

Data-Intensive Biocomputing in the Cloud

Nabeel Meeramohideen Mohamed

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Wu-chun Feng, Chair
Heshan Lin
Ali Raza Ashraf Butt

July 3, 2013
Blacksburg, Virginia

Keywords: Cloud Computing, Next Generation Sequencing, MapReduce, GATK, Workflow
Copyright 2013, Nabeel Meeramohideen Mohamed

Data-Intensive Biocomputing in the Cloud

Nabeel Meeramohideen Mohamed

(ABSTRACT)

Next-generation sequencing (NGS) technologies have made it possible to rapidly sequence the human genome, heralding a new era of health-care innovations based on personalized genetic information. However, these NGS technologies generate data at a rate that far outstrips Moore's Law. As a consequence, analyzing this exponentially increasing data deluge requires enormous computational and storage resources, resources that many life science institutions do not have access to. As such, cloud computing has emerged as an obvious, but still nascent, solution.

This thesis intends to investigate and design an efficient framework for running and managing large-scale data-intensive scientific applications in the cloud. Based on the learning from our parallel implementation of a genome analysis pipeline in the cloud, we aim to provide a framework for users to run such data-intensive scientific workflows using a hybrid setup of client and cloud resources. We first present *SeqInCloud*, our highly scalable parallel implementation of a popular genetic variant pipeline called genome analysis toolkit (GATK), on the Windows Azure HDInsight cloud platform. Together with a parallel implementation of GATK on Hadoop, we evaluate the potential of using cloud computing for large-scale DNA analysis and present a detailed study on efficiently utilizing cloud resources for running data-intensive, life-science applications. Based on our experience from running SeqInCloud on Azure, we present *CloudFlow*, a feature-rich workflow manager for running MapReduce-based bioinformatic pipelines utilizing both client and cloud resources. CloudFlow, built on the top of an existing MapReduce-based workflow manager called Cloudfgene, provides unique features that are not offered by existing MapReduce-based workflow managers, such as enabling simultaneous use of client and cloud resources, automatic data-dependency handling between client and cloud resources, and the flexibility of implementing user-defined plugins for data transformations. In general, we believe that our work will increase the adoption of cloud resources for running data-intensive scientific workloads.

This work was supported in part by NSF CCF-1048253, as part of the NSF Computing in the Cloud Program with Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or Microsoft.

Dedication

To my loving brother, pattima, parents and friends

Acknowledgments

I, Nabeel, would like to express my deepest gratitude towards all of these people. They were guides to me in this path, without whom, I would have never seen this day happen.

The torchbearer, Prof. Wu-chun Feng, who guides all of us in the team, towards a successful career path. Words fall short, when I find, to thank him in my endeavor. As my advisor, he was patient, inspirational, and willing to guide me along the way. His capability to nurture one's talents and bring out their hidden potential is awesome. I have to add that the bi-weekly meetings conducted by him gave me the nidus; ignited new thought patterns; nourished the growth of my work and chiseled its outcome. I owe every bit of my success in this thesis project to him.

I have to thank Dr. Heshan Lin, for his support and screening of my ideas; imparting them with a particular sense of direction, at times of need. He was always available to me, when I had doubts, when I was stuck, and when I had panic attacks. Thanks a lot, Dr. Heshan. I would also like to thank you for being a committee member.

I would like to thank Prof. Ali R. Butt for his insights and comments on my thesis work, for providing me with career guidance, and for being a part of the defense committee.

Our world brightens in the research lab with Prof. Eli Tilevich around. His sense of humor and capability to lighten the moment was invaluable during my long hours in the lab. Thanks, Professor.

My team members in the SyNeRGy Lab. We co-existed during the past two years with all our ups and downs with mutual support and shoulders to lean on. With them, my hours in lab were made more memorable. Their constructive criticism during the team meetings, especially from Thomas Scogland, Balaji Subramaniam and Umar Kalim, added catalyst to the fuel. I also need to thank Dr. Nataliya E. Timoshevskaya for giving wonderful suggestions, with respect to defense presentation slides and thesis documentation.

Virginia Tech – Oh! My dream. Without you, I would be less a person when I die. I even owe the stone in which you are made of. Such was your impact. Each of my department faculty and staff, graduate school office bearers – you will remain etched in my life forever. The concern all of you exhibit is simply mind-blowing.

Friends in CRC – who were my buffer, at all times – Krish, Vignesh, Sriram, Lokendra and Rajesh. Thanks for humming.

I was never born with them, they became a part of me, and I can now never dislodge them from my heart. My roommates: Ananth, Kiran, M.A. and Subbu.

My Family, Pattima and associates - without whom, I as a person and a dream chaser, would never have had the base to take-off. I am for you all.

Thanks again, Everyone.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Related Work	3
1.3	Contributions	7
1.4	Document Overview	10
2	Background	11
2.1	Cloud Service Models	11
2.2	Microsoft Azure	12
2.3	Apache Hadoop	14
2.4	Genome Analysis Toolkit	16
2.4.1	Overview of File Formats	17
3	SeqInCloud: Sequence Analysis in the Cloud	20
3.1	Overview	20
3.2	Design and Implementation	21
3.2.1	SeqInCloud Workflow Stages	21
3.3	SeqInCloud Optimizations	24
3.3.1	Compute Optimizations	24
3.3.2	Data Transfer Optimizations	38
3.4	Applicability of SeqInCloud to Other Cloud Platforms	45

4	CloudFlow: Distributed Workflow Manager for the Cloud	48
4.1	Cloudgene Overview	49
4.2	Design and Implementation	51
4.2.1	Cloudgene Architecture	51
4.2.2	CloudFlow Architecture	53
4.2.3	CloudFlow Features	53
4.3	CloudFlow Optimizations	61
4.4	Case Study	64
4.5	Applicability of CloudFlow to Other Cloud Platforms	69
5	Conclusions and Future Work	70
5.1	Conclusions	70
5.2	Future Work	72
	Bibliography	73

List of Figures

1.1	Sequencing Progress vs Compute and Storage	2
1.2	DNA Sequencing Cost of Human-sized Genome	3
1.3	Comparison of CloudFlow with Other Solutions	5
2.1	Cloud Service Models	12
2.2	Components of Azure Blob Service	13
2.3	HDInsight Service Storage Architecture	14
2.4	MapReduce Phases	15
2.5	Variant Analysis Pipeline with GATK from Broad Institute	17
2.6	Sequence Read Formats	18
2.7	Variant File Format	19
3.1	SeqInCloud Workflow	21
3.2	Input BAM Size vs Output Variant File Size	23
3.3	Contig-based Partitioning	25
3.4	Loci-based Partitioning	26
3.5	Loci-based Partitioning - The Problem	26
3.6	Local Realignment - The Problem	27
3.7	Design of IndelRealigner Stage	28
3.8	Mix of Contig and Loci-based Approach for Local Realignment Stage (SeqIn- Cloud 1.0 vs SeqInCloud 2.0)	29
3.9	SeqInCloud 2.0: Optimized SeqInCloud Workflow	30
3.10	Feasible Input/Output Storage Resource for the “Mix” Mapping.	31

3.11	Comparison of Baseline and SeqInCloud Execution Time of Entire Workflow (Except Alignment Stage) for Datasets NA12878 and NA21143.	33
3.12	Execution Time of the Major Stages in SeqInCloud for the 24.3 GB NA10847 Dataset.	34
3.13	Execution Time of the Major Stages in SeqInCloud for the 11 GB NA21143 Dataset.	35
3.14	Performance Comparison of SeqInCloud 1.0 vs SeqInCloud 2.0	37
3.15	Execution Time of the Workflow and the Individual Stages for NA19066 Dataset with Increasing Number of Nodes.	38
3.16	Execution Time of “All HDFS”, “Mix” and “All Blob” Mappings.	39
3.17	Reference-based Compression	40
3.18	Acceleration Using GPU	42
3.19	Dataset Used for Evaluation	42
3.20	Breakeven Point for Reference-based Compression	43
3.21	Performance Improvement and Storage Savings for NA10847, NA21143 and NA12878 Datasets Due to Compression Using a 14-node On-premise Hadoop Cluster.	44
3.22	Evaluation of GPU Acceleration Using CUSHAW	45
4.1	Cloudgene Modules	50
4.2	Cloudgene Manifest File	50
4.3	Cloudgene Job Submission Architecture	52
4.4	CloudFlow - High Level Architecture	54
4.5	CloudFlow User Interface	55
4.6	CloudFlow DAG Configuration	56
4.7	CloudFlow DAG Example Configuration	57
4.8	CloudFlow Hybrid Cloud Support	58
4.9	CloudFlow Data-Dependency Configuration	59
4.10	CloudFlow Data-Dependency Table	60
4.11	CloudFlow Data-Dependency Flowchart	65

List of Tables

4.1	Data-Dependency Table – Initial Values	68
4.2	Data-Dependency Table – After Data Transfer	68

Chapter 1

Introduction

1.1 Motivation

Today, next-generation sequencing (NGS) technologies generate data at a rate much faster than that of the growth of compute and storage capacity [1, 2, 3]. The growth of compute capacity is given by Moore’s law, which states that the number of transistors doubles every two years. The growth of storage capacity is given by Kryder’s law, which states that the storage capacity or the storage density doubles every thirteen months. Fig. 1.1 shows that the rate of growth of sequence data is faster than the growth of compute and storage capacity [4].

In addition, to further support this trend, the cost of sequencing a human-sized genome has come down rapidly from 95 million US dollars in 2001 to twenty thousand US dollars in 2011 [5], as shown by Fig. 1.2. As a consequence, storing and analyzing genomic data has become a fundamental “big data” challenge due to the high cost associated with owning and maintaining on-premise compute resources. Cloud computing offers an attractive model where users can access compute resources on-demand and scaled according to their

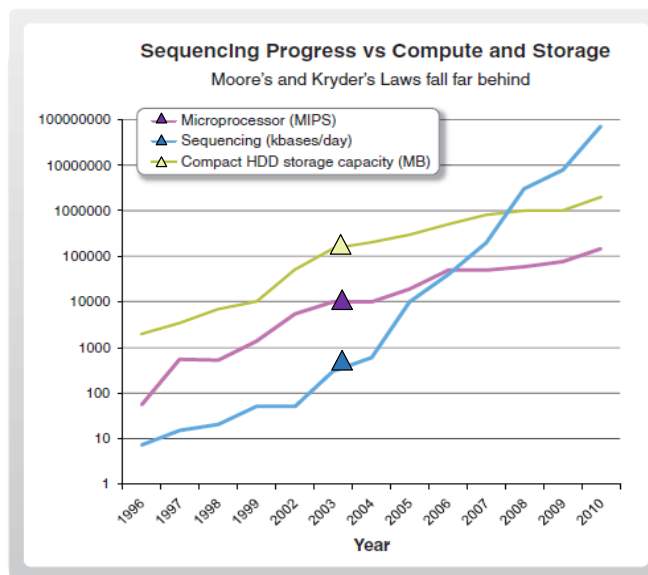


Figure 1.1: Sequencing Progress vs Compute and Storage

needs. The cloud computing model also enables the easy sharing of public dataset and helps to facilitate large-scale collaborative research, accelerated using Azure content delivery network (CDN) [6] support. As such, cloud computing has gained increasing traction in the bioinformatics community.

In spite of the above advantages with cloud computing, its adoption is not as fast as expected. Life scientists are reluctant to move to the cloud due to data-security concerns, data-transfer overhead, and unavailability of an easy-to-use interface to configure cloud-enabled workflows. Our work seeks to address these issues by studying the challenges behind running large-scale data-intensive workloads in the cloud and further use this knowledge to design a feature-rich MapReduce-based workflow manager to configure and monitor bioinformatic workflows. Here, *data-intensive* workloads refer not only to the volume of data accessed from storage or transferred via network (I/O rate), but also the processing required to convert the data to an usable form (compute rate). So, data-intensive workloads are typically both I/O and compute intensive.

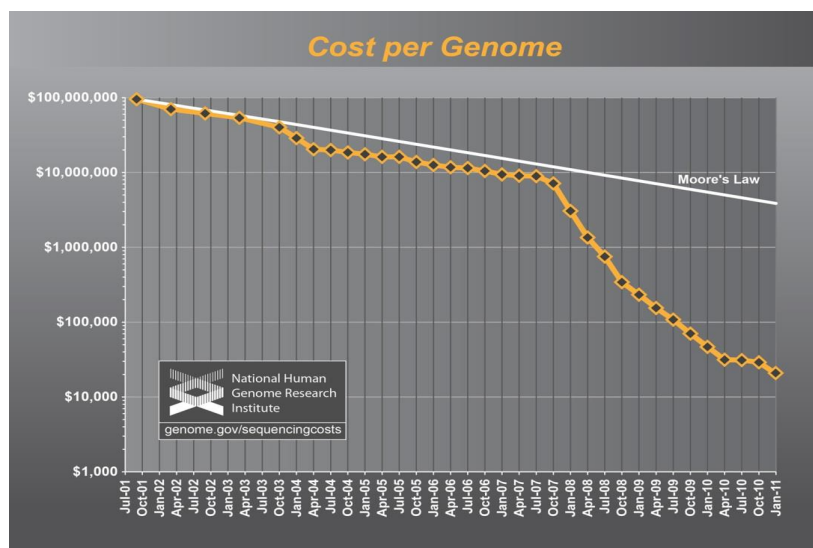


Figure 1.2: DNA Sequencing Cost of Human-sized Genome

1.2 Related Work

In recent years, there has been a steep increase in the number of bioinformatic applications [7] and workflows [8, 9, 10, 11, 12, 13, 14] that use MapReduce [15] framework, a large percentage of which runs in the cloud [16, 17, 18, 19].

Crossbow [12] and Myrna [10] implement workflows for single-nucleotide polymorphism (SNP) discovery and RNA-Seq differential expression analysis, respectively, in the cloud. Crossbow uses Bowtie [20] to align reads in the map phase, sorts alignments by genomic region, and uses SOAPsnp [21] for SNP discovery. Both use Hadoop Streaming [22] to implement the workflow. CloudBurst [23] is a parallel read-mapping algorithm that is optimized using Hadoop MapReduce. The Genome Analysis ToolKit (GATK) [24, 25] is a MapReduce-like framework, which provides various sequence analysis tools that are extensively used by SeqInCloud. While GATK does *not* support distributed parallelism, it does provide a command-line scripting framework, GATK-Queue [26], to implement workflows. GATK-Queue can run jobs in batch processing systems like Oracle Grid Engine [27]. GATK-

Queue recommends partitioning the workflow by chromosome, but this causes an uneven distribution of workload, resulting in stragglers adversely affecting the performance of the workflow.

Pireddu et al. [13, 28] discuss Seal, a workflow that uses Pydoop [29] and BWA to implement short-read mapping and duplicate removal. Seal provides its own implementation of de-duplication and covariate table calculation using the MapReduce framework. In HugeSeq [8], Lam et al. discuss a three-stage workflow, which uses GATK in their pipeline. HugeSeq does not use the MapReduce framework and runs on Sun Grid Engine (SGE) clusters. HugeSeq also partitions the dataset by chromosome, which exhibits a load imbalance problem similar to GATK-Queue. In SIMPLEX [14], Fischer et al. discuss a cloud-enabled autonomous exome analysis workflow, which is implemented as a web service and shipped as a cloud image for ease of use. It uses GATK for recalibration and SNP discovery, but does not parallelize it using the MapReduce framework.

Abhishek Roy et al. [9] discusses a deep analysis pipeline that uses association mining to discover patterns with both statistical significance and biological meanings. The authors specify that their pipeline considers MapReduce-style processing for implementing quality score recalibration. Leo et al. [30] evaluated BLAST, GSEA, and GRAMMAR using the MapReduce framework and observed that the MapReduce cluster is efficiently utilized only for a fraction of the jobs running time and the best job execution time is achieved when the number of map tasks is same as the total number of CPU cores in the cluster. In CloudBLAST [31], Matsunga et al. parallelized BLAST using MapReduce framework and observed that the virtualization overhead is minimal if the application is compute intensive and the I/O working set fits completely in the aggregate cluster memory.

Compared to these parallel GATK studies, SeqInCloud delivers a significantly more portable and scalable design and offers several cloud-specific optimizations that can be applied in any

of the above environments.

There is limited work in the area of cloud and Hadoop [32] aware workflow managers that are used to compose and run MapReduce based pipelines. Our solution, “CloudFlow” built on top of Cloudfgen [33] seeks to fill this void. We based our solution on Cloudfgen as it is easy to configure and intuitive to use. Cloudfgen is built using components that are platform-independent like ExtJS [34], Restlet [35], and JSON [36] and offers a rich feature set compared to the existing approaches.

Fig. 1.3 provides a comparison of CloudFlow with existing MapReduce-based workflow managers.

Tools Features	CloudFlow	Cloudfgen	Oozie	Kepler + Hadoop	Amazon EMR	Azure HDInsight	CloVR	Tavaxy
<i>MapReduce support</i>	✓	✓	✓	✓	✓	✓	✗	✗
<i>DAG Support</i>	✓	✗	✓	✓	✗	✓	✓	✓
<i>Automated data dependency</i>	✓	✗	✗	✗	✗	✗	✗	✗
<i>Hybrid cloud support</i>	✓	✗	✗	✗	✗	✗	✗	✓
<i>On demand cloud provisioning</i>	✓	✗	✗	✗	✓	✗	✓	✓
<i>Extensibility</i>	✓	✓	✓	✓	✗	✗	✓	✗
<i>Plugin support</i>	✓	✗	✗	✗	✗	✗	✗	✗
<i>Triggers</i>	✗	✗	✓	✓	✗	✓	✗	✓

Figure 1.3: Comparison of CloudFlow with Other Solutions

Oozie [37] is a workflow scheduler used to configure, run, and monitor MapReduce jobs. The configuration of MapReduce workflows via Oozie is not very intuitive, as users have to provide low-level configuration details like the mapper/reducer class, InputFormat, and so on. Oozie does not implement automated data-dependency handling mechanisms between client and cloud clusters in a hybrid environment and hence puts the burden on the user to

ensure that the data is ready and available.

Kepler [38] is in the early stages of integration with Hadoop and aims to provide an easy-to-use interface to compose, execute, and monitor MapReduce applications in Kepler workflows. Kepler also requires a low-level configuration like Oozie and is not very intuitive to use for domain scientists. In addition, Kepler does not integrate with existing cloud providers and hence does not have any on-demand provisioning or automated data-dependency handling support.

Clovr [39] is implemented as a portable virtual machine (VM) image that provides several automated pipelines. Clovr VMs can utilize either client or cloud resources, but not both simultaneously. Clovr supports automatic provisioning of cluster resources during pipeline execution and offers customizable VM images that can execute on multiple platforms. Clovr is built using different components like Ergatis [40], a workflow system and Vappio [41], which is built on top of the Amazon EC2 API [42] and is used for managing EC2 clusters. Clovr currently runs on grid resources and is not MapReduce-enabled.

The commercial cloud providers, i.e., Amazon and Microsoft, provide their own workflow managers, Amazon EMR [43] and Azure HDInsight with Oozie support [44], respectively. Amazon EMR lacks most of the features. Azure HDInsight uses Oozie for MapReduce workflows and hence has the same feature set as Oozie.

Tavaxy [45] is a workflow management system that combines Taverna [46] and Galaxy [47] sub-workflows. Tavaxy supports simultaneous use of local infrastructure and remote resources using web services and triggers. However, Tavaxy does not support MapReduce-based workflows.

