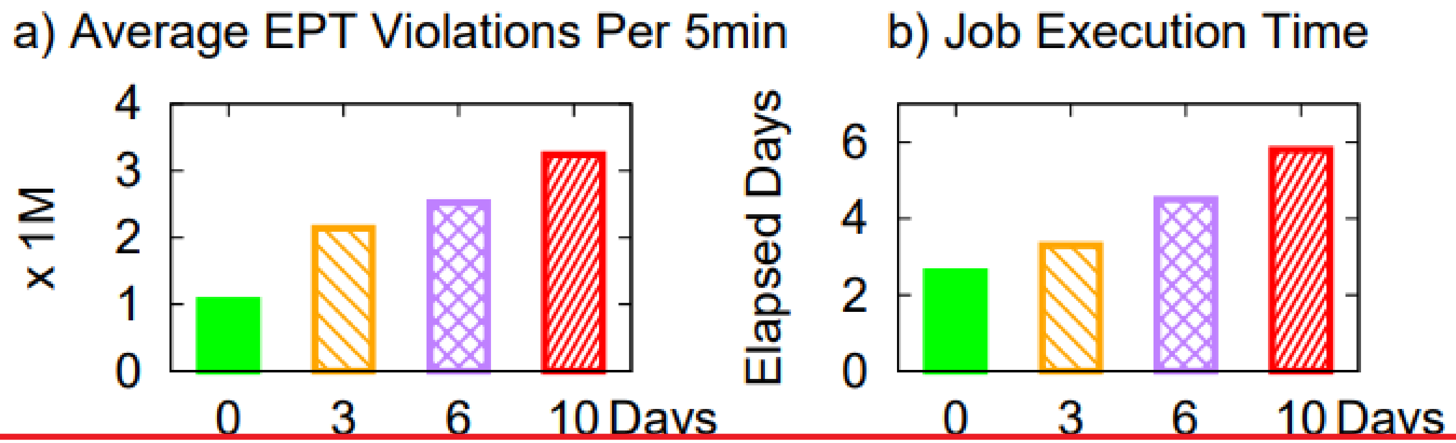
3.3 Cause: Extended Page Table

- **Extended Page Table** (EPT) is an implementation of Second Level Address Translation that manages memory address translations from guest OS to host OS.
- A miss in EPT, similar to a page fault, is called an **EPT violation**, and it will result in a **VMExit (expensive event)**.

3.3 Slowdown caused by EPT

EPT violations in the long running experiments

- 5 DNA alignment jobs using 30 CPUs running repeatedly on a fresh VM for more than 10 days.
- We collect the number of VMExits that handles EPT violation observed over every other 5 minutes.

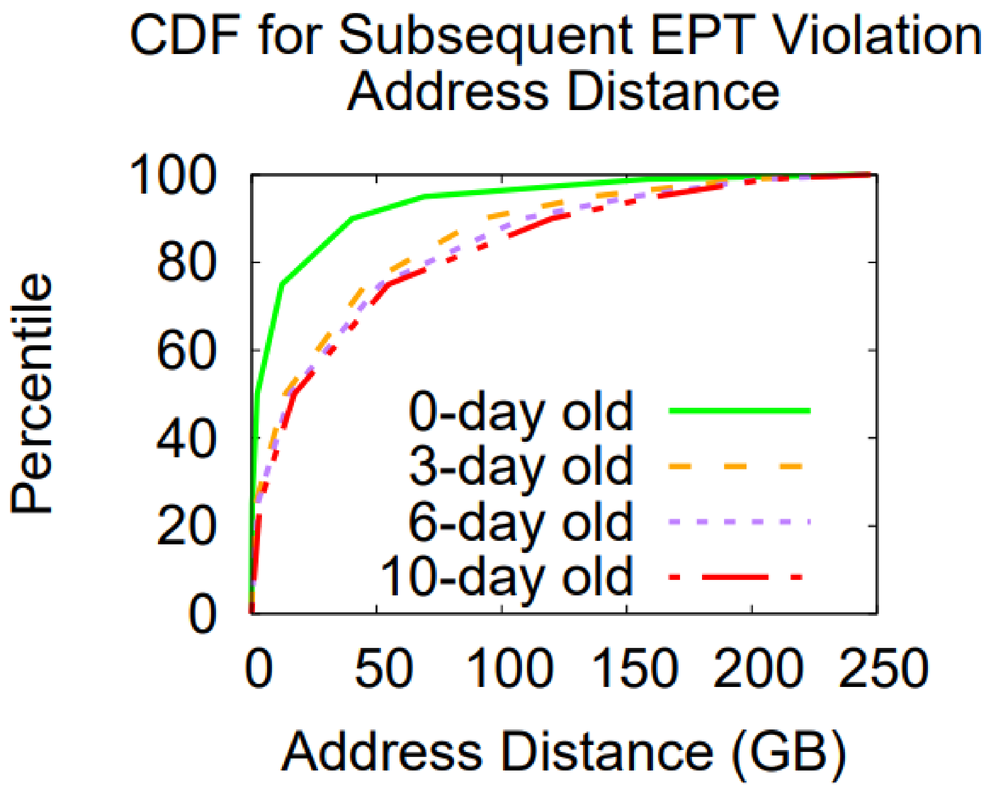


The longer VM has been running, the higher the number of EPT violations, and the longer execution time.

