THE UNIVERSITY OF CHICAGO

IMPROVING THE PERFORMANCE OF LONG-RUNNING SCIENTIFIC PIPELINES IN A BIOINFORMATICS PIPELINE PLATFORM

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
HAO TONG

CHICAGO, ILLINOIS
AUGUST 2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This dissertation was made possible by the patience and guidance of my committees, Prof. Robert Grossman, Prof. Haryadi Gunawi, and Dr. Zhenyu Zhang. Over the years, I have learned tremendous experiences in both academics and life.

Special thanks are given to my wife, Siyu Zhang, who I met at the start of my Ph.D. program in Chicago. Thank you for being on my side for all the years. The journey to the doctorate is challenging, and it would not be possible for me to hold up without your support and understanding, especially in the year 2020 when there is a pandemic happening.

I also thank all my friends who have been keeping me inspired and encouraged: Huiling Feng, Fangzhou Yu, Rong Fan, Lang Yu, Hao Wan, Yuan Gao, Chen Li, Yuxi Chen, and my old friends since undergraduate, Beijie Li, Shang Dang, Qirui Xu, Chuiwen Ma, Rucheng Du, etc. The list goes on endlessly and my gratitude is beyond words.

Many thanks go to many members in the CTDS team. I have asked countless questions and requests, to get a better understanding of bioinformatics pipeline platforms, and to acquire computing resources to conduct experiments. My sincere appreciations are given to Bilal Baqar, Kyle Hernandez, Shenglai Li, Greg Singer, Christian Dompierre, Jeremiah Savage, Sean Sullivan, and Raymond Powell. Thank you all for contributing ideas, inspirations, implementations to the project and my dissertation.

ABSTRACT

The Genomic Data Commons (GDC) is a data platform for managing, processing, analyzing, and sharing cancer genomics data. The data-processing component of the GDC is called the GDC Pipeline Automation System (GPAS). GPAS currently uses an on-premise cluster that uses virtual machines (VMs) and bare-metal machines to run multiple bioinformatics pipelines.

The GPAS has been used in production for over two years, and valuable pipeline statistics are scattered in multiple databases across the platform. This dissertation presents a platform-wide statistics collecting service for the GPAS, and, based on the synthesized statistics, several performance issues have been identified and investigated.

The first performance issue examined is that jobs on VMs exhibit highly varied performance. In particular, there can be a very long tail, with some VMs taking significantly longer than others to execute the same jobs. Through an analysis of jobs statistics and traces, we find that the root cause is the virtual machine memory management layer in the VM hypervisor. When the layer is overwhelmed by intense searches for memory mappings from virtual machine to the physical host, it causes the performance of the VM to degrade.

The second performance issue examined concerns job scheduling. Through an analysis of production statistics, we find that GPAS's overall work progress can be delayed by days, even if only a small percentage of jobs fail. A few other drawbacks of the current simple job-scheduling model have been listed with evidence in the dissertation. A more sophisticated, task-based scheduling model is proposed in this dissertation.

Lastly, a thorough literature review is presented in this dissertation towards a vision for the GPAS with further improved pipeline performance.

CHAPTER 1

INTRODUCTION

Bioinformatics is an interdisciplinary subject that develops and uses methods and computer programs to help humans to understand biological data. Biological databases are doubling in size every 15 months [78]. Biological data is usually processed with one or more bioinformatics pipelines or workflows, and a number of distributed computing platforms have been developed and deployed to run these workflows. Since bioinformatics workloads are often both data-intensive and computation-intensive, and since the computing platforms often use virtual machines (VMs), a number of performance-related issues arise.

This dissertation is concerned with the bioinformatics pipelines and workflows used by the Genomic Data Commons (GDC) [39], which has processed over 2.5 petabytes of genomic data for over 80,000 individuals using a system called the GDC Pipeline Automation System (GPAS). The GPAS uses a mixture of virtual machines and bare-metal machines in a large on-premise cluster.

With pipeline execution-related data and statistics scattered in multiple databases across the GPAS, it was never an easy job for the GPAS team to get an overview of, nor in-depth analysis on, the performance of pipeline jobs. This dissertation presents a statistics synthesizing service (`biosyncdb`) and an all-in-one database (`res_biodb`) to store all the related statistics from across the platform. Based on the statistics in the database, a report on the overall system status and performance since the GPAS officially launched is presented.

Further, inspired by the job statistics, this dissertation reveals a significant VM performance problem that surfaced in the GPAS. We noticed that a significant number of jobs in the GPAS exhibited much slower performance than other jobs of similar types, with as much as 1,000% degradation. Following an in-depth investigation and experiments, we speculated that the problem could be related to memory fragmentation of aging VMs, causing frequent Translation Lookaside Buffer (TLB) misses and hurting performance of jobs.

Moreover, this dissertation also reviews the job-scheduling model that GPAS provides and finds that the current scheduling is inefficient because job failures bring high extra costs: running jobs in parallel causes computing resource under-utilization and contention. A task-based scheduling model is presented and evaluated through simulation and small-scale experiments. This dissertation also discusses further topics in terms of the challenges and requirements to build an efficient task-based scheduler for the GPAS.

Lastly, by building with a thorough literature review on others' experiences of building a bioinformatics application/pipeline platform, this dissertation presents future research possibilities and a vision for the GPAS with further improved pipeline performance.

In summary, the contributions of this dissertation are:

1. A statistics synthesizing service is created for the GPAS, and an all-in-one statistics database is created. The production statistics collected from the GPAS in this database facilitate most of the topics in the dissertation and are invaluable for future research.

2. This dissertation shows how real bioinformatics workloads can cause "aging VMs" after several days and performance degradation by as much as 1,000%. Several possible mitigation scenarios are evaluated.

3. The job-scheduling model in the GPAS is carefully examined and discussed, and a task-based scheduling model is proposed with evaluations, and discussed.

4. This dissertation presents a literature review on bioinformatics application/pipeline platforms and develops a vision for the GPASwith further improved pipeline performance.

CHAPTER 2

BACKGROUND

This chapter covers comprehensive background information regarding bioinformatics pipeline platforms. The topics range over bioinformatics, scientific computing, high performance computing, bioinformatics application and pipeline computing frameworks, virtual machine technologies.

2.1 Overview of Bioinformatics

2.1.1 Brief Introduction to Bioinformatics

Since biology's molecular era began in the mid 20th century, biologists have been inventing and improving their methods of conducting experiments, acquiring genetic data, and analysis at the molecular scale. To be more specific, with their new understanding of molecular genetics, biologists started to conduct studies focused on DNA and/or RNA samples.

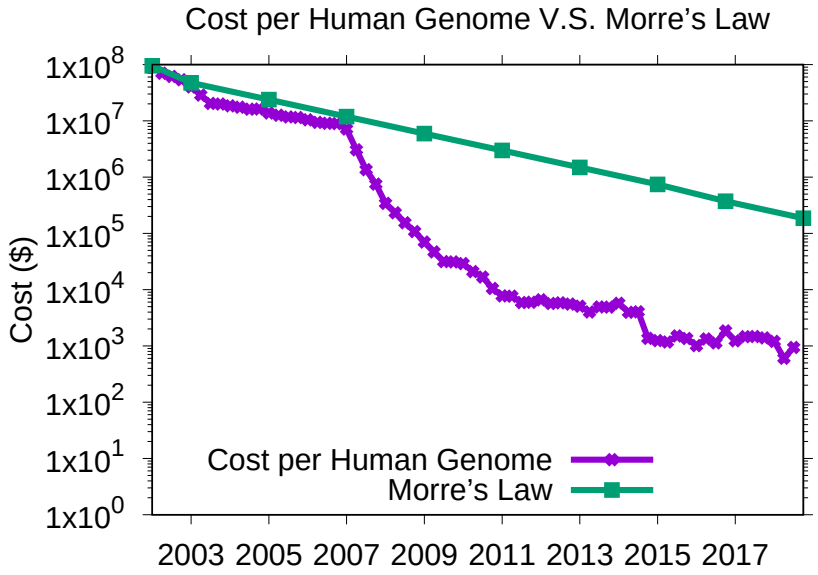

Figure 2.1: Sequencing Cost per Human Genome vs. Moore's law

Since the invention of DNA sequencing in 1977, tremendous volumes of data have been produced. According to an article in 2001, biological databases were doubling in size every 15 months [78]. Over the decades, the cost for whole human genome sequencing has been dropping more than exponentially, as shown in Figure 2.1[1]. Compared with the famous Moore's law[2], which essentially depicts the falling cost of increasing computing power over the years, the cost and efficiency of human genome sequencing has outpaced the advancement of computing power. The year 2003 marked the completion of the human gnome project, and generation of biological data has further been accelerated thanks to the advancement of sequencing technologies including next-generation sequencing (NGS) [108]. The raw data collected by NGS machines are called **sequence reads**.

The biggest challenge brought by the surge of biological data is how to achieve high-speed and cost-effective analysis. Utilizing computers is ideal because computers make it possible for people to handle large amounts of data in order to acquire data-driven insights efficiently. As a result, in recent years, a wide range of bioinformatics methods, techniques, algorithms and tools have been developed to extract information and to generate human understandable reports from experimental data [62].

The aims of bioinformatics are three-fold [78]. First, at its simplest bioinformatics organizes data in a way that allows researchers to access existing information and to submit new entries as they are produced; the second aim is to develop tools and resources that aid in the analysis of data; the third aim is to use these tools to analyze the data and interpret the results in a biologically meaningful manner.

The process of bioinformatics analysis for NGS data can be divided into three phases [62]. The first phase is to obtain raw sequence reads from sequencer and pre-process raw data; the second phase is to assemble the reads to a human reference genome; lastly, the

1. Figure is reproduced from Wetterstrand, K.A. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP). Available at: `www.genome.gov/sequencingcostsdata`

2. https://en.wikipedia.org/wiki/Moore%27s_law

third phase is to produce human understandable interpretation for the results of the second phase.

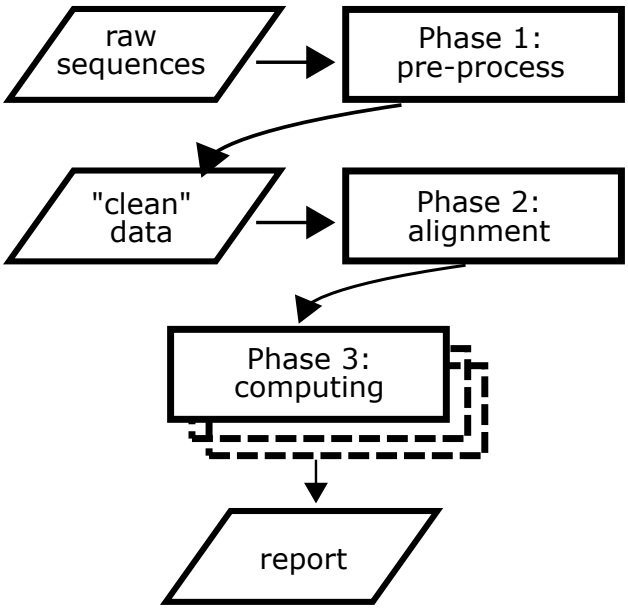

Figure 2.2: General Process of Bioinformatics Analysis

Take DNA sequence reads as an example, as shown in Figure 2.2. Before the data can be used for analysis, it must be pre-processed to be "cleaned up". Various tools and packages can be used for pre-processing, including the Genome Analysis Toolkit (GATK) [81]. The chosen methods of pre-processing data can vary from institute to institute, resulting in slightly different "clean" data even thought the raw sequence reads are for the same purpose. This difference could affect subsequent analysis as well.

Once the data has been "cleaned", it needs to be assembled to the reference. In this process, the sequences are aligned to the reference genome using tools such as Burrows-Wheeler Aligner (BWA) [71]. After alignment, in the third phase, one needs to identify variants which are the locations on the aligned sequences where the nucleotides differ from the reference genome. The procedure to find variants is named "variant calling" in bioinformatics, and a large number of tools are available including GATK, MuSE [34], MuTect2 [23], VarScan2 [59], SomaticSniper [67], etc. Different tools implement different algorithms and potentially

give different results in terms of variant types and confidence. The variants called from the reads need to be annotated so that humans can interpret them, using tools such as SnpEff [24]. Finally, researchers need to conduct further analysis to compare the results between cases and controls.

Another key concept in bioinformatics is **experiment strategy**, which identifies the type of genomic sequences, *e.g.*, DNA sequencing vs. RNA sequencing. Even though the raw data acquired from the first phase are just characters representing nucleotides, a different experiment strategy such as RNA-sequencing does require another category of tools. Other strategies including targeted or genome sequencing may have different tools or a different reference database during the study.

In addition, other utilities would be used in different occasions. Examples include Biobambam [112], SAM tools [72], FastQC [11], Picard Tools [3], for different purposes such as data conversion, data viewing, quality control, metrics collection.

Since the tools are developed by different groups of researchers, written in different languages, and intended to serve different purposes, those tools' workload characteristics such as parallelization capabilities, input/output ratio are quite different too. With wider adoption of solid-state drive (SSD) storage, researchers have found that not all tools exhibit significant performance improvements when storage is switched to SSDs [68]. The reason may be that the tool supports only single thread execution, or that it may have been optimized for spinning disks and hence not benefit from SSDs by design.

In short, bioinformatics comes from the needs of handling large volumes of data in a cost-effective, high-speed, high-throughput manner. A variety of analytic tools have been developed to serve all kinds of purposes. Those tools largely contribute to enhancing the understanding of genomic data.

2.1.2 Bioinformatics Pipelines

The typical practice of bioinformatics involves multiple phases and they are primarily running computer programs other than the first phase of gathering experimental data. To better automate and repeat the process of a study, one needs to arrange all required computer programs into a script-like format so that all the computation tasks can be executed automatically in order, in a bioinformatics pipeline or workflow. In the context of bioinformatics, the terms "pipeline" and "workflow" refer to the same thing and are often used interchangeably.

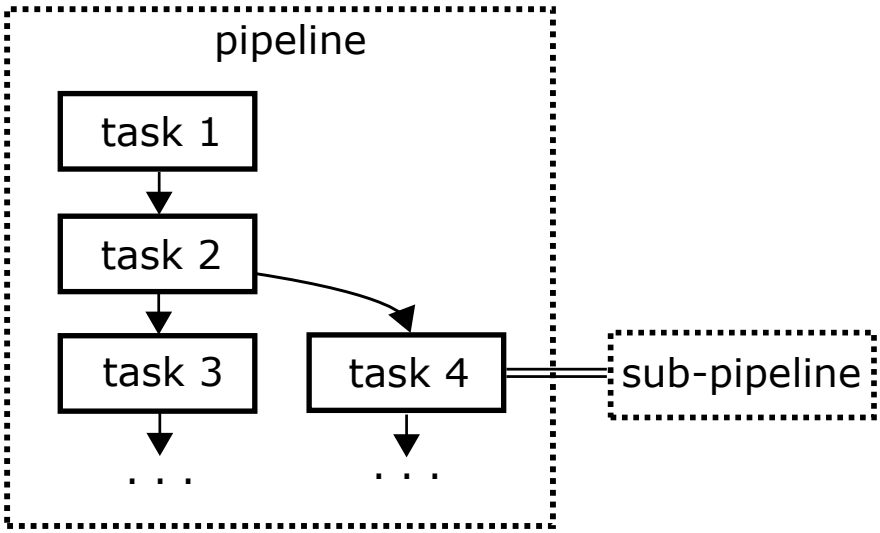

Figure 2.3: Directed Graph Representation of a Pipeline

To be precise, a pipeline represents a set of computing tasks in a specified order. The output of one task may be the input of another task, and some tasks may be independent. Depending on the data dependencies between tasks, a pipeline can be represented as a directed graph as shown in Figure 2.3, where each vertex represents a computing task, and an edge between two vertices represents a data dependency. Usually, non-adjacency of vertices implies that they could be executed in parallel. It is common that a task in a pipeline references another pipeline. Even though the referenced pipelines could be executed on their own, we call them "sub-pipelines" in the presence of the parent pipeline. We consider

all sub-pipeline executions to be part of the overall parent pipeline.

There are several emerging workflow language standards for describing pipelines. Two of the more popular ones are the Common Workflow Language (CWL) [10] and the Workflow Description Language(WDL) [4]. Specific execution engines have been developed to execute pipelines written in a certain language. CWLtool and Cromwell are the primary execution engines for CWL and WDL respectively. Both are sufficient to run pipeline jobs. But to fully utilize computing resources on any particular platform, they rely on designs and policies implemented by that platform. In addition, other third-party execution engines support CWL or WDL, but may not support up-to-date language standards.

Bioinformatics pipelines are data-driven, i.e. whenever a task's inputs are ready, the task can be performed. Thus, if inputs for certain tasks are intentionally not provided, those tasks will be skipped during pipeline execution. Often, researchers pack all possible tasks into one pipeline, and they can perform slightly different analyses using the same pipeline by using different inputs, as shown in Figure 2.4.

A pipeline job is a single pipeline execution that processes a specific set of input data. Thus, jobs are distinguishable by the input data and the pipeline definition. Jobs are expected to perform similarly if they come from the same pipeline, because the computing tasks are similar. However, due to the data-driven nature of pipelines, input data also affects job performance. As shown in Figure 2.4, in addition to data itself, input data includes a few further characteristics of which project and experiment are the most important. The project represents for the source (institutions or organizations) of input data, and datasets from the same project are usually similar. Experiment strategy, e.g. DNA sequencing or RNA sequencing, determines the work required for preprocessing and other computing tasks. As a result, jobs are expected to show similar performance when they come from the same pipeline, and their input data project and experimental strategy are the same. Job performance is typically measured by data processing rate, i.e. the amount of data per unit time

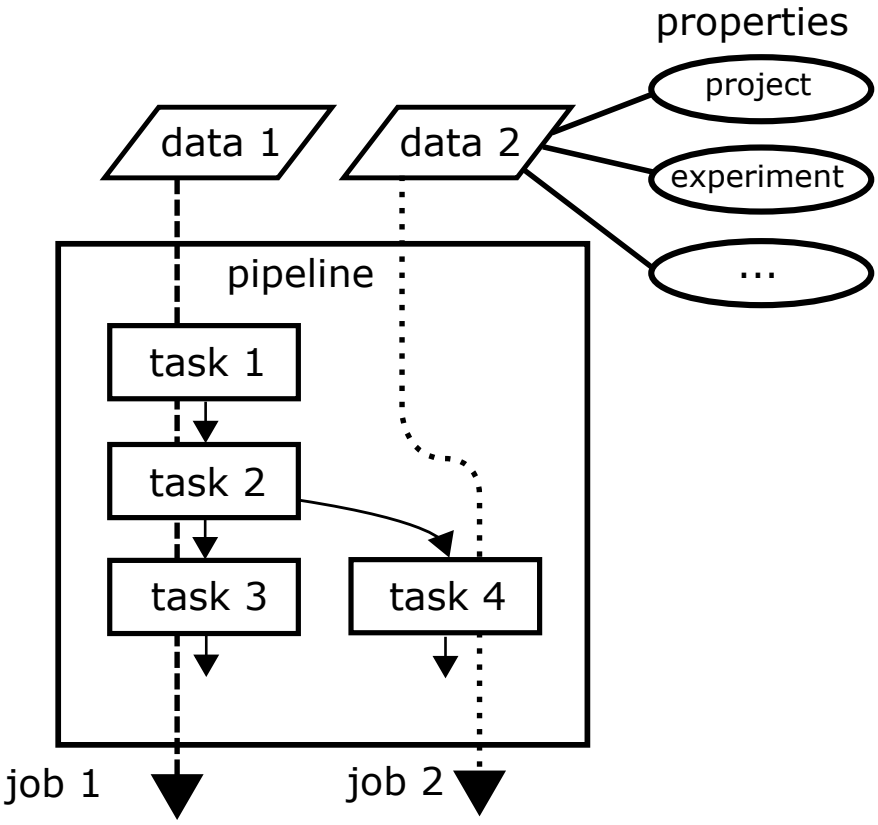

Figure 2.4: Representation of Pipeline Jobs

that a job processes.

Note that bioinformatics pipelines are fundamentally different from instruction pipelining[3] in computer science contexts. However, executing bioinformatics pipeline jobs on a distributed cluster shares many similarities with instruction pipelining. So far, bioinformatics pipeline scheduling is not as sophisticated as pipelining is, because of the complexity of mixing up so many different computation tasks. We will explain current, popular pipeline job execution schemes in a later section.

3. https://en.wikipedia.org/wiki/Instruction_pipelining

2.1.3 Genomic Databases

Genomic data creation has reached a phenomenal rate, and it becomes invaluable if all the variant information extracted can be shared by all related clinicians and researchers. Bioinformatics today not only includes computation tasks, but also facilitates genomic data-sharing to stimulate research in molecular biology. Interest in sharing has led to the idea of creating genomic databases, so that all the knowledge of stored genomic data would be available.

The models for genomic databases are chosen from relational databases, graph database, NoSQL database and flat files. Each option has its own advantage and drawbacks: relational databases are the most efficient but they require specialized software and proficiency in computer skills; graph database and NoSQL database are popular alternatives to relational databases but store data in models other than the tabular relations; flat files have a modest querying capacity and do not require too much in the way of computer skills.

In addition to the storage that a genomic database provides, it also provides indexing and searching functionalities on the stored data, *e.g.*, one will be able to search for a certain variant on a genome in the database without any extra help from other software. However, genomic databases are not associated with how the data is produced or computed.