Compared to the above discussed MapReduce-based workflow systems, none of them provide support for handling automatic data-dependency in hybrid cloud environments, where the

client and cloud resources can be simultaneously utilized by the workflow scheduler. In addition, CloudFlow offers an unique flexibility for the users to define data-transforming plugins, like a plugin for compression/encryption that can be used before and after transferring data between client and cloud clusters.

1.3 Contributions

In this work, we address the following objectives,

- Understand the challenges of running data-intensive scientific applications in the cloud, e.g., Microsoft Azure.
- Devise a robust data management and analysis software to accelerate data-intensive bioinformatic pipelines on Azure. Optimize the data management framework to employ semantic-based transformations and accelerate data transfers to the cloud.
- Based on our experiences of running such data-intensive workloads, design a flexible workflow management system to compose, run, and monitor data-intensive bioinformatic pipelines using a hybrid setup of client and cloud resources.

We first focus on accelerating a widely used, genome-analysis pipeline [48] built atop Burrows-Wheeler Aligner (BWA) [49] and the Genome Analysis Toolkit (GATK)¹ framework [24, 25] on Microsoft Azure [50], a *platform-as-a-service* (PaaS) cloud environment. Parallel implementations of the GATK pipeline in cluster environments have been investigated in several previous studies [13, 8, 26]. While these implementations can be deployed on *infrastructure-as-a-service* (IaaS) clouds such as Amazon EC2, they require external software packages (e.g.,

¹Our current implementation of SeqInCloud uses the latest open-source version of GATK (i.e., version 1.6). We note that our design approach and optimization techniques should be applicable to subsequent GATK versions.

Pydoop and Oracle Grid Engine) that are not available on PaaS clouds such as Microsoft Azure. In addition, existing parallel GATK implementations are designed for clusters where node failures are rare, and thus not suitable for cloud environments where node failures are rather norm.

To address the above issues, we present *SeqInCloud*, short for “sequencing in the cloud” and pronounced as “seek in cloud.” SeqInCloud seamlessly integrates all the stages in the GATK pipeline with the Hadoop [32] framework in order to maximize portability. By doing so, SeqInCloud can be easily deployed on PaaS and IaaS clouds as well as on on-premise clusters. The tight integration with Hadoop also enables SeqInCloud to leverage Hadoop’s fault-tolerant features to transparently handle node failures in cloud environments.

In addition, SeqInCloud offers a number of novel features that are critical to the cost and performance in cloud environments:

- In existing parallel GATK implementations, parallelism is achieved by partitioning the input data by contig² (e.g., chromosome). Due to the limited number of contigs and the large variation in contig sizes, such a partition-by-contig approach suffers from limited scalability and load imbalance, resulting in wasted cloud resources. To address this issue, SeqInCloud adopts a highly scalable design that allows data processing to be partitioned by loci or genome location, a much finer level of parallelism.
- To optimize network costs, SeqInCloud enables application-level compression by converting the Binary Alignment/Map (BAM) [51] format to a reference-based compression format like CRAM [52] before transferring data to the cloud. The compressed CRAM file is typically 40% to 50% smaller than the original BAM file. In addition, SeqInCloud optimizes storage costs by converting the CRAM file to a lossless BAM

²A sequence contig is a contiguous, overlapping sequence read from paired-end sequencing.

file for downstream analysis in the cloud.

- SeqInCloud can also use graphics processing units (GPUs) to improve workflow performance by pre-processing the data using a GPU-based aligner, before transferring it to the cloud. Overall, we observed 33% to 50% improvement in workflow performance with the data-transfer optimizations.
- To improve I/O performance, SeqInCloud maps input and output data across the storage hierarchy on Azure, including the local filesystem, Azure Blob [53], and Hadoop Distributed Filesystem (HDFS), according to their I/O characteristics. Experiments show that our storage-mapping approach can achieve a performance improvement of 20% compared to uniformly storing all data on HDFS.

Based on our experiences from running large-scale, data-intensive bioinformatic workflows on Azure, e.g., SeqInCloud, we have designed a MapReduce-based workflow management system that we call *CloudFlow*. CloudFlow allows users to easily compose flexible data-processing pipelines on distributed cloud resources, e.g., a hybrid environment of client and cloud resources. CloudFlow offers the following features on top of Cloudgene, which makes it superior for the distributed execution of MapReduce pipelines, such as SeqInCloud, in a hybrid cloud environment:

- Support for processing DAG-based³ MapReduce pipelines
- Support for on-demand provisioning of cloud resources
- Support for automated data-dependency handling (automatic data transfers) between different workflow stages running in a hybrid environment

³DAG: directed acyclic graph

- User-defined plugins for byte-level or context-based data transformations before exchanging data between client and cloud resources

Impacts of Our Work: There are two majors impacts of our work,

- SeqInCloud accelerates a popular genetic variant discovery pipeline, thus enabling life scientists to quickly determine the cause for diseases like cancer, for example.
- The goal of cloud-enabled software frameworks like SeqInCloud and CloudFlow is to increase the adoption of cloud computing among domain scientists by addressing the challenges and shortcomings of running data-intensive scientific applications in the cloud. This enables domain scientists to focus their energy towards optimizing their domain science rather than worrying about the computer science aspects like configuration and setup, data management, and so on.

1.4 Document Overview

The rest of this thesis is organized as follows. Chapter 2 presents background information about (i) Microsoft Azure and HDInsight service, which is the Windows implementation of Apache MapReduce, (ii) Apache Hadoop and MapReduce, and (iii) Genome Analysis Toolkit. In Chapter 3, we discuss the design and implementation of *SeqInCloud* to support fine-grained, loci-based partitioning, the various compute- and data-transfer optimizations to efficiently utilize the cloud resources and discuss the results. In Chapter 4, we present the current design of Cloudgene and provide detailed information on the different features that CloudFlow implements on top of Cloudgene. We also discuss the various data-transfer optimizations that CloudFlow incorporates. Finally, in Chapter 5 we give a brief summary of our tools and discuss the future work.

Chapter 2

Background

2.1 Cloud Service Models

The *cloud clients* provide an interface to communicate with the cloud resources. These are usually web browsers, mobile application, or some kind of terminal emulators. *Software as a Service (SaaS)* is a software delivery model where the software and its associated data are centrally hosted on the cloud. The *Platform as a Service (PaaS)* model provides a computing platform and solution stack as a service. Here the cloud users create software using the tools or libraries provided by the cloud provider. The *Infrastructure as a Service (IaaS)* model provides physical or virtual machines as a service. Here, the entire software stack from the operating system to the application software has to be installed and managed by the cloud users. These service models are depicted in Fig. 2.1 [54].

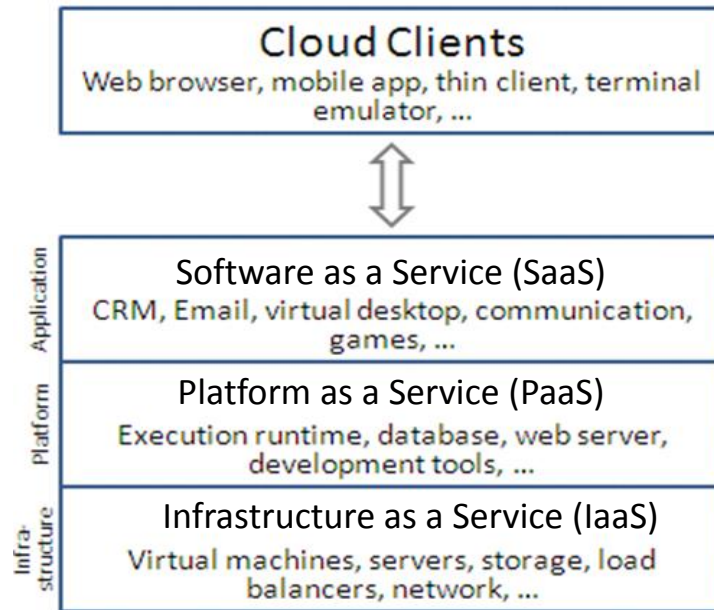


Figure 2.1: Cloud Service Models

2.2 Microsoft Azure

Windows Azure [50] is a cloud service offered by Microsoft that enables one to deploy, build, and manage applications across Microsoft-managed data centers. Microsoft offers a wide range of cloud-based solutions. A brief description of the relevant ones are provided below. Our work extensively makes use of the big data and storage service.

- **Infrastructure** is an infrastructure-as-a-service (IaaS) offering, where the users can spin up new Windows Server or Linux virtual machines on-demand and pay only for their usage.
- **Big Data** enables users to unearth insights from big data and drive business decisions. This is enabled with Azure HDInsight [44], which is a PaaS-based big data solution powered by a Windows implementation of Apache Hadoop. HDInsight services for Windows Azure makes Apache Hadoop available as a service in the cloud.

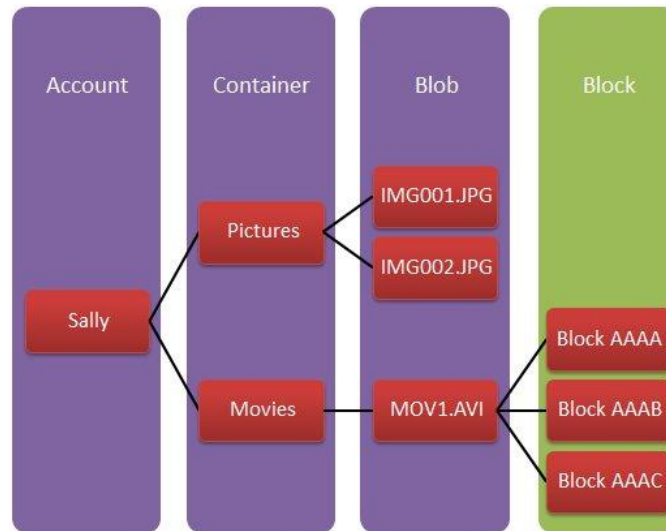


Figure 2.2: Components of Azure Blob Service

- **Storage, Backup and Recovery** is powered using Windows Azure storage services that are provisioned across Microsoft's eight region-wide data centers. Azure storage provides scalable, secure, and highly reliable storage services like blob, tables, and queues. Blob storage is a service for storing large amounts of unstructured data in the cloud accessed via http[s]. The Blob service contains the components as shown in Fig. 2.2 [55]:

Storage Account: This is the highest level of the namespace for accessing blobs. An account can contain an unlimited number of containers limited by a total size of 100 TB.

Storage Container: A container can store an unlimited number of blobs. All blobs must be in a container. When mapped to a typical file system hierarchy, containers could be visualized as a directory that can hold other directories or files.

Blob: A blob [53] can be a page or block blob. A block blob can be up to 200 GB in size and is used to represent large files. Page blobs are a collection of 512-byte pages and are optimized for random read and write operations. The page blob can be up to 1 TB in size.

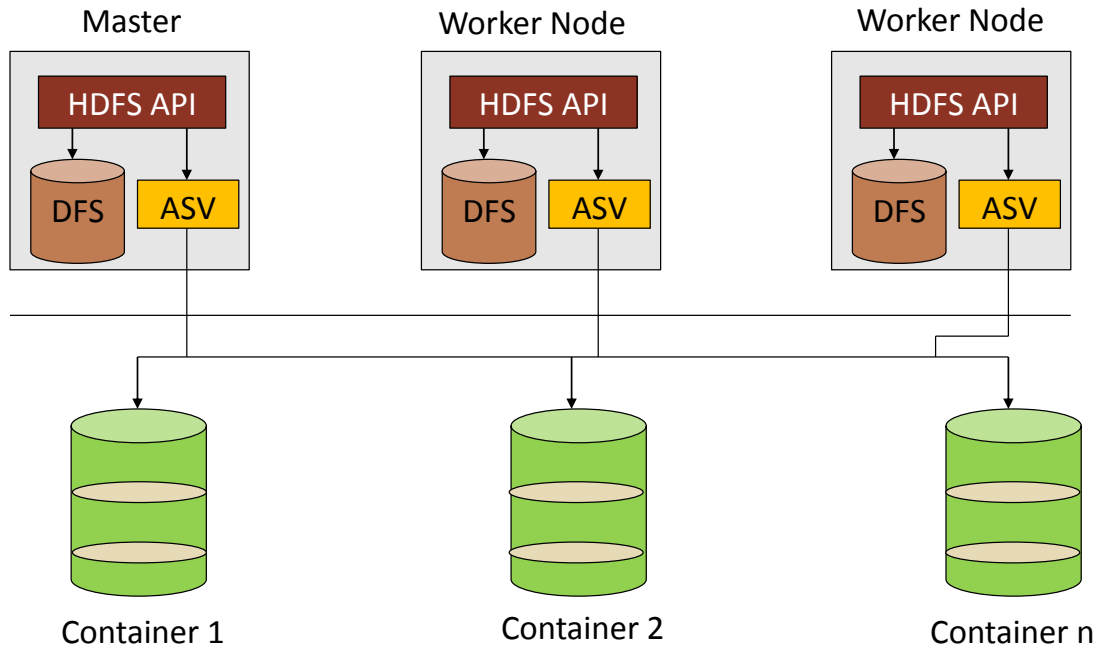


Figure 2.3: HDInsight Service Storage Architecture

The HDInsight service provides access to both the distributed file system that is both locally attached to the compute nodes, accessed by HDFS URI (`hdfs://`) and the data stored in blob storage containers, accessed by ASV URI (`asv://`).

2.3 Apache Hadoop

Hadoop is an open-source implementation of a computational paradigm named MapReduce, derived from Google's MapReduce [15] and Google File System (GFS) [56] papers. Hadoop has two major subsystems: MapReduce framework and Hadoop Distributed File System (HDFS). MapReduce, as shown in Fig. 2.4, is a functional programming framework, which is used to process massive amounts of data in a distributed fashion on large-scale commodity clusters. The MapReduce framework exposes two major primitives: `map` and `reduce`. The `map` phase takes input key-value pairs and translates them into intermediate key-value pairs.

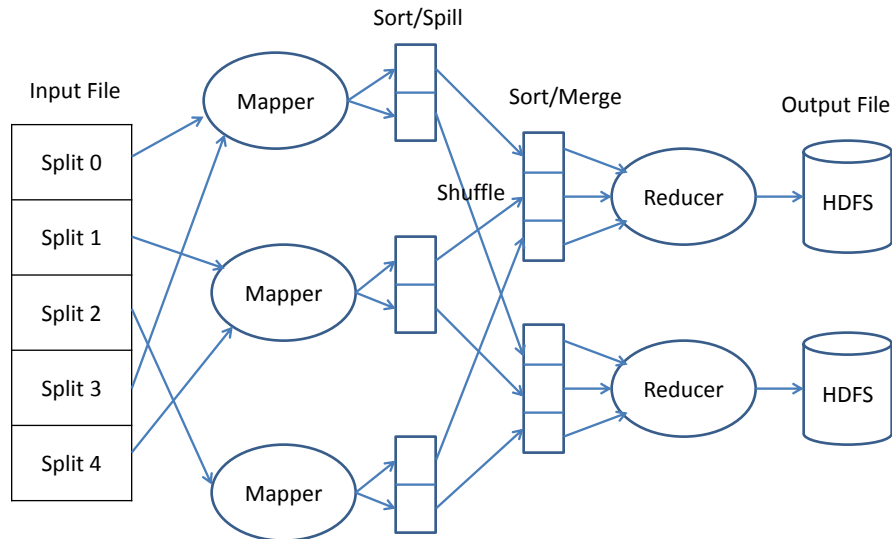


Figure 2.4: MapReduce Phases

The *reduce* phase converts the intermediate key-values into final key-value pairs and writes the output to HDFS. The input and output data are stored in HDFS, while the intermediate data are stored in the local file-system. In addition to defining the map and reduce functions, the users of the framework have to define the *InputFormat* for the job. The *InputFormat* defines how the data needs to be split for processing by multiple mappers and also defines a *RecordReader*, which identifies logical record boundaries from the split and emits them as key-value pairs to the mappers. The output from each mapper is then sorted by the framework, where an optional *combiner* could be used to combine identical keys and reduce inter-node traffic between mapper and reducer. The output keys from the mappers have to be partitioned among multiple reducers and this is defined by a *Partitioner*.

HDFS consists of a master *Namenode* process that manages the filesystem namespace and maintains the metadata for the entire filesystem tree. The *Datanode* process runs on each cluster node and stores the file data in its local filesystem. The data in HDFS is split into 64 MB blocks (by default) and is distributed among the cluster nodes. The *Namenode* maintains the list of blocks and node mappings for each file in the filesystem. The MapReduce

framework assigns input splits to a mapper process on a best-effort basis, such that the data can be locally accessed by the mapper without remote accesses.

2.4 Genome Analysis Toolkit

GATK is a software library, written in Java, that provides tools for analyzing next-generation sequencing (NGS) data. GATK is organized into *data traversals* and *data walkers*. The data traversals access the data set and provide data to the data walker, which processes the data. The data traversal is done either on a per-read basis or on a per-genome location basis. Our solution modifies GATK to read and write data from/to HDFS and write variants in variant call format (VCF) to HDFS. In addition to this, a few walkers were modified to output data directly to the reducer, rather than writing it to the local filesystem.

GATK inherently supports three kinds of parallelism: *shared memory* parallelism, where parallelism is achieved on a single node using multiple threads, *scatter-gather* (SG) parallelism, where multiple GATK instances that run on the same/different node are assigned with its own exclusive genome interval to work on, and *distributed* parallelism, which is a coordinated SG parallelism where multiple GATK instances work on the same dataset and is no longer a supported feature. The SG parallelism works well for walkers that are locus-based and hence independent per site. Otherwise, the level of parallelism is determined by the number of unique contigs in the data set. The SG parallelism is automated using a command-line scripting framework called *GATK-Queue*. GATK-Queue is used to build genome analysis pipelines that run on batch processing compute farms like Grid Engines.

The variant analysis pipeline recommended by MIT Broad Institute has three phases: (1) *NGS data processing*, which takes raw reads as input and outputs analysis ready reads, (2) *variant discovery and genotyping*, which takes analysis-ready reads as input and outputs

raw variants, and (3) *integrative analysis*, which takes raw variants as input and produces analysis-ready variants. This is depicted in Fig. 2.5 [48].

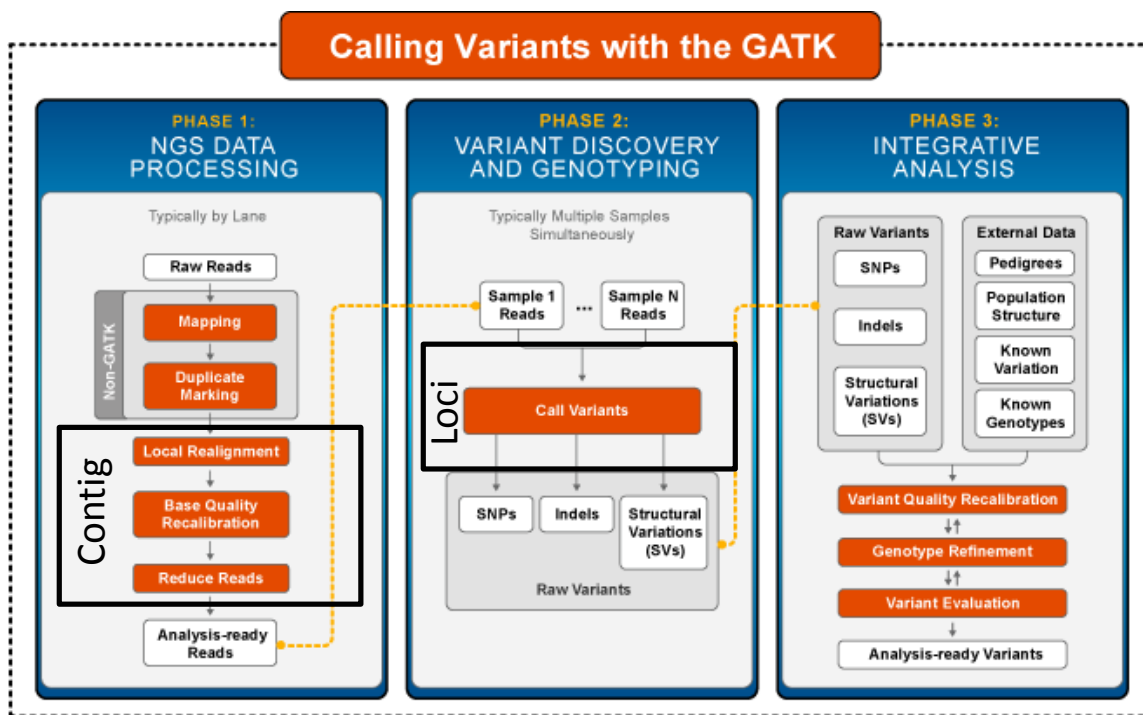


Figure 2.5: Variant Analysis Pipeline with GATK from Broad Institute

2.4.1 Overview of File Formats

This section gives a brief overview of the different file formats used in the genome analysis pipeline. Fig. 2.6 gives an instance for each format.

