Accelerating Data-Intensive Genome Analysis in the Cloud

Nabeel M Mohamed Heshan Lin Wu-chun Feng
Department of Computer Science
Virginia Tech
Blacksburg, VA 24060
{nabeel, hlin2, wfeng}@vt.edu

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. Consequently, 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.

In this paper, we present *SeqInCloud*, our highly scalable implementation of a genome analysis pipeline on the Microsoft Hadoop on Azure (HoA) public cloud. 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 data-intensive, life-science applications with SeqInCloud.

Keywords: Cloud Computing, Microsoft Azure, MapReduce, GATK, Next Generation Sequencing.

1 Introduction

Today, next-generation sequencing (NGS) technologies generate data at a rate much faster than that of the growth of compute and storage capacity [1]. 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 needs. The cloud computing model also enables the easy sharing of public datasets and helps to facilitate large-scale collaborative research. As such, cloud computing has gained increasing traction in the bioinformatics community.

In this paper, we focus on accelerating a widely used genome-analysis pipeline built atop Burrows-Wheeler Aligner (BWA) [2] and the Genome Analysis Toolkit (GATK)[1] framework [3, 4] on Microsoft Azure [5], a *platform-as-a-service* (PaaS) cloud environment. Parallel implementations of the GATK pipeline in cluster environments have been investigated in several previous studies [6, 7, 8]. While these implementations can be deployed on *infrastructure-as-a-service* (IaaS) clouds such as Amazon EC2, they require external software packages (e.g., 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 [9] 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 cost and performance optimizations in cloud environments:

- In existing parallel GATK implementations, parallelism is achieved by partitioning the input data by contig (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

[1]Our 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.

issue, SeqInCloud adopts a highly scalable design that allows data processing to be partitioned by loci, a much finer level of parallelism.

- To optimize network costs, SeqInCloud enables application-level compression by converting the Binary Alignment/Map (BAM) [10] format to a reference-based compression format like CRAM [11] 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 file for downstream analysis in the cloud.

- To improve I/O performance, SeqInCloud intelligently maps input and output data across the storage hierarchy on Azure, including the local filesystem, Azure Blob, 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.

2 Related Work

In recent years, there has been a steep increase in the number of bioinformatic applications and workflows that use the MapReduce framework, a large percentage of which runs in the cloud. Crossbow [12] and Myrna [13] implement workflows for Single-Nucleotide Polymorphism (SNP) discovery and RNA-Seq differential expression analysis, respectively, in the cloud. Crossbow uses Bowtie [14] to align reads in the map phase, sorts alignments by genomic region and uses SOAPsnp [15] for SNP discovery. Both use Hadoop Streaming to implement the workflow.

The Genome Analysis ToolKit (GATK) [3, 4] 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 [8], to implement workflows. GATK-Queue can run jobs in batch processing systems like Oracle Grid Engine. In [6, 16], Pireddu et al. discuss Seal, a workflow that uses Pydoop and BWA to implement short-read mapping and duplicate removal. Seal provides its own implementation of deduplication and covariate table calculation using the MapReduce framework. In HugeSeq [7], 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. In SIMPLEX [17], 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. 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.

3 Methodology

Fig. 1 shows a SeqInCloud workflow, implemented using the Microsoft Hadoop on Azure (HoA) cloud framework. SeqInCloud uses Hadoop MapReduce framework and runs the workflow in a distributed fashion using multiple compute nodes provisioned in the cloud. The parallelism is achieved by partitioning input data by loci for the entire workflow. The workflow starts with the alignment stage, which uses a 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* walkers[2] from GATK. The realigned reads are fixed for discrepancy in mate information using Picard's *FixMateInformation*, deduplicated using Picard's *MarkDuplicates*, and re-indexed. The quality score of the de-duplicated reads are recalibrated using *CountCovariates* and *TableRecalibration* walkers. This is followed by the identification and filtering of structural variants (SNP and INDELS) using *UnifiedGenotyper* and *VariantFiltration* walkers. Finally, the variants are merged using *CombineVariants* walker.