2.1.4 Data Commons

The increasing sizes of genomic data corpora make it harder and harder for researchers to access, manage, and analyze the data [43], because they need to find a platform that can handle the computation. The research community has begun to work on creating large-scale, distributed computing platforms, as a result of which, a number of well-known platforms such as Taverna [92], Galaxy [7], Toil [113] have arisen. However, one platform can't fit all needs [26]: different aspects of research require different ways of handling data, thus systems need to be implemented in correspondingly different ways.

Although it is hard to build a uniform platform for all types of bioinformatics study, a uniform platform would be inevitably beneficial for the ease of data management and research. The crucial part is data harmonization which enables one single platform to process data in different ways. "Data Commons" was hence introduced, aiming to co-locate data, storage and computing infrastructure with commonly used software services, tools and applications for analyzing and sharing data. The goal was create a resource for the research community [38, 114]. The Data Commons eliminates the burden of purchasing and managing local storage or clusters by letting researchers submit or download data, and analyze data "in-place".

The Genomic Data Commons (GDC) [39, 52] is a data commons, that starts with biomedical and genomic data. The data processing component, GDC Pipeline Automation System (GPAS), is the large-scale computing platform that accesses data from GDC and conducts analysis.

2.1.5 Bioinformatics Pipeline Platforms

Bioinformatics study relies heavily on computation, and the complexity of bioinformatics pipelines requires a high-performance and robust platform to ensure successful job executions.

Taverna [92] is an early tool designed specially for bioinformatics workflows, and was published in 2003. Along with Taverna, a new workflow language, SCUFL was created. The primary goal was to satisfy bioinformaticians' needs to design workflows from web-based interfaces. Much of the effort involved has focused on building a reliable collection of components to achieve that goal.

Similarly, Galaxy [7] provides the capability for researchers to design workflows through a web-based interface. Besides, it uses a built-in, light-weight job-running system, and a number of other workload managers including SLURM [119], HTCondor [77], Apache Mesos [55] and Kubernetes [18] to schedule and manage job executions. Galaxy has also been

developed into different versions to run on major clouds including Amazon Web Services and Microsoft Azure.

Toil [113] is a portable, open-source workflow package that can run workflows on a large scale in cloud-based or high-performance computing environments. Toil relies on Apache Spark to rapidly deploy services and schedule jobs. One special scheduling policy that Toil provides is file caching and data streaming for jobs. Toil is also able to run on multiple cloud services.

A recent development is to decouple the pipeline execution engine from the computing resources. That is, execution engines for workflow language, such as CWL, WDL, NextFlow [31] provide basic execution capability, while computing resources such as VM management, job submission, job scheduling reside on external services. Cloud providers such as Google Cloud Life Sciences have started to provide life sciences-oriented cloud service tailored for computation-intensive bioinformatics research.

Though solutions for executing pipeline jobs continue to advance, the ultimate goal remains the same, which is to achieve high-throughput and cost-effective computing.

2.2 Technical Review of Bioinformatics Pipeline Platform

Bioinformatics research requires huge a mount of computing resources because of the huge data size and time-consuming analytic computations. Various institutions and companies have created a number of platforms for high-throughput bioinformatics pipeline execution.

2.2.1 Abstract of a Bioinformatics Pipeline Platform

The general architectures of bioinformatics pipeline platforms are similar. As shown in Figure 2.5, at the top level the platform will provide a user interface through, depending on implementation, a web app or a command line tool. This interface is used to invoke a pipeline execution engine to execute the pipeline job.

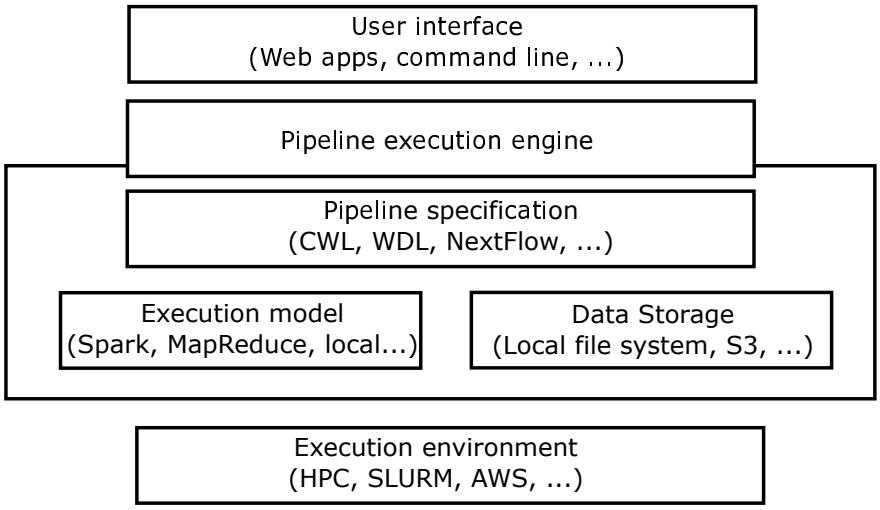

Figure 2.5: Abstract of Bioinformatics Pipeline Platform

At the second level, the pipeline execution engine must know the pipeline specification which may be written in CWL, WDL, NextFlow, etc. Currently, each workflow language tends to be best supported by only one corresponding execution engine, so researchers usually choose the language and execution engine at the same time. Users give parameters to the pipeline execution engine, including information such as input data file path, so that a job can be created. As bioinformatics datasets tend to be large, the data storage is usually a large local file system, or a remote system hosted in the cloud.

The most interesting part of a bioinformatics pipeline platform is the execution model that the execution engine provides for tasks. Some engines aim to use unmodified tools, so they may provide a limited parallelized execution model for the tasks in a pipeline. Other engines may be implemented with more advanced frameworks such as MapReduce [29] and Spark [120], which requires that all tools must be re-implemented with the framework. Such re-implementation may be difficult with most tools.

Last, at the bottom layer, the platform must provide options for the execution environment. Most of the platforms are built with distributed systems in mind, and get support from many high-performance computing (HPC) infrastructures and commercial clouds. Some platforms also provide the option to run locally. The key difference between platforms

is the degree to which a platform integrates the pipeline execution with the underlying execution environment. Greater integration allows the platform itself to have more control over the job scheduling in its execution environment so that the execution environment can be specially optimized for individual workloads.

2.2.2 Bioinformatics Pipeline Execution Model

As explained earlier, a bioinformatics pipeline is essentially a set of computations that are arranged in a certain order. Hundreds of standalone computation tools have already been developed by various bioinformatics researchers, and the purpose of pipelines is to organize those tools together so that researchers can create a sequence of processes in order to execute and reuse the same computations.

The most common and simplest execution model for pipelines is that the pipeline just organizes unmodified tools, and the pipeline execution engine starts the requested tools directly in order, as implemented in NextFlow, CWLtool, Cromwell. The biggest advantage of this model is its simplicity; any bioinformatics tool can be put into one of these pipelines without modification. It performs poorly, however, in scaling up workloads and maximizing parallelism, because it is hard to figure out how to optimize for all of the various tools individually and in combination.

Other pipeline engines may be implemented with the MapReduce framework or Spark framework:

- MapReduce is a classical distributed task-execution model where a job is split into independent, small-sized tasks through mapping, and results from tasks get synthesized into final results through reducing. GATK has plenty of tools and small pipelines implemented with MapReduce.

- Spark is a distributed computing framework where more transparent data and job partitioning are provided through a resilient distributed dataset (RDD). Since RDD

will put the whole work dataset into memory, executions in Spark are much faster.

The adoption of MapReduce and Spark would require new tools to be written in those frameworks, or existing tools to be rewritten.

In addition to these framework abstractions, we also introduce three early bioinformatics pipeline platforms:

- Galaxy [7] provides a web-based user interface so that users can develop workflows easily with pre-provided tools. Galaxy adopts the simple model: the tools are executed in separate processes. Those tools are not necessarily the same as most popular standalone tools. But, being built on a high-performance computing (HPC) cluster, Galaxy does implement a standalone job scheduler to avoid long waiting times.

- ADAM [90] is a research project that leverages Apache Spark and Parquet to achieve high speed increases for bioinformatics pipelines. However, most of the analysis tools must be rewritten to be used in a Spark framework, which involves extensive, difficult work.

- GESALL [104], on the other hand, tries to improve pipeline performance without modifying the tools, by interfacing with them through "Wrapper Technology". Essentially, GESALL achieves application-level parallelism by partitioning input data properly. In three evaluated tools, performance is increased by a large margin, but there were also slight differences in the final results compared to the tools' original execution models. Due to the limitations of the evaluation, it is possible that larger differences may appear for other tools. Besides, under most contract-based scenarios, it is not acceptable for the platform to create different results that the client may not expect.

To summarize, designing of a pipeline platform and selecting a pipeline execution model must consider the numerous existing analysis tools and the requirement for programming

skills among bioinformaticians. Most of the analysis tools are mature and difficult to be correctly re-implemented in other frameworks. Thus, the emphasis in bioinformatics pipeline platform design is focused more on how to use existing, unmodified tools to easily create a large-pipeline execution environment. Workflow languages such as NextFlow, CWL and WDL can all easily pack unmodified tools together for execution on a cloud platform such as Google Cloud, Amazon Web Services, etc. Pipeline performance increases rely more on finding ways to utilize computing resources and scheduling jobs more efficiently.

2.2.3 Bioinformatics Pipeline Job Scheduling

Although there are numerous options for building a platform with various pipeline execution engines and execution environment, in this section, we focus specifically on the type of platforms that can be built with NextFlow, CWLtool or Cromwell, and that use a cloud service as the execution environment.

In most cases, cloud services aim to reduce the difficulty for bioinformaticians to create and manage virtual machines. They provide cost-optimized VMs for long-running bioinformatics jobs, and batch-mode pre-emptible VMs for fault-tolerant workloads.

As a result, the development and optimization of bioinformatics pipeline engines are often separate from computing environments. There are two strategies for scheduling jobs on a cloud-based cluster. One strategy is to achieve task parallelization on a single worker node for a single job, and then run multiple jobs on the same node to maximize resource utilization and parallelism. The other strategy is to make a single job utilize multiple nodes so that independent tasks of the job can run in parallel on separate worker nodes.

In the first strategy, where tasks are parallelized on the single node, the pipeline execution engine is responsible for executing the job, and for achieving possible parallelism for tasks within the job. The platform uses APIs of the execution environment to specify computing resource requirements for the job, and the job will be submitted to a worker node

transparently. In this case, the computing efficiency largely relies on the implementations of the pipeline execution engine. At the time of writing, the stable version of CWLtool only supports serialized execution of a pipeline job. CWLtool beta version, Cromwell and NextFlow support tasks parallelization within a job.

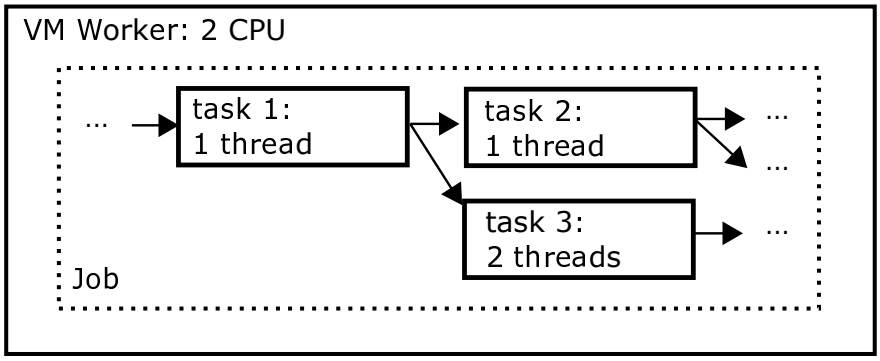

Figure 2.6: Abstract of a Pipeline Job Execution

As shown in Figure 2.6, the pipeline execution engine is able to parallelize as many tasks as possible that use different numbers of threads within one job. However, due to lack of fine-granularity resource management, as shown in Figure 2.6, the job would under-utilize CPU resource during the execution of Task 1 if it runs on a 2-CPU VM, and then oversubscribe CPU resource during the executions of Tasks 2 and 3. Scheduling difficulties would increase when more jobs are added to the same node.

Alternatively, the execution engine can choose to execute one task at a time, which is the same as what the current stable version of CWLtool supports. Thus, the overall job CPU limit is easy to set and guarantee, and appropriate VMs can be allocated for jobs.

To scale up jobs on a large-scale computing cluster, an external job submission and scheduling service can be built to distribute jobs across all of the cluster's available nodes, allowing for effects of resource availability on the individual node and resource requirements of jobs.

The other strategy depends on the integration with cloud providers that pipeline execution can provide. NextFlow and Cromwell have implemented mature integrations with

Google Cloud, Amazon Web Services, etc. By providing the configuration or requirement of the computing cluster, users can have the pipeline execution engine automatically scale up and down in response to the resource requirements of currently running tasks. As of now, NextFlow and Cromwell only support managing one job on one cluster at a time. As a result, although the tasks of a job are parallelized across an elastic computing cluster, there is no easy way to scale up more jobs on the same cluster.

So far we have talked only about scheduling under the limits of CPUs. When we add more constraints such as memory and I/O intensity, it will become more difficult to achieve efficient job scheduling. As a matter of fact, successful execution and correctness matter much more than efficiency to the GPAS team. Thus, in the GPAS, jobs are usually allocated more than enough computing resource, ensuring that jobs can complete successfully.

2.2.4 Frameworks for Bioinformatics Tools

Numerous tools have been created to meet the needs of bioinformatics research. As we mentioned before, more platforms tend to support unmodified bioinformatics tools wrapped in pipelines, than support rewriting their own implementations. Bioinformatics developers have started to look at more advanced programming frameworks to improve tool performance.

Until now, most mainstream tools have been implemented in traditional multi-threaded models in C, Java or Python, such as BWA. Due to the design of the algorithms used, the benefits from multi-threaded parallelization differ between tools.

Research has tried to turn some traditional tools into ones with more advanced frameworks, as BWA, for example, has been turned into SparkBWA [6]. GATK is a collection of useful genome analysis tools, some of which are built with MapReduce in mind, and some others have now been tested with the Spark framework. However, there are still a number of other tools that cannot be re-implemented within the Spark framework.

A complication brought about by implementing tools in more scalable frameworks is that

one needs to find a way to utilize a highly scaled tool on a separate cloud when it is just one of the many tasks in the pipeline job. For tools in the Spark framework, one needs to set up a Spark cluster and configure the job for tasks to use that Spark cluster; the situation is likewise for tools in the MapReduce framework.

Doing this would add another layer of problems in managing and utilizing another Spark or MapReduce cluster, which can be rather difficult for current implementations of all the pipeline execution engines. More often, those tools would run locally just to utilize the multiple CPUs when they are embedded in a pipeline.

2.3 Organization of the GDC and GPAS

2.3.1 Organization of the GDC

The main purpose of the GDC is to serve as a data warehouse where users can conveniently submit and download data. To build a robust and high performance data service, a lot of design decisions are focused on improving data querying performance and data reliability.

Figure 2.7 shows the organization of the GDC. At the top right, where the GDC is mainly servicing external users, F5 application services guarantee security, firewall, load balancing, etc. APIs and a web portal are provided by the GDC so that users can submit and download data securely to and from the GDC. Submitted data is ultimately uploaded to S3-compatible storage, such as Cleversafe and Ceph in the GDC. There are two important design decisions that keep data service from the GDC reliable and fast.

Graph-oriented data model and data build services: The data model is designed to maintain the consistency, integrity, and availability for existing data and metadata, while accommodating data for multiple ongoing or new projects, as well as providing reliable query capability for both external and internal users.

To meet these requirements, the GDC implements a flexible but robust graph-oriented

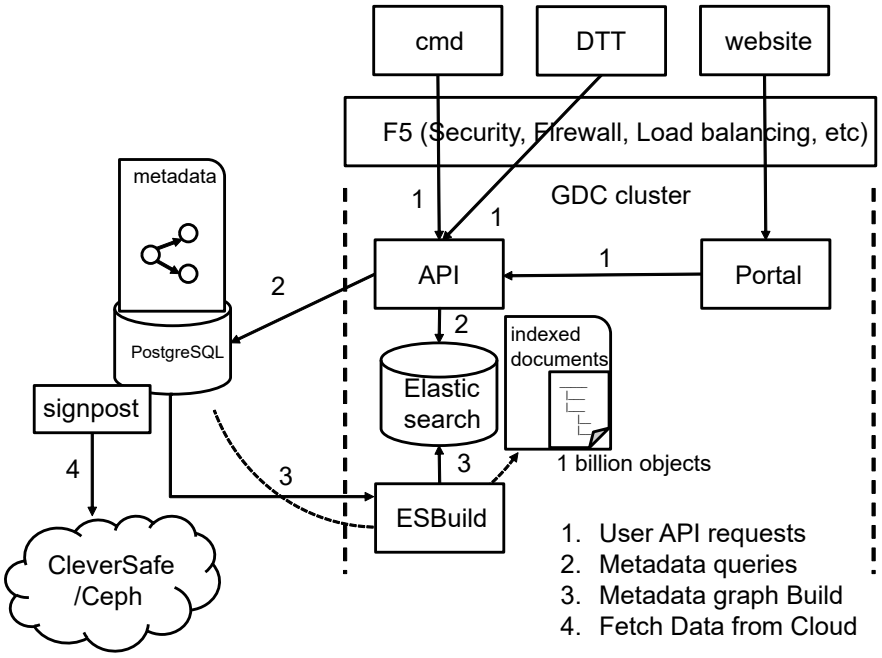

Figure 2.7: System Organization of the GDC

metadata store using PostgreSQL, an index service for data files using Signpost, and indexed document stores for API and front end performance in Elasticsearch facilitated by a data-build service, ESBuild.

Data entities for bioinformatics projects are so complicated that the relationships between different entities are hard to maintain in a traditional relational database. Thus, a graph-oriented data model is created as shown in Figure 2.8. The figure shows only a small portion of the entire model, enough to clearly show the relationships between entities. The relationships are represented in the edges. These edges are generic and are transparent to database users; thus the model is flexible and extendible.

The true data files that contain sequencing data and analyzed results are stored in the cloud (Cleversafe and Ceph), and they are represented as a node with an UUID in the graph data model. Signpost keeps the mapping of UUIDs and the data files' true locations in the cloud.

As simple and flexible as the graph data model is, its downside is poor query performance

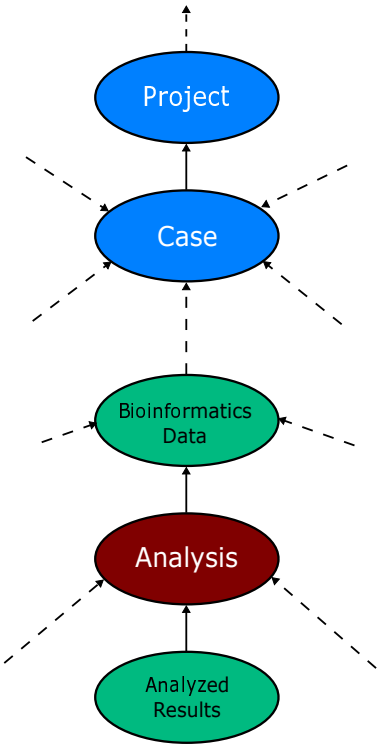

Figure 2.8: Graph-oriented Data Model in the GDC

in PostgreSQL, especially when the website needs to present a whole bucket of information from multiple data entities in the graph data model. To allay this limitation, Elasticsearch is used to provide fast queries for metadata to users from the website. ESBuild, as a data-build service, was thus created to traverse the graph data model and convert the graph model into flat JSON to be indexed into Elasticsearch. To support highly integrated and flexible data queries with fast performance, the Elasticsearch JSON schemata are designed in a highly denormalized fashion, with related data entities pre-joined via subdocument embedding.

File-centric and case-centric indexes are built to allow users to search and retrieve files and participants respectively. Figure 2.9 shows a representation for a file-centric index built by ESBuild. File-related metadata is joined from PostgreSQL by ESBuild and embedded directly in the same document with the file. In addition, other related files are also embedded as subdocuments. Thus, when the index in Elasticsearch is queried, the search process avoids a lot expensive join operations in the relational database.

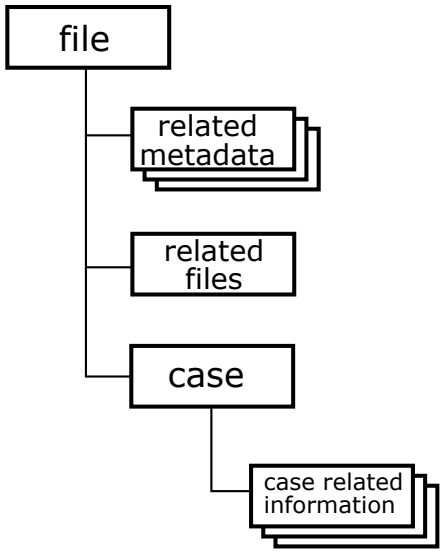

Figure 2.9: Elasticsearch Data Model for the GDC

Currently, the GDC graph database takes up 62 GB on-disk; thus the ESBuild process can be resource- and computation-intensive. Numerous performance improvements have been made to ESBuild to enable fast caching and project-based parallelization of the process. This is discussed in the next section.

Data backup and recovery: As data is important to users, the other major responsibility is to keep all data safe. The GDC currently stores all data with one copy on the premises, one copy in Amazon Data backup services (S3/IA and Glacier), and one copy on BlackPearl tapes .