Unaligned Reads (*FASTQ file*). The DNA sequences or unaligned reads emitted by sequencers are stored in the FASTQ format. The FASTQ file uses four lines per read. The first line begins with a '@' character and is followed by a sequence identifier and optional description. The second line contains the read or DNA sequence. The third line contains the '+' character and the last line encodes the quality values of the read.

Aligned Reads (*SAM/BAM/CRAM file*). The aligned reads can be stored in a text format called Sequence Alignment Map (SAM) [51], compressed format called Binary Alignment Map (BAM), or a reference-based compressed format called CRAM [52]. An aligned read has the necessary information to determine the exact location where the read aligns to a reference genome, with additional tags that specify the mismatches/gaps in the alignment. A single-end aligned read is independent, whereas as paired-end aligned read is always associated with another read, termed as its mate pair. Both the reads in a mate pair stores alignment information of the other. The maximum allowed difference between the alignment location of the mate pairs is termed as the insert size. The CRAM format uses reference-based compression, i.e., instead of completely storing the sequence, the difference between the sequence and reference data is stored. This reduces the file size by 40% to 50% on average. The SAM/BAM file can be re-generated later from the CRAM file with the help of the reference genome.

.fastq file - DNA Sequence (Raw Read)

```
@SRR016607.1157 BI:302GJAAXX090504:8:1:10:1927 length=50
TATCCAGAGCTGTCTCCCTCTCTTTCAAGGCTCGATTCTGTC
+
IIIIIIIIIIIIIIIIIIIIIGIHIIII@BII;1I:H@HE6:I7A
```

.sam / .bam file – Single-end Aligned Read

```
SRR016607.1157 163 chr22 16050000 15 48S31M21S = 0 0 TATCCAGAGCTGTCTCCCTCTCTT
TCAGTTTCAAGGCTCGATTCTGTC IIIIIIIIIIIIIIIIIIIIIIGIHIIII@BII;1I:H@HE6:I7A RG:Z:foo
```

Figure 2.6: Sequence Read Formats

Genetic Variants (*VCF file*). The genetic variations between the input DNA sequence and the reference genome is represented in a variant call format (VCF) file [57], as shown

in Fig. 2.7. VCF is a text file format that contains multiple data lines. Each line provides variation in a single genome location (loci), if there is a difference between the input DNA sequence and the reference at this loci. The variation could be a single nucleotide (SNP), insertion, or a deletion.

.vcf file – Variant file

```
20 61795 . G T 1437.08 . AC=1;AF=0.50;AN=2;BaseQRankSum=-3.496;DP=106;  
21 Dels=0.00;FS=6.389;HRun=0;HaplotypeScore=0.5784
```

Figure 2.7: Variant File Format

Chapter 3

SeqInCloud: Sequence Analysis in the Cloud

3.1 Overview

Fig. 3.1 shows a SeqInCloud workflow, implemented using the Azure HDInsight cloud framework. SeqInCloud [58] uses the Hadoop MapReduce framework and runs the workflow in a distributed fashion using multiple compute nodes provisioned in the cloud. The workflow starts with the alignment stage, which uses our own distributed implementation of *BWA* and supports both single- and paired-end sequence alignment. The aligned reads are sorted, merged, and fed into a local realignment stage, which uses the *RealignerTargetCreator* and *IndelRealigner* walkers¹ from GATK. The realigned reads are fixed for discrepancy in mate information using Picard's *FixMateInformation*, de-duplicated using Picard's *MarkDuplicates*, and re-indexed. The quality score of the de-duplicated reads are recalibrated using

¹GATK is structured into walkers and traversals. GATK walkers are analysis modules that process data fed by the GATK traversals.

CountCovariates and *TableRecalibration* walkers. This is followed by the identification and filtering of structural variants (SNP and INDELS) using *UnifiedGenotyper* and *VariantFilteration* walkers. Finally, the variants are merged using *CombineVariants* walker. SeqInCloud takes FASTQ file (couple of them for paired end alignment) as input and emits both structural variants in VCF format and analysis ready reads in BAM format. The input and output varies if one attempts to selectively run different combination of stages, using the command line interface.

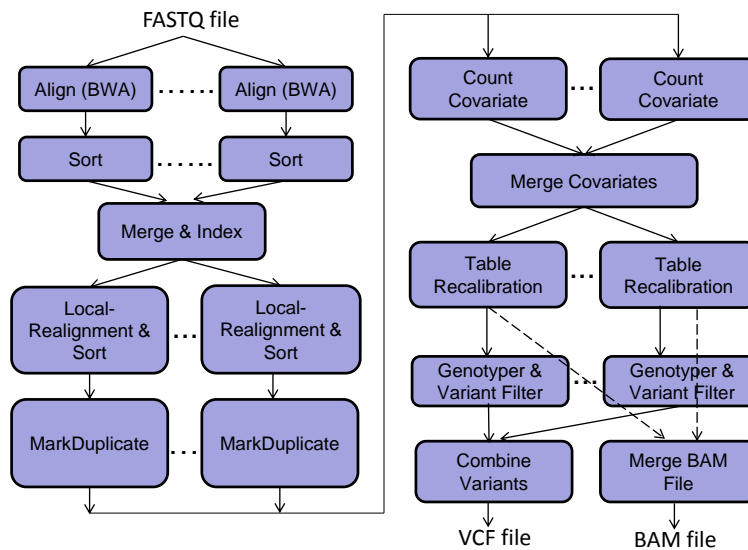


Figure 3.1: SeqInCloud Workflow

3.2 Design and Implementation

3.2.1 SeqInCloud Workflow Stages

Here we present design details on the various stages in SeqInCloud, in particular, sequence alignment, local realignment, and base quality recalibration and variant calling.

Sequence Alignment

SeqInCloud uses BWA [59] to run both single- and paired-end sequence alignment in the MapReduce framework. SeqInCloud utilizes the Windows port of BWA from [60]. The input FASTQ file(s) are split into multiple fragments by the mappers. The number of fragments are same as the number of reduce slots in the cluster. These fragments are aligned in parallel by the reducers. Considering the human reference genome, per compute node, BWA requires about 3-4 GB of memory, and the Hadoop daemons require about 2 GB of memory. If the compute nodes are medium-sized Azure virtual machine (VM) instances that have a fixed memory limit of 3.5 GB, there would be resource constraints and running BWA under such constraints results in “out of heap space” memory errors. To address these errors and to provide more flexibility in VM provisioning, SeqInCloud provides the flexibility to offload the sequence-alignment stage either completely or partially to on-premise resources. The resulting BAM files are then transferred to the cloud using application-level compression (e.g., conversion to CRAM), as described in Section 3.3.2.

Local Realignment

The local realignment stage consists of two steps: (1) identifying suspicious alignment intervals that require realignment and (2) running the realigner. The suspicious intervals are identified using GATK’s `RealignerTargetCreator`, which is a locus-based walker that is capable of processing read sequences independently by intervals. The realignment is done using GATK’s `IndelRealigner`, which is a read-based walker, that mandates a single GATK instance to process read sequences from the same contig.

Base Quality Recalibration & Variant Calling

The base quality recalibration consists of two steps: CountCovariate and TableRecalibration. The CountCovariate stage determines new empirical quality score used for recalibration. This is followed by the TableRecalibration step, which rewrites the quality score of the reads with the empirical quality values calculated by the CountCovariate stage. The structural variants are identified using UnifiedGenotyper, which is a locus-based GATK walker used for SNP and indel calling. A single MapReduce job is used for both TableRecalibration and UnifiedGenotyper stage to improve performance. In addition, the recalibrated BAM files from the TableRecalibration stage are written to the local filesystem (local FS), which provides 10- to 15-fold faster write throughput than HDFS (verified using Hadoop TestDFSIO benchmark). The UnifiedGenotyper processes recalibrated BAM files directly from the local FS. The recalibrated BAM files and variants (using CombineVariants walker) are finally merged. The variants are stored in a variant call format (.vcf) file, as discussed in the section 1.4. The variant files are very small compared to the input BAM file, and the size of the variant file completely depends on the number of genetic variants in the input dataset (Fig. 3.2). As a result, there cannot be a definite model to predict the size of the variants from the input dataset size.

Input BAM size (GB)	Output VCF size (MB)
6.25	179
12.5	533
25	604
50	754

Figure 3.2: Input BAM Size vs Output Variant File Size

The InputFormat and RecordReader for handling BGZF-compressed BAM files are used from

the Hadoop BAM [61] library. The RecordReader provided by Hadoop BAM is extended in SeqInCloud to define genomic intervals or loci for each GATK instance invoked by the Hadoop mapper process.

3.3 SeqInCloud Optimizations

In this section, we present several techniques aimed at optimizing the execution cost of SeqInCloud in cloud environments. This section discusses both the compute and data transfer optimizations implemented by SeqInCloud.

3.3.1 Compute Optimizations

Fine-Grained Partitioning

SeqInCloud partitions the dataset by loci corresponding to each MapReduce split rather than by contig. This ensures high scalability and well-balanced work distribution among mappers/reducers.

As shown in figure 3.3, the alignment stage is embarrassingly parallel and hence can be partitioned based on the number of cores/nodes. However, the intermediate stages can be partitioned only by chromosome/contig (a single GATK instance assigned to process a single contig/chromosome) to ensure functional correctness, with the final stage (Genotyper) being an exception. As a result, contig-based partitioning heavily relies on the distribution of reads across contigs in the input dataset. For example, if the reads are clustered to a particular contig, the mapper/reducer processing this contig runs for a longer duration. This creates an imbalance in the workload distribution and skews the overall execution time, which in turn leads to under-utilization of cluster resources. In addition, contig-based partitioning

imposes an upper bound on scalability because it cannot scale beyond the number of unique contigs in the input dataset, irrespective of the number of available cluster nodes.

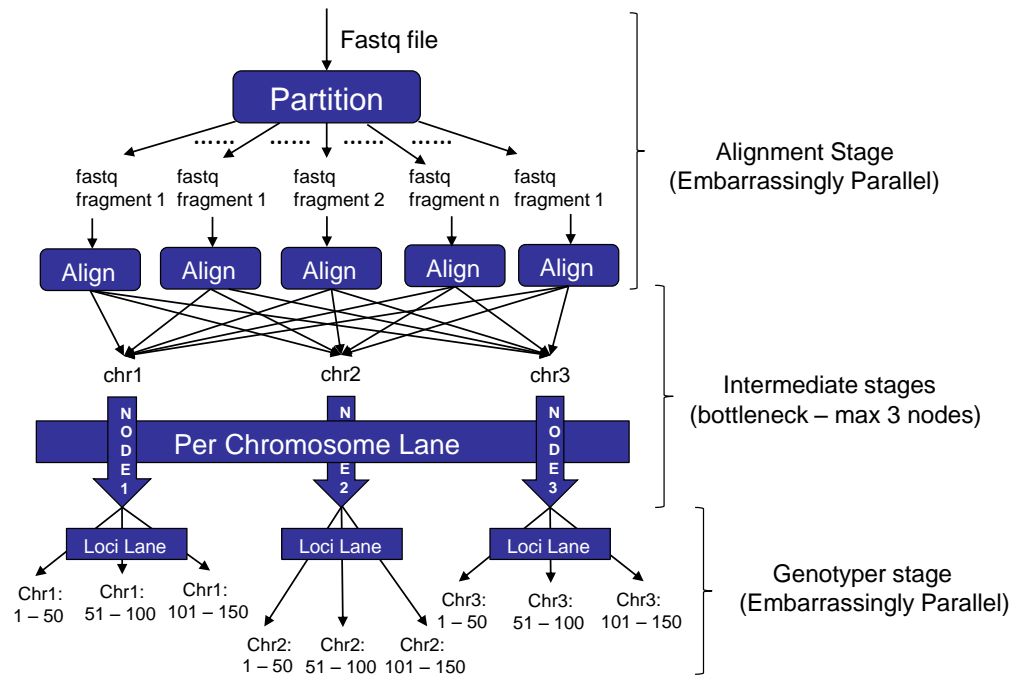


Figure 3.3: Contig-based Partitioning

In the case of data partitioning by loci, as shown in Fig. 3.4, the input dataset can be partitioned at a finer granularity than contig-based partitioning. This enables multiple GATK instances or mappers to process data belonging to the same contig/chromosome, which improves scalability of the pipeline and ensures equal distribution of work among the mappers. In addition, each partition is processed by a mapper process and corresponds to a MapReduce split for which the mapper has local access to. This reduces inter-node data accesses. After partitioning the input data by loci, multiple GATK instance can process each partition. Each instance further subdivides the genome interval corresponding to its partition into sub-intervals and processes them. However, implementing a genome analysis pipeline that is purely based on loci-based processing introduces performance bottlenecks to certain stages in the pipeline, which will be discussed later.

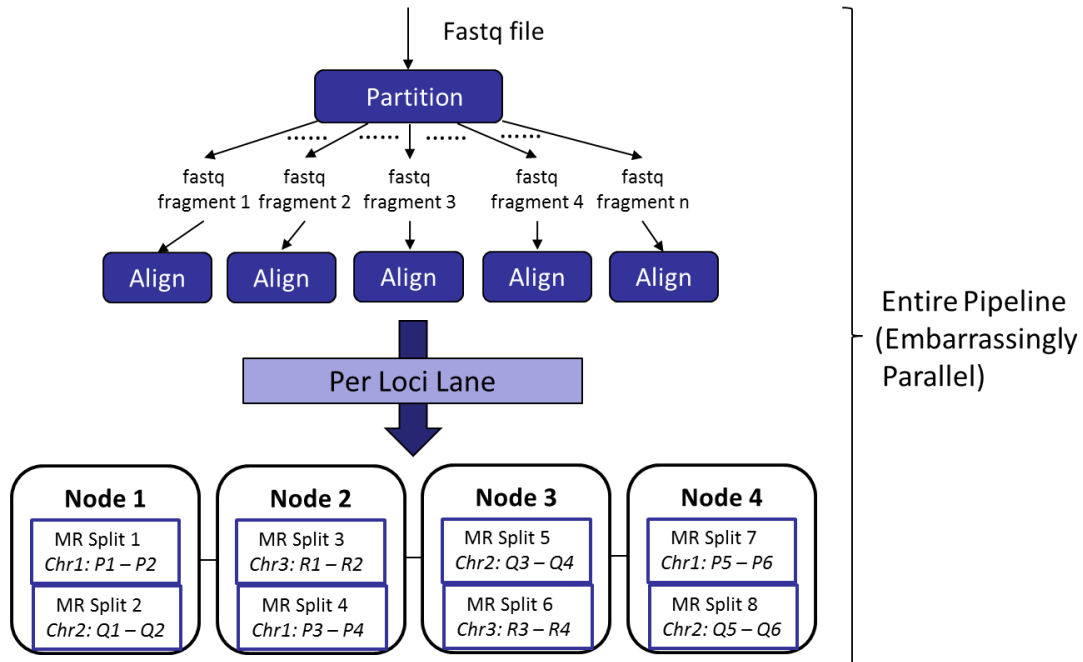


Figure 3.4: Loci-based Partitioning

Enabling loci-based partitioning and processing introduces functional incorrectness, as the reads belonging to the same contig/chromosome would be processed by different processes, as shown in Fig. 3.5.

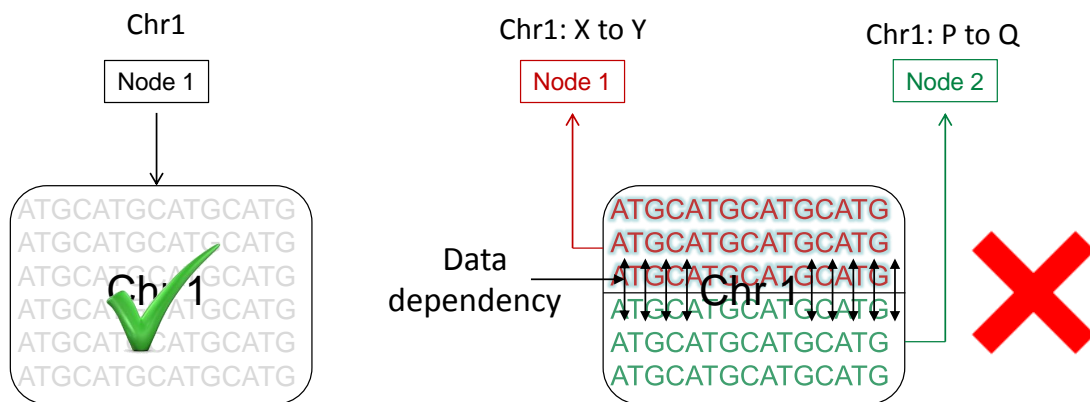


Figure 3.5: Loci-based Partitioning - The Problem

SeqInCloud provides the following mechanisms to enable loci-based processing for the Local Realignment and CountCovariate stages.

Local Realignment Stage:

Enabling loci-based partitioning for the local realignment stage results in the following problem: If a read is realigned, its new alignment location has to be updated in its mate pair and vice versa. This is not possible if realignment for a read and its mate pair is handled by different GATK instances, as it leads to incorrect results, shown in Fig. 3.6. In this figure, the read processed by “GATK Instance 2” is realigned from its original location 16050000 to 16050010. This new location is not updated in its mate pair as it is processed by another instance “GATK Instance 1”. The mate pair still points to the original location (16050000) of its mate. Due to this restriction, the maximum parallelism that can be achieved for the indel realignment step is equal to the number of contigs the input BAM file spans across. So, in a sample data set, if all the reads are aligned to a single contig (e.g., chr20), the realignment step cannot run in parallel using multiple GATK instances.