3.3 Slowdown caused by EPT

EPT violations in the long running experiments

- *Address distance of subsequent EPT violations,* measures the level of memory fragmentation.
- When the addresses of subsequent EPT violations are farther apart, the memory tends to be more fragmented, and more EPT violations will occur.



Outline

I. Background

II. Workloads in GPAS

III. Investigate the performance issues

IV. Performance management

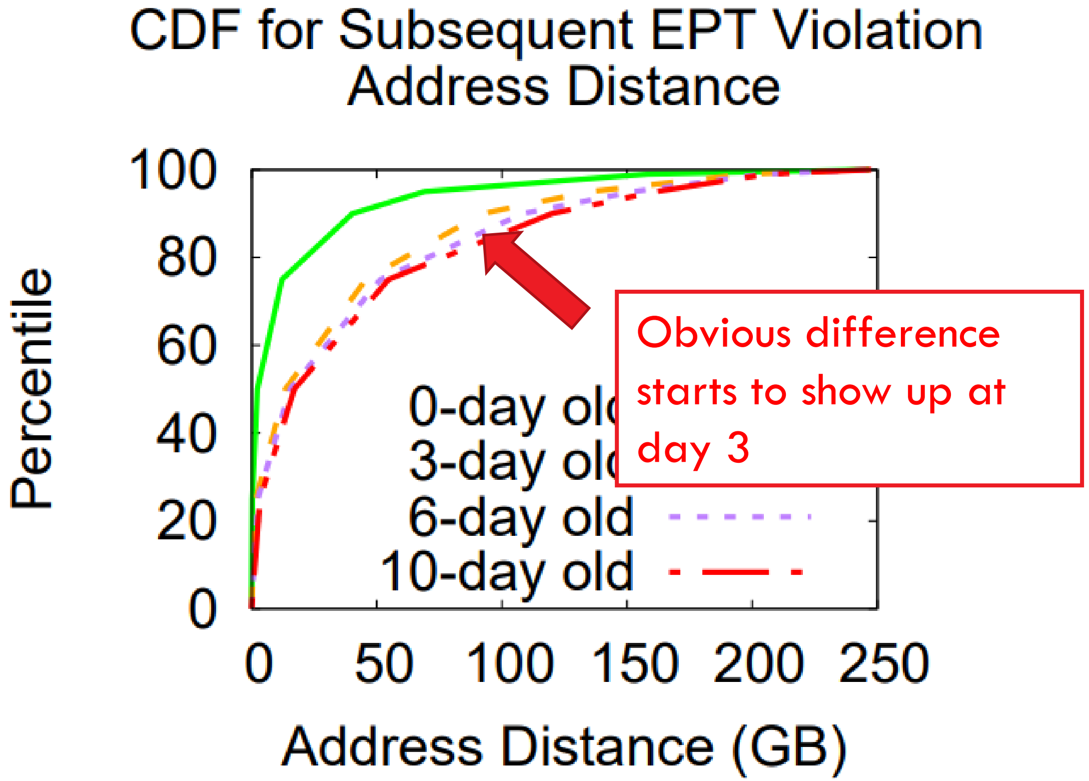
   1. Performance monitoring
   2. Degradation mitigation

V. Conclusion and future challenges

4.1 Monitoring

Host-Level EPT Violation Monitoring

- *Address distance of subsequent EPT violations,* measures the level of memory fragmentation.
- When the addresses of subsequent EPT violations are farther apart, the memory tends to be more fragmented, and more EPT violations will occur.
- **Requires tracing on the host, no obvious impact is observed in experiment.**



CDF for Subsequent EPT Violation Address Distance

Obvious difference starts to show up at day 3

4.1 Monitoring

VM Level /proc/stat Monitoring

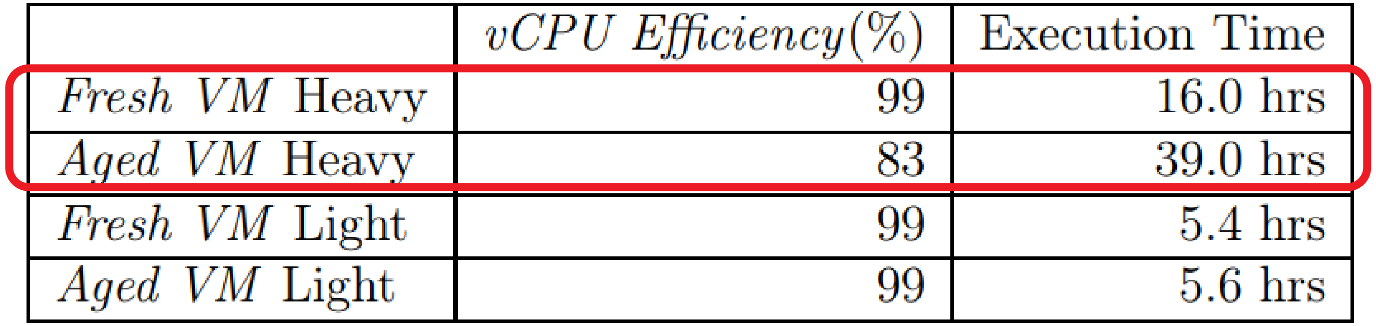
- Define $\boldsymbol{vCPU\ Efficiency}$