Frequently accessed data is maintained under Amazon S3 Standard Service, and infrequently accessed data is maintained under Amazon S3 Standard IA. Retired, deprecated, or archived data is maintained in Amazon Glacier. The GDCuses AWS Object Lifecycle Management to manage rules for moving data objects between storage tiers.

2.3.2 Organization of the GPAS

As shown in Figure 2.10, the system consists of several micro services, implemented using both community open source software and in-house software.

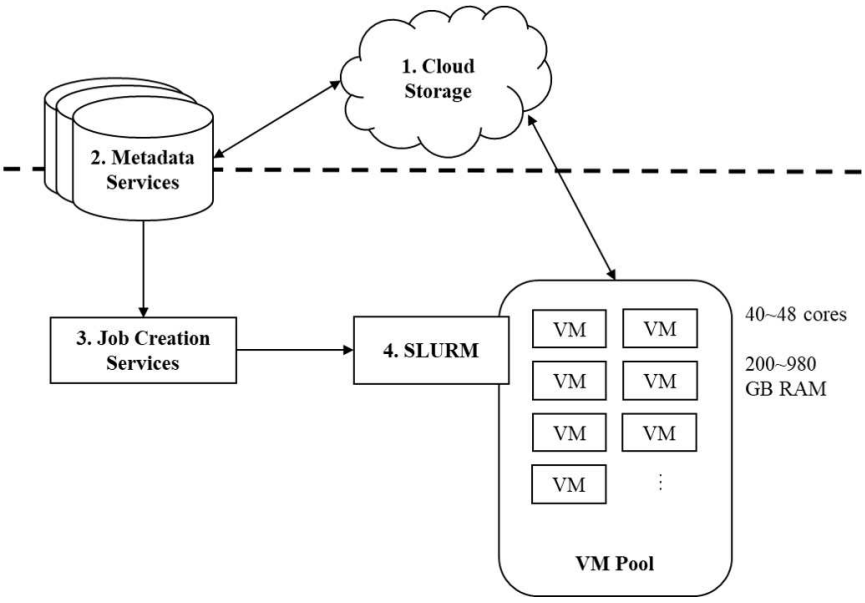

Figure 2.10: System Organization of the GPAS

1. Cloud Storage: Cloud storage is the data warehouse where the majority of raw data in the GDC is stored. GPAS uses cloud storage to conveniently download and process raw data and upload analyzed results. The GDC is designed to be S3-compatible, so users can select their favorite S3 data services. In GPAS, data is downloaded through a download client only when computing workflows in the system are created to analyze data. Data quantity for an analysis job ranges from a few gigabytes to hundreds or even thousands of gigabytes. Currently, in the GDC, the total data volume stored in cloud storage is 7.2 PB including both public data and internal unpublished data.

2. Metadata Services: This part is simplified in the figure, but metadata services bridge between the GPAS and the GDC. The metadata database for the GPAS is created through a few steps. Firstly, it clones the database from the GDC's database, and later the cloned database is reduced; only necessary nodes that pipeline jobs need are kept. This final database is also called the "BioGraph" in the GPAS. As in the GDC, metadata is stored within a graph. Metadata does not have a critical impact on workflow performance.

3. Job Creation Services: Job creation services mark the starting point of a job's

life. They comprise two components: "Work Creator" and "BioJSS" (Bioinformatics Job Scheduling Service). Work Creator queries the BioGraph for new jobs, and it combines data nodes and pipeline nodes to create CWL jobs. BioJSS assembles new jobs and hands them to the cluster manager. When jobs are submitted to the cluster, BioJSS periodically polls the cluster manager for job status and records it in database.

4. SLURM: GPAS chooses SLURM as the cluster manager for VM Pool. Although SLURM itself has its own scheduling algorithms, it is merely a job distributor in the GPAS, with little control over the scheduling of jobs.

5. VM Pool: Currently, in the GPAS, the VM Pool is on-premise and managed by OpenStack. The policy of VM allocation is simply to assign one VM per physical node. Almost all pipeline jobs, until recently, were executed in VMs. The VM Pool includes VMs residing on a variety of physical nodes, with CPUs ranging from 40 to 48 cores, memory ranging from 500 to 900 GB, and storage ranging from hard disk RAID to SSD RAID.

2.4 Computing in Bioinformatics Platforms

The large size of bioinformatics datasets and the large number of bioinformatics pipeline jobs requires a platform with tremendous computing power. Thus, only large-scale, high-performance computing clusters are used to support the bioinformatics platform. It is important to understand the type of computing required of bioinformatics pipeline platforms.

2.4.1 Computing Paradigms

Modern computers provide powerful computation capabilities, however, complex problems — in various fields such as science, engineering and business and etc. — require tremendously more computation power. Clusters of computers are deployed for this purpose. Computations in such scenarios will try to utilize the computing power from multiple computers to finish complete workloads.

Depending on the type and time required by a computation job, three paradigms are proposed:

- **High-performance computing (HPC)** classifies computing that requires large amounts of computing power for the workloads, and uses intensive communications between each computation task on distinct computers via a Message-Passing Interface (MPI). The computation tasks are considered to be tightly coupled and can usually be completed in a short period of time (within seconds). Thus, the latency of communications via MPI is crucial. Application performance in HPC are measured in floating-point operations per second (FLOPS).

- **High-throughput computing (HTC)** [77] also classifies computing that requires large amounts of computing power. However, the tasks for a computation job are usually loosely coupled or independent, and require long periods of time. Scientists using HTC are more interested in number of floating-point operations per week or per month that they can *extract* from the underlying computing cluster, rather than the FLOPS that the cluster can provide.

- **Many-task computing (MTC)** [99] is proposed to bridge the gap between HPC and HTC. It is aimed at computations that consist of a larger number of tasks, each of which takes a relatively short per-task execution time (seconds to minutes), yet is also data-intensive (tens of MB of I/O per CPU-second). Tasks in MTC can be individually scheduled on many different computing resources, and their performance is measured variously in FLOPS, tasks per second, MB per second, etc.

It is important to classify an application appropriately into one of the three computing paradigms, because benchmarking and metrics are quite different for them. However, since the applications from all three paradigms share the same requirement for computing resources of their underlying environments, the discussion for computing environment is

usually focused on HPC.

2.4.2 HPC Computing Environment

HPC was initially introduced as the form of supercomputers with a high level of performance compared to a general-purpose computer. Supercomputers are powerful but too expensive for most applications.

Grid computing, distributed computing, and cloud computing were then introduced to provide more affordable and flexible computing power. The boundaries between them are, however, blurry in some cases. A grid computing environment is usually built for applications that are embarrassingly parallel [32] and do not need communications between the parallel running tasks. Distributed computing [13] aims to achieve more control over the assignment of tasks to distributed resources. Cloud computing, in contrast, attempts to provide HPC-as-a-service through virtualization technologies so that HPC users can enjoy the benefits of scalability in cluster size, resources on demand, and inexpensive cost. A cluster of VMs acquired from a cloud platform can be easily used for completely parallel applications as in a grid computing on-premise cluster, or for tasks that need more control over assignment as in a distributed computing on-premise cluster, through installing different stacks of software. Moreover, because of the security and flexibility that VMs provide, more on-premise clusters have been deploying worker nodes with VMs, just as in the VM pool in the GPAS shown in Figure 2.10.

Recent research interest in cloud-based HPC computing environments lies on the performance available from a cloud provider compared to a traditional, on-premise cluster. Because of the probable overhead brought by VMs and performance impact brought by the multitenancy that cloud providers commonly use, performance variation and predictability in the cloud have been studied [69]. Besides, multiple studies [51, 61] of Amazon Web Services and Microsoft Azure have proven that only the computation-optimized instances provided by the

cloud providers, instead of the general-purpose instances, are suitable for building a HPC computing cluster. And the best cloud solution for a HPC platform varies from case to case in terms of the applications to be run on the platform. Besides, hybrid environments where workloads can run on an on-premise cluster while also using on-demand resources from a cloud platform is becoming more pervasive in the industry. Such combinations pose more challenges to getting the best performance from the unknown underlying characteristics of the cloud server to build a cost-efficient hybrid environment [88].

2.4.3 Bioinformatics Applications in High Throughput Computing

Because of their long-running times, bioinformatics applications are usually regarded as in the category of high-throughput computing.

HTCondor [77] was the first framework implemented for HTC purposes, and it stresses the amount of computing power that applications can utilize in the platform. The key mechanism in HTC is an efficient algorithm to satisfy resource requirements from tasks in a distributed environment [100]. With efficient resource management at the center of HTC, resource monitoring [54] in HTC is of prime importance. A more recent HTC system, HTCaaS [58] expands the resource management to handle a combination of grids and clouds.

It has been pointed out that HTC is very helpful for bioinformatics applications that tend to take months to complete [111], and also that achieving high data throughput for bioinformatics is a very complex problem. While there is relatively little research on bioinformatics in the HTC context, HTCondor has been modeled into a few systems for bioinformatics computing solutions [110, 57]. The Discovery Net system [103], and the galaxy [7] platform also try to bring HTC to bioinformatics.

2.4.4 Pipeline/Workflow Scheduling in the Cloud

Pipelines or workflows, being much more complicated workloads than single applications, raise interesting challenges to scheduling in the cloud where underlying hardware status is usually unknown.

Performance fluctuations on cloud-based VMs have attracted considerable attention [117] because it is difficult to guarantee user-perceived performance. Thus, workflow scheduling needs to be fluctuation-aware in order to make predictive decisions [73].

With the cloud allowing users to pay only for what they use, schedulers have to find a balance between time and cost. The majority of research in scheduling seeks to minimize the cost while meeting a deadline constraint [12, 101]. Thus, it is important to make accurate predictions or estimates for workflows, and various approaches [47, 53, 86] have been adopted. Another category of scheduling proposals focuses on reducing costs to meet quality-of-service requirements for workflows [82, 79, 98, 22]. Moreover, a large number of scheduling algorithms [46, 116, 101, 118] take a task-based approach in order to achieve finer-granularity scheduling results.

For more comprehensive taxonomies on scheduling algorithms for scientific workflows, please refer to these two surveys: [102, 45].

2.5 Virtual Machine (VM) Technology Background

2.5.1 Overview of Virtual Machines and VM Hypervisors

Regardless of how large a computation platform might be, people usually adopt virtual machine technology for rapid deployment, fast development and testing, and security requirements rather than using a huge number of bare-metal machines.

As shown in Figure 2.11, the architecture of virtual machines includes the VMs themselves, VM manager software, and host hardware. The VM manager software is often called a

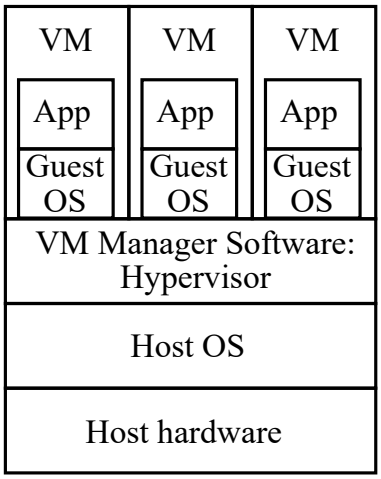

Figure 2.11: Architecture of Virtual Machines

VM hypervisor which is responsible for isolating VMs from each other, distributing hardware resources to individual VMs, and helping to execute instructions from VMs.

Virtual machine technology itself is a large topic and we don't want to dive too deeply into it here. It will be enough to focus on VM hypervisors and VM memory management mechanisms since they are most concerned with VMs' performance.

Depending on where VM hypervisors are situated, they are often classified into two categories. Type-1 hypervisors run directly on the system hardware, creating virtualized resources for guest operating systems directly on the hardware. Type-2 hypervisors, on the other hand, run on a host operating system that provides virtualization services such as I/O device support and memory management.

Type-2 hypervisors require a host OS to be installed first on the hardware, and they manage resources with assisted from the host OS. The advantages of this type are that it supports a wide range of hardware thanks to the host OS's additional layer of resource abstraction, and the installation of the hypervisor and VMs are much easier. However, the VMs' performance is reduced because of the additional layer of the host OS, and moreover, the stability of the VMs depends largely on the stability of the host OS.

On the other hand, type-1 hypervisors directly run on hardware, the distance between

VMs and the host's hardware resources is small, so the performance of VMs managed by type-1 hypervisors are usually higher.

Type-1 hypervisors include VMware ESX, Microsoft Hyper-V, and many Xen variants; type-2 hypervisors include VMware Workstation, VirtualBox, etc.

A popular hypervisor choice is Linux's Kernel-based Virtual Machine (KVM) because it ships with every Linux distribution. KVM can be classified as type 1 because it effectively converts the whole host operating system into a hypervisor. However, virtualized resources for VMs are still derived from the abstraction of the host OS, so KVM can also be classified as type 2.

In this paper, we classify KVM as a type 2 hypervisor due to its memory management mechanism as will be explained in the next section. KVM maintains a two-level memory address mapping from guest OS to host OS to hardware. However, type-1 hypervisors such as Xen use direct paging that directly maps guest OS memory address to hardware memory address, which effectively reduces the memory access cost for the VMs.

2.5.2 Important VM technology: Second-Level Address Translation

There are many notable technologies to improve VM performance and security. As this dissertation is mainly concerned with bioinformatics workloads that often require large amount of memory, the performance of VM memory management is vital to the performance of workloads. For type-2 hypervisors, such as KVM that OpenStack relies on, memory access performance is optimized by Second-Level Address Translation (SLAT). The implementation of SLAT in Intel processors is called the "Extended Page Table" (EPT), and we will use EPT directly since Intel processors are commonly used in GPAS cluster.

For an application to work properly in operating system, operating system ensures the translation between application's memory address space (Virtual Address, VA) and machine physical address (PA) space. Virtual machines add an additional guest operating system (the

Guest OS) layer to the stack, with the virtual machine hypervisor occupying the application layer on the physical machine, and applications are installed inside the virtual machine. The virtual machine hypervisor prevents the application from being aware of its environment, and takes care of translating the application's Guest VA (GVA) to Guest OS's PA (GPA), then translating the GPA to the address in virtual machine manager's virtual address space on the host (HVA), and finally translating it to the true physical address on the host (HPA), assisted by the Host OS .

This two-layer abstraction introduces three translation mappings for each memory access. Initially, VM hypervisors used software-based memory virtualization by implementing shadow page-tables that store the mappings from GVA to HPA for each guest process. This approach does not introduce address translation overhead for normal memory accesses in the virtual machines after the shadow page tables are set up, because the translation look-aside buffer (TLB) on the processor caches GVA to HPA mappings. However, since the hypervisor needs to maintain a shadow page table for each guest process, the memory requirement is higher. Besides, the overhead and implementation difficulties that come from maintaining synchronization between the shadow page tables and the guest page table are also increased.

In order to increase memory performance further for VMs in modern computers, Intel and AMD have both implemented solutions for second-level address translation (SLAT) with Intel VT-x since 2005 and AMD-V since 2006, respectively. This approach is hardware-based memory virtualization. For Intel, the Extended Page Table (EPT) is introduced so that the host OS builds this table to store GPA to HPA mappings. Intel processors use tags on the translation look-aside buffer (TLB) to avoid flushing the whole TLB for each context-switch, so that EPT entries can be stored in the TLB without frequent flushing penalties. During virtualization, the VM can access the TLB directly. As shown in Figure 2.12, EPT entries in the TLB cache GPA to HPA mappings for the guest, so when the guest OS is accessing memory, as long as the mappings are in the TLB, virtual CPUs can directly handle the

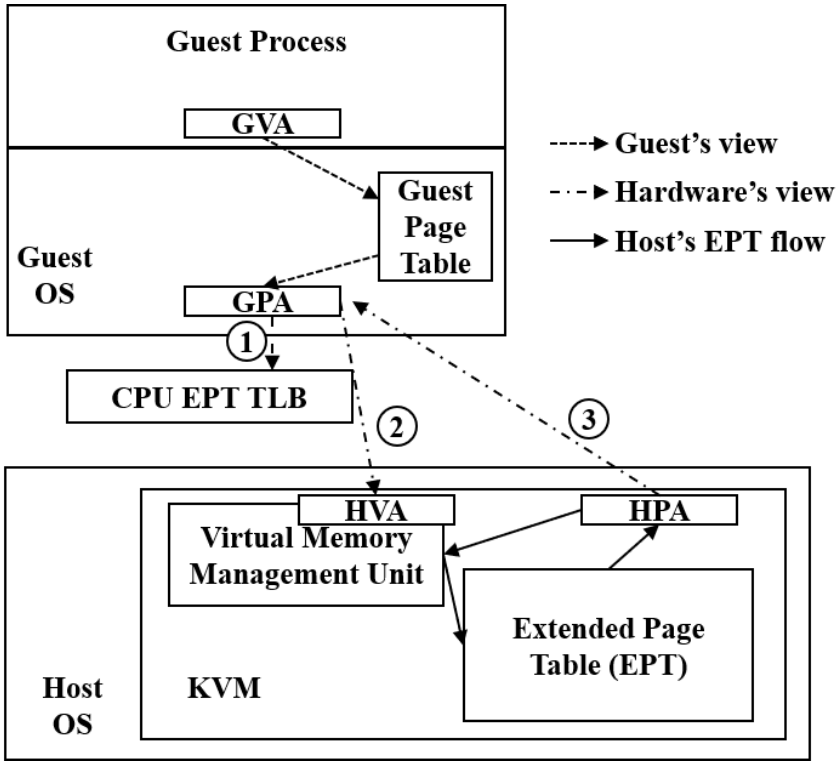

Figure 2.12: VM Memory Access with Extended Page Table (EPT)

address translation for the VM. That avoids switches between VM and host, shown as ① in Figure 2.12, and also avoids the need to synchronize address translations between mappings and the guest page table.

However, when the guest OS tries to access a GPA that does not have a mapping in the TLB, a TLB miss occurs and the system needs to switch from VM to host. This switching is referred as a VM exit event and is expensive (shown as ②). The host OS intercepts the GPA, and looks it up in the EPT in the virtual memory management unit, shown as ③.

Such a TLB miss as occurs in ② is also referred as an EPT violation. The host must do extra work to look for the mapping. This mechanism is easy to achieve because there is only one table per VM, compared to the per-guest shadow page-table process mentioned earlier.

This mechanism works well with good spatial-locality workloads on relatively little memory, as the TLB size is limited. However, if the locality is bad, in the worst case, accessing a GVP can cause as many as 24 table accesses in guest and host in a 4K-page setup. Because

both the guest page table and the host mapping table are implemented in 4 levels, a GVA needs to access four extra GVAs for each level of its guest page table. In the worst case, each extra GVA access causes a EPT violation, and then each of those causes the host to walk four levels of the mapping table. In the case of KVM implementation for EPT management, only one lock is designed for the virtual MMU mapping table, so when multiple processes in the guest OS are causing EPT violations, there will be serious contention on the lock.

2.5.3 Related Evaluations and Researches on SLAT

Over the years, there have been a handful of evaluations based on benchmarks for VM performance with the adoption of SLAT, or of Intel's EPT. These evaluations are mainly from virtualization vendors such as VMware.

In 2009, an evaluation [20] compared between software-based memory virtualization (shadow page tables) and hardware-based memory virtualization (EPT) memory virtualization. Benchmarks used in the evaluation included Kernel Microbenchmarks, Apache Compile, Oracle Swingbench, Specjbb 2005, Order-Entry benchmark, SQL Server Database Hammer, Citrix XenApp. EPT was found to have significant performance increase over shadow page tables except SPECjbb2005, which is an industry-standard, server-side Java benchmark. This exception is because Java's usage of the heap and associated garbage collection increases TLB misses. Huge pages can be used to reduce TLB misses and were found to increase performance.

Another performance analysis [21] in 2013 found a considerable performance drop for EPT compared with native performance on a bare-metal node. The workloads evaluated included OLTP workload, Hadoop, and latency-sensitive applications. Performance was evaluated using hardware event counters and cycle-accurate measurements. The evaluation found most of the virtualization overheads to be small, but that EPT costs indeed bring a large overhead to virtual machines.

A more recent study [83] in 2016 analysed memory load and cycles in hardware events. CPU-bound benchmarks and database benchmarks are evaluated in this study. This study found that the performance impact of TLB misses from EPT is not severe thanks to the increasingly larger TLBs and caches on modern CPUs. Yet another study [109] in 2016 found that performance increases from EPT are almost negligible compared with software-based solutions, as measured using the SPEC CPU2006 [44] benchmarks.

In our experience, we find that there indeed is no obvious performance impact in the short term, however, as VMs live longer, the impact becomes harder to ignore.

All the evaluations by other researchers [49, 87] agree that adopting a huge page can reduce TLB misses. However, with server memory and applications' memory demands both increasing, the demand for an even larger page size is emerging as well. It is found that, for big-memory workloads that use a wide range of memory, such as databases, in-memory caches will still suffer from TLB misses even with huge pages [17].

Further reasons to avoid using huge pages for VMs are because huge pages amplify memory allocation requests when VMs, and it defeats the purpose of page sharing — thus huge pages must be broken into small pages [41]. This is one of the reasons that led to the problem of page splintering. Some of the research on huge pages are addressing this problem and may mitigate the difficulties of using huge pages [97, 96]. However, the study [83] finds little impact from page splintering. Some other researchers have addressed possibilities for making huge pages more flexible by tracking page movements [94], dynamic page sizes [8, 74, 37, 40, 93], dynamic huge page-based memory allocation and ballooning [107, 70, 48] and even fundamental redesigns [65].