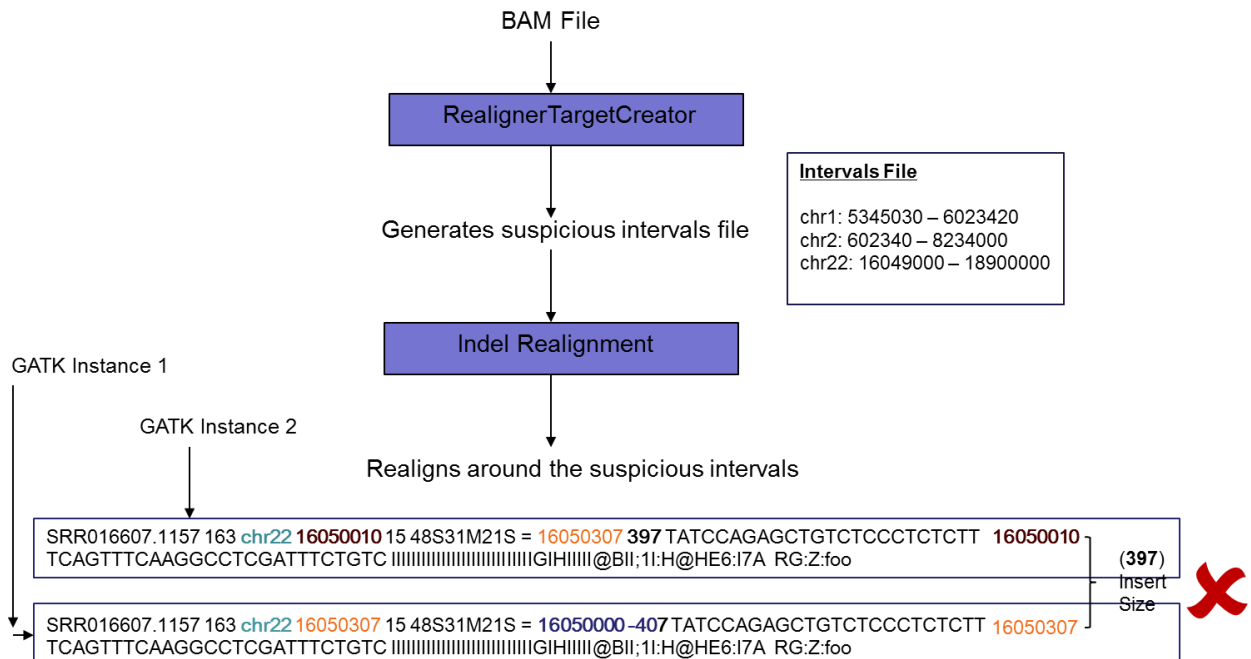


Figure 3.6: Local Realignment - The Problem

SeqInCloud introduces a novel and scalable solution that enables multiple GATK instances

to process read sequences from the same contig. This is achieved by using information on the *maximum insert size* between a read and its mate pair that GATK considers for realignment. GATK's IndelRealigner defines this as 3000 bases by default. Our solution, as shown in Fig. 3.7, adjusts the genomic interval provided as an input to each GATK instance, such that there is a window of maximum insert size base locations on either side of the actual interval the split spans across. For example, if a split spans across an actual interval of chr1: x-y, the adjusted interval would be chr1: (x-3000)-(y+3000), capped by the length of the contig. Invoking each instance of IndelRealigner in this fashion includes additional reads that provide the necessary mate information to realign reads in the actual interval. The reads in the dummy region are realigned and emitted as part of the MapReduce split they belong to.

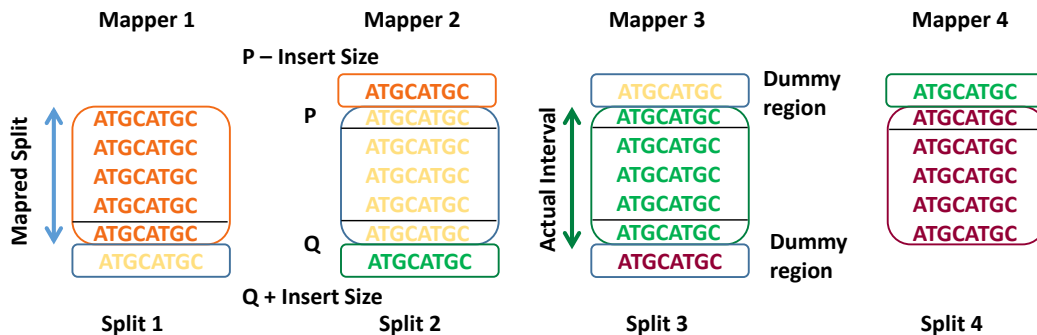


Figure 3.7: Design of IndelRealigner Stage

As shown in Fig. 3.8, the current design of SeqInCloud, also referred to as *SeqInCloud 1.0*, does not employ loci-based partitioning in the local realignment stage because this creates data-dependency between adjacent partitions while processing sequences that belong to the same contig. As a result, the sorted alignments from the previous stage are merged into a single BAM file, indexed and then fed as input to the local realignment stage.

This imposes a performance bottleneck, since the merge and index step is carried out by a single processing instance. The improved design, hereafter referred to as *SeqInCloud 2.0*,

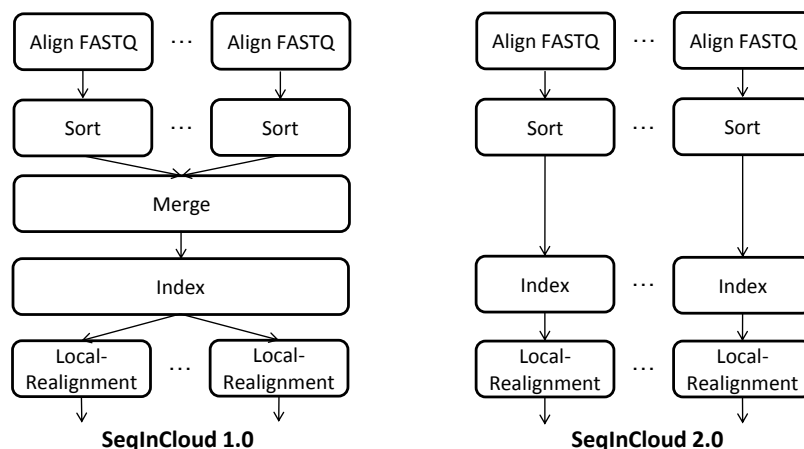


Figure 3.8: Mix of Contig and Loci-based Approach for Local Realignment Stage (SeqInCloud 1.0 vs SeqInCloud 2.0)

addresses this bottleneck by completely eliminating the merge step in the local realignment stage. It achieves this by using a combination of contig-based partitioning and loci-based processing approach. The sorted BAM records are partitioned by contig, indexed in parallel and then fed as input to the local realignment stage. The local realignment step can now process each contig independently using loci-based approach. This eliminates the merge step as well as utilizes multiple processing instance to index the BAM fragments in parallel improving the performance of the pipeline. The resulting optimized workflow, SeqInCloud 2.0 is depicted in Fig. 3.9.

CountCovariate Stage:

The CountCovariates walker from GATK mandates that a single contig/chromosome should be processed by a single GATK instance to ensure functional correctness. SeqInCloud addresses this issue by having the mapper processes partition at the loci level and communicate the covariate values to a single reducer process. The reducer aggregates identical covariates from all mappers and calculates a new empirical quality score using Phred scores.² In order

²Phred is the most widely used basecalling program due to its high base calling accuracy.

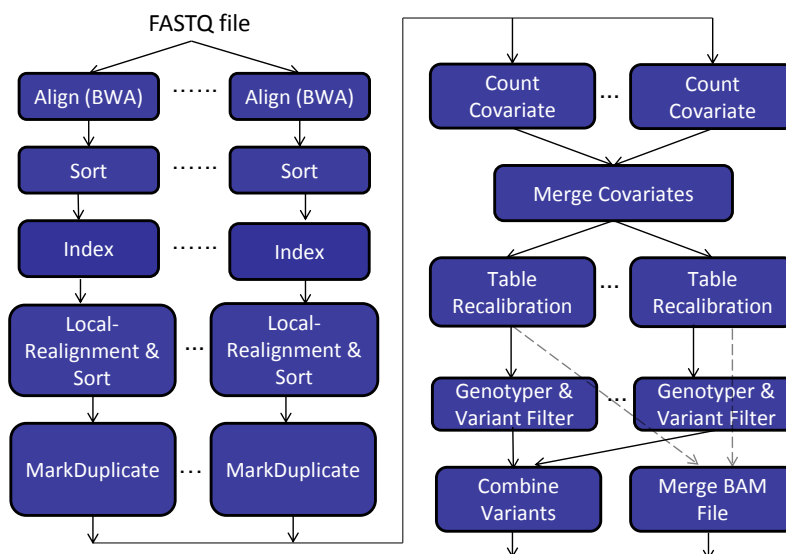


Figure 3.9: SeqInCloud 2.0: Optimized SeqInCloud Workflow

to reduce the load on a single reducer process, each node uses a combiner to perform a reduction on the map output, i.e., to locally aggregate identical covariates. The final covariate file from the reducer is used further for the recalibration of reads.

Storage Tiering

SeqInCloud uses different storage resources that are available in the HDInsight environment, such as Azure Blob, HDFS, and local filesystem. Blob is a Windows Azure storage service that stores unstructured data in a shared volume. Blob storage is both local- and geo-replicated for disaster recovery. The HDInsight service supports a new scheme `asv://` for MapReduce jobs to access data directly from blobs, similar to the standard `hdfs://` scheme used for accessing HDFS files.

To measure the read and write throughput of the local FS, Blob, and HDFS, we benchmarked the systems and, as expected, found that the local FS performed far better than the other two storage resources. Blob has higher write throughput than HDFS (3x), and HDFS has

higher read throughput than Blob (1.4x). In addition to HDFS, MapReduce can directly process the files that are available in blobs.

We have defined three storage mappings, which use different combinations of storage resources for input/output in the workflow. The “All HDFS” mapping uses only HDFS, the “All Blob” mapping uses blobs wherever possible, and the “Mix” mapping is structured as in Fig. 3.10. This is done so that the best-suited storage resource based on the requirement of each stage and throughput is chosen for input/output (I/O). For example, local FS cannot be used in places where the data needs to be persistent after the completion of a job. In this case, the blob is the preferred storage to store the final persistent output of the workflow due to its higher write throughput and durability (when compared to HDFS).

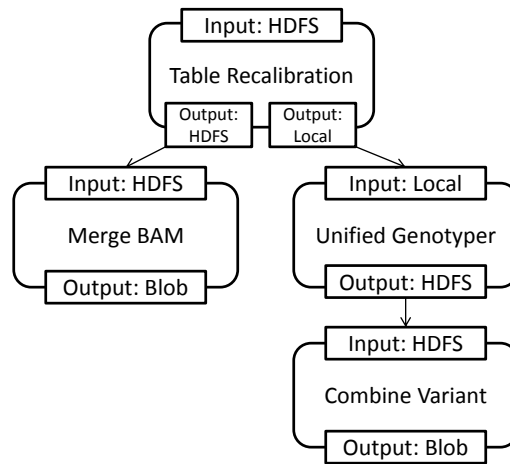


Figure 3.10: Feasible Input/Output Storage Resource for the “Mix” Mapping.

Results and Discussion

We have evaluated SeqInCloud on a 32-node Azure cluster, where each node is a medium Hadoop on Azure (HoA) [62] instance. The medium instance is provisioned as a virtual machine with two cores, 3.5 GB of RAM, and 500 GB of disk space. For the rest of the paper, we will refer to each VM as a compute node. The compute nodes run Windows Server

2008 R2 Enterprise and Hadoop 0.20.203. The MapReduce cluster is configured with 64 map slots and 32 reduce slots. All experiments were run with a default HDFS block size of 256 MB. We used the following datasets from 1000 Genomes Project [63] in our experiments for running the stages that follow the alignment stage: a 6-GB BAM file (NA12878) mapped to chr20 and an 11-GB (NA21143) and 30-GB (NA10847) BAM file mapped to an entire reference genome. The known variants database [64] used for count covariates stage is dbSNP_135.b37.

Baseline Performance vs. SeqInCloud Performance

SeqInCloud partitions the input data by loci for the entire workflow. This results in maximal utilization of cloud resources. Fig. 3.11 shows total execution time (in minutes) for local realignment, quality recalibration, and genotyper stages in the workflow, using contig- and loci-based partitioning. Contig-based partitioning serves as the baseline and uses local FS for input/output. In general, existing parallel GATK implementations use contig-based partitioning and rely on shared storage systems like Network File System (NFS) to access the input/output data. Due to the lack of shared storage in HoA cloud environment, we used the following procedure to obtain the baseline results. The entire BAM file and the reference genome were distributed to the local FS of all cluster nodes, and each node was dynamically assigned with a set of unique contigs. The baseline time corresponds to the parallel time taken by the nodes to complete the above specified stages for its assigned contig.

Both baseline and SeqInCloud measurements were taken using a 32-node HoA cluster. As discussed earlier, the baseline run mimics a traditional on-premises cluster environment, which typically uses either a shared network filesystem (NFS) or a local filesystem for data storage. In HoA environment, due to the non-availability of shared storage, the baseline run uses the local filesystem of each node in the cluster to store and access data, whereas SeqInCloud uses the Hadoop distributed filesystem (HDFS). Moreover, in the case of baseline

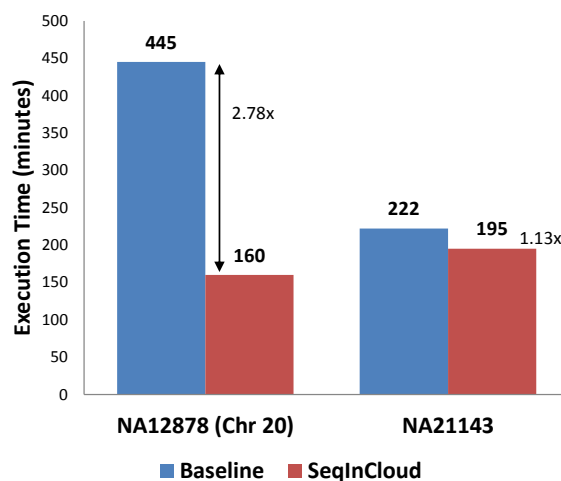


Figure 3.11: Comparison of Baseline and SeqInCloud Execution Time of Entire Workflow (Except Alignment Stage) for Datasets NA12878 and NA21143.

run, the single-contig NA12878 dataset can effectively utilize only a single cluster node for the Local Realignment, CountCovariate, and TableRecalibration stages. This is because of the functional requirement of contig-based or coarse-grained data partitioning, which mandates that a contig should be processed only by a single process or compute instance. As a result, the other 31 nodes in the cluster remain unutilized while running these three stages. For the rest of the stages, all the 32 nodes in the cluster were completely utilized. This affects the baseline workflow performance and accrues usage cost for idle resources. Since SeqInCloud uses fine-grained or loci-based data partitioning, it uses the entire 32 node HoA cluster to run all the stages in the pipeline. As a result, the run time of SeqInCloud is nearly 2.7-fold faster than the baseline run time for the single-contig NA12878 dataset. In the case of the 84-contig NA21143 dataset, where sequences are aligned to the entire reference genome, SeqInCloud is nearly 1.13-fold faster than the baseline. The performance improvement for NA21143 dataset is not as significant because the total number of map slots (64) is less than the number of contigs (84) in the input dataset. We would see an increasing improvement in performance as we keep increasing the number of map slots beyond 84, which is the baseline

upper bound on scalability for the 84-contig NA21143 dataset.

Evaluation of Scalability

We evaluate the strong-scaling behavior of SeqInCloud by doubling the number of virtual cores and measuring run time for a fixed workload size. We study scalability using the 24.3-GB NA10847 dataset (lossless compressed) and the 11-GB NA21143 dataset. The MapReduce split size was set to the HDFS block size of 256 MB. The number of virtual cores was varied between 8, 16, 32, and 64. Fig. 3.12 and Fig. 3.13 show the run time of the major time-consuming stages in the workflow, i.e., IndelRealigner, CountCovariate, TableRecalibration and UnifiedGenotyper. SeqInCloud exhibits near-linear scaling until 32 cores, after which the number of map waves becomes too small to observe much performance improvement.

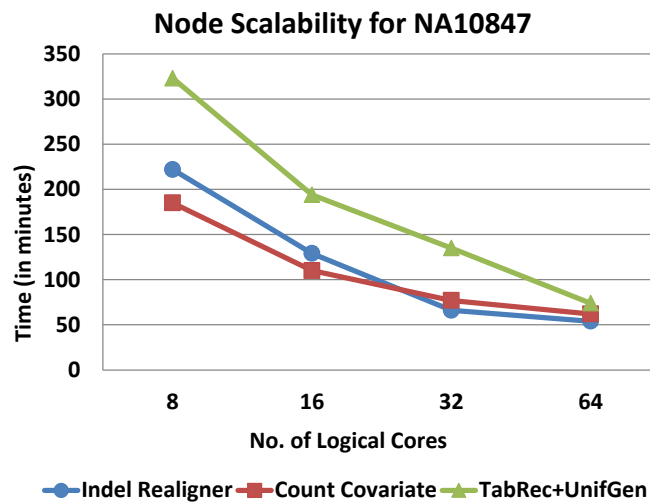


Figure 3.12: Execution Time of the Major Stages in SeqInCloud for the 24.3 GB NA10847 Dataset.

In SeqInCloud, strong scaling depends on two major factors:

- **Number of Map Waves**, which is given by the number of map tasks divided by the total number of map slots in the cluster. Due to the fixed workload requirement of

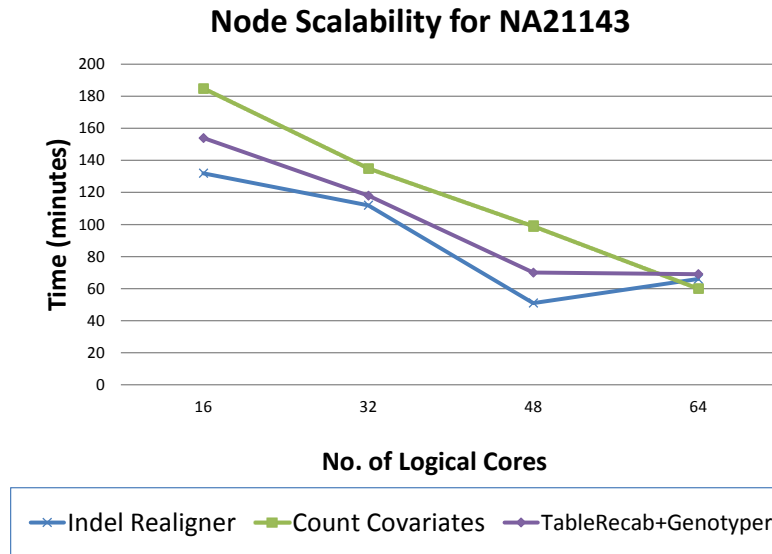


Figure 3.13: Execution Time of the Major Stages in SeqInCloud for the 11 GB NA21143 Dataset.

strong scaling, the number of map task remains the same, as we scale up/down the number of virtual cores. However, as we double the number of virtual cores, the number of map slots also doubles, and this halves the number of map waves. Since SeqInCloud does not depend on the nature of the input dataset and the map tasks almost run for the same duration, the number of map waves is one of the major components that determines scalability of SeqInCloud. From our strong-scaling numbers, we observe that doubling the number of virtual cores results in diminishing returns when the number of map waves becomes smaller (less than 3). This result serves as a guideline, as it enables one to know the maximum number of cluster nodes to be provisioned to ensure maximum resource utilization, and in turn, to optimize resource usage cost.

- **Number of Reducers**, which is set to 9/10 of the number of reduce slots in the cluster to have a single reduce wave. As we double the number of virtual cores, the number of reduce slots also doubles. However, due to the fixed workload size, the size of data that needs to be written by each reducer halves. Thus, the time taken by the

reduce phase halves when we double the cluster size.

Mixing the Contig- and Loci-based Approaches:

SeqInCloud 2.0 employs a mix of contig- and loci-based approach in the local realignment stage. This removes the performance bottlenecks along the pipeline. The newer version runs on the latest Microsoft HDInsight service platform, which offers better network and storage performance than the Microsoft Hadoop on Azure (HoA) platform. All the above contributes to a performance improvement of 16% to 36% for different node configurations and improved scalability as compared to SeqInCloud 1.0.