$$vCPU\ Efficiency = \frac{all_{in\ VM}}{all_{in\ Host}} \times 100\%$$

$$= \frac{all_{in\ VM}}{t \times 100} \times 100\%$$

- $vCPU\ Efficiency$ less than 100% means that VM loses time slices from the host due to intensive hypervisor activities.

- This method is simple with low overhead
- But can only detect the slowdown when slowdown occurs. It may be misleading when a lot CPUs are idle
  - Heavy workloads use 40 CPUs
  - Light workloads use 8 CPUs

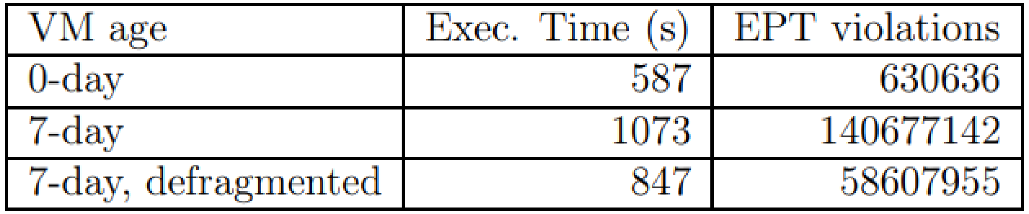


	vCPU Efficiency(%)	Execution Time
Fresh VM Heavy	99	16.0 hrs
Aged VM Heavy	83	39.0 hrs
Fresh VM Light	99	5.4 hrs
Aged VM Light	99	5.6 hrs

4.2 Mitigations

1. Defragmenting Memory

- Built-in defragmentation tool
- Defragmentation increases performance, decreases EPT violations. However, it cannot make the VM back to the fresh state.

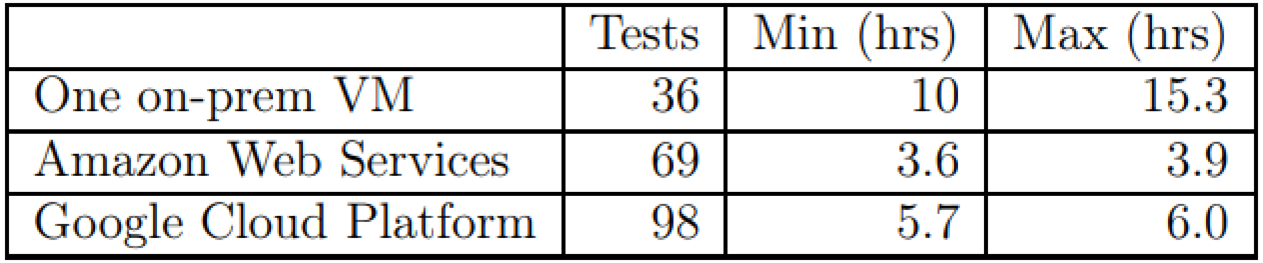


VM age	Exec. Time (s)	EPT violations
0-day	587	630636
7-day	1073	140677142
7-day, defragmented	847	58607955

4.2 Mitigations

2. Using Public Cloud

- Amazon Web Services, Google Cloud
  - Specialized hypervisor
  - Undisclosed optimizations



	Tests	Min (hrs)	Max (hrs)
One on-prem VM	36	10	15.3
Amazon Web Services	69	3.6	3.9
Google Cloud Platform	98	5.7	6.0

- Test
  - Repeatedly running DNA alignment workflow
  - On-premise VM starts to slowdown since the 4th day
- Conclusion
  - Provide reliable performance
  - Costs are generally higher.
  - **Limitation of our test**: We use dedicated hosts. However, for other normal instances, performance may be affected unknowingly because of many other factors such as multi-tenancy in the cloud.

Conclusion and future challenges

1. To the best of our knowledge, we are the first to conduct a prolonged performance evaluation of virtualization stack for jobs that are both long running and memory intensive.
2. We hope the contributions of this paper can help other deployments similar to ours and lead to new research activities (e.g., memory aging tools).
3. GPAS runs many different heterogenous long-running workloads. Developing hybrid scheduling algorithms that use both on-premise clouds and public clouds presents some interesting research challenges.

Thank you!