The most effective, but also the hardest, approach is to reduce TLB misses. Some researchers focus on introducing translation cache structures [19, 14], or limiting direct paging for virtual machines [17, 36]. Other researcher [15, 9, 56] have proposed a speculation mechanism to reduce TLB misses.

CHAPTER 3

PIPELINE PERFORMANCE SUMMARY FOR THE GPAS

3.1 Overview

The organization of the GPAS intuitively scatters job execution-related information across multiple databases in the GPAS, which makes it difficult for the team to review system status and performance on the GPAS.

This chapter presents a platform statistics synthesizing service `biosyncdb` that pulls statistics into an all-in-one database. Using this database, the current status of pipeline performance and system achievements are illustrated.

This database serves as the most critical part of this dissertation and is invaluable for future researches on the GPAS' performance.

3.2 Platform Statistics Synthesizing Service `biosyncdb` and Statistics Database `res_biodb`

In §2.3.2, the organization of the GPAS is described. A major drawback of the loosely coupled components is that data related to pipeline job performance is scattered over multiple databases. There are three databases and one data service in the GPAS:

1. **Graph-oriented database in PostgreSQL**: Relationships between projects, pipelines/workflows, jobs, input data, output data are recorded in the representation of edges. However, no information beyond UUIDs or names is stored.

2. **BioJSS Database**: As mentioned in §2.3.2, the "BioJSS" service handles the scheduling and dispatching of jobs, and it records important information about input/output documentation, pipeline configuration and job status at its backend database. However, it replaces failed job records if the job is attempted again.

3. **SLURM Database**: The SLURM database records all necessary information for a job in terms of running status, resource allocation, submit time, start time, end time, etc . Besides, it keeps failed job records as well.

4. **File Indexing Service**: As explained in §2.3.1, Signpost serves as an indexing service to provide necessary information for a file when the file's UUID is given, such as file name, download URL in the cloud storage, and file size.

Naturally, it is impossible to measure the performance of a system without knowing how much data it processes. Besides, as explained in §2.1.2, job performance is only comparable if the jobs are in the same projects, of the same pipeline, and processing data with the same experimental strategy. Thus, to assess the performance of GPAS, data from multiple sources must be combined.

One of the major contributions in this dissertation is that a platform-wide statistics synthesizing service (`biosyncdb`) is created, and a statistics database (`res_biodb`) is thus created.

Table 3.1 shows an overview of the database scheme for the `res_biodb`. The algorithm of `biosyncdb` is very straightforward; it pulls data from the databases as shown in **Source** column for each table in the `res_biodb`, in order. The performance of `biosyncdb` is constrained by two factors:

1. The APIs provided by the graph-oriented database developed by the GDC are limited and do not allow arbitrary JOIN operations. Thus, in order to get information such as how many jobs are using the same input file, instead of querying the file table and joining with the job table, `biosyncdb` must traverse all the job nodes and build a map to store the affiliation between files and jobs.

2. The backend database of the file indexing service is not exposed for security considerations. And the query `http` point is also used for job execution purposes in the GPAS.

Table name	Purpose	Source
`program`	Program and project compose the project id	1. graph-oriented database
`project`	As above	1. graph-oriented database
`workflow`	Workflow name, UUID information	1. graph-oriented database
`workflow_biojss_info`	Additional git repository information	2. BioJSS database
`job`	Job UUID, workflow affiliation	1. graph-oriented database
`job_graph_info`	additional information on experimental strategy, project affiliation.	1. graph-oriented database
`job_biojss_info`	additional information on job input/output document, etc.	2. BioJSS database
`datafile`	UUID, data size, etc.	1. graph-oriented database, 4. file indexing service
`datafile_job`	Job and data file affiliation	1. graph-oriented database
`job_slurm_info`	Job states, submit time, start time, end time, and resource allocation	3. SLURM database
`job_metrics`	SLURM recorded average resource utilization metrics	3. SLURM database
`job_to_job`	Dependence between jobs	1. graph-oriented database
`workflow_to_workflow`	Dependence between workflows	1. graph-oriented database
`last_sync`	Last timestamp that the service runs incrementally	

Table 3.1: Synthesized Statistics Database Scheme Overview

In order not to swarm the query point, `biosyncdb` limits the frequency of information queries about e.g. data size from the file-indexing service.

As with ESBuild in the GDC, it also takes longer and longer times for `biosyncdb` to synthesize all the data. To optimize performance, `biosyncdb` synchronizes data in batches and only checks for newly changed data since the last synthesis. Currently, `biosyncdb` runs once a day and it takes around one hour to finish synthesizing the new data since the previous day.

Currently, `res_biodb` stores all the available and necessary information for more than 904,484 data files, 362,400 job attempts (including successes, failures, test attempts that are not presented in the graph-oriented database), and the total size of the database is merely 2.3 GB. This database is valuable because most of the remaining discussion in this dissertation is based on the data stored in `res_biodb`, and it provides enough data to generate an overall impression on the status of the GPAS. Besides, other unused data such as `job_to_job` and `workflow_to_workflow` will be valuable to future research in the relationship between jobs and workflows.

3.3 Job Performance (*processing rate*) on the GPAS' VM Cluster

3.3.1 Job Performance Definition: processing rate

The majority of the jobs in the GPAS process a large amount of input data and take a long time to complete. Thus, a simple metric for understanding job performance and its degradation is the processing rate:

$$processing\ rate = \frac{job\ execution\ time}{input\ data\ size} \tag{3.1}$$

Performance is measured in time per gigabyte; thus a larger *processing rate* value means the performance is worse (as more time is needed for processing one GB input data).

3.3.2 Performance in the GPAS and Performance Tails

In this section, we present *processing rate* for projects that have the most pipeline jobs.

In this section, an overview of the job performance in the GPAS is given and a critical issue of performance is discussed.

As shown in Figure 3.1, Gaussian kernel density estimation is applied to the *processing rate* for four pipelines. As expected, the distribution of *processing rate* roughly complies with

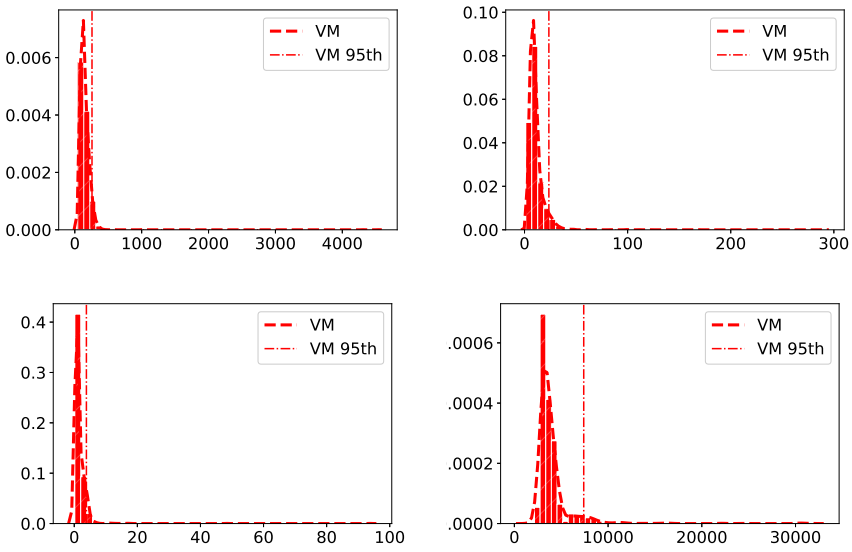

Figure 3.1: Gaussian Kernel Density Estimations for *Processing rate* (seconds/GB) of Four Pipelines on VMs

a normal/Gaussian distribution. However, what concerns us the most is that the slowest *processing rate* is abnormally bad, as shown to the far right of the graphs.

In Figure 3.1, the vertical line shows where the 95th percentile *processing rate* resides, and to the far right of the vertical line of each graph, there is a long "tail" reaching to even longer times required to process one GB of input data. We refer this as the "**tail *processing rate***" for a pipeline. Jobs that suffer from tail *processing rate* tend to contribute more to the total execution time for all the jobs for a pipeline. In the case of these four pipelines, jobs that are slower than 95% of all the jobs, contribute to the total execution time for all the jobs by 11.4%, 15.2%, 22.2% and 13.8%, respectively. In the worst case, tail jobs in another pipeline contribute 53.1% for the total execution time.

This **tail *processing rate*** issue will be discussed in detail in Chapter §4.

One method to reduce the tail effects is to run jobs on physical hosts directly instead of using VMs. This type of computing node is called "bare-metal nodes". Since June 2019, the GPAS has started to run a small portion of jobs on bare-metal nodes, as it has been proven that bare-metal nodes more reliably provide stable performance for jobs. Figure 3.2

provides a more intuitive comparison of *processing rate* between jobs on VMs and jobs on bare-metal nodes, where jobs on a VM do have longer tails than jobs on bare-metal nodes. Jobs on bare-metal nodes that are slower than 95% of all the jobs, contributing to the total execution time for all the jobs by 7.1%, 6.3%, 6.8% and 7.4%, respectively. However jobs of the same pipeline that are slower than 95% on VMs contribute 8.0%, 24.8%, 6.2% and 10.4%, respectively. For the pipeline mentioned earlier where slow jobs on VMs contribute 53.1% of execution time, the jobs of this pipeline that run on bare-metal nodes contributes only 8.1%. More details will be discussed in Chapter §4.

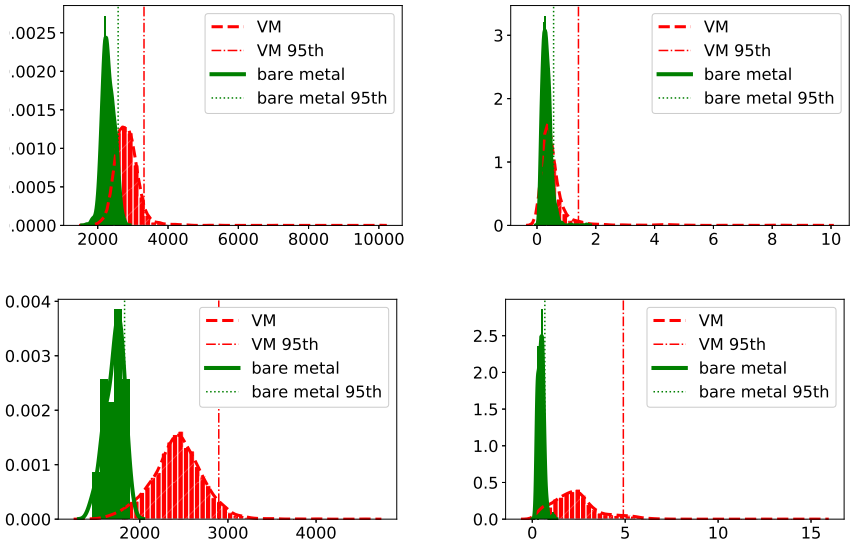

Figure 3.2: *Processing rate* (Seconds/GB) Comparison for Jobs on Bare-Metal Nodes and VMs

Since the main goal of GPAS is to achieve high throughput for pipelines, the most urgent task for us is to eliminate the tail jobs. One approach that we have started to adopt is distributing jobs on the bare-metal nodes directly. We are also actively exploring other options to run jobs more efficiently in the VMs.

3.3.3 Variation in Non-tail Performance

As shown in the previous section, jobs on bare-metal nodes exhibit much more stable performance than those on VMs (at least for the VMs with current configurations in the GPAS), and the distribution of *processing rate* on bare-metal node is more centered to the mean *processing rate*, and 95th-percentile *processing rate* is closer to the center as well. However, there still exists variation. This section discusses some factors that correlated with the variation of *processing rate*. All the statistics are collected from bare-metal nodes to avoid interference from VM performance instability.

Through linear regression and significance-test analysis, it is found that, among other parameters known before a job starts (such as number of input files, number of CPUs allocated, etc.), input data size exhibits the strongest correlation with *processing rate*. Due to the small number of jobs running on bare-metal nodes, only a few pipelines are suitable for regression analysis; hence four pipelines are discussed in this section.

To better illustrate the correlation between input size and *processing rate*, jobs are sorted by their *processing rate*, and divided into 20 buckets according to the percentile of *processing rate*. Average input size of each bucket is shown in Figure 3.3, 3.4, 3.5. There can be either positive or negative correlation between input size and *processing rate* for different pipelines. Definitions of the pipelines have been investigated to help explain the correlations. Hence, at least three relationships emerge from this analysis:

1. **Positive correlation** (larger input size means slower/larger *processing rate*): As shown in Figure 3.3, *processing rate* is positively correlated with input size. At the same time, according to the graph to the right, *processing rate* of this pipeline is generally very large, more than 1500 seconds/GB. Indeed, the workloads in this pipeline contain applications to align sequences to a human genomic database, which, in a nutshell, boils down to string-searching. Thus, it is common for the pipeline to have $\mathcal{O}(N^2)$ computation complexity, and it exhibits positive correlation between input data size

and *processing rate*.

2. **Negative correlation** (larger input size means faster/smaller *processing rate*): Contrary to the previous result, Figures 3.4 and 3.5 show that *processing rate* is negatively correlated with input size. This demonstrates that *processing rate* in these two pipelines is much faster than for the jobs in the previous one. By inspecting the definitions of these two pipelines, we find that these two pipelines are designed to do simple work involving downloading and extracting files. With a high-bandwidth network and I/O, *processing rate* i s higher. However, since the execution time for such pipeline jobs is short, the overhead of spawning containers and other components cannot be ignored. As a result, smaller input size leads to worse *processing rate*.

3. **External factors interference**: In addition, there are some cases that may not comply with the previous behaviors. Figure 3.6 shows two different distributions for job processing rate of this pipeline. In the significance test, jobs in this pipeline show a much higher p-value (0.038). The p-value indicates the probability to observe data as extreme or more extreme given that the null hypothesis is true, and thus the smaller the p-value, the stronger the evidence that we should reject the null hypothesis. In this case, the p-value of 3.8% provides an evidence to reject the null hypothesis and favor the alternative hypothesis that input size has an impact on processing rate. We also observed a much stronger evidence with p-value of 0.001% for the former pipelines. By inspecting job logs, it is found that jobs at the right peak (slower jobs) spent significantly longer times downloading data. That implies that there might have been network congestion in the cluster during that time. Hence, in rare cases, we need to take the external environment into account to understand the *processing rate* of jobs.

To conclude, through statistical analysis of the jobs, although there are some degrees of variation, we find that *processing rate* for pipelines is highly correlated with a job's input size. The exact correlation depends on the workloads associated with the pipeline. Sometimes,

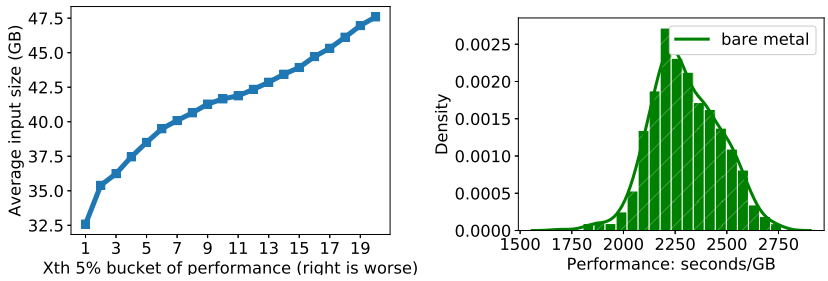

Figure 3.3: Input Size and *Processing rate* Positively Correlated

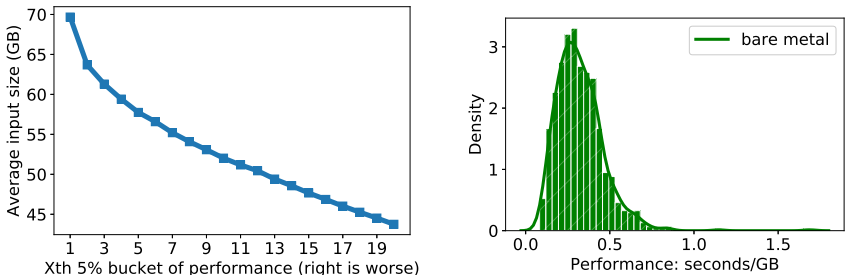

Figure 3.4: Input Size and *Processing rate* Negatively Correlated (1)

the *processing rate* might also be subject to external factors during some parts in the job. It is important to understand the nature of the job so that good interpretation can be formed.

Chapter §5 will discuss the importance of defining a finer-granularity pipeline scheduling mechanism so that it will be easier to model the *processing rate* of jobs.

3.4 Overall System Status

Thanks to the data collected by `biosyncdb`, here we present an overview on the system status of the GPAS, which has not been conclusively investigated before.

3.4.1 Sustainable Data Processing

Starting March 2018, the GPAS has been devoted into large-scale bioinformatics pipeline computing. Figure 3.7 and 3.8 show the accumulated input consumption and output production over the months. The first few months included development, testing and small-scale

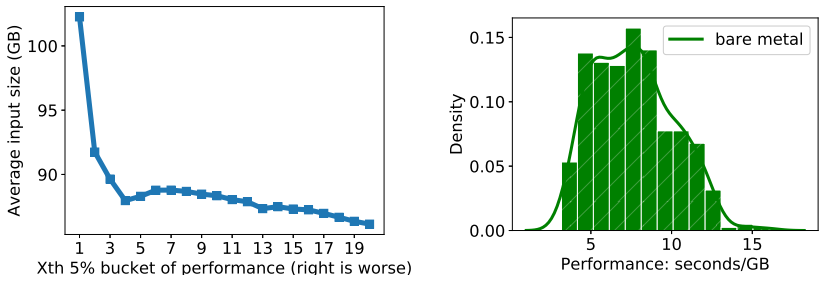

Figure 3.5: Input Size and *Processing rate* Negatively Correlated (2)

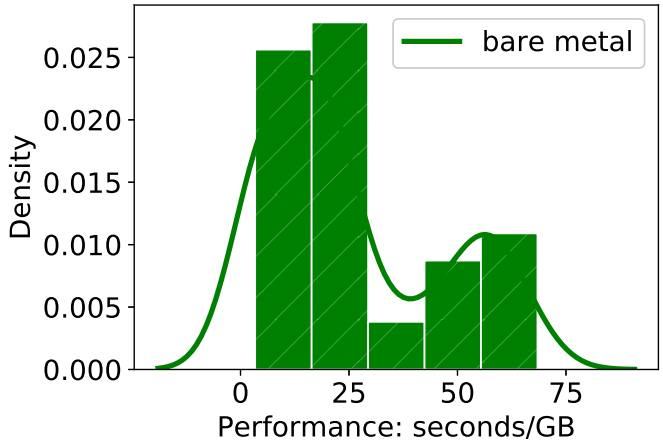

Figure 3.6: Abnormal *Processing rate* Distribution

production in GPAS, thus the rate is relatively low. Beginning from July 2018, the GPAS started large-scale production pipelines in the platform and the data-consumption and production rates stabilized over subsequent months.

3.4.2 Sustainable Computing Power

Figure 3.9 shows the accumulated computation time that the GPAS has utilized in the cluster. Over the two years, 200 job computing years have been utilized for computation task.

A large room for improvement on computing power utilization exists due to the design of job scheduling in the GPAS. As shown in Figure 3.10, the CPU allocation exposes a lot of

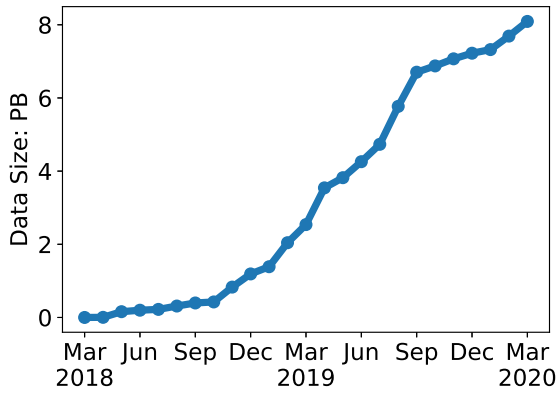

Figure 3.7: Accumulated Input Data Consumption in the GPAS

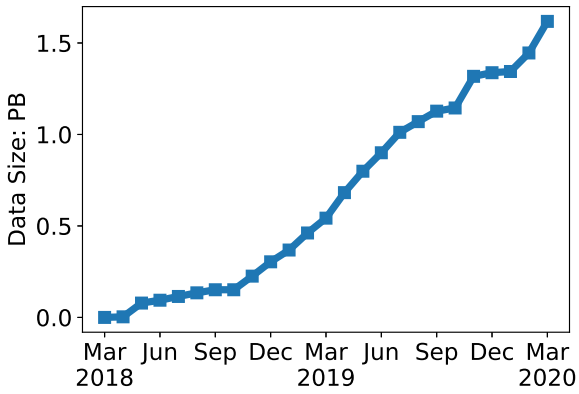

Figure 3.8: Accumulated Output Data Production in the GPAS

inefficiencies in the current design of the GPAS. This resulted from various reasons. Current practice in the GPAS is that never to allocate all the CPUs to the jobs on a VM. This is to ensure that there is no contention on CPUs because most the applications in bioinformatics are computation-intensive. The other reason is that jobs in the GPAS are contract-based. In most cases, pipelines must get permission before they can be submitted into the GPAS.

More details will be discussed in Chapter 5 on what measures can be adopted to improve the efficiency of job scheduling.