Here we discuss the results of employing a mix of contig- and loci-based approach for the local realignment stage.

The experiments for this optimization were conducted on the Microsoft Azure HDInsight service cluster, where each node is a large virtual machine instance configured with 4 cores, 7 GB of RAM and 1 TB of local disk space. The compute nodes are installed with Windows Server 2008 R2 Enterprise and Hadoop 1.1.0. The MapReduce cluster is configured with a HDFS block size of 256 MB. The experiments use a single 50-GB dataset (NA19066) and two supporting files: the known sites database (dbsnp_135.b37) and the reference genome (human).

Performance of SeqInCloud 1.0 vs SeqInCloud 2.0:

Fig. 3.14 compares the workflow execution time of SeqInCloud 1.0 and SeqInCloud 2.0 for all stages excluding the alignment stage in the pipeline, using base versus mix approach. The performance improvement varies between 16% and 36% due to the following reasons: The local realignment stage is optimized such that the merge step is eliminated and the index step is run in parallel. In addition, controlling the number of reducers in the local realignment stage optimizes the number of parallel processing instances required for running

the Mark-Duplicates stage. All these improvements are shown in the table that is embedded in Figure 3.14. The table provides a breakdown of the execution time (in hours) for the stages that are of interest.

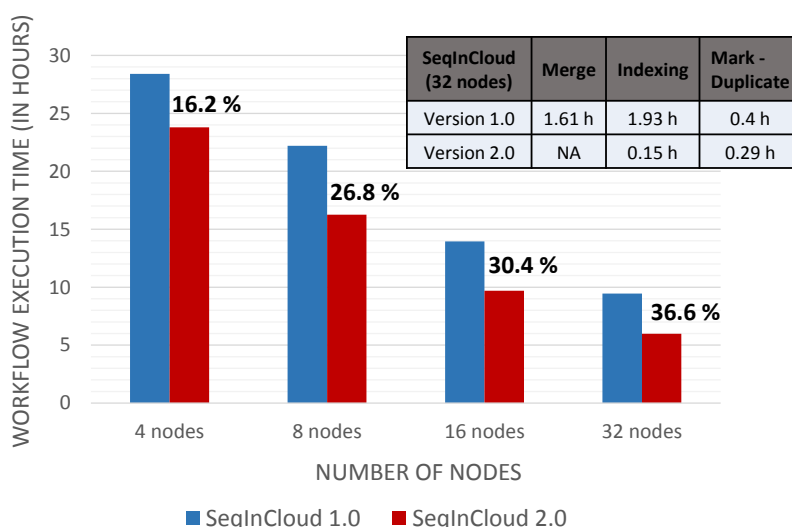


Figure 3.14: Performance Comparison of SeqInCloud 1.0 vs SeqInCloud 2.0

Scalability of SeqInCloud 2.0: We evaluated strong scaling by varying the number of nodes (4, 8, 16 and 32) and measuring the execution time of all stages except the alignment stage in the pipeline. SeqInCloud 2.0 exhibits near-linear scaling as shown in Fig. 3.15. The graph also plots a best-fit linear trend line with an R-squared value of 0.9697, which is a good fit as the R-squared value is much closer to 1. In addition, we can observe that the scalability deviates a little from the linear trend line at the data point corresponding to 32 nodes. This is because of the same reason described earlier, which is the dataset size was not sufficient enough to utilize all the cores of a 32-node cluster, i.e., the number of map waves is only around 1.5 for processing a 50 GB dataset using 32 nodes and 256 MB HDFS block size.

Evaluation of Performance Due to Storage Tiering

We evaluated the performance of SeqInCloud using different combinations of storage re-

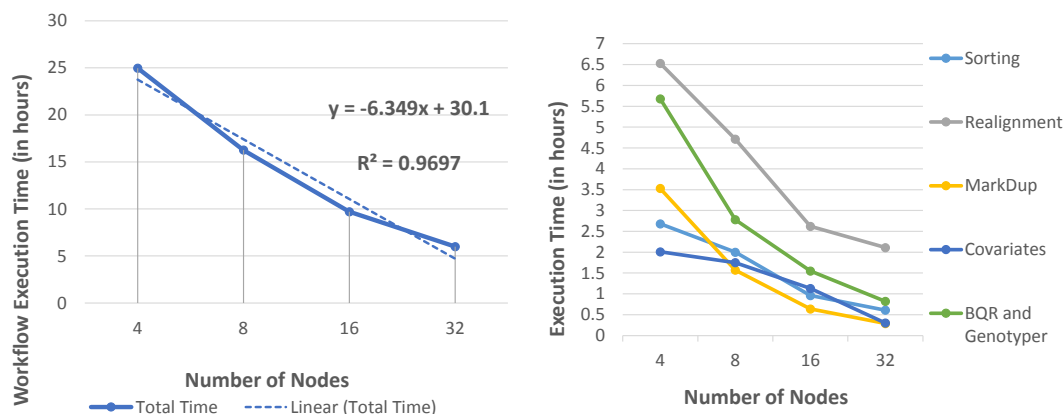


Figure 3.15: Execution Time of the Workflow and the Individual Stages for NA19066 Dataset with Increasing Number of Nodes.

sources to identify the right mix that delivers the best performance. The results correspond to the execution time of SeqInCloud from TableRecalibration until the final merge stage and compares the “All blob” and “Mix” mappings with the “All HDFS” mapping. The improved runtime of “Mix” or “All Blob” mapping in Fig. 3.16 is due to the higher write throughput of the blob/local filesystem. For the TableRecalibration and UnifiedGenotyper stages, the “All Blob” mapping showed an improvement of 20%. For the Merge Variant stage, “Mix” and “All Blob” mappings showed an improvement of 29%. For the Merge BAM stage, “Mix” mapping showed an improvement of 26.4%. Finally, the overall run time of “All Blob” mapping is better than the other two mappings. “All Blob” showed a performance improvement of 20% and “Mix” showed a performance improvement of 19% over “All HDFS” mapping.

3.3.2 Data Transfer Optimizations

Reference-based Compression

SeqInCloud uses compression to optimize network and storage costs in the cloud. It uses the CRAM [52] format, which is a reference-based compression mechanism that encodes

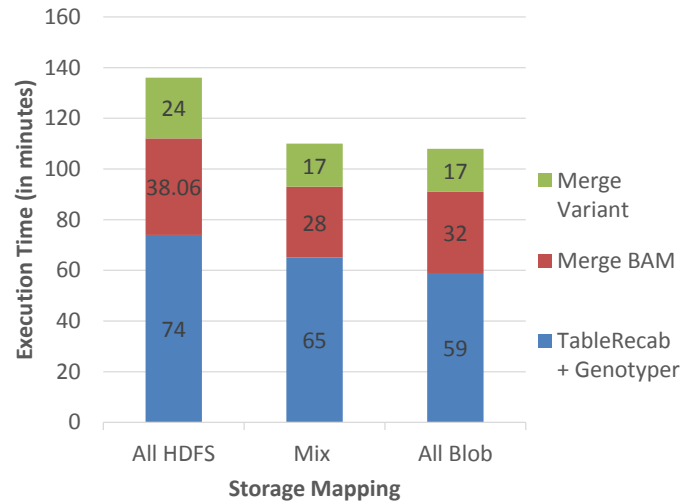


Figure 3.16: Execution Time of “All HDFS”, “Mix” and “All Blob” Mappings.

and stores only the difference between a read sequence and reference genome. The CRAM toolkit [65], offered by the European Nucleotide Archive, contains tools and interfaces that provide programmatic access for compression/decompression. In order to ensure sensitivity and correctness of downstream analysis, SeqInCloud uses lossless compression by preserving quality scores but excluding unaligned reads as well as read names and tags from each BAM record.

The reference-based compression is implemented as shown in Fig. 3.17. After aligning reads in parallel, each reducer writes its BAM file to HDFS. This is followed by a parallel sort of the reads using the TotalOrderPartitioner interface provided by the MapReduce framework. The sorted BAM records are converted to CRAM format by multiple reducers in parallel using the CRAM toolkit. The CRAM files are then transferred to the cloud using the secure file transfer service provided by the HoA framework. These CRAM files are typically 40% to 50% smaller than the BAM files, thus significantly reducing network traffic and costs. Once the data transfer is completed, a remote MapReduce job is triggered, which uses multiple mappers to decompress CRAM records to BAM records in parallel. The decompression

results in a lossless BAM file, which is smaller than the original BAM file, thus reducing storage costs.

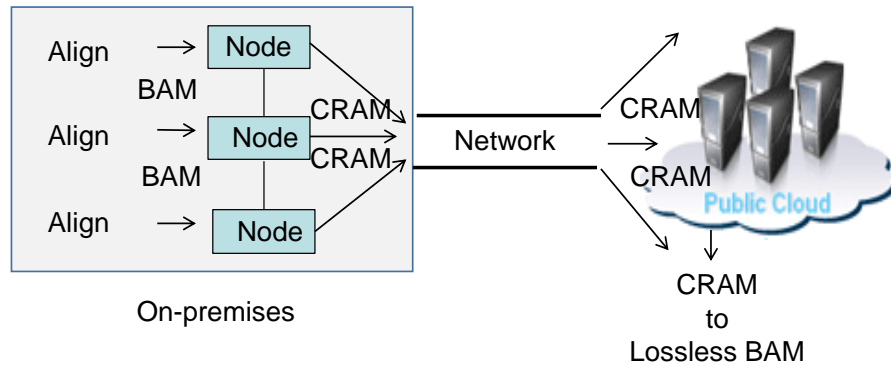


Figure 3.17: Reference-based Compression

From the above, compression is applicable under two scenarios: (1) when the sequence alignment stage is carried out using on-premise Hadoop resources and the BAM file needs to be transferred to the cloud and (2) when the final workflow result (i.e., the merged BAM file) needs to be persistently stored in the Blob. For the latter, instead of storing data in BAM format, it can be stored either in CRAM or lossless BAM format, thus bringing down the storage cost considerably.

It is worth noting that GATK 2.0 has introduced a new walker *ReduceReads*, which performs a lossy compression of the NGS BAM file and reduces its size by 100-fold. The reduced BAM file has just the information necessary to make accurate SNP and indel calls using the UnifiedGenotyper walker. Using the CRAM format for compression has broader applicability than GATK *ReduceReads*, as the lossless BAM file can be used by other downstream analysis tools. In addition, *ReduceReads* compression takes much longer than CRAM compression. For instance, for a fragment of the NA12878 dataset of size 754 MB, the *ReduceReads* compression took 112 minutes vs. 10 minutes for the CRAM compression.

Acceleration Using the GPU

SeqInCloud uses compression to reduce data-transfer time from on-premise resources to the cloud and improve overall performance. We use CUSHAW [66], a parallel GPU-based, short-read aligner to realize our proposed client-plus-cloud model. CUSHAW is a CUDA-compatible, short-read aligner based on Burrows-Wheeler transform (BWT). CUSHAW provides support to either produce both aligned and unaligned reads to a SAM file or produce aligned reads to a SAM file and unaligned reads to a FASTQ file. For single-end alignment, we set the maximum number of mismatches in the seed and full length to 0. This ensures that reads that perfectly match the reference will be part of the SAM file and rest of the reads will be part of the FASTQ file. The FASTQ file containing the unaligned reads will be later aligned in the cloud.

As shown in Fig. 3.18, we use both the client and cloud simultaneously such that there is an overlap of communication and computation between client and cloud. The unaligned reads in the FASTQ file will be aligned using BWA in the cloud. The reads aligned using CUSHAW in the client environment will be transferred to the cloud and sorted and merged along with the other reads aligned by BWA in the cloud environment. This creates an opportunity to overlap the data-transfer and alignment of imperfectly matched reads in the cloud with the processing of perfectly matched reads and its transfer to the cloud. From our experiments, we observed that we can overlap nearly 20% to 35% of the maximum time, the maximum time being the time spent for the data-transfer and alignment of reads in the cloud. Also, CUSHAW is able to perfectly align nearly 55% to 65% of the total number of reads.

Results and Discussion

Evaluation of Reference-based Compression

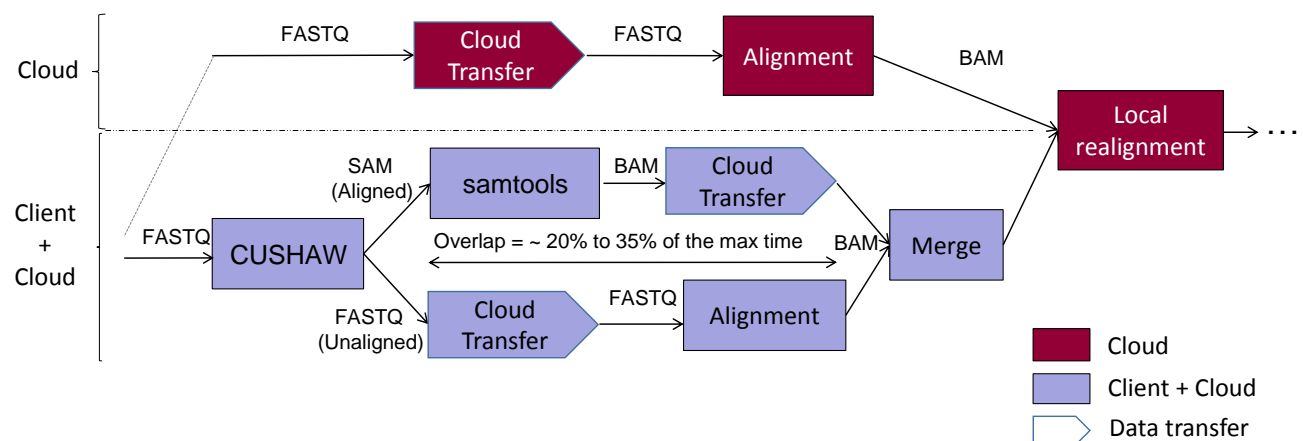


Figure 3.18: Acceleration Using GPU

We evaluated the cost savings due to compression on a 14-node on-premise Hadoop cluster, where each node consisted of two quad-core Intel Xeon E5462 processors with 8 GB of RAM. The dataset used for evaluation is presented in the Fig. 3.19.

Dataset	Size	Mapped to
NA12878	6 GB	Chromosome 20
NA21143	11 GB	Human genome
NA10847	30 GB	Human genome
ERR001268	1.7 GB	Human genome

Figure 3.19: Dataset Used for Evaluation

The sequence alignment and sorting stage in the workflow were carried out using these on-premise resources. As discussed earlier, using the CRAM format instead of BAM reduces the amount of data transferred to the cloud by 40% to 50%. However, this improvement in the data-transfer time comes with an additional overhead of compression from BAM to CRAM on-premise and decompression from CRAM to lossless BAM in the cloud.³ This overhead should be considered while evaluating the impact on workflow performance when using the CRAM format instead of the BAM format. Here, the workflow performance refers

³The compression and decompression is achieved using interfaces from the CRAM toolkit.

to the time taken to run the workflow until the alignment stage, including the data transfer, compression and decompression time, if any. The workflow performance is said to break-even when the performance using BAM format is equal to the performance using CRAM format, as shown in Fig. 3.20.

Network transfer time while using BAM format = t_{bam}

Network transfer time while using CRAM format = t_{cram}

$$\Delta t = t_{\text{bam}} - t_{\text{cram}}$$

Compression time = t_c

Decompression time = t_d

Workflow performance “break-even” when, $t_c + t_d = \Delta t$

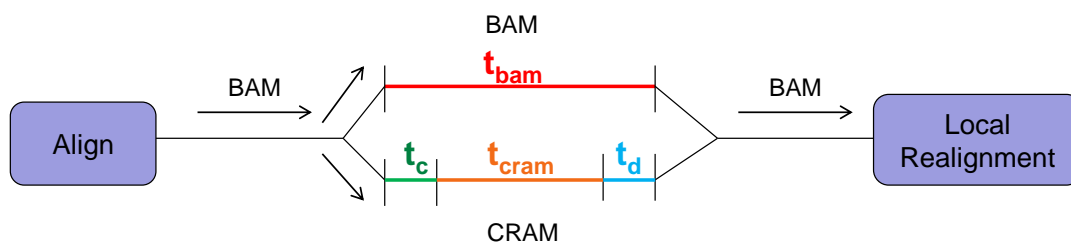


Figure 3.20: Breakeven Point for Reference-based Compression

While using the CRAM format, the workflow performance reaches break-even when the sum of compression and decompression time equals the delta improvement in the data-transfer time. At break-even, we only observe storage savings without any impact on workflow performance. The storage savings correspond to the percentage reduction in the size of the lossless BAM file when compared with the original BAM file. For the datasets used in our experiments, we observed break-even when using four to six on-premise nodes. When the number of on-premise nodes was greater than the number of nodes used for achieving break-even, we achieved an improvement in the workflow performance. Conversely, when the number of on-premise nodes was less, we observed a dip in the workflow performance.

Factor	NA10847	NA21143	NA12878
Performance	34.5 %	21 %	23 %
Storage savings	20.3 %	16.3 %	43 %

% improvement in workflow performance and % storage savings

Figure 3.21: Performance Improvement and Storage Savings for NA10847, NA21143 and NA12878 Datasets Due to Compression Using a 14-node On-premise Hadoop Cluster.

Fig. 3.21 shows the improvement in workflow performance and storage savings when using the CRAM format instead of the BAM format. Here the number of on-premise nodes (14) is greater than the break-even number of nodes (4-6). The performance improvement varies across datasets, as the efficacy of compression while using reference-based compression mechanisms like CRAM, largely depends on the nature of alignments in the input BAM file. The nature of alignments refers to factors like the number of perfectly aligned reads, the length of read names, the number of tags, the number of unaligned reads, and so on. This determines the improvement in workflow performance, as it influences the compression, decompression and data transfer times. The decompression at the cloud results in a lossless BAM file, which has trimmed read names and does not contain tags, unaligned reads etc., when compared with the original BAM file. As a result, the storage savings also varies across datasets.

Evaluation of GPU Acceleration Using CUSHAW

We have evaluated the baseline performance, where the FASTQ files are completely transferred to the cloud and aligned versus the performance of using GPU for accelerating the alignment stage. The client machine consists of a single quad-core Intel Core i5-2400 with 8 GB RAM and 1 Tesla C2050 GPU, and for the cloud environment, we used a local cluster comprised of a five-node cluster with a pair of quad-core AMD Opteron processors and 32

GB RAM.

Fig. 3.22 shows the effect of using GPU acceleration for theoretically varying network bandwidth between the client and cloud environment. The result shows that as we keep increasing the network bandwidth, the data transfer time between the client and cloud environment becomes negligible. As a result, the difference in execution time between the baseline and GPU acceleration is determined solely by the difference in the alignment time, i.e., for aligning the complete FASTQ file vs pre-processing and aligning a fragment of the FASTQ file. Here the execution time refers to the sum of data transfer and alignment time.

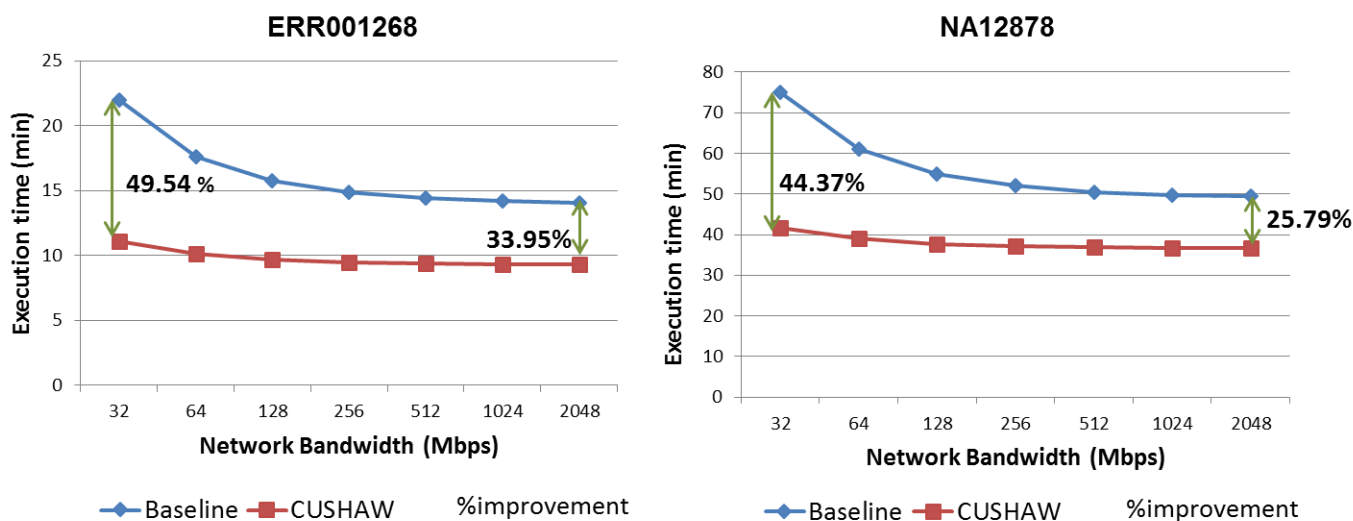


Figure 3.22: Evaluation of GPU Acceleration Using CUSHAW

3.4 Applicability of SeqInCloud to Other Cloud Platforms

SeqInCloud is a pure MapReduce-based application written in Java. It is referred as a pure MapReduce application as it is completely platform-independent and does not use any

Hadoop streaming-based components in its parallel implementation. Hadoop streaming-based parallel implementations are typically platform-dependent as it enables one to run any platform-specific executable/binary in parallel using Hadoop. As a result, the base SeqInCloud implementation, without any optimizations, can run on any on-premises or IaaS / PaaS cloud environments like Amazon EMR, Amazon EC2, Rackspace cloud servers, and Google compute engine.

Here, we discuss the applicability of various SeqInCloud optimizations across different cloud environments.

Compute Optimizations: Since *fine-grained partitioning* is only a data decomposition and processing technique and does not use any cloud provider-specific solution, this optimization is applicable to any cloud environment. *Storage tiering* evaluates the mapping of input, intermediate, and output data across different storage resources. We use the local file-system, HDFS, and Azure blob in the Microsoft HDInsight cluster. The physical storage for HDFS comes from the local disks of the cluster. The local file-system is also from the local disks, but is not managed by HDFS. It is used by the MapReduce jobs to stage temporary data and is not valid across multiple MapReduce jobs. Both local FS and HDFS are usually available in all cloud provider environments. The third storage resource, Azure blob is a cloud storage offering from Microsoft. Other cloud providers typically have their own cloud storage offering. For example, Amazon offers the Simple Storage Service (S3). In a MapReduce job, the storage URI has an access scheme which delineates the backing store for HDFS, for e.g. `hdfs://` uses local disks, `asv://` uses Azure blob and `s3://` uses Amazon's storage service. Functionally, we have not tested if our implementation can work with a S3 URI in Amazon's EMR. The performance numbers from SeqInCloud will also be different across different cloud provider's storage offerings.

Data-Transfer Optimizations: *Reference-based compression* is achieved using the CRAM

toolkit which offers interfaces for programmatic compression/decompression. Since the CRAM toolkit is written in Java, it is platform-independent and can run “as is” on any on-premise resource plus cloud environment. The parallel compression/decompression achieved using multiple cluster nodes is also implemented as a MapReduce application and hence platform-independent, too. *Acceleration using the GPU* requires running a GPU-based aligner, e.g., CUSHAW using the on-premises environment. CUSHAW is platform-dependent and can run only in a Linux environment with a GPU. On the other hand, on the cloud side, we use only the basic SeqInCloud implementation. Hence this optimization will work for all cloud provider environments provided the on-premises environment is equipped with a single Linux-based system that has NVIDIA CUDA-enabled GPUs based on Fermi architecture or newer.

Chapter 4

CloudFlow: Distributed Workflow Manager for the Cloud

CloudFlow is a distributed workflow management system for MapReduce clusters. CloudFlow is built on top of an existing solution, Cloudfone, which enables users to compose, run, and monitor MapReduce pipelines, using either a private or public cloud environment. Cloudfone already provides an easy-to-use interface to compose and monitor flexible MapReduce workflows, and it supports normal MapReduce jobs, streaming jobs, and MapReduce pipelines.

CloudFlow adds many interesting features on top of Cloudfone, like directed acyclic graph (DAG) support, simultaneous use of both client and cloud resources (hybrid cloud), on-demand automated data transfer between client and cloud resources to satisfy data-dependency for MapReduce jobs, on-demand cloud provisioning, and plugin support for users to implement their own byte-level or context-based compression plugins.

4.1 Cloudfuge Overview

Cloudfuge provides a graphical execution platform to improve the usability of MapReduce programs. The MapReduce programs can be run either using private in-house clusters or public clouds. Cloudfuge consists of two independent modules: *Cloudfuge-Cluster* and *Cloudfuge-Mapred*, as shown in Fig. 4.1. Cloudfuge-Cluster provides the functionality to instantiate a cluster using public cloud resources, i.e., Amazon EC2. This requires the user to provide the cluster size, credentials, and other associated information. Cloudfuge-Mapred gets installed automatically on the head node of the provisioned cluster. Cloudfuge-Mapred provides the graphical interface to compose and monitor MapReduce jobs or pipelines. MapReduce jobs or pipelines should be first defined using the manifest file provided by the Cloudfuge framework. Once the manifest file is defined, Cloudfuge-Mapred presents the user with an auto-generated GUI to configure MapReduce jobs or pipelines. Cloudfuge provides support for running normal MapReduce jobs, streaming jobs, and MapReduce workflows.

Cloudfuge is implemented as a client-server framework. The client is designed using the JavaScript framework, *Sencha ExtJS*. The server is designed using the RESTful web framework, *Restlet*. The client and server communicate using *JSON*, a platform-independent interchange format. The manifest file is defined using a popular name-value format file, *Yet Another Markup Language (YAML)*. The user defines parameters in the manifest file, as shown in Fig. 4.2, which is then processed by the client framework and presented in an interactive graphical wizard for the users to key-in the input values.

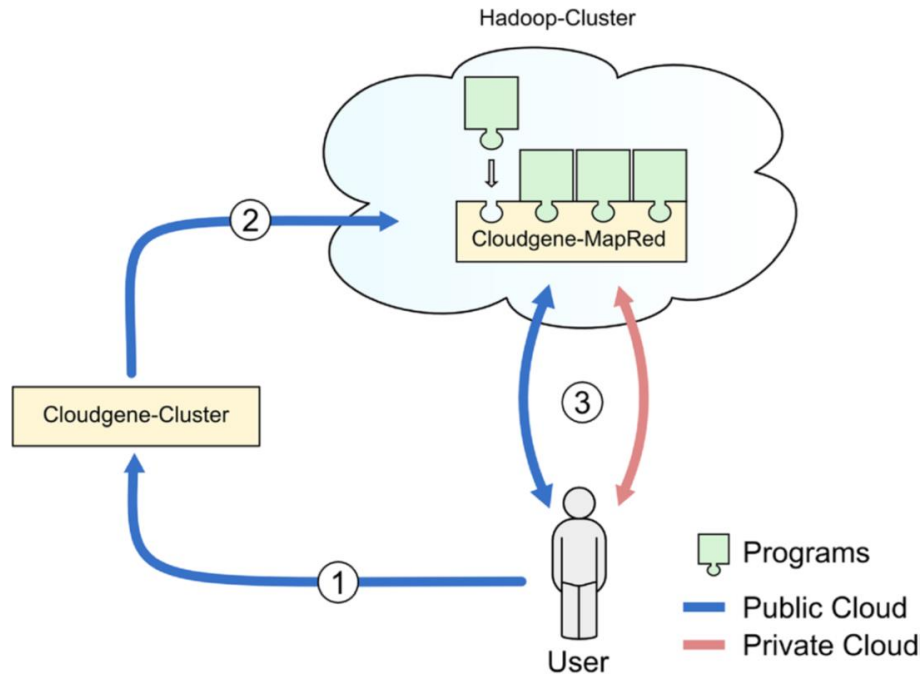


Figure 4.1: Cloudgene Modules

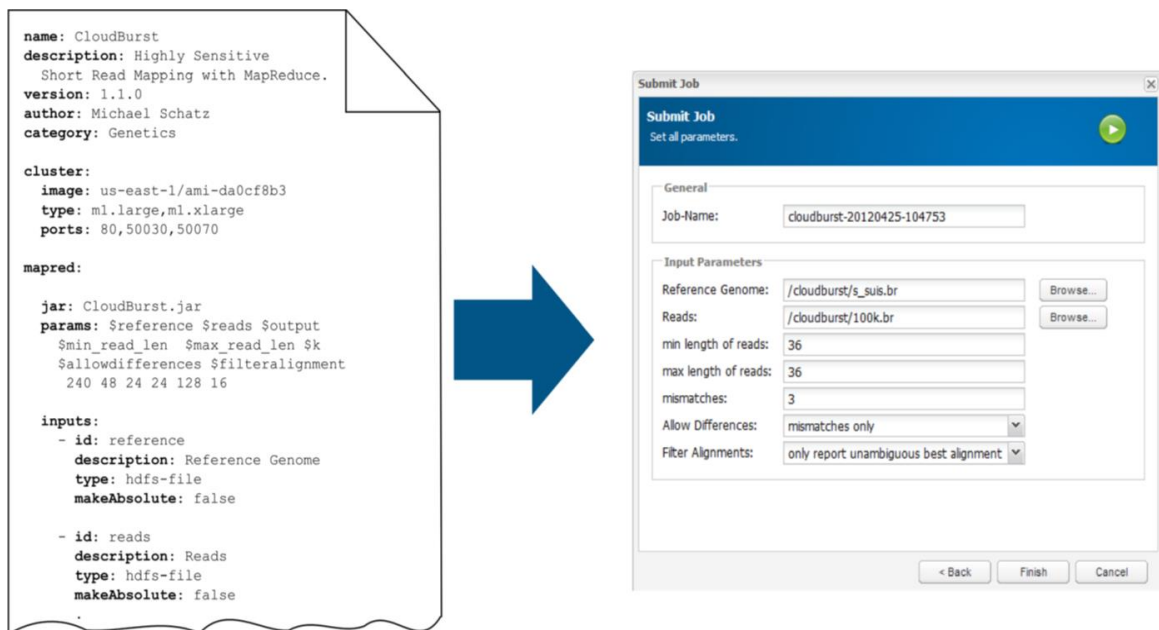


Figure 4.2: Cloudgene Manifest File

4.2 Design and Implementation

CloudFlow is built on top of Cloudfuge and provides additional functionality, few of which the other existing solutions provide and the rest offered uniquely by CloudFlow. The major goal of CloudFlow is to provide a workflow manager that efficiently utilizes a hybrid client-plus-cloud environment for running MapReduce-based pipelines. Here, we first discuss the existing design of Cloudfuge and the design of new features introduced by CloudFlow.

4.2.1 Cloudfuge Architecture

As specified in the earlier sections, Cloudfuge is implemented as a client-server framework. Cloudfuge-mapred server is implemented using Restlet, a RESTful web framework. Cloudfuge-mapred server runs either on the head node of a private on-premises cluster or the head node of a public rented cloud cluster. The user configures the MapReduce job or pipeline using a manifest file. The Cloudfuge client parses the manifest file and dynamically creates a graphical wizard for the user to configure MapReduce job or pipeline. Once the user provides the input values and submits the job, the job parameters and their corresponding values are communicated from the client (browser) to the Cloudfuge server in JSON format.

In this section, the focus will be on the design of Cloudfuge's job submission module, as it is a necessary prerequisite to understand the design of CloudFlow. The job inputs submitted from the web client are sent to the Cloudfuge-mapred server. Cloudfuge-mapred's server initialization code creates a *job queue*, where the jobs submitted by users are enqueued in order, as specified in Fig. 4.3. The enqueued job could be a simple MapReduce job or a MapReduce pipeline comprising of multiple MapReduce jobs or stages. A *job-queue thread* repeatedly polls the job queue for the arrival of new jobs. As soon as a new job arrives, the job-queue thread spawns a *worker thread* and associates the worker thread with the newly

arrived job. The worker thread dequeues the job and submits it for execution using the available cluster resources. It is evident from this design that the execution of different jobs are independent and can be processed in parallel by different worker threads. The worker thread writes the job status periodically to a database, which the Cloudgene server queries during the monitoring of the submitted jobs from the client interface.

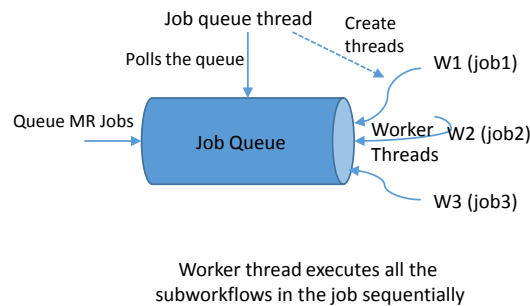


Figure 4.3: Cloudgene Job Submission Architecture

The above design has the following major shortcomings:

- Cloudgene can be configured to use either the cloud or client resource, but not both simultaneously. In a hybrid environment, where the users have in-house clusters in addition to the publicly rented cloud resources, Cloudgene cannot utilize both simultaneously for optimized execution of MapReduce pipelines.
- Cloudgene does not support DAG-based MapReduce pipelines. With the current design, the execution of the MapReduce pipeline is always sequential.
- It is the user's responsibility to ensure that the data dependencies are met before executing various stages within a MapReduce pipeline.

4.2.2 CloudFlow Architecture

CloudFlow is currently designed to work with Microsoft Azure cloud. Its design is modular such that it is easy to integrate CloudFlow with any other cloud providers, if desired. CloudFlow has the minimal requirement of having a single-node, on-premise Hadoop configuration. The modified Cloudfuge-mapred server, hereafter referred to as CloudFlow-mapred server, runs in the head node of the on-premise cluster and acts as a driver for the workflow management system. It controls the provisioning of cloud resources, processes workflow DAGs, spawns local/remote MapReduce jobs and automatically transfers data between the client and cloud resources to ensure data dependency between different stages of the pipeline. Similar to Cloudfuge, the user configures the workflow in a manifest file. The client component generates graphical wizards and takes input from the user. The input is communicated to the CloudFlow-mapred server running in the local on-premise cluster. The Cloudfuge-Mapred server then drives the pipeline execution towards completion. A high-level overview of CloudFlow architecture is shown in Fig. 4.4.

4.2.3 CloudFlow Features

With the existing design of Cloudfuge, the execution of all stages in a MapReduce pipeline is mandatory. Moreover, all the stages execute either on a private cluster or in a public cloud. The execution of the pipeline is sequential in nature and does not support any kind of parallelism. The input dataset for a stage could either be the output of its previous stage or should be made available by the user in an appropriate location.

CloudFlow addresses the above shortcomings by providing users with the flexibility to enable/disable different stages in a MapReduce pipeline at runtime. In addition, it enables users to specify the execution location of each stage in the pipeline, either the client or cloud

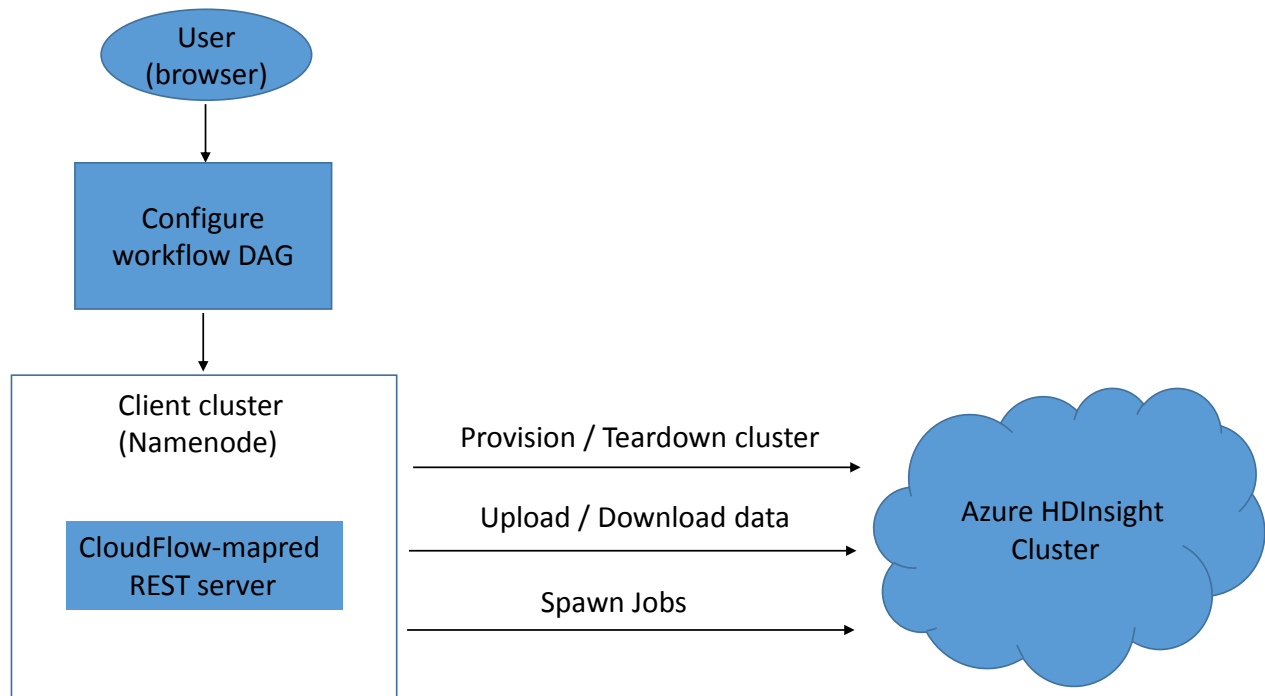


Figure 4.4: CloudFlow - High Level Architecture

resources. This is shown clearly in the Fig. 4.5, where a check box is available near each stage in the user interface to enable/disable its execution. The “Where to run” field collects the desired execution location of each stage in the pipeline. The users choose either client or cloud resource based on availability of the resource and security requirement of the dataset.

The below subsections discuss the different features offered by CloudFlow.

Directed Acyclic Graph (DAG) Support

CloudFlow supports DAG-based MapReduce pipelines. In order to support DAG-based workflows, the users should be capable of specifying the dependency information between different stages within a MapReduce pipeline. This support is implemented by using the following extensions to the manifest file, as shown in Fig. 4.6.