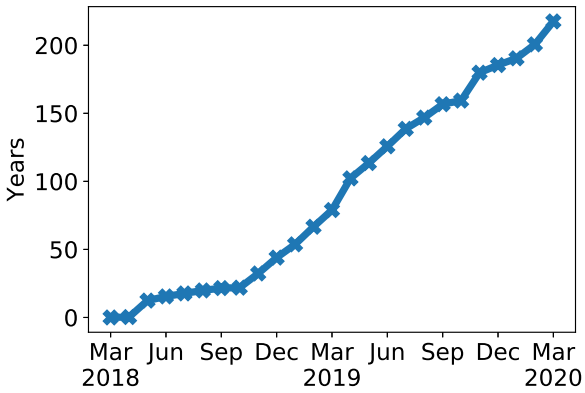

Figure 3.9: Total Computing Time by the GPAS in Years

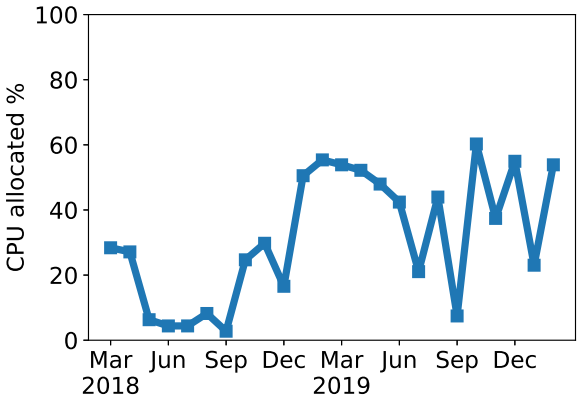

Figure 3.10: Monthly Average CPUs Allocated in GPAS

3.4.3 Monthly Job Execution Details

Figure 3.11 shows the number of job attempts for each month, and it shows dramatic changes in number of job attempts over the months. This corroborates the variance in monthly CPU allocation covered in the previous section. Admittedly, current implementation of the GPAS still needs improvements for finer-granularity resource management. Also, the team is actively exploring the possibility of running non-priority jobs in the background to improve resource utilization.

Figure 3.12 shows the success rate for jobs in each month. The GPAS managed to maintain a success rate of around 90% in most months. However, sometimes it might have faced interruptions to production jobs or test jobs, which is also reflected in the figure.

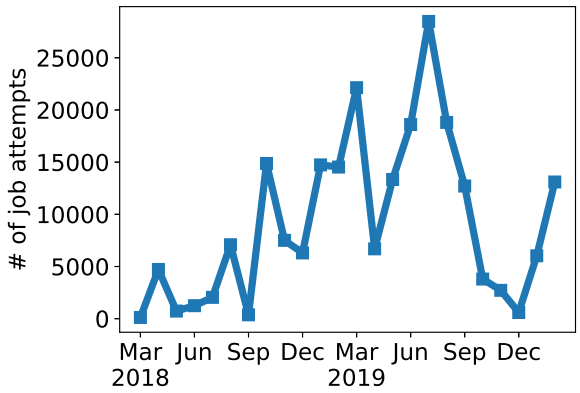

Figure 3.11: Monthly Job Attempts in the GPAS

Starting in June 2019, bare-metal nodes turned out to provide much more stable performance than the VMs, thus the GPAS phased out a small portion of the VMs and used bare-metal nodes directly. The bold green line in the figure shows that the success rate on bare-metal is relatively high. Since jobs that run on bare-metal nodes are few in number, the overall success rate is still dominated by jobs that run on VMs.

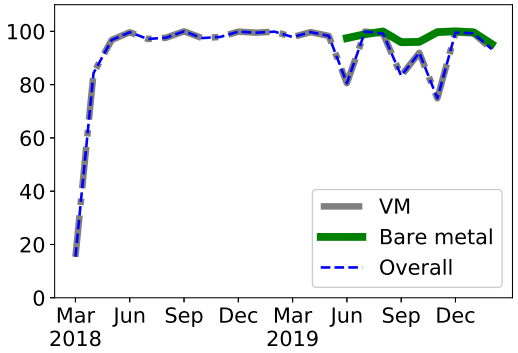

Figure 3.12: Pipeline Jobs Success Rate

3.5 Summary

This chapter presents the all-in-one database (`res_biodb`) created to facilitate the performance study on the GPAS, and a definition to measure job performance is given. By examining the data collected in `res_biodb`, this chapter presents various metrics that can be

acquired:

- An issue is exposed in which some jobs that run on VMs in the GPAS exhibit abnormally poor performance (tail performance).
- Variation in non-tail performance is studied and the input size is found to be the most significant correlate with the job performance.
- A system status overview on the two-year production of GPAS is presented to showcase the achievements and to identify space for improvement.

CHAPTER 4

PERFORMANCE IMPLICATIONS OF AGING VMS IN THE GPAS

4.1 Overview

As briefly discussed in §3.3.2, some jobs that run on VMs exhibit **tail *processing rate***. Since the workloads in the GPAS are I/O intensive, memory-consuming, and most importantly long-running (the longest job takes 39 days), the combination of computational demands makes the issue hard to troubleshoot. This chapter expands the discussions and shows the steps we take to pinpoint the root cause and evaluate fixes to this issue.

4.2 Motivation

This section shows that some jobs exhibit tail performance, and why this matters.

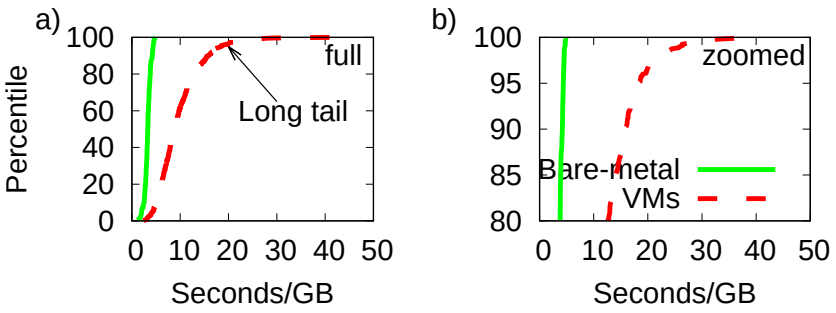

Figure 4.1: CDF of *processing rate*

Figure 4.1 shows the CDFs of *processing rate* of jobs on VMs and bare-metal nodes. There are 796 and 247 jobs on VMs and bare-metal, respectively, and all the jobs run the same type of pipeline ("variant-filtration.pindel"), hence all the jobs should display similar performance. In Figure 4.1a, the nearly-vertical solid line represents *stable processing rate* of jobs on bare-metal nodes. However, the dashed line shows that *processing rate* on VMs is generally worse than *processing rate* on bare-metal nodes. While this is expected, the issue

lies in the *tail* performance, especially at high percentiles. According to the zoomed graph in Figure 4.1b, starting from the 80th percentile, *processing rate* on VMs is 3 times worse than on bare-metal nodes, and at higher percentiles, the performance worsens by an even larger magnitude.

We also would like to note again that the jobs in Figure 4.1 come from only one type of job pipeline ("variant-filtration.pindel") that generally requires no more than 40 seconds/GB. However, there are other, more CPU/memory-intensive pipelines that require as much as 1000 – 3000 seconds to process each GB. The problem raised in this dissertation becomes worse for these type of pipelines.

To show that the behavior in Figure 4.1 is not because of degraded machines or hardware, we take six VMs, each running on its own physical host, and show the statistics of the job performance for these 6 VMs in Figure 4.2 in a boxplot. In every VM (*e.g.*, VM1 on machine1), users can run the same job pipeline repeatedly. As shown in the figure, the *processing rate* in a single VM varies (even though the same pipeline on the same machine) and the performance across the VMs also varies (even though the VMs have the same configuration).

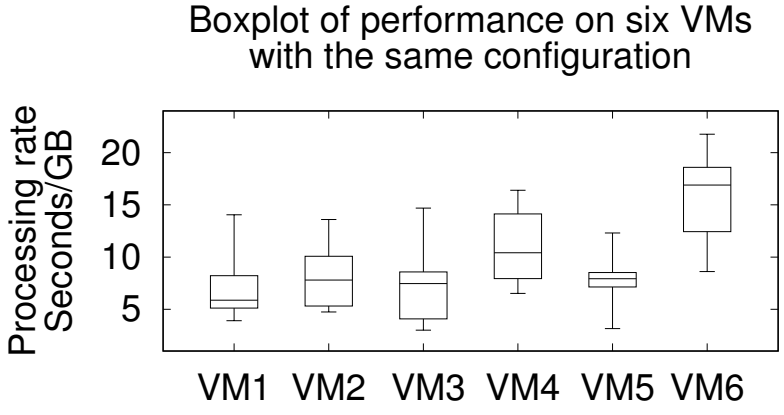

Figure 4.2: *Processing rate* variance on VMs

4.3 Investigation on the Tail Performance and Aging VMs

In order to understand the root cause beneath the issue, we conducted a set of experiments that included micro-benchmarks and real workloads. All experiments were conducted on a pair of VMs with the same virtual hardware setup and the same software setup, except that one of the VMs was cold-restarted before the experiment. We refer this cold-restarted VM as the *Fresh VM*. The other VMs (*Aged VM*) had been running for a few days and were selected for close monitoring after slow jobs were observed.

Each VM was the only tenant on its host and configured to have 40 vCPUs, 226 GB RAM and 2.5 TB storage. The hosts were equipped with two 2.20 GHz, 12-core, 24-thread Intel Xeon E5-2650 v4 processors, 504 GB RAM and 7.3 TB SSD RAID-5 storage. Linux-4.4 kernel, libvirt 1.3.1 and OpenStack Nova 13.1.4 were installed for virtual machine support.

4.3.1 Application-Level Measurement

In GPAS, we noticed that jobs started running slower on VMs that have been running for several days. We ran a simple application to reproduce this phenomenon. To simplify the experiment, we broke down a widely-used pipeline in GPAS (Somatic Variant Calling) and selected only one of the tasks, a Java application called VarScan2 [59]. Job pipelines in GPAS are spawned as multiple processes, hence VarScan2 can run as multiple processes.

Since the input data is the same for all the tests, here we do not use *processing rate*, but instead *execution time* as the performance metric. A higher execution time implies worse performance (the same as in *processing rate*).

We define an "n-process VarScan2 *task*" as a task that uses n VarScan2 processes to process the input data. We define a "*test*" as an experiment that concurrently runs 5 tasks of 8-process VarScan2 on a single VM. The input data is replicated 5 times and each task performs the same computation on the same content but using distinct replicas of the data. After a test completes, we measure and record the *average execution time* of the 5 tasks in

the test, and then repeat the test until 5 days have elapsed.

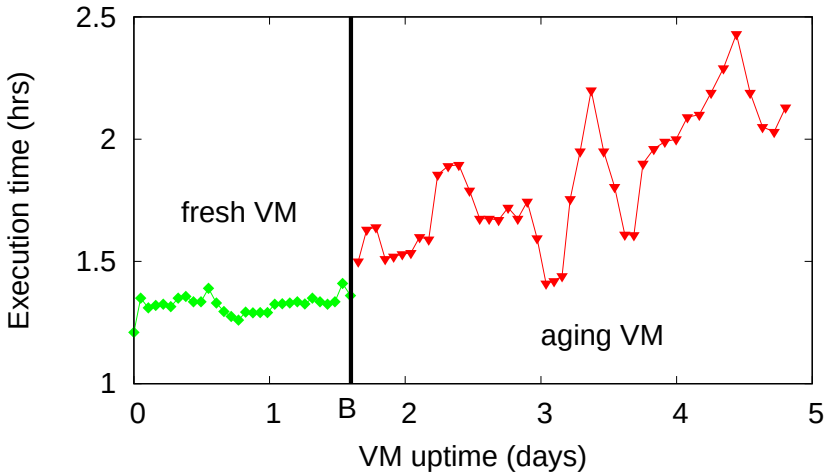

Figure 4.3: VarScan2 experiment

Figure 4.3 shows the average execution time of the tasks across several days. Every point represents a test; the vertical axis measures the average execution time of the 5 concurrent tasks in the test. We can observe that, before day 1.6 (marked by line B), test execution is relatively fast with low variance. However, after approximately 1.6 days, the performance starts to degrade. We hence observe that VarScan2 tasks perform normally on a *Fresh VM*, but perform much worse on an *Aged VM*.

4.3.2 Kernel-Level Measurement

To understand the slow performance in an *Aged VM*, we conducted further experiments including micro-benchmarks and in-kernel measurements in the *Aged VM*.

We use `sysbench` [60], a configurable multi-threaded benchmark tool that provides a variety of tests for benchmarking CPUs, multi-threading, memory operation, and file I/O performance. We performed many varieties of experiment and found that most benchmarking results do not reveal much difference between an *Aged VM* and a *Fresh VM*, except for file I/O performance. Not only does the I/O throughput show different results; but, more

interestingly, the monitored CPU utilization on *Aged VM* and *Fresh VM* are quite different from that of the host OS.

A common way to monitor CPU utilization is by calculating the differences between accumulated values in the pseudo-filesystem (`/proc/stat`). Simply speaking, the values represent how many time slices have been used for each process type (user, system, etc) and for each CPU. There are many tools that profile CPU utilization including `top` and `scollector`. We use the latter as it collects and saves raw data, allowing us to use other tools to calculate and visualize statistics in desired ways.

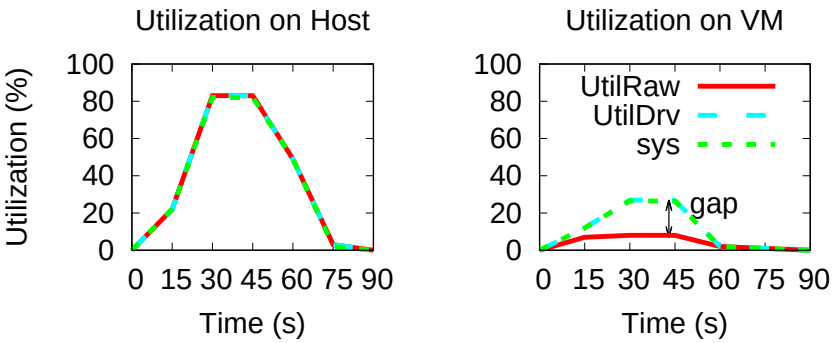

Figure 4.4: CPU utilization of `sysbench`

Figure 4.4 shows the comparison of CPU utilization collected from the host OS versus utilization in an *Aged VM*. In the VM, we ran a `sysbench` file I/O test (on an SSD). It is worth noting that there are two lines in the figure (`UtilRaw` and `UtilDrv`) representing the two ways we calculate CPU utilization. `UtilRaw` is the raw CPU utilization number (in %) that `scollector` outputs every second. In addition to the raw CPU utilization, `scollector` also outputs more detailed information including `cpuUserSlices`, `cpuSysSlices`, and `cpuIdleSlices`. We define `UtilDrv` (for derived) by summing the user and system time slices and dividing by all slices ((`cpuUserSlices`+`cpuSysSlices`)/`allSlices`). Again the difference is that `UtilDrv` simply sums the `cpuUserSlices` and `cpuSysSlices` without considering the `idleSlices`. The figure also shows another line `sys`, which represents `cpuSysSlices` divided by all the slices (in %).

We make the following important observations from the figure.

1. System (`sys`) CPU utilization is high on the host and equal to the overall CPU utilization. A similar observation can be made of the *Aged VM*. We consider this abnormal because the workload is I/O-bound.

2. Peak CPU utilization observed on the host is 82%, while it is only 27% on the *Aged VM*. Note again that there is only one VM on the host and no other heavy workloads running on the host. This implies that *the hypervisor works intensively, a hidden CPU overhead*.

3. On the VM, there is a *discrepancy* between the two ways we measure CPU utilization (the gap between `UtilDrv` and `UtilRaw`). Normally, these two lines should overlap as in the host-level measurement (left graph). What happens here is that `scollector` assumes that when a process gets a time slice (10 ms), it always gets the entire time slice. However, our method, `UtilDrv`, shows a "loss of time" due to the hidden overhead in the hypervisor. CPU time that is supposed to be used for tasks in the VM was in fact used for other system tasks in the host.

4. On a separate measurement on a *Fresh VM* (not shown for space), we found no such high system CPU usage, nor a discrepancy between the two lines.

4.3.3 Memory Fragmentation

A major problem of aging resources is fragmentation. We started suspecting that there was a memory fragmentation problem where sequential guest pages were not mapped sequentially onto the physical pages. To get more evidence, we ran concurrent processes and analyzed the read operations. Roughly speaking, a file read operation goes through five steps: 1) check the page cache to see whether the data is already in memory; 2) conduct a synchronized read request at the file- system level; 3) send out an asynchronized read request at the block

level; 4) wait for completion; and, 5) copy the data from the kernel to the user space. Of all these steps, Step (5) is the most memory-intensive .

We conducted an experiment that records the time spent in each of the steps. We set up four tests. The first test runs a single 1-process VarScan2 task, and the other four tests run 1, 4, and 5 (concurrent) 8-process tasks, respectively.

	1×1	1×8	4×8	5×8
Steps #1–4	< 1	< 2	< 12	< 20
Step #5	**5**	**92**	**290**	**512**
Total	6	94	306	552

Table 4.1: Read latency break-down

Table 4.1 shows the average time that *every* process spends on each I/O step. Note that most of the time is spent in Step #5 (memory copying). With just 1 process (1×1), a process takes only 5 seconds in total for memory copying. With 40 processes (5×8), *every* process now takes 512 seconds in step #5, which is roughly a *100× slowdown*. We note that there is other contention in the SSD (as can be seen in steps #1–4), but, even with 40 processes, the I/O waiting time is not as severe as the slowdown from memory-copying. We also note that we have a 48-core machine, hence CPU contention should be almost negligible with 40 processes.

4.3.4 The Root Cause: EPT Violation

As our experiments all point to memory management overhead, it is time for us to quantify the root cause. After considerable investigation, we found that the root cause resides in the *extended page table (EPT)*, which is a technology invented to increase virtual memory performance for VMs. The hypervisor's use of EPTs is designed to be transparent to VM users. To our knowledge, EPT is used by only certain hypervisor implementations such as Linux KVM. In a nutshell, EPTs serve as a page table that stores the mapping between the VM memory address and the host physical memory address. Modern CPUs use the

translation look-aside buffer (TLB) to store a small subset of the EPT entries. Whenever there is a TLB miss, an *EPT violation* occurs, which causes the hypervisor to interrupt the VM to handle the violation.

Figure 4.5 shows an experiment with 5 jobs using 30 cores, running repeatedly on a fresh-state VM for ten days. Figure 4.5a shows the average of number EPT violations (in millions) observed in every 5 minutes. The figure clearly shows that, the longer the VM has been running, the higher the number of EPT violations. Figure 4.5b confirms the correlation between the number of EPT violations and the job execution time.

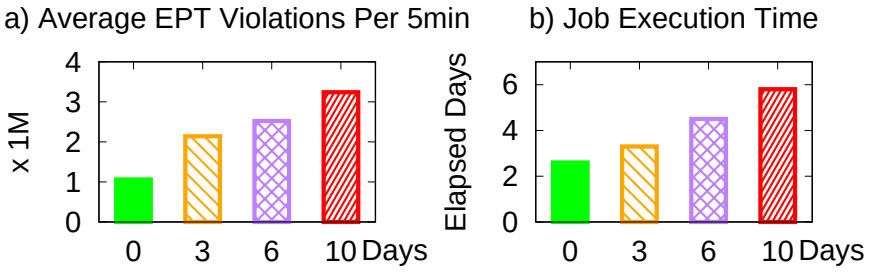

Figure 4.5: EPT violation

In conclusion, VM aging leads to more frequent EPT violations, causing the hypervisor to interrupt the VM more frequently. In the next two sections, we describe how to monitor and mitigate the problem.

4.4 Monitoring

We suggest two methods to detect VM aging: monitoring EPT violation (in the host) or CPU utilization gap (in the VM).

4.4.1 Host-Level EPT Violation Monitoring

One direct way to measure the problem is to count the number of EPT violations observed in the hypervisor, however, the result is relative — how do we know whether the number

represents a higher number of EPT violations than is normal? We found another, more concrete, metric to measure this problem: *address distance of subsequent EPT violations* (which basically attempts to measure the extent of memory fragmentation). For example, if two subsequent violations at times T and $T + t$ are about translation misses of guest pages #100 and #2000, respectively, then the distance recorded is 1900×4 KB. Essentially, we argue that, when the addresses of subsequent EPT violations are farther apart, the memory tends to be more fragmented and more EPT violations will occur, causing more time to be spent managing EPT violations.

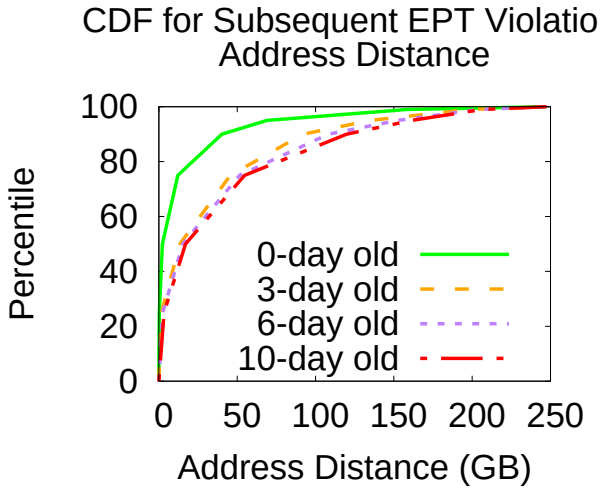

Figure 4.6: EPT violation monitoring

We return to the experiment in Section 4.3.4 and, this time, plot the distribution of address distances of subsequent EPT violations. Figure 4.6 shows different distributions categorized based on the age of the VM. Notice that older VMs have distinctly larger address distances. For example, in a 10-day old VM, we can see a distance of at least 50 GB appear in roughly 40% of the time (address distance 50 GB, percentile 0.6).