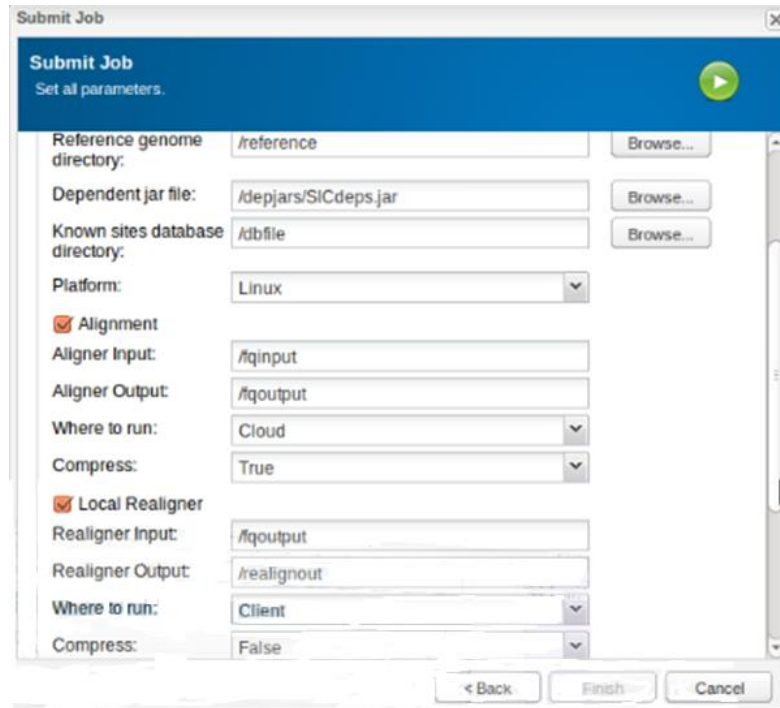


Figure 4.5: CloudFlow User Interface

1. The *'alias'* parameter, which is used to associate each stage in the pipeline with a name. The dependency information can be specified using the alias name of each stage.
2. The *'dependency'* parameter, which specifies the control and data dependency information of each stage in the pipeline, if applicable. The value of this parameter specifies a single or a list of alias, which corresponds to other stages in the pipeline that needs to complete before this stage can start its execution.

Once the user submits a job, which corresponds to a MapReduce pipeline, the CloudFlow implementation parses the dependency information for each stage in the pipeline and constructs a directed acyclic dependency graph. The vertex corresponds to a stage in the pipeline and the edges specify the dependency between stages. Only those stages that are enabled by the

```

- name: Table Recalibration
  params: -nfv -nm -nmd -na -nra -nv -ref $reference -b $s1input -o $s1output
         -djarloc $dpjarloc -p $s3platform -dbfile $dbfile
  alias: s1
  dependency: s0 } DAG Support

- name: MergeResults
  params: -nqr -nfv -nmd -na -nra -nv -ref $reference -b $s1output -o $finoutput
         -djarloc $dpjarloc -p $s3platform
  alias: s2
  dependency: s1

```

Figure 4.6: CloudFlow DAG Configuration

user at runtime are part of this graph. The dependency graph is constructed by the server thread as soon as the user submits a new job. This job is then enqueued into the job queue. In the traditional Cloudfuge architecture, a worker thread dequeues the job and runs the pipeline in a sequential fashion. In CloudFlow, the worker thread instead queries a *DAG processor* for the next set of candidate stages to run. The DAG processor processes the constructed dependency graph and returns the set of vertices or stages that has an in-degree of zero. An in-degree of zero implies that all data and control dependencies are met for that stage. These stages can then run in parallel using both the client and cloud resources simultaneously. Once a particular stage in the pipeline completes, the corresponding vertex is removed from the dependency graph and the worker thread immediately queries the DAG processor for the next set of eligible stages to run.

For example, in Fig. 4.7, the stage ‘s1’ has an in-degree of zero. The candidate set for the first query would be ‘s1’. After ‘s1’ completes, the worker thread queries the DAG processor, which returns ‘s2’ and ‘s3’. Now, the candidate set is {s2, s3}. After ‘s2’ and ‘s3’ completes, ‘s4’ is executed.

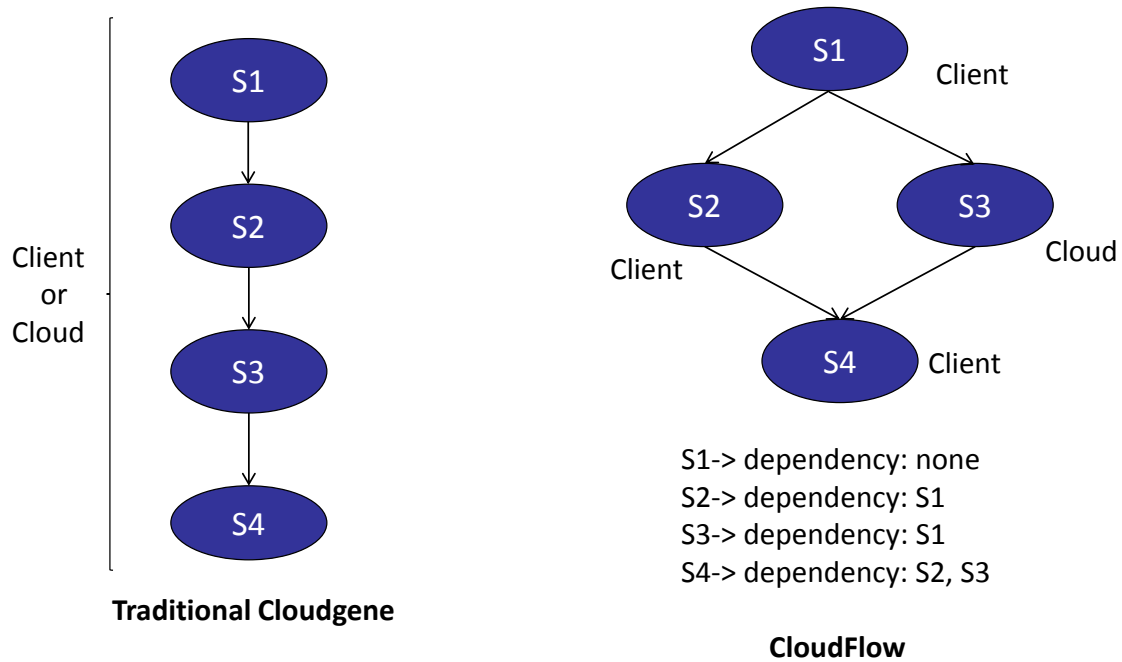


Figure 4.7: CloudFlow DAG Example Configuration

Hybrid Cloud Support

Hybrid cloud support corresponds to the simultaneous use of client and cloud resources for the execution of MapReduce pipelines. As discussed earlier, Cloudgene uses a worker thread per MapReduce pipeline and executes all the stages sequentially, either using the client or cloud resources. In order to run multiple stages in parallel using both client and cloud resources simultaneously, CloudFlow introduces the *step queue*, in addition to the job queue in Cloudgene. The step queue sits below the DAG processor and enables asynchronous execution of stages using distributed resources. The set of candidate vertices or stages determined by the DAG processor is fed into this step queue, by the worker thread, for execution. The step queue is periodically polled by a *step queue thread*. As soon as a vertex or stage is queued, the step queue thread spawns a *sub-worker thread* for this stage and then continues polling. This results in an asynchronous thread processing each stage of the

pipeline in parallel. As a result, multiple stages can be run independently and simultaneously using the desired resources provided by the user.

Fig. 4.8 provides a good overview of the hybrid cloud support provided by CloudFlow.

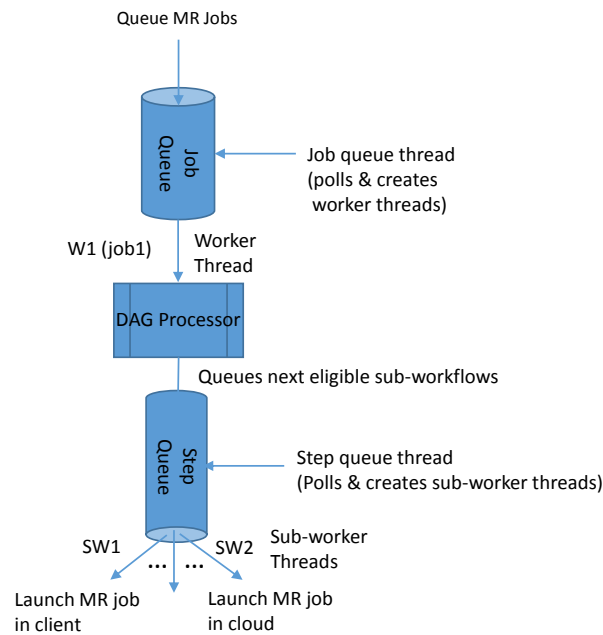


Figure 4.8: CloudFlow Hybrid Cloud Support

The sub-worker thread uses Apache Templeton [67] or WebHCat interface to spawn local or remote MapReduce jobs. Templeton provides a REST-based interface for launching both normal and streaming MapReduce jobs. The job launch via WebHCat returns a job ID (Job ID of the Templeton controller job), which can be used to query the status of running jobs.

Automatic Data Dependency Handling

CloudFlow implements automatic data-dependency handling for MapReduce pipelines. In traditional Cloudfone, it is the user's responsibility to ensure that the data is available either by manually importing the data or by having the data available from the output of the previous stage. In order to provide this support in CloudFlow, the users should specify

the input and output dataset required for each stage in the MapReduce pipeline. This support is implemented using the following two extensions in the manifest file, as shown in Fig. 4.9.

1. The *'stepinput'* parameter, which is used to specify the input dataset (file or directory) required by each stage in the pipeline. The input data should be prefixed with the appropriate access scheme (HDFS or asv) based on where the data is currently available (HDFS or blob).
2. The *'stepoutput'* parameter, which is used to specify the output data (file or directory) produced by each stage in the pipeline. The output data gets automatically prefixed with the appropriate access scheme based on where stage is configured to run.

```

- name: Table Recalibration
  params: -nfv -nm -nmd -na -nra -nv -ref $reference -b $s1input -o $s1output
          -djarloc $dpjarloc -p $s3platform -dbfile $dbfile
  stepinput: $reference,$s1input,$dpjarloc,$dbfile
  stepoutput: $s1output
} Data dependency support

- name: MergeResults
  params: -nqr -nfv -nmd -na -nra -nv -ref $reference -b $s1output -o $finoutput
          -djarloc $dpjarloc -p $s3platform
  stepinput: $reference,$s1output,$dpjarloc
  stepoutput: $finoutput

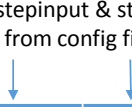
```

Figure 4.9: CloudFlow Data-Dependency Configuration

CloudFlow parses the “stepinput” and “stepoutput” values and initializes a global table with this information. This table contains the name of the data (with the access scheme removed) and the current location of the data (client, cloud or both). The sub-worker thread consults this table before launching the MapReduce job corresponding to a particular stage in the pipeline. Based on the execution location (either client or cloud) of a stage, the re-

quired dataset might be already available in the correct location or CloudFlow automatically transfers it to the correct location. A sample global table is shown in Fig. 4.10.

Parse stepinput & stepoutput
from config file



Filename	Location
dir1/dir2/file1	Client
dir1/file2	Cloud
Dir2/file1	Both

Figure 4.10: CloudFlow Data-Dependency Table

User-defined Plugin Support

Before the data is transferred (from client to cloud or vice versa), CloudFlow invokes the data transformation plugins registered by the user, if any. These plugins refer to certain functionality, such as compression or encryption, as implemented by the user. For example, the compression transformation might usually involve either byte-level or reference-based, i.e., contextual, compression. Reference-based compression typically requires an additional application-specific support file to assist the compression and decompression process. The transformation routine is invoked by the sub-worker thread before the data is transferred and the reverse-transformation routine is invoked by the same thread after the data is transferred and before it is used. CloudFlow optimizes the transformation, reverse-transformation, and data transfer phases using MapReduce processing. This will be discussed later in the optimization section.

On-demand Cloud Provisioning

The main driver behind CloudFlow is the CloudFlow-mapred server that runs in the client cluster and implements on-demand cloud provisioning. Having the workflow management driver at the client gives us the flexibility to provision cloud resources only when there is a need to offload computations to the cloud. This greatly helps in optimizing cloud costs as the cloud resources need not be provisioned statically before the start of the MapReduce pipeline. On-demand provisioning also avoids under-utilization of cluster resources and saves energy cost.

Facilitating Easy Debugging

CloudFlow supports running the entire MapReduce pipeline using the local client resources before offloading it to the cloud. This is similar to the support offered by local Oozie runner. This feature ensures that the pipeline runs to completion without issues before provisioning cloud resources. A successful dry run before the actual execution is highly advantageous as it enables one to debug any library, configuration or data availability related issues locally before moving the computation to the cloud. Provisioning cloud resources and then debugging issues incurs unnecessary cost for idle resources.

4.3 CloudFlow Optimizations

CloudFlow optimizes data transfers from client to cloud and vice versa using MapReduce. In addition, MapReduce is used to optimize the user-defined transformations that are defined by plugins. Using MapReduce for data transfers and transformations results in higher utilization of cluster resources. It also gives us the aggregate bandwidth and aggregate processing

capacity of the cluster to transfer and transform data, respectively.

CloudFlow uses the below algorithm to transfer data from client to cloud after applying transformations.

Step 1:

Generate a list of files to transfer from client to cloud

Let the file list be 'flist'

Step 2:

- a. Split 'flist' among the nodes of the MapReduce cluster using a custom InputFormat
- b. Let each partition of 'flist' be 'part_flist'

Step 3:

In each mapper on the client

```
if (transformation enabled) {  
  a. Apply transformation to 'part_flist' owned by this mapper  
  b. Send the transformed files to the cloud (Blob, in-case of  
    Azure)  
  c. Remove the transformed files from local HDFS  
} else {  
  Send 'part_flist' to the cloud  
}
```

Step 4:

```
if (transformation enabled) {  
  a. Construct a file list with the names of the transformed  
    files. Let the list be 'part_flist_names'. 'part_flist_names'  
    correspond to the transformed names of each file in 'part_flist'
```


- b. Send the file list 'part_flist_names' to the cloud
 - c. Create a remote MapReduce job to split 'part_flist_names' list among cloud nodes
 - d. In each mapper on the cloud
 - d.1. Call the reverse-transformation routine for the files owned by this mapper
 - d.2. Delete the original transformed file from the Blob
- }

Here is the algorithm used by CloudFlow to transfer data from cloud to client after applying the transformation.

Step 1:

Generate a list of files to transfer from cloud to client

Let the file list be 'flist'

Let 'final_list = flist'

Step 2:

if (transformation enabled) {

a. Send 'flist' to the cloud

b. Create a remote MapReduce job to split 'flist' among cloud nodes

c. In each mapper on the cloud

c.1. Call transformation routine to transform the files owned by each mapper

d. Construct a file list 't_flist' containing the names

- of the transformed files on the client
- e. Let ‘final_list = t.flist’
- }

Step 3:

- a. Split ‘final_list’ among multiple nodes in the client cluster, using a custom InputFormat
- b. In each mapper on the client
 - b.1. Receive a portion of final_list owned by this mapper from the cloud
 - b.2. if (transformation enabled) {
 - b.2.1. Reverse-transform the files that are part of the ‘final_list’ owned by this mapper
 - b.2.2. Delete the transformed file from the cloud
 - }

The flowchart in Fig. 4.11 provides a high-level overview of the entire process.

4.4 Case Study

We use SeqInCloud as a case study to test the feature set of CloudFlow. We built CloudFlow based on our insights and learning from running SeqInCloud on Azure. As discussed earlier, SeqInCloud is a data-intensive genome analysis pipeline that runs on Azure platform using Hadoop MapReduce. In this section we present how SeqInCloud can make use of the feature set that CloudFlow offers.

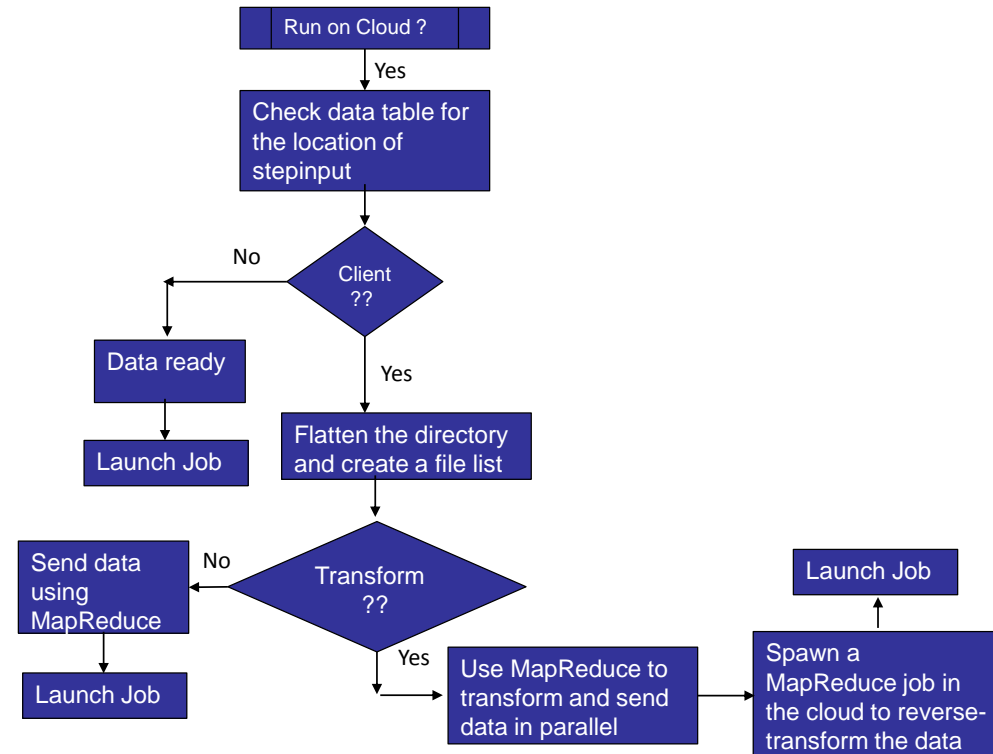


Figure 4.11: CloudFlow Data-Dependency Flowchart

DAG & Hybrid Cloud Support: The stages in SeqInCloud are sequential in nature except the last stage where the merging of the BAM output (“`.bam`” fragments) and merging of variant output (“`.vcf`” files) can be run simultaneously. The merging of the BAM fragments is done by a single process that runs on the head node of the cluster. Moving this computation to the cloud does not help as the scale of the cluster is immaterial here. However, the merging of variant files scales linearly as we increase the cluster size. Hence, running the merge variant stage using large-scale resources like the cloud will be beneficial. This is a good instance of hybrid resource usage as the merging of BAM fragments and merging of variant files will run simultaneously using both client and cloud resources. Below is a snippet of the YAML file for this configuration.

```
- name: Table Recalibration
  alias: s1
  dependency: none
  runlocation: Client

- name: Merge BAM
  alias: s2
  dependency: s1
  runlocation: Client

- name: Merge Variant
  alias: s3
  dependency: s1
  runlocation: Cloud
```

Automatic Data-Dependency Handling: Let us assume that the “TableRecalibration and Genotyper” stage is configured to run using on-premise resources. Two steps fork from this stage. Out of these two, the merge BAM step is configured to run using on-premise resources, and the merge variant step is configured to run using cloud resources. Below is a snippet of the YAML files that shows the input and output configuration for these stages. The parameter values that has a ‘\$’ sign in the front will be taken as input from the users using the generated user interface.

```
- name: Table Recalibration
```

```
alias: s1
dependency: none
runlocation: Client
stepinput: $reference,$s1input,$dbfile,$ipjar
stepoutput: $s1mergeout, $s1variantout

- name: Merge BAM
  alias: s2
  dependency: s1
  runlocation: Client
  stepinput: $reference,$s1mergeout,$ipjar
  stepoutput: $s2out

- name: Merge Variant
  alias: s3
  dependency: s1
  runlocation: Cloud
  stepinput: $reference,$s1variantout,$ipjar
  stepoutput: $s3out
```

As described earlier CloudFlow parses the “stepinput” and “stepoutput” values and initializes a global table. Initially, the global table looks as shown in Table 4.1.

After stage “s1” completes, the DAG processor returns s2, s3 as the candidate stages to run. As configured by the user, “s2” will be run using client resources and “s3” using cloud resources. The data-dependency module parses the “stepinput” and “stepoutput”

Name	Location
\$reference	Client
\$s1input	Client
\$dbfile	Client
\$ipjar	Client
\$s1mergeout	Client
\$s1variantout	Client
\$s2out	Client
\$s3out	Cloud

Table 4.1: Data-Dependency Table – Initial Values

parameters of each stage and triggers the data transfer as required. In our case, the stage “s2” has all the required data available at the client. But for stage “s3”, the input needs to be transferred to the cloud before the stage could run. The data-dependency module makes use of the MapReduce framework to transfer data in parallel to the cloud. After the data transfer, the global table looks as shown in Table 4.2.

Name	Location
\$reference	Both
\$s1input	Client
\$dbfile	Client
\$ipjar	Both
\$s1mergeout	Client
\$s1variantout	Both
\$s2out	Client
\$s3out	Cloud

Table 4.2: Data-Dependency Table – After Data Transfer

User-defined Plugin Support: As described earlier, the users can register plugins for data transformation before transferring data to the cloud. The transformation might be a normal byte-level compression or a domain-specific transformation. For example, in SeqInCloud, the input to the stages are BAM files. Instead of transferring the BAM file directly to the cloud, the transformation routine could implement either the reference- or GPU-based compression,

as discussed in SeqInCloud data-transfer optimizations. The data-dependency module will ensure that the transformation and reverse-transformation routines are invoked during the data transfer. There is not much opportunity here to compress the input of merge variant stage as the variant files are already smaller in size.

4.5 Applicability of CloudFlow to Other Cloud Platforms

Since CloudFlow is a ExtJS and REST-based server implementation, the majority of its features are platform-independent, such as DAG support, hybrid cloud support, and user-defined plugin support. As such and to facilitate broader adoption, CloudFlow is architected in a modular fashion so that it is easier to extend to a variety of other cloud provider environments.

Currently, the only aspects of CloudFlow that are specific to Microsoft HDInsight are the data-management interface and the on-demand cloud provisioning interface. The former uses the Azure Blob API to make the data available in the Azure Blob for running MapReduce jobs. The latter uses the Azure management API to provision cloud resources. Thus, in order to extend CloudFlow to other cloud environments, the data-management interface and on-demand cloud provisioning interface need to be instantiated.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this section, we present a summary of our work on SeqInCloud, a highly scalable realization of a widely used, genome analysis pipeline on the Windows Azure platform and CloudFlow, a workflow manager for running MapReduce-based pipelines using hybrid client and cloud resources.

In addition to the realization of SeqInCloud, this thesis presents an evaluation of running SeqInCloud, a data-intensive bioinformatic pipeline, on Microsoft Azure. We evaluate the strong-scaling behavior of SeqInCloud by varying the number of virtual cores from 8 to 64 and observe that SeqInCloud shows near-linear scalability. SeqInCloud optimizes network and storage costs with the help of a compressed sequence format, i.e., CRAM or by the use of GPUs for data pre-processing. It also optimizes the I/O throughput by mapping the data onto different storage resources on Azure, according to their characteristics. SeqInCloud is easy to configure and does not require installation of any additional packages.

Based on our experience and learning from running SeqInCloud on Azure, we also design and realize an efficient framework for running such data-intensive workflows using a hybrid setup of client and cloud resources. Specifically, we present CloudFlow, a workflow manager for running MapReduce-based pipelines like SeqInCloud using distributed resources. CloudFlow supports the simultaneous use of client and cloud resources, resulting in an optimized workflow that reduces execution time, when there are multiple stages in the workflow that run concurrently. CloudFlow also supports automated data transfers between the client and cloud resources to satisfy data dependencies before the execution of stages in the pipeline. CloudFlow provides users with the flexibility to define plugins for data transformations. The data transformations and transfers are optimized by using MapReduce processing, which efficiently utilizes the aggregate bandwidth and capacity of all nodes in the MapReduce cluster. From our learning and analysis, we observe that MapReduce is an excellent fit for large-scale, data-intensive scientific workloads. Our work on SeqInCloud can significantly benefit life scientists as it “accelerates” the time to genetic variant discovery. This, in turn, accelerates the identification of causes for diseases and aids in certain fields like population-scale sequencing and comparative genomics, for example.

Cloud computing is in the early stage of adoption among life scientists. The major issues for the shift in focus towards the cloud are as follows: data security, software configuration and setup overhead, data transfer overhead, and lack of availability of an easy-to-use interface to configure bioinformatic workflows. CloudFlow addresses these issues and aims to increase the adoption of cloud computing for running data-intensive scientific workloads.

5.2 Future Work

In the future, we intend to bundle SeqInCloud as a virtual machine image and offer it to the community via public cloud storage services like Azure Blob. We plan to improve the pipeline by adding new variant analysis stages from GATK. We also plan to build a model to predict the execution time of different stages within the pipeline, based on various factors like dataset size, compute and I/O characteristics, MapReduce parameters, and so on. Once the execution time of the workflow is modeled, we can provision only the required number of nodes for each stage and thereby meet the performance and cost budget requirements of the user.

For CloudFlow, we would like to improve the feature set by providing support for data triggers and locality, for example. We plan to build a model to determine the optimal location (either client or cloud cluster) to execute different stages in a workflow. This model will be based on the client and cloud cluster configuration, I/O characteristics, network transfer cost, as well as compute and storage costs. We also plan to extend CloudFlow to support a mix of MapReduce and non-MapReduce stages in a workflow. Another feature that is worth incorporating is the use of both client and cloud resources simultaneously for running a single stage of the workflow and later merging the results.

Bibliography

- [1] Lincoln D. Stein, “The case for cloud computing in genome informatics,” *Genome Biology*, vol. 11, no. 5, p. 207, 2010.
- [2] Michael C. Schatz, Ben Langmead, and Steven L. Salzberg, “Cloud computing and the DNA data race,” *Nature Biotechnology*, vol. 28, no. 7, pp. 691–693, 2010.
- [3] Monya Baker, “Next-generation sequencing: adjusting to data overload,” *Nature Methods*, vol. 7, pp. 495–499, 2010.
- [4] Scott D. Kahn, “On the Future of Genomic Data,” in *SCIENCE*, vol. 331, 2011, pp. 728–729.
- [5] Wetterstrand KA., “DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP),” accessed: 08-18-2013. [Online]. Available: www.genome.gov/sequencingcosts
- [6] “Windows Azure CDN,” accessed: 08-18-2013. [Online]. Available: <http://www.windowsazure.com/en-us/home/features/caching/>
- [7] Ronald C. Taylor, “An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics,” in *Proceedings of the 11th Annual Bioinformat-*

- ics Open Source Conference*, ser. BOSC 2010, vol. 11, no. S12. BMC Bioinformatics, 2010, p. S1.
- [8] Hugo Y.K. Lam, Cuiping Pan, Michael J. Clark, Phil Lacroute, Rui Chen, Rajini Haraksingh, Maeve O’Huallachain, Mark B. Gerstein, Jeffrey M. Kidd, Carlos D. Bustamante, and Michael Snyder, “Detecting and annotating genetic variations using the HugeSeq pipeline,” *Nature Biotechnology*, vol. 30, no. 3, pp. 226–229, 2012.
- [9] Abhishek Roy, Yanlei Diao, Evan Mauceli, Yiping Shen, and Bai-Lin Wu, “Massive genomic data processing and deep analysis,” in *Proceedings of the VLDB Endowment*, vol. 5, no. 12. ACM, 2012, pp. 1906–1909.
- [10] Ben Langmead, Kasper D. Hansen, and Jeffrey T. Leek, “Cloud-scale RNA-sequencing differential expression analysis with Myrna,” *Genome Biology*, vol. 11, no. 8, p. R83, 2010.
- [11] Jourdren L., Bernard M., Dillies MA., and Le Crom S., “Eoulsan: a cloud computing-based framework facilitating high throughput sequencing analyses,” *Bioinformatics*, vol. 28, no. 11, pp. 1542–1543, 2012.
- [12] Ben Langmead, Michael C. Schatz, Jimmy Lin, Mihai Pop, and Steven L. Salzberg, “Searching for SNPs with cloud computing,” *Genome Biology*, vol. 10, no. 11, p. R134, 2009.
- [13] Luca Pireddu, Simone Leo, and Gianluigi Zanetti, “MapReducing a genomic sequencing workflow,” in *MapReduce ’11 Proceedings of the second international workshop on MapReduce and its applications*. ACM, 2011, pp. 67–74.
- [14] Maria Fischer, Rene Snajder, Stephan Pabinger, Andreas Dander, Anna Schossig, Johannes Zschocke, Zlatko Trajanoski, and Gernot Stocker, “SIMPLEX: Cloud-Enabled

- Pipeline for the Comprehensive Analysis of Exome Sequencing Data,” *PLoS ONE*, vol. 7, no. 8, 2012.
- [15] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: simplified data processing on large clusters,” in *Communications of the ACM*, vol. 51, no. 1. ACM, 2008, pp. 107–113.
- [16] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox, “MapReduce in the Clouds for Science,” in *Cloud Computing Technology and Science*. IEEE, 2010, pp. 565–572.
- [17] Enis Afgan, Dannon Baker, Nate Coraor, Brad Chapman, Anton Nekrutenko, and James Taylor, “Galaxy CloudMan: delivering cloud compute clusters,” *BMC Bioinformatics*, vol. 11, no. S12, p. S4, 2010.
- [18] Konstantinos Krampis, Tim Booth, Brad Chapman, Bela Tiwari, Mesude Bicak, Dawn Field, and Karen E. Nelson, “Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community,” *BMC Bioinformatics*, vol. 13, no. 42, 2012.
- [19] Thilina Gunarathne, Tak-Lon Wu, Jong Youl Choi, Seung-Hee Bae, and Judy Qiu, “Cloud computing paradigms for pleasingly parallel biomedical applications,” *Concurrency and Computation: Practice and Experience*, vol. 23, no. 17, pp. 2338–2354, 2011.
- [20] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg, “Ultrafast and memory-efficient alignment of short DNA sequences to the human genome,” *Genome Biology*, vol. 10, no. 3, p. R25, 2009.
- [21] Li R., Li Y., Fang X., Yang H., Wang J., Kristiansen K., and Wang J., “SNP detection for massively parallel whole-genome resequencing,” *Genome Research*, vol. 19, no. 6, pp. 1124–32, 2009.

- [22] “Hadoop Streaming,” accessed: 08-18-2013. [Online]. Available: <http://hadoop.apache.org/docs/stable/streaming.html>
- [23] Michael C. Schatz, “CloudBurst: highly sensitive read mapping with MapReduce,” *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.
- [24] McKenna A., Hanna M., Banks E., Sivachenko A., Cibulskis K., Kernytsky A., Garimella K., Altshuler D., Gabriel S., Daly M., and DePristo M.A., “The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data.” *Genome Research*, vol. 20, pp. 1297–1303, 2010.
- [25] DePristo M.A., Banks E., Poplin R., Garimella K., Maguire J., Hartl C., Philippakis A., del Angel G., Rivas M.A., Hanna M., McKenna A., Fennell T., Kernytsky A., Sivachenko A., Cibulskis K., Gabriel S., Altshuler D., and Daly M., “A framework for variation discovery and genotyping using next-generation DNA sequencing data,” *Nature Genetics*, vol. 43, no. 5, pp. 491–498, 2011.
- [26] “GATK Queue,” accessed: 08-18-2013. [Online]. Available: <http://gatforums.broadinstitute.org/discussion/1306/overview-of-queue>
- [27] “Oracle Grid Engine,” accessed: 08-18-2013. [Online]. Available: http://en.wikipedia.org/wiki/Oracle_Grid_Engine
- [28] Luca Pireddu, Simone Leo, and Gianluigi Zanetti, “SEAL: a distributed short read mapping and duplicate removal tool,” *Bioinformatics*, vol. 27, no. 15, pp. 2159–2160, 2011.
- [29] Simone Leo and Gianluigi Zanetti, “Pydoop: a Python MapReduce and HDFS API for Hadoop,” in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 819–825.

- [30] Simone Leo, Federico Santoni, and Gianluigi Zanetti, “Biodoop: Bioinformatics on Hadoop,” in *International Conference on Parallel Processing Workshops*. IEEE, 2009, pp. 415–422.
- [31] Matsunaga A., Tsugawa M., and Fortes J., “CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications,” in *ESCIENCE*, vol. 62. ACM, 2008, pp. 222–229.
- [32] “Hadoop,” accessed: 08-18-2013. [Online]. Available: <http://hadoop.apache.org>
- [33] Sebastian Schonherr, Lukas Forer, Hansi Weissensteiner, Florian Kronenberg, Gunther Specht, and Anita Kloss-Brandstatter, “Cloudbene: A graphical execution platform for MapReduce programs on private and public clouds,” *BMC Bioinformatics*, vol. 13, no. 1, p. 200, 2012.
- [34] “Sencha ExtJS,” accessed: 08-18-2013. [Online]. Available: <http://www.sencha.com/products/extjs>
- [35] “Restlet Framework,” accessed: 08-18-2013. [Online]. Available: <http://restlet.org>
- [36] “JavaScript Object Notation,” accessed: 08-18-2013. [Online]. Available: <http://www.json.org/>
- [37] “Apache Oozie,” accessed: 08-18-2013. [Online]. Available: oozie.apache.org
- [38] Jianwu Wang, Daniel Crawl, and Ilkay Altintas, “Kepler + hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems,” in *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, ser. WORKS '09, no. 12. ACM, 2009, pp. 12:1–12:8.
- [39] Samuel Angiuoli, Malcolm Matalaka, Aaron Gussman, Kevin Galens, Mahesh Vangala, David Riley, Cesar Arze, James White, Owen White, and W. Florian Fricke, “Clovr: A

- virtual machine for automated and portable sequence analysis from the desktop using cloud computing,” *BMC Bioinformatics*, vol. 12, no. 1, p. 356, 2011.
- [40] Joshua Orvis, Jonathan Crabtree, Kevin Galens, Aaron Gussman, Jason M. Inman, Eduardo Lee, Sreenath Nampally, David Riley, Jaideep P. Sundaram, Victor Felix, Brett Whitty, Anup Mahurkar, Jennifer Wortman, Owen White, and Samuel V. Angiuoli, “Ergatis,” *Bioinformatics*, vol. 26, no. 12, pp. 1488–1492, 2010.
- [41] “Vappio,” accessed: 08-18-2013. [Online]. Available: <http://vappio.sourceforge.net/>
- [42] “Amazon Elastic Compute Cloud,” accessed: 08-18-2013. [Online]. Available: <http://aws.amazon.com/ec2/>
- [43] “Amazon Elastic Map Reduce,” accessed: 08-18-2013. [Online]. Available: <http://aws.amazon.com/elasticmapreduce/>
- [44] “Windows Azure HDInsight,” accessed: 08-18-2013. [Online]. Available: <https://www.hadooponazure.com>
- [45] Mohamed Abouelhoda, Shadi Issa, and Moustafa Ghanem, “Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support,” *BMC Bioinformatics*, vol. 13, no. 1, pp. 77+, 2012.
- [46] Wolstencroft Katherine, Haines Robert, Fellows Donal, Williams Alan, Withers David, Owen Stuart, Soiland-Reyes Stian, Dunlop Ian, Nenadic Aleksandra, Fisher Paul, Bhagat Jiten, Belhajjame Khalid, Bacall Finn, Hardisty Alex, Nieva de la Hidalga Abraham, P. Balcazar Vargas Maria, Sufi Shoaib, and Goble Carole, “The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud,” *Nucleic Acids Research*, 2013.

- [47] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team, “Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences,” *Genome Biology*, vol. 11, no. 8, p. R86, 2010.
- [48] “BroadInstitute best practices for variant calling with the GATK,” accessed: 08-18-2013. [Online]. Available: <http://www.broadinstitute.org/gatk/guide/topic?name=best-practices>
- [49] Heng Li and Richard Durbin, “Fast and accurate long-read alignment with BurrowsWheeler transform,” *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2009.
- [50] “Windows Azure,” accessed: 08-18-2013. [Online]. Available: www.windowsazure.com/en-us/
- [51] Li H., Handsaker B., Wysoker A., Fennell T., Ruan J., Homer N., Marth G., Abecasis G., and Durbin R., “The Sequence Alignment/Map format and SAMtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.
- [52] Hsi-Yang Fritz M., Leinonen R., Cochrane G., and Birney E., “Efficient storage of high throughput DNA sequencing data using reference-based compression,” *Genome Research*, vol. 21, pp. 734–740, 2011.
- [53] “Windows Azure Blob,” accessed: 08-18-2013. [Online]. Available: <http://www.windowsazure.com/en-us/develop/net/how-to-guides/blob-storage/>
- [54] “Cloud Service Models,” accessed: 08-18-2013. [Online]. Available: <http://www.inspurglobal.com/article-366.html>
- [55] “Windows Azure Blob Storage Concepts Simplified,” accessed: 08-18-2013. [Online]. Available: <http://www.azurecloudpro.com/windows-azure-blob-storage-concepts-simplified/>

- [56] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, “The google file system,” in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, 2003, pp. 29–43.
- [57] “Variant Call Format,” accessed: 08-18-2013. [Online]. Available: <http://www.1000genomes.org/node/101>
- [58] Nabeel M. Mohamed, Heshan Lin, and Wu-chun Feng, “Accelerating data-intensive genome analysis in the cloud,” in *5th International Conference on Bioinformatics and Computational Biology (BICoB)*. ISCA, 2013, pp. 297–304.
- [59] Heng Li and Nils Homer, “A survey of sequence alignment algorithms for next-generation sequencing,” *Briefings in Bioinformatics*, vol. II, no. 5, pp. 473–483, 2010.
- [60] Dong Xie, “Bioinformatics On Windows,” accessed: 08-18-2013. [Online]. Available: <http://bow.codeplex.com/releases>
- [61] Matti Niemenmaa, Aleksi Kallio, Andre Schumacher, Petri Klemelae, Eija Korpelainen, and Keijo Heljanko, “Hadoop-BAM: directly manipulating next generation sequencing data in the cloud,” *Bioinformatics*, vol. 28, no. 6, pp. 876–877, 2012.
- [62] “Hadoop on Azure,” accessed: 08-18-2013. [Online]. Available: <https://www.hadooponazure.com>
- [63] The 1000 Genomes Project Consortium, “A map of human genome variation from population-scale sequencing,” *Nature*, vol. 467, no. 7, pp. 1061–1073, 2010.
- [64] Sherry S.T., Ward M.H., Kholodov M., Baker J., Phan L., Smigielski E.M., and Sirotkin K., “dbSNP: the NCBI database of genetic variation,” *Nucleic Acids Research*, vol. 29, no. 1, pp. 308–311, 2001.

- [65] “CRAM Toolkit,” accessed: 08-18-2013. [Online]. Available: http://www.ebi.ac.uk/ena/about/cram_toolkit
- [66] Liu Yongchao, Schmidt Bertil, and L. Maskell Douglas, “Cushaw: a cuda compatible short read aligner to large genomes based on the burrows-wheeler transform,” *Bioinformatics*, vol. 28, no. 14, pp. 1830–1837, 2012.
- [67] “Apache Templeton,” accessed: 08-18-2013. [Online]. Available: http://people.apache.org/~thejas/templeton_doc_latest/