This monitoring method requires access to the host. It can provide performance alerts in advance. For example, the distance distribution in a 3-day old VM can be clearly distinguished from that in a fresh VM. Here, the resulting job execution time has been increased

by 27% (which was hard to observe in the middle of the job). Thus, this kind of monitoring allows us to predict job performance degradation even before the job ends. In terms of performance overhead, sampling violations in an online manner may bring an impact to system performance. However, our experiment, where the trace is enabled for five minutes every ten minutes, does not show a negative impact on performance.

4.4.2 VM-Level /proc/stat Monitoring

While the above method requires host-level access, we now present another method that can be done at the VM level. As explained before, the Linux kernel implements the `proc` pseudo-filesystem which provides an interface to kernel data structures. The `/proc/stat` file provides time-slice statistics across user, system, and idle processes from the time when the system boots up. In our deployment, the time slice is set to 10ms.

Inspired by the discrepancy shown in Figure 4.4 earlier, we observe that it is possible for users to monitor that discrepancy at the VM level. **The discrepancy is caused by time slices in the VM being actually — and significantly — less than 10 ms because the hypervisor is handling EPT violations.** Thus, we can introduce a simple metric, *vCPU Efficiency*, which is the sum of all the time-slice values in `/proc/stat` (user, sys, and idle values) divided by the real time slices that have elapsed (since the last time we read `/proc/stat`). The metric should be near 100%, but when EPT violation is high, it is expected that the efficiency will be much lower. Figure 4.7 shows a simple shell script to calculate *vCPU Efficiency*.

	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

Table 4.2: *vCPU Efficiency*

```
stat_period=300
cpunum=‘grep -c ^processor /proc/cpuinfo‘
read cpu user nice system idle ... < /proc/stat
total_HZ_prev=$((user+system+nice+softirq+steal+...))
sleep $SLEEP
read cpu user nice system idle ... < /proc/stat
total_HZ_cur=$((user+system+nice+softirq+steal+...))
ElapsedHZ=$((total_HZ_cur-total_HZ_prev))
vCPU_Efficiency=$((ElapsedHZ/cpunum/stat_period));
echo $vCPU_Efficiency
```

Figure 4.7: Shell Script to Calculate *vCPU Efficiency*

Table 4.2 shows *vCPU Efficiency* and the job execution time for the same experiments we ran before. Here, we select one "heavy" and one "light" application, where the heavy application uses all 6 available cores per job and the light application uses 1 core per job. We can see that, for heavy workloads, there is a large difference in *vCPU Efficiency* and execution time. For example, when *vCPU Efficiency* drops from 99% to 83%, the resulting job execution time increases from 16 to 39 hours.

The disadvantage of this method is that we cannot detect VM aging unless we run a heavy application whose performance can be impacted by the aging. Table 4.2 shows that, for a light application, there is no visible difference in the *vCPU Efficiency* or execution time, even though the VM is already degraded at runtime. We also want to emphasize that VM aging *cannot* be crudely defined by the number of days a VM has been up running. In our deployment, VMs age quickly (after 3–6 days) because they had to deal with complex bioinformatics pipelines.

4.5 Mitigations

This section describes several ways that we tried to address the problem. It is up to the system requirements and administrators to decide which mitigation technique is most appropriate. Table 4.3 summarizes the pros and cons of the mitigation techniques we discuss below.

	Pros	Cons
Using Huge Pages §4.5.1	Performance is close to bare-metal nodes.	Huge page VMs are more complicated to configure, and less flexible. Benefits of huge pages could be offset with even larger memory size and memory usage.
Restarting VM §4.5.2	Performance is best after restarting.	The system must support job checkpointing. Longer down time
Defragmenting Memory §4.5.3	Performance is improved, short down time.	Cannot provide the best performance. Improvement is only temporary
Running on bare-metal §4.5.4	Performance is best and sustainable.	Difficult to manage and maintain. Security concerns.
Using public clouds §4.5.5	Based on one-week experiments, the performance is best and sustainable. Easy to manage	Security concerns.

Table 4.3: Pros and cons of five mitigation methods

4.5.1 Using Huge Pages

Just like a standard page table, EPT size increases as memory grows larger, hence a higher probability of misses when the page unit is only 4 KB. Increasing the page size (*i.e.*, using 1 MB "huge" pages) can effectively shrink the size of the EPT table. However, in GPAS, this is not an easy option to adopt. Conceptually, huge-page VMs are less flexible. Configuring and debugging huge- page VMs in a production environment takes time. For academic platforms, administrator time is more limited than in large industries. Hence, it is not easy to reconfigure and troubleshoot all the machines with a huge page configuration. In

addition, a huge page size that is "huge" enough for now cannot be a permanent solution given anticipated increases in memory and application memory usage in the future [17]. Another way is to keep the 4 KB page unit but allocate smaller VMs to reduce memory fragmentation. However, in GPAS, resource requirements differ across jobs; many of them require large VM memory.

4.5.2 Restarting VMs to Avoid Performance Degradation

Another method is to restart the VMs occasionally to "reset" the memory fragmentation. According to Figure 4.5, it is possible to detect EPT violations early before performance degrades significantly. With such a detection, we can decide when is a proper time to restart. We conducted a simple experiment with the same jobs as in Figure 4.5. Table 4.4 shows that restarting increases the number of jobs finished per day. Here, "proactive restart" entails restarting the VM after every job finishes (*i.e.*, do not reuse the VM across jobs) and "slowdown-triggered restart" entails restarting when the monitored *vCPU Efficiency* drops below 90%. The throughput numbers in Table 4.4 might also suggest that restarting in the middle of a long job might improve its execution time. However, this requires checkpointing the job progress, which feature is not supported in GPAS given the complexity of bioinformatics jobs.

	Jobs per day
Proactive restart	1.92
Slowdown-triggered restart	1.69
No restart	1.23

Table 4.4: Rebooting VM mitigation

4.5.3 Defragmenting Memory

The burst of EPT violations in aging VMs is essentially caused by the loss of data locality of the VM memory on the host physical memory. We can use the built-in memory defragmen-

tation tool in Linux to reorganize the memory layout. A simple experiment with VarScan2 workloads shows that defragmentation indeed helps decrease EPT violations. As shown in Table 4.5, the test runs 21% faster after defragmentation when the VM has been heavily used for 7 days. The number of EPT violations during the test is decreased by 58%. However, comparing to the performance of the fresh state, this method is still 44% slower and EPT violations are nearly 100 times greater. In other words, memory defragmentation can be a temporary method to improve performance by a small margin, but restarting VMs leads to a better outcome.

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

Table 4.5: Memory defragmentation

4.5.4 Running on Bare Metal

In GPAS, the most viable alternative is to run jobs directly on bare-metal nodes without VMs. The caveat is that not all research projects (jobs) can run in this mode; some research projects require strict security to be bundled in a secure VM. Thus, we make GPAS support hybrid bare-metal and VM deployments, especially for jobs that prefer performance over flexibility and security. Recently, some jobs have been running on bare-metal nodes directly. The statistics presented below are from job pipelines that have more than one hundred jobs on bare-metal nodes.

In Figure 4.8, a Gaussian kernel density estimation is calculated for jobs of four particular pipelines on bare-metal nodes and VMs. As shown, bare-metal jobs exhibit much less variance than VM jobs (bare-metal jobs exhibit narrower distribution across the *processing rate* axis). Besides, bare-metal jobs also show a more superior *processing rate* than that of VMs. In addition, the 95th percentile *processing rate* is marked in the graphs; it is clear

that the *processing rate* tail of bare-metal jobs is relatively short.

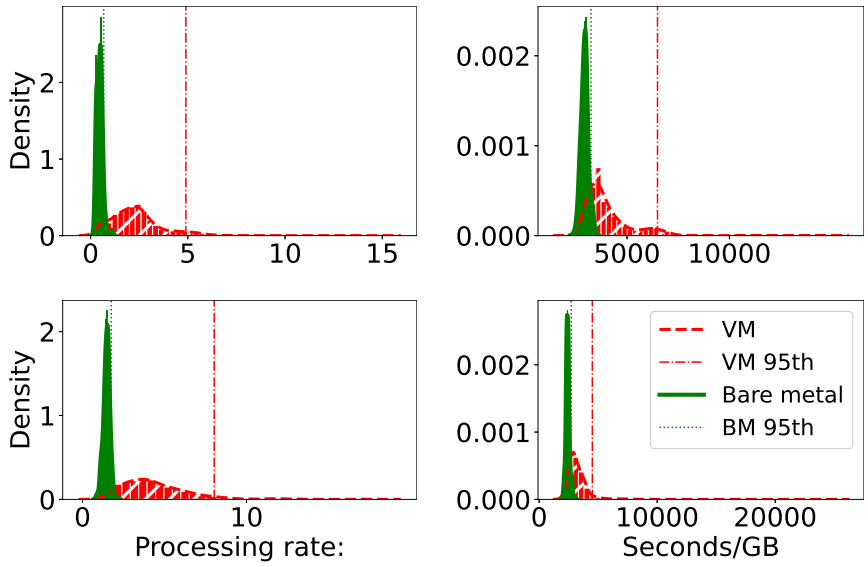

Figure 4.8: *Processing rate* Gaussian density estimation

Among all the fifteen pipelines, of which each has over one hundred bare-metal jobs, job performance has been improved a lot when compared to VMs. The 95th percentile *processing rate* is improved by 22 – 95%, the average by 18 – 87%, and the variance by 19 – 100% for all the pipelines. The *processing rate* improvement (in %) at different percentiles is shown in Table 4.6. In summary, given these benefits, running the GDC platform in a hybrid mode (bare-metal and VM supports) becomes a viable option, albeit more management complexity and overhead.

25^{th}	*median*	75^{th}
14 – 78	18 – 79	19 – 81
95^{th}	97^{th}	*max*
22 – 95	20 – 97	28 – 100

Table 4.6: Running on bare-metal

4.5.5 Using Public Clouds

The problem we unveil in here pertains to OpenStack/KVM deployment. Different virtualization stacks are likely to employ different approaches. Many vendors [2, 1] have stated, in their documentation, that configuring a huge page memory is beneficial to VMs with a large amount of memory, but there is no concrete recommendation on how large the memory is when a huge page must be configured for the VM, and it is possible for most people to ignore the necessity. On the other hand, hypervisors such as Xen use another approach to implement virtual machine memory management which directly maps guest virtual address to host physical address (called direct paging), thus they do not incur any overhead for resolving the mapping from guest to host as KVM does.

To find out if this problem appears in other virtualization stacks, we tried running jobs on commercial cloud providers. There are some caveats associated with using public clouds, for example to do with privacy and security. Indeed, because of protected data policy, we cannot run the same VarScan2 experiments on public cloud clouds. Thus, to evaluate public clouds, specifically Amazon Web Services and Google Cloud Platform, below we use open-access data from GDC and DNA alignment workloads. As a baseline, for a one-week experiment, the minimum execution time on our VM is 10 hours and the maximum is 15.3 hours. In an aging, 4-day old VM, the experiment begins to take more than 13 hours.

	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

Table 4.7: DNA Alignment Pipeline Experiments

Amazon Web Services developed their own hypervisor, based on Xen. We rented a dedicated host (z1d) and allocated a large memory VM (12xlarge). The one-week experiment does *not* show any performance degradation. The minimum execution time is 3.6 hours and the maximum is 3.9 hours. As mentioned before, we suspect that the Xen direct-paging

approach successfully avoids the address translation overhead, albeit providing less flexibility for memory sharing.

Google Cloud Platform built their hypervisor based on KVM [5]. We rented a 96-core, sole-tenant host to avoid sharing with other users, and allocated a 90-core, 576 GB VM on the host. We repeated the same experiment while scaling the maximum number of jobs to 14. The experiment results also do *not* show any degradation either. The minimum and maximum execution times are 5.7 and 6.0 hours, respectively. Although this platform uses KVM, the same problem might not appear due to one of the following reasons: it uses huge pages, it uses software-based memory management instead of EPT, the CPUs have larger TLBs.

In summary, public cloud platforms are highly prevalent nowadays, but for many bioinformatics tasks with stringent security and privacy measures, using public clouds, although they are fast, is not an option. For all of the reasons stated in this section, our GDC platform resorts to hybrid VM/bare-metal supports and occasional VM restarts.

4.6 Summary

To the best of our knowledge, we are the first to conduct a prolonged performance evaluation of a virtualization stack, and in particular the OpenStack/KVM stack for bioinformatics platforms. We have shown that bioinformatics workloads are uniquely long-running and memory intensive to the point of causing virtual memory to be highly fragmented after days of operation, hence causing a high increase in EPT violations and consequent interrupts to the host, a phenomenon that appears in what we call "aging VMs". We have presented a detailed diagnosis as well as two monitoring techniques (at both host level and VM level) and a range of mitigation techniques (including their pros and cons) that can be adopted by the GPAS.

CHAPTER 5

PIPELINE JOB SCHEDULING IN THE GPAS

5.1 Overview

Due to the complicated composition of bioinformatics pipeline, the GPAS uses a very simple job scheduling model that simply parallelizes jobs on a VM node, which is also refered as "embarrassingly parallel" in HPC context.

This simple scheduling model have shown drawbacks in the GPAS, causing job failures extremely costly, computing resource under-utilization and contention. These drawbacks make it a necessity to adopt a task-based scheduling model.

This chapter presents the evidences of the drawbacks in the simple scheduling model, and discusses the benefits of a task-based scheduling model, and the challenges to implement a good task-based scheduler.

5.2 Current Job Scheduling in the GPAS

As shown in Figure 2.10, the core parts in the GPAS are a job scheduling service and a VM pool managed by SLURM. SLURM is grid-computing cluster management software that features multiple job-scheduling algorithms. Since the CWLtool does not provide APIs to connect with computing environment, the external job-scheduling service plays the role of scheduling jobs, and SLURM is only responsible for distributing jobs to VMs according to the jobs' computing resource requirements. Besides, the GPAS relies on a S3-compatible storage service to acquire data through a flat-file genomics database service, and jobs' status information is kept track of in a local relational database.

Job scheduling design in the GPAS differs from NextFlow and Cromwell, mainly because the version of CWLtool in GPAS does not support parallelized task execution for a job. In the GPAS, information about jobs is put first into the graph-oriented database, then the job

scheduling service picks up job information and assembles job configuration in JSON format and submits it to SLURM on a first-in, first-out basis. Thus the GPAS has the ability to schedule a lot of jobs on a large cluster. However, NextFlow and Cromwell connect to a cloud computing cluster directly without any external components. Only tasks inside one job at a time can be executed on the specific cloud cluster.

The ultimate goal of the GPAS is to achieve highly coordinated job scheduling so that tasks from different jobs can be executed on one node to maximize resource utilization. However, in its current state, a certain number of jobs are just simply run in parallel on a node, which is more similar to the "embarrassingly parallel" setting as introduced in §2.4.2. Besides, the GPAS always allocates the largest possible computing resource that a job may need, even though the allocated resource may not always be fully utilized during the job's execution.

5.3 Drawbacks of the Existing Job Scheduling Model

5.3.1 Extra Cost Caused by Job Failures

Other than the "embarrassingly parallel" computing fashion, the GPAS does not use job progress caching mechanisms, nor checkpointing, for jobs. There are three main reasons:

1. The GPAS expects the error rate for jobs to be low, so retrying a job from scratch doesn't hurt too much.

2. There can be various types of errors or reasons that terminate a job; intermediate results for a job may not be useful.

3. Sometimes a job needs to be retried on another node, so the intermediate results are not easily accessible.

Because of this design choice, it may take some extra time for a failed job to be rescheduled

and retried until the job eventually completes successfully. What's worse, it is possible for the job to fail multiple times and to be retried multiple times.

	VM			Bare-metal			Overall		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Execution minutes	283	24617	2098	1227	8573	3673	283	24617	2134
Attempts	1	4	1.28	1	2	1.02	1	4	1.28
Error rate	926/4171 (22.2%)			2/95 (2%)			928/4266(21.75%)		

Table 5.1: Basic Statistics for Jobs of Somatic Variant Calling Pipeline

Table 5.1 shows summary statistics for Somatic Variant Calling pipeline jobs. Some jobs were run on bare-metal nodes to provide a baseline for the jobs running on VMs. All the jobs reflected in the table have succeeded in the end, though there might be some failed attempts. In the leftmost column, "Execution minutes" are measured during the last successful execution for a job, "Attempts" are measured by counting all attempts including the final successful attempts, and "Error rate" is calculated as the failed attempts divided by all the attempts. It's worth noting that this way of calculating error rate differs from the calculation of success rate in §3.4.3, Figure 3.12. All the jobs in this table are regarded as successful jobs in Figure 3.12 because the success rate in §3.4.3 only cares about the final status of the jobs, instead of the outcome of each job attempt.

The statistics show that 22.2% Somatic Variant Calling pipeline jobs in the GPAS have been attempted more than once, and extensive time has been spent retrying jobs.

We define "Job delayed time" as the period between the time when a job was first attempted and the time when the job was successfully finished, whether in the same first attempt or in another later attempt. "Job delayed time" can vary a lot, because some jobs may fail due to random errors and then succeed simply by making another attempt. However, other jobs may need manual investigation to fix errors; such interventions bring in nondeterministic overheads.

Figure 5.1 shows the distribution of "Job delayed time" in hours for jobs on VMs and

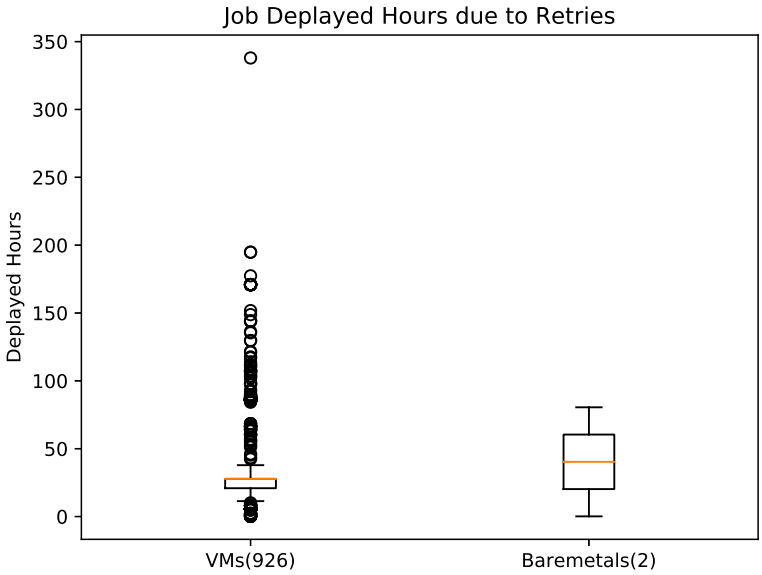

Figure 5.1: Job Delayed Hours due to Retries

bare-metal nodes, and the majority of the jobs have been delayed by around 1 day until they can finally start to run successfully. What's worse, there are a considerable number of bad cases that are delayed by 100 hours on VMs, which significantly affects the efficiency of the GPAS. Again, the difference between VMs and bare-metal nodes mainly come from the different error types typical of VMs and bare-metal nodes.

The long delay time exposes a critical issue in the GPAS because of the job- level scheduling and neglect of job progress caching. More importantly, since a pipeline job generally takes a long time to complete, more efficient pipeline debugging methods are necessary for jobs to quickly recover from errors.

5.3.2 Resource Under-utilization and Contention

Apart from reducing the overhead of error recovery, it is crucial to improve the utilization of computing resources so the system can achieve high job throughput.

Based on the scheduling design of the GPAS, multiple jobs may be scheduled to run on

one VM host, and tasks within a job are executed in serial order. It is possible that tasks would ask for different amounts of resources. The current practice in the GPAS would always grant the maximum resource for a job that all tasks in this job may request. In consequence, computing resource may not always be fully utilized for the job.

Among the computing resources that can be allocated to a job, memory and storage can only be allocated in large volumes because the real-time memory and storage usage can be constantly changing. CPU resource, on the other hand, is more rigid because, for a multi-threading/multi-processing task, it is more likely that all the allocated CPUs will be used during the whole execution of the task. However, the challenge of allocating CPUs efficiently in the GPAS is that not all the tasks in a job are multi-threading/multi-processing. This leads to varied average CPU utilization for different pipelines.

Another issue which probably slows down jobs is the I/O contention. Apparently, adopting SSDs can avoid I/O contention to some degree, however, a small number of VMs in the GPAS use spinning disks and disks are still widely used to support cheaper computation.

Figure 5.2 shows the real-time CPU and I/O utilization for an experiment that runs five Somatic Variant Calling jobs on a VM that has 48 cores, 226 GB RAM, and 1 TB hard disk storage. The experiment starts these five jobs at the same time in the VM, and 8 CPUs are allocated for each job. The whole experiment runs for around 2 days, and Figure 5.2 captures I/O and CPU utilization for the second day.

The top graph shows the CPU under-utilization for the five jobs; system-wide CPU utilization never reaches the maximum utilization, 83.3%. For most of the time, the jobs wait for I/O to complete. Between **point A** and **point B**, there is a window with low I/O activity, but the CPU activity is also relatively low. This is because the jobs reach a task for which not all 8 CPUs are used.

The bottom graph shows the I/O bandwidth. For most of the time, I/O utilization is always full, and it is mixed with reads and writes. However, the bandwidth of reading

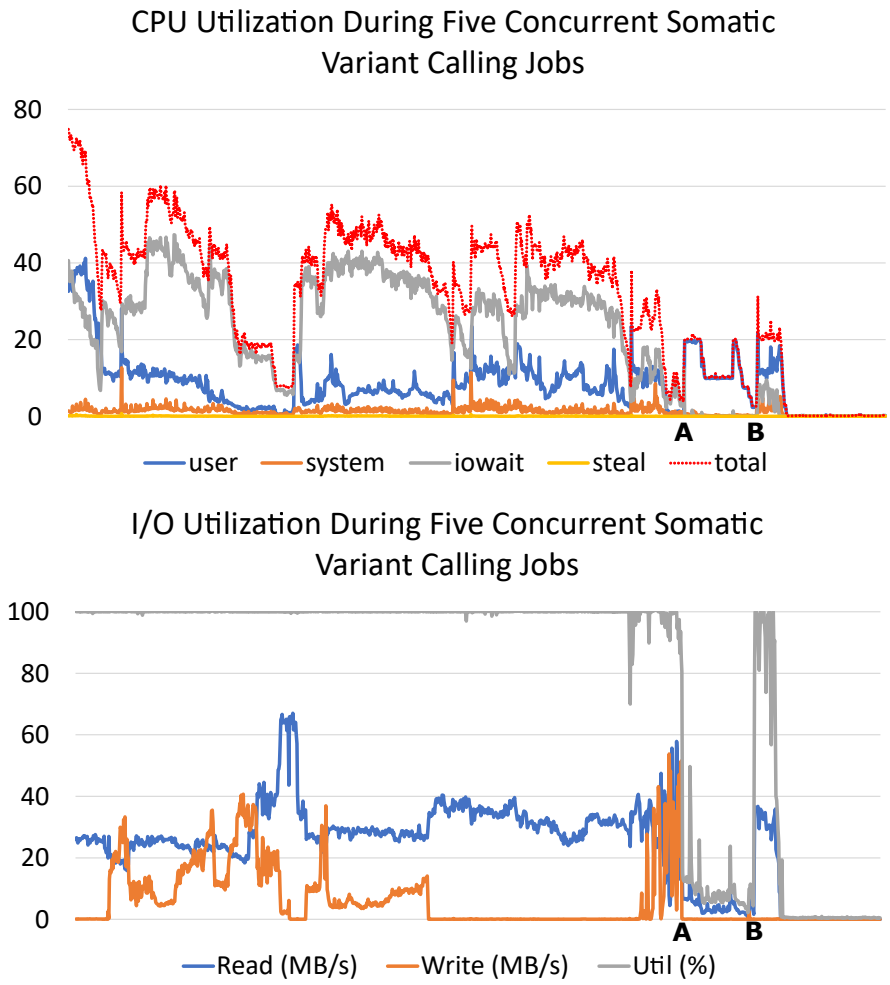

Figure 5.2: I/O and CPU Utilization of 5 Somatic Variant Calling Jobs Running in VM

and writing is much lower than maximum bandwidth of the hard drive; that is because the spinning disk cannot handle the read and write contentions from all five jobs.

In addition, graphs like those in Figure 5.2 can only be explained together with job logs which contains the timestamps for all the tasks. However, there are too many tasks in a job and too many different jobs in the cluster, making it difficult to identify the exact tasks that result in resource under-utilization and contention on a worker node.

5.4 Task-based Scheduling Model

5.4.1 Proposed Task-based Scheduling Model

As introduced in §2.2.3, some pipeline execution engines support parallelized execution for tasks, such as CWLtool, Cromwell and NextFlow. However, the implementation in those execution engines only considers the execution for a single job. In §2.2.3 Figure 2.6 has already illustrated the situation where parallelized tasks may require more CPU resources than the worker has.

To achieve a better solution, we need to implement a global control to at least coordinate all tasks in scheduled jobs on a VM host, so that computing resource for individual job can be guaranteed. The key difference between the proposed scheduling and current scheduling is that a whole pipeline job is broken down into tasks. And scheduling is conducted in a finer granularity on the task level.

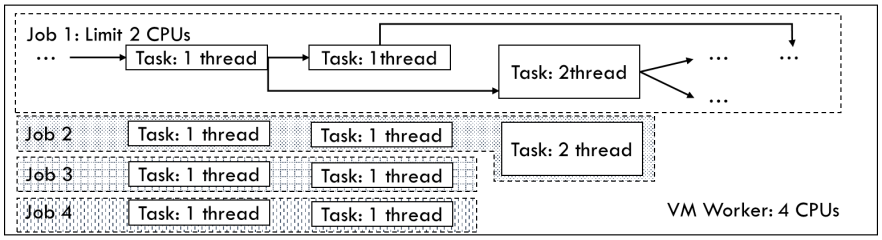

Figure 5.3: Task-based Scheduling Model

Figure 5.3 illustrates the proposed scheduling model that is able to schedule jobs based on tasks, and that guarantees the CPU limits for individual tasks, while respecting the CPU limit of the VM worker. As the figure shows, there won't be more than 2 threads used for Job 1, even if the second and the third task could be executed in parallel without CPU limitations. Secondly, during the execution of the second task of Job 1, tasks from other jobs can be executed to saturate the available CPU resource of the VM.

5.4.2 Theoretical Assessment Through Simulations

	Old 5 Jobs	Task-based 5 Jobs	15 Jobs	20 Jobs
Simulated CPU Utilization	82%	83%	94%	96%
Time (Minutes) Reduction (%)	3904	3852 (-1.3%)	10174 (-13.1%)	13260 (-15.1%)

Table 5.2: Simulated Execution for Somatic Variant Calling Workflow Jobs

	Old 5 Jobs	Task-based 5 Jobs	15 Jobs	20 Jobs
Simulated CPU Utilization	89%	90%	93%	95%
Time (Minutes) Reduction (%)	4862	4839 (-0.4%)	13917 (-4.6%)	18252 (-6.1%)

Table 5.3: Simulated Execution for Bamfasq-align Workflow Jobs

Table 5.2 and Table 5.3 show the simulated execution time for two pipelines: Somatic Variant Calling and Bam FastQ Alignment. Estimated execution time for tasks are extracted from one job of that pipeline, and the experiments simulate running 5 jobs in parallel, and 15 and also 20 jobs in a batch. Each job can use at most 8 threads, and at most 40 threads can be used at the same time by all the jobs. The percentage of time reductions use the execution time of original 5 jobs multiplied by 3 or 4, for 15 or 20 batch jobs, respectively. We can see that there are considerable improvements on CPU utilization and execution time when we batch more jobs.

We also notice that improvements shown in Table 5.3 are smaller than in Table 5.2. This is because the original CPU utilization is high in the Bam FastQ Alignment pipeline. Thus, the improvement brought by the new scheduling depends on the pipeline design. It is reasonable that, if the pipeline doesn't have any single thread bottlenecks as mentioned before, there won't be any benefits to adopting task-based scheduling if CPU utilization is the only concern.

Now that the minimum scheduling units are tasks, it is possible to checkpoint task outputs

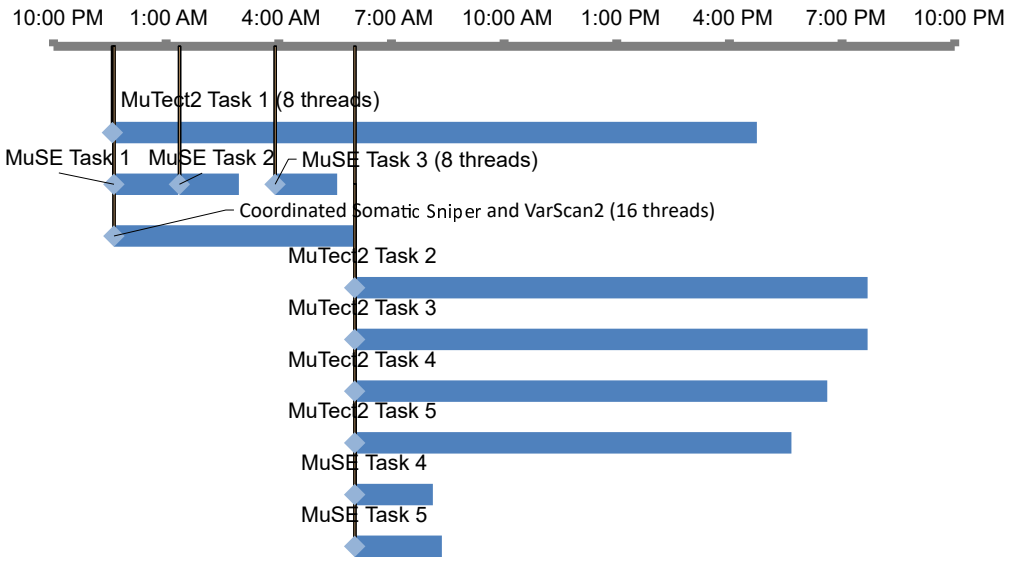

Figure 5.4: Prefecthing Data Before Computation Optimization for Tasks

that are original intermediate results in the current scheduling model. Thus, recoverability is improved.

5.4.3 Task-Specific Optimizations

In addition, tasks, which are usually executing a single tool, are easier to classified by their workload characteristics. Based on tasks' requirements for computing resources, a more sophisticated scheduling algorithm is able to arrange tasks with different demands together to maximize resource utilization. Another prototype of this scheduling model implements scheduling optimization for the Somatic Variant Calling pipeline, where the execution of MuSE, MuTect2, SomaticSniper and VarScan2 is specially optimized.

To be precise, through benchmarking, we learned that SomaticSniper and VarScan2 are bounded equally by I/O and CPU when they are executed on disk-based storage, and MuTect2 and MuSE are simply CPU-bound. As a result, we implemented a scheduling component that coordinates the execution of SomaticSniper and VarScan2 tasks by prefetching the needed data for each thread and then starting computation. Since the other two tasks are CPU-bound, we just start them when there are CPU resources available.

Figure 5.4 shows the execution timeline for each task when we test five jobs, each consuming eight threads on a 48-core VM with spinning disk as main storage. Comparing to the original scheduling, the execution time for the jobs is improved by 18.6%, and the improvement for the four major computation tasks are as much as around 50%.

Experiment Scenario	**Execution Time**
w/o prefetching, *Fresh VM*	71 min
w/ prefetching, *Fresh VM*	66 min
w/o prefetching, *Aged VM*	123 min
w/ prefetching, *Aged VM*	87 min

Table 5.4: Prefetching Optimization for Tasks When the VM is Fresh or Aged

Moreover, this optimization can also improve job performance when the VM degrades because of the issue discussed in Chapter 4. Table 5.4 shows the execution time for an experiment that runs the same VarScan2 tasks as previously. When the VM is fresh, it takes 71 minutes to finish the tasks when data is not prefetched, and 66 minutes (including the time spent prefetching the data) with the prefetching optimization. Although the five-minutes improvement is not a big deal, when the VM is in a slow state, it proves that prefetching optimization is extremely helpful. Running the tasks without prefetching when the VM is slow takes 123 minutes (73% slowdown), but, with prefetching, it takes only 87 minutes (22 %) slowdown. This is because the design of prefetching limits the number of concurrent I/Os. Thus, even though the VM degrades I/O performance, I/O bandwidth can remain high for single reading process.

5.4.4 Challenges of Building Task-based Scheduling for Pipelines

To implement an efficient task-based scheduler, there are a number of challenges that we need to tackle.

1. **Type of tasks**: The rule of thumb is that the scheduler must be accurately aware task type. As discussed throughout this dissertation, VarScan2 and Somatic Sniper tasks

in the Somatic Variant Calling pipeline are I/O and computation-mixed workloads, and when they are running on spinning disks, it is best to separate reading from computation. Mutect2 and MuSE tasks are computing-intensive workloads, and without data prefetching on spinning disk, the computation is not obviously interfered with by reading data.

2. **Storage type**: Since the effect of prefetching differs depending on the storage type, the scheduler must also take the storage type of worker nodes into account, though Table 5.4 shows that prefetching on a VM with SSD storage may also help improve performance.

3. **VM degradation issue**: As extensively discussed in Chapter 4, the VM degradation issue seriously affects I/O performance on VMs with SSDs. We also find that task-based scheduling can potentially accelerate the VM degradation when we make too many I/O intensive workloads. In an experiment where the scheduler tries to schedule a batch of ten jobs, 20 parallel I/O intensive tasks are started and *vCPU Efficiency* drops to 92% during the first hour since the VM was freshly started, which decline is not observed when the jobs are running in "embarrassingly parallel" mode.

4. **Balance between number of parallel tasks and other computing resources**: When more tasks are parallelized, more memory and storage space are certainly required to accommodate all the tasks. Especially for bioinformatics workloads which are common to consume large volume of data, it is easy to cause task failures by providing insufficient memory and storage.

5. **Efficient task designs**: Last but not least, the design of the tasks is important, especially for accurate resource management. Table 5.5 shows the average I/O read bandwidth and CPU utilization for running 1, 8, 16, 32, 40 Somatic Sniper tasks in parallel with only one thread allowed for each task on a VM with 48 CPUs and SSD.

	Average CPU Utilization	Average Read Bandwidth (Maximum: 2GB/s for the SSD)
1 Task	5 %	1.0 MB/s
8 Tasks	41 %	6.6 MB/s
16 Tasks	76 %	10.2 MB/s
32 Tasks	87 %	11.0 MB/s
40 Tasks	89 %	11.1 MB/s

Table 5.5: Somatic Sniper Multiple Task Experiment

We note that, for one task, the average CPU utilization is already more than twice the theoretical value (2.1%), and it is similar for 8 parallel tasks and 16 parallel tasks. For 32 and 40 tasks, it seems that the CPU utilization is saturated. This is because, in the design of Somatic Sniper task, there is actually an extra thread running in parallel that is not controlled by task configuration. In fact, this task uses SAMtools in parallel to convert the format of Somatic Sniper task's input in advance. Besides, the conversion is also computing-intensive, and SAMtools becomes a bottleneck causing the reading bandwidth to be far lower than the maximum bandwidth of the SSD.

5.5 Summary

This chapter discusses the drawbacks of the current simple job-scheduling model including the high cost of retrying the whole job when a job fails, and computing resource under-utilization and contention due to lack of finer-granularity scheduling.

A task-based scheduling model is proposed and evaluated through simulation to be useful in improving computing resource utilization and job performance. Moreover, the task-based scheduling model opens up opportunities for task-specific optimizations. A simple optimization that prefetches data into memory before computation is proven to benefit jobs that run on slow storage, or on a VM with degraded performance.

Finally, this chapter discusses five challenges that need tackling to implement a good task-based scheduler, including identifying the computing type of all tasks, choosing differ-

ent strategies for different storage media, being aware of VM degradation issues, the balance between parallel tasks and the increased consumption of computing resources due to parallelization, and efficient task designs.

CHAPTER 6

TOWARDS A VISION FOR THE GPAS WITH IMPROVED PERFORMANCE

6.1 Bioinformatics Research in the Cloud Computing Era

The majority of recent research in bioinformatics cloud computing falls into two categories: highly parallelized applications and building a pipeline computing platform specifically for bioinformatics. While a few works proposed new pipeline execution engines [85, 35, 75], CWLtool, Cromwell and Nextflow remain the best choices for the GPAS because they allow almost any application to be assembled into a pipeline.

6.1.1 Bioinformatics Applications

As bioinformatics is an interdisciplinary subject, a large number of researchers in bioinformatics are not experts in software development. Most bioinformatics applications were designed to run locally until recently, when cloud computing started to play an important role.

It is never an easy job to make bioinformatics applications to exploit distributed computing resources. More research is emerging to make new applications with new computing frameworks, or to transform classic tools for use with new frameworks. A typical example is the Basic Local Alignment Search Tool (BLAST)[1], which has been actively transformed into mutiple versions: mpiBLAST [27], ScalaBLAST [91], GridBLAST [64], CloudBLAST [80]. mpiBLAST is actually a HPC version of BLAST where each computing node searches a portion of the database and MPIs are used to communicate between tasks. GridBLAST adopts the "embarrassingly parallel" computing model in grid computing. ScalaBLAST makes use

1. https://blast.ncbi.nlm.nih.gov/Blast.cgi

of a distributed database as the "shared memory" and makes it possibile to run BLAST on thousands of processors. CloudBLAST combines MapReduce and virtualization, and runs on two geographically separate clusters. Similarly, a large number of alignment or sequence mapping applications have been created: CloudAligner [89], CloudBurst [105], Crossbow [66], SparkBWA [6]. What is common for this kind of application is that the operations inside the application are actually independent from each other. Thus, transforming these applications to perform the operations on a distributed cluster is natural.

It has also been pointed out that the applications implemented in MapReduce are actually within the Many-Task Computing paradigm, which is introduced in §2.4.1.

6.1.2 Bioinformatics Pipeline Platforms

A large number of bioinformatics pipeline platform researchers focus on the ease that platforms bring to bioinformaticians for creating and executing pipelines, and how they integrate into cloud-computing environments. The platforms include SciApps [115], BioWorks [42], BioWMS [16], Taverna [92], Workflow-based PSE [110], Discovery Net [103] and Galaxy [63, 76]. It is common that how pipeline jobs are scheduled is not included in the literature, but some of these platforms use existing middleware to control computing in the cloud, *e.g.* BioWMS uses Hermes [25]; Workflow-based PSE and Galaxy use HTCondor.

Though not often talked about in the context of bioinformatics pipeline platforms, pipeline or workflow scheduling is critical to good pipeline performance. Parallelism is explored for bioinformatics workflows [84, 33], and it is believed that the parallelism for general scientific workflows (see §2.4.4) is beneficial to bioinformatics pipelines as well. Pipelines eventually need to be broken into tasks where a task is usually a sole bioinformatics application. What is more commonly researched is the mechanisms and policies to provision resource for bioinformatics applications and scheduling applications in the cloud environment [106, 95, 30, 50, 28].

6.2 A Vision for GPAS: Hybrid of HTC and MTC

In scientific computing, it is important to accurately classify the computing paradigm for an application. However, through the literature review of bioinformatics research, we find that it is hard to classify bioinformatics pipeline platform simply as one of HTC or MTC. This is because of the large variety of applications and the diversity of bioinformatics algorithms. But we must admit that the finer control over smaller tasks in MTC, though requiring more sophisticated management, can achieve better system performance.

Thus, there are several research directions that could lead to performance improvements for the GPAS:

1. After adopting the task-based scheduling model, the GPAS should choose whether deadline-constraint scheduling is a better fit, or scheduling for lower cost is better.

2. Currently, since the tasks are mixed up in pipeline jobs, there hasn't been any proper prediction model for the execution time of tasks or jobs. With pipeline broken into tasks, work should be done to create a proper prediction for task execution time.

3. How to deploy MTC tasks simultaneously with other, non-MTC tasks. The GATK applications, some of which are implemented in MapReduce or Spark, are commonly used in pipelines in the GPAS. It may be possible to make some applications run with MapReduce on a subset of nodes in the cluster, while other applications run in "embarrassing parallel" on other nodes.

4. Find more methods to convert non-MTC applications into MTC. A good example in the GPAS currently is that, because VarScan2 and Somatic Sniper do not support multi-threading, input files have been split so that multiple processes can be spawned to compute in parallel.

5. Develop good middleware which is able to schedule some applications in MTC and

other applications in HTC.

CHAPTER 7

CONCLUSIONS

This dissertation presents an in-depth review on the bioinformatics pipeline platform, GPAS. Starting with the creation of an all-in-one database, this dissertation opens up opportunities to look into GPAS job statistics.

A significant VM performance problem is revealed: a significant number of jobs in the GPAS exhibited much slower performance than other jobs of similar types, with as much as 1,000% degradation. To the best of our knowledge, we are the first to conduct a prolonged experiment with bioinformatics workloads on the VM stack.

Moreover, this dissertation also reviews the job scheduling model that the GPAS provides, and finds that the current scheduling is inefficient because job failures bring high extra costs, and because parallel-running jobs cause computing resource under-utilization and contention.

Lastly, combined with a thorough literature review on others' experience of building a bioinformatics application/pipeline platform, this dissertation presents future research possibilities that may further improve the performance of pipelines in the GPAS.

In summary, the contributions of this dissertation are:

1. A statistics synthesizing service is created for the GPAS and an all-in-one statistics database is created. The production statistics collected from the GPAS in this database facilitates most of the topics in the dissertation, and it is invaluable for future research.

2. This dissertation shows how real bioinformatics workloads can cause "aging VMs" after several days and performance degradation by up to 1,000%. Several possible mitigation scenarios are evaluated.

3. The job-scheduling model in the GPAS is carefully examined and discussed, and a task-based scheduling model has been proposed with evaluations and discussed.

4. This dissertation presents a literature review on bioinformatics application/pipeline platforms and develops a vision for the GPAS with further improved pipeline performance.

REFERENCES

[1] How is the hugepages feature enabled for virtual machines? https://docs.oracle.com/cd/E64076_01/E64081/html/vmcon-vm-hugepages.html, 2018.

[2] Huge pages in vmware vcloud nfv openstack edition. https://docs.vmware.com/en/VMware-vCloud-NFV-OpenStack-Edition/3.0/vmwa-vcloud-nfv30-performance-tunning/GUID-1F05987F-012B-4BC4-9015-CDE3C991C68C.html, 2018.

[3] Picard toolkit. http://broadinstitute.github.io/picard/, 2019.

[4] Workflow description language specification. https://github.com/openwdl/wdl, 2019.

[5] Google compute engine faq. https://cloud.google.com/compute/docs/faq, 2020.

[6] José M Abuín, Juan C Pichel, Tomás F Pena, and Jorge Amigo. Sparkbwa: speeding up the alignment of high-throughput dna sequencing data. *PloS one*, 11(5), 2016.

[7] Enis Afgan, Dannon Baker, Bérénice Batut, Marius Van Den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn A Grüning, et al. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic acids research*, 46(W1):W537–W544, 2018.

[8] Mohammad Agbarya, Idan Yaniv, and Dan Tsafrir. Memomania: From huge to huge-huge pages. In *Proceedings of the 11th ACM International Systems and Storage Conference*, pages 112–112, 2018.

[9] Chloe Alverti, Vasileios Karakostas, Georgios Goumas, and Nectarios Koziris. Pact: G: Speculative offset address translation.

[10] Peter Amstutz, Michael R Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, et al. Common workflow language, v1.0. 2016.

[11] Simon Andrews et al. Fastqc: a quality control tool for high throughput sequence data, 2010.

[12] Nazia Anwar and Huifang Deng. Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. *Future Internet*, 10(1):5, 2018.

[13] Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

[14] Thomas W Barr, Alan L Cox, and Scott Rixner. Translation caching: skip, don't walk (the page table). *ACM SIGARCH Computer Architecture News*, 38(3):48–59, 2010.

[15] Thomas W Barr, Alan L Cox, and Scott Rixner. Spectlb: a mechanism for speculative address translation. *ACM SIGARCH Computer Architecture News*, 39(3):307–318, 2011.

[16] Ezio Bartocci, Flavio Corradini, Emanuela Merelli, and Lorenzo Scortichini. Biowms: a web-based workflow management system for bioinformatics. *BMC bioinformatics*, 8(1):1–14, 2007.

[17] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D Hill, and Michael M Swift. Efficient virtual memory for big memory servers. *ACM SIGARCH Computer Architecture News*, 41(3):237–248, 2013.

[18] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[19] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. Accelerating two-dimensional page walks for virtualized systems. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 26–35, 2008.

[20] Nikhil Bhatia. Performance evaluation of intel ept hardware assist. *VMware, Inc*, 2009.

[21] Jeffrey Buell, Daniel Hecht, Jin Heo, Kalyan Saladi, and R Taheri. Methodology for performance analysis of vmware vsphere under tier-1 applications. *VMware Technical Journal*, 2(1):19–28, 2013.

[22] Eun-Kyu Byun, Yang-Suk Kee, Jin-Soo Kim, and Seungryoul Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011–1026, 2011.

[23] Kristian Cibulskis, Michael S Lawrence, Scott L Carter, Andrey Sivachenko, David Jaffe, Carrie Sougnez, Stacey Gabriel, Matthew Meyerson, Eric S Lander, and Gad Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature biotechnology*, 31(3):213, 2013.

[24] Pablo Cingolani, Adrian Platts, Le Lily Wang, Melissa Coon, Tung Nguyen, Luan Wang, Susan J Land, Xiangyi Lu, and Douglas M Ruden. A program for annotating and predicting the effects of single nucleotide polymorphisms, snpeff: Snps in the genome of drosophila melanogaster strain w1118; iso-2; iso-3. *Fly*, 6(2):80–92, 2012.

[25] Flavio Corradini and Emanuela Merelli. Hermes: agent-based middleware for mobile computing. In *School on Formal Methods-Moby*, pages 234–270. Springer, 2005.

[26] Vasa Curcin and Moustafa Ghanem. Scientific workflow systems-can one size fit all? In *2008 Cairo International Biomedical Engineering Conference*, pages 1–9. IEEE, 2008.

[27] Aaron E Darling, Lucas Carey, and W Chun Feng. The design, implementation, and evaluation of mpiblast. Technical report, Los Alamos National Laboratory, 2003.

[28] Gabriel SS de Oliveira, Edward Ribeiro, Diogo A Ferreira, Aletéia PF Araújo, Maristela T Holanda, and Maria Emilia MT Walter. Acosched: A scheduling algorithm in a federated cloud infrastructure for bioinformatics applications. In *2013 IEEE International Conference on Bioinformatics and Biomedicine*, pages 8–14. IEEE, 2013.

[29] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI '04*.

[30] Frédéric Desprez and Antoine Vernois. Simultaneous scheduling of replication and computation for data-intensive applications on the grid. *Journal of Grid Computing*, 4(1):19–31, 2006.

[31] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature biotechnology*, 35(4):316, 2017.

[32] Inês Dutra, David Page, Vitor Santos Costa, Jude Shavlik, and Michael Waddell. Toward automatic management of embarrassingly parallel applications. In *European Conference on Parallel Processing*, pages 509–516. Springer, 2003.

[33] Vincent C Emeakaroha, Paweł P Łabaj, Michael Maurer, Ivona Brandic, and David P Kreil. Optimizing bioinformatics workflows for data analysis using cloud management techniques. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, pages 37–46, 2011.

[34] Yu Fan, Liu Xi, Daniel ST Hughes, Jianjun Zhang, Jianhua Zhang, P Andrew Futreal, David A Wheeler, and Wenyi Wang. Muse: accounting for tumor heterogeneity using a sample-specific error model improves sensitivity and specificity in mutation calling from sequencing data. *Genome biology*, 17(1):178, 2016.

[35] Mark WEJ Fiers, Ate van der Burgt, Erwin Datema, Joost CW de Groot, and Roeland CHJ van Ham. High-throughput bioinformatics with the cyrille2 pipeline system. *BMC bioinformatics*, 9(1):96, 2008.

[36] Jayneel Gandhi, Arkaprava Basu, Mark D Hill, and Michael M Swift. Efficient memory virtualization: Reducing dimensionality of nested page walks. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 178–189. IEEE, 2014.

[37] Jayneel Gandhi, Christopher J Rossbach, and Timothy Merrifield. Decoupling memory metadata granularity from page size, September 12 2019. US Patent App. 15/916,173.

[38] Robert L Grossman, Allison Heath, Mark Murphy, Maria Patterson, and Walt Wells. A case for data commons: toward data science as a service. *Computing in science & engineering*, 18(5):10, 2016.

[39] Robert L Grossman, Allison P Heath, Vincent Ferretti, Harold E Varmus, Douglas R Lowy, Warren A Kibbe, and Louis M Staudt. Toward a shared vision for cancer genomic data. *New England Journal of Medicine*, 375(12):1109–1112, 2016.

[40] Fan Guo, Yongkun Li, Yinlong Xu, Song Jiang, and John CS Lui. Smartmd: A high performance deduplication engine with mixed pages. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 733–744, 2017.

[41] Fei Guo, Seongbeom Kim, Yury Baskakov, and Ishan Banerjee. Proactively breaking large pages to improve memory overcommitment performance in vmware esxi. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 39–51, 2015.

[42] Youngmahn Han. Bioworks: a workflow system for automation of bioinformatics analysis processes. In *2011 International Conference on Ubiquitous Computing and Multimedia Applications*, pages 76–81. IEEE, 2011.

[43] Allison P Heath, Matthew Greenway, Raymond Powell, Jonathan Spring, Rafael Suarez, David Hanley, Chai Bandlamudi, Megan E McNerney, Kevin P White, and Robert L Grossman. Bionimbus: a cloud for managing, analyzing and sharing large genomics datasets. *Journal of the American Medical Informatics Association*, 21(6):969–975, 2014.

[44] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[45] Muhammad H Hilman, Maria A Rodriguez, and Rajkumar Buyya. Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions. *ACM Computing Surveys (CSUR)*, 53(1):1–39, 2020.

[46] Muhammad Hafizhuddin Hilman, Maria Alejandra Rodriguez, and Rajkumar Buyya. Task-based budget distribution strategies for scientific workflows with coarse-grained billing periods in iaas clouds. In *2017 IEEE 13th International Conference on e-Science (e-Science)*, pages 128–137. IEEE, 2017.

[47] Muhammad Hafizhuddin Hilman, Maria Alejandra Rodriguez, and Rajkumar Buyya. Task runtime prediction in scientific workflows using an online incremental learning approach. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 93–102. IEEE, 2018.

[48] Jingyuan Hu, Xiaokuang Bai, Sai Sha, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. Hub: Hugepage ballooning in kernel-based virtual machines. In *Proceedings of the International Symposium on Memory Systems*, pages 31–37, 2018.

[49] Jerry Huck and Jim Hays. Architectural support for translation table management in large address space machines. In *Proceedings of the 20th annual international symposium on computer architecture*, pages 39–50, 1993.

[50] Eduardo Huedo, Rubén S Montero, and Ignacio M Llorente. Adaptive grid scheduling of a high-throughput bioinformatics application. In *International Conference on Parallel Processing and Applied Mathematics*, pages 840–847. Springer, 2003.

[51] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *2010 IEEE second international conference on cloud computing technology and science*, pages 159–168. IEEE, 2010.

[52] Mark A Jensen, Vincent Ferretti, Robert L Grossman, and Louis M Staudt. The nci genomic data commons as an engine for precision medicine. *Blood*, 130(4):453–459, 2017.

[53] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[54] Gideon Juve, Benjamin Tovar, Rafael Ferreira Da Silva, Dariusz Król, Douglas Thain, Ewa Deelman, William Allcock, and Miron Livny. Practical resource monitoring for robust high throughput computing. In *2015 IEEE International Conference on Cluster Computing*, pages 650–657. IEEE, 2015.

[55] Dharmesh Kakadia. *Apache Mesos Essentials*. Packt Publishing Ltd, 2015.

[56] Gokul B Kandiraju and Anand Sivasubramaniam. Going the distance for tlb prefetching: an application-driven study. In *Proceedings 29th Annual International Symposium on Computer Architecture*, pages 195–206. IEEE, 2002.

[57] Edward Kent, Stefan Hoops, and Pedro Mendes. Condor-copasi: high-throughput computing for biochemical networks. *BMC systems biology*, 6(1):91, 2012.

[58] Jik-Soo Kim, Seungwoo Rho, Seoyoung Kim, Sangwan Kim, Seokkyoo Kim, and Soonwook Hwang. Htcaas: leveraging distributed supercomputing infrastructures for large-scale scientific computing. In *Proceedings of the 6th ACM Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS'13) held with SC13*. Citeseer, 2013.

[59] Daniel C Koboldt, Qunyuan Zhang, David E Larson, Dong Shen, Michael D McLellan, Ling Lin, Christopher A Miller, Elaine R Mardis, Li Ding, and Richard K Wilson. Varscan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome research*, 22(3):568–576, 2012.

[60] Alexey Kopytov. Sysbench: a system performance benchmark. *http://sysbench. sourceforge. net/*, 2004.

[61] Charlotte Kotas, Thomas Naughton, and Neena Imam. A comparison of amazon web services and microsoft azure cloud platforms for high performance computing. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4. IEEE, 2018.

[62] L Koumakis, C Mizzi, and G Potamias. Bioinformatics tools for data analysis. In *Molecular Diagnostics*, pages 339–351. Elsevier, 2017.

[63] Michael T Krieger, Oscar Torreno, Oswaldo Trelles, and Dieter Kranzlmüller. Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows. *Future Generation Computer Systems*, 67:329–340, 2017.

[64] Arun Krishnan. Gridblast: a globus-based high-throughput implementation of blast in a grid computing framework. *Concurrency and Computation: Practice and Experience*, 17(13):1607–1623, 2005.

[65] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J Rossbach, and Emmett Witchel. Coordinated and efficient huge page management with ingens. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 705–721, 2016.

[66] Ben Langmead, Michael C Schatz, Jimmy Lin, Mihai Pop, and Steven L Salzberg. Searching for snps with cloud computing. *Genome biology*, 10(11):1–10, 2009.

[67] David E Larson, Christopher C Harris, Ken Chen, Daniel C Koboldt, Travis E Abbott, David J Dooling, Timothy J Ley, Elaine R Mardis, Richard K Wilson, and Li Ding. Somaticsniper: identification of somatic point mutations in whole genome sequencing data. *Bioinformatics*, 28(3):311–317, 2011.

[68] Sungmin Lee, Hyeyoung Min, and Sungroh Yoon. Will solid-state drives accelerate your bioinformatics? in-depth profiling, performance analysis and beyond. *Briefings in bioinformatics*, 17(4):713–727, 2015.

[69] Philipp Leitner and Jürgen Cito. Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. *ACM Transactions on Internet Technology (TOIT)*, 16(3):1–23, 2016.

[70] Guoxi Li, Wenzhi Chen, Kui Su, Zhongyong Lu, and Zonghui Wang. Hzmem: New huge page allocator with main memory compression. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 51–64. Springer, 2017.

[71] Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[72] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[73] Weiling Li, Yunni Xia, Mengchu Zhou, Xiaoning Sun, and Qingsheng Zhu. Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. *IEEE Access*, 6:61488–61502, 2018.

[74] Xinyu Li, Lei Liu, Shengjie Yang, Lu Peng, and Jiefan Qiu. Thinking about a new mechanism for huge page management. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 40–46, 2019.

[75] Burkhard Linke, Robert Giegerich, and Alexander Goesmann. Conveyor: a workflow engine for bioinformatic analyses. *Bioinformatics*, 27(7):903–911, 2011.

[76] Bo Liu, Ravi K Madduri, Borja Sotomayor, Kyle Chard, Lukasz Lacinski, Utpal J Dave, Jianqiang Li, Chunchen Liu, and Ian T Foster. Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses. *Journal of biomedical informatics*, 49:119–133, 2014.

[77] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP journal*, 11(1):36–40, 1997.

[78] Nicholas M Luscombe, Dov Greenbaum, and Mark Gerstein. What is bioinformatics? a proposed definition and overview of the field. *Methods of information in medicine*, 40(04):346–358, 2001.

[79] Malawski Maciej. Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press*, 2012.

[80] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *2008 IEEE Fourth International Conference on eScience*, pages 222–229. IEEE, 2008.

[81] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 20(9):1297–1303, 2010.

[82] Jasraj Meena, Malay Kumar, and Manu Vardhan. Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. *IEEE Access*, 4:5065–5082, 2016.

[83] Timothy Merrifield and H Reza Taheri. Performance implications of extended page tables on virtualized x86 processors. In *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 25–35, 2016.

[84] Luiz AVC Meyer, Shaila C Rössle, Paulo M Bisch, and Marta Mattoso. Parallelism in bioinformatics workflows. In *International Conference on High Performance Computing for Computational Science*, pages 583–597. Springer, 2004.

[85] Hiroyuki Mishima, Kensaku Sasaki, Masahiro Tanaka, Osamu Tatebe, and Koh-ichiro Yoshiura. Agile parallel bioinformatics workflow management using pwrake. *BMC research notes*, 4(1):331, 2011.

[86] Farrukh Nadeem and Thomas Fahringer. Optimizing execution time predictions of scientific workflow applications in the grid through evolutionary programming. *Future Generation Computer Systems*, 29(4):926–935, 2013.

[87] Juan Navarro, Sitararn Iyer, Peter Druschel, and Alan Cox. Practical, transparent operating system support for superpages. *ACM SIGOPS Operating Systems Review*, 36(SI):89–104, 2002.

[88] Marco AS Netto, Rodrigo N Calheiros, Eduardo R Rodrigues, Renato LF Cunha, and Rajkumar Buyya. Hpc cloud for scientific and business applications: taxonomy, vision, and research challenges. *ACM Computing Surveys (CSUR)*, 51(1):1–29, 2018.

[89] Tung Nguyen, Weisong Shi, and Douglas Ruden. Cloudaligner: A fast and full-featured mapreduce based tool for sequence mapping. *BMC research notes*, 4(1):1–7, 2011.

[90] Frank Austin Nothaft, Matt Massie, Timothy Danford, Zhao Zhang, Uri Laserson, Carl Yeksigian, Jey Kottalam, Arun Ahuja, Jeff Hammerbacher, Michael Linderman, et al. Rethinking data-intensive science using scalable analytics systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 631–646. ACM, 2015.

[91] Chris Oehmen and Jarek Nieplocha. Scalablast: a scalable implementation of blast for high-performance data-intensive bioinformatics analysis. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):740–749, 2006.

[92] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[93] Ashish Panwar, Sorav Bansal, and K Gopinath. Hawkeye: Efficient fine-grained os support for huge pages. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 347–360, 2019.

[94] Ashish Panwar, Aravinda Prasad, and K Gopinath. Making huge pages actually useful. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 679–692, 2018.

[95] Sachin Pawaskar and Hesham H Ali. A dynamic energy-aware model for scheduling computationally intensive bioinformatics applications. In *2010 International Conference on High Performance Computing & Simulation*, pages 216–223. IEEE, 2010.

[96] Binh Pham, Ján Veselỳ, Gabriel H Loh, and Abhishek Bhattacharjee. Large pages and lightweight memory management in virtualized environments: Can you have it both ways? In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 1–12, 2015.

[97] Binh Pham, Jan Vesely, Gabriel H Loh, and Abhishek Bhattacharjee. Using tlb speculation to overcome page splintering in virtual machines. 2015.

[98] Deepak Poola, Saurabh Kumar Garg, Rajkumar Buyya, Yun Yang, and Kotagiri Ramamohanarao. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *2014 IEEE 28th international conference on advanced information networking and applications*, pages 858–865. IEEE, 2014.

[99] Ioan Raicu, Ian T Foster, and Yong Zhao. Many-task computing for grids and supercomputers. In *2008 workshop on many-task computing on grids and supercomputers*, pages 1–11. IEEE, 2008.

[100] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No. 98TB100244)*, pages 140–146. IEEE, 1998.

[101] Maria A Rodriguez and Rajkumar Buyya. Scheduling dynamic workloads in multitenant scientific workflow as a service platforms. *Future Generation Computer Systems*, 79:739–750, 2018.

[102] Maria Alejandra Rodriguez and Rajkumar Buyya. A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. *Concurrency and Computation: Practice and Experience*, 29(8):e4041, 2017.

[103] Anthony Rowe, Dimitrios Kalaitzopoulos, Michelle Osmond, Moustafa Ghanem, and Yike Guo. The discovery net system for high throughput bioinformatics. *Bioinformatics*, 19(suppl_1):i225–i231, 2003.

[104] Abhishek Roy, Yanlei Diao, Uday Evani, Avinash Abhyankar, Clinton Howarth, Rémi Le Priol, and Toby Bloom. Massively parallel processing of whole genome sequence data: an in-depth performance study. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 187–202. ACM, 2017.

[105] Michael C Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[106] Izzet F Senturk, Ponnuraman Balakrishnan, Anas Abu-Doleh, Kamer Kaya, Qutaibah Malluhi, and Ümit V Çatalyürek. A resource provisioning framework for bioinformatics applications in multi-cloud environments. *Future Generation Computer Systems*, 78:379–391, 2018.

[107] Sai Sha, Jing-Yuan Hu, Ying-Wei Luo, Xiao-Lin Wang, and Zhenlin Wang. Huge page friendly virtualized memory management. *Journal of Computer Science and Technology*, 35:433–452, 2020.

[108] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135, 2008.

[109] Tom Spink, Harry Wagstaff, and Björn Franke. Hardware-accelerated cross-architecture full-system virtualization. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(4):1–25, 2016.

[110] Choong-Hyun Sun, Byoung-Jin Kim, Gwan-Su Yi, and Hyoungwoo Park. A model of problem solving environment for integrated bioinformatics solution on grid by using condor. In *International Conference on Grid and Cooperative Computing*, pages 935–938. Springer, 2004.

[111] Mark Swindells, Mark Rae, Martyn Pearce, Stuart Moodie, Rob Miller, and Pat Leach. Application of high-throughput computing in bioinformatics. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 360(1795):1179–1189, 2002.

[112] G Tischler. biobambam2. *URL https://github. com/gt1/biobambam2*, 2017.

[113] John Vivian, Arjun Arkal Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D Deran, Audrey Musselman-Brown, et al. Toil enables reproducible, open source, big biomedical data analyses. *Nature biotechnology*, 35(4):314, 2017.

[114] Samuel L Volchenboum, Suzanne M Cox, Allison Heath, Adam Resnick, Susan L Cohn, and Robert Grossman. Data commons to support pediatric cancer research. *American Society of Clinical Oncology Educational Book*, 37:746–752, 2017.

[115] Liya Wang, Zhenyuan Lu, Peter Van Buren, and Doreen Ware. Sciapps: a cloud-based platform for reproducible bioinformatics workflows. *Bioinformatics*, 34(22):3917–3920, 2018.

[116] Yuxin Wang, Shijie Cao, Guan Wang, Zhen Feng, Chi Zhang, and He Guo. Fairness scheduling with dynamic priority for multi workflow on heterogeneous systems. In *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 404–409. IEEE, 2017.

[117] Yunni Xia, MengChu Zhou, Xin Luo, Qingsheng Zhu, Jia Li, and Yu Huang. Stochastic modeling and quality evaluation of infrastructure-as-a-service clouds. *IEEE Transactions on Automation Science and Engineering*, 12(1):162–170, 2013.

[118] Meng Xu, Lizhen Cui, Haiyang Wang, and Yanbing Bi. A multiple qos constrained scheduling strategy of multiple workflows for cloud computing. In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 629–634. IEEE, 2009.

[119] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[120] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *HotCloud '10*.