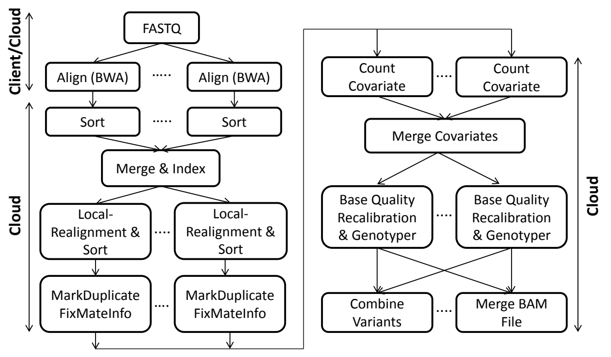

Figure 1: SeqInCloud Workflow

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

Below 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 to run both single- and paired-end sequence alignment in the MapReduce framework. SeqInCloud utilizes the Windows port of BWA from [18]. The input FASTQ files are split into multiple fragments by the mappers and are aligned in parallel by the reducers. The number of fragments owned by each cluster node depends on its memory and processing capacity. Per compute node, BWA requires about 3-4 GB of memory, and the Hadoop daemons require about 2 GB of memory. Each compute node in our resource allocation is a medium-sized Azure virtual machine (VM) instance that has a fixed memory limit of 3.5 GB, and the cluster is configured with a Java heap space of 1 GB. Running BWA under such resource constraints results in "out of heap space" memory errors. To address errors arising from running BWA in such memory constrained cloud environments and to provide more flexibility in VM provisioning, SeqInCloud allows users to offload the sequence-alignment stage 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 3.1.2. The compression can also be used when the input reads for alignment stage are stored in the BAM format.

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. 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. 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.

To address the above restriction, we provide 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. 2, 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.

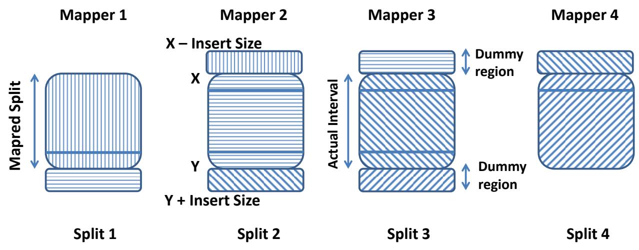

Figure 2: Design of IndelRealigner Stage

Base Quality Recalibration & Variant Calling. The base quality recalibration consists of two steps: CountCovariates and TableRecalibration. For CountCovariates, which is a locus-based GATK walker, the reducer aggregates identical covariates from all mappers and calculates a new empirical quality score using Phred scores[3]. This is followed by the TableRecalibration step, which rewrites the quality score of the reads with the empirical quality values calculated by CountCovariates 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 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 are finally merged.

The InputFormat and RecordReader for handling BGZF-compressed BAM files are used from the Hadoop BAM [19] library. The RecordReader provided by Hadoop BAM is extended in SeqInCloud to

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

define genomic intervals for each GATK instance invoked by the Hadoop mapper.

3.1 Cost Optimization in the Cloud

In this section, we present several techniques aimed at optimizing the execution cost of SeqInCloud in cloud environments.

3.1.1 Cost Optimization by Increasing Scalability

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. The 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 that is processing this contig runs for a longer duration. This creates an imbalanced workload and skews the overall execution time, which in turn, leads to underutilization 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.

3.1.2 Cost Optimization by Using Compression

SeqInCloud uses compression to optimize network and storage costs in the cloud. It uses the CRAM [11] format, which is a reference-based compression mechanism that encodes and stores only the difference between a read sequence and reference genome. The CRAM toolkit [20], 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.

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.

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. In 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 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 to this, ReduceReads compression takes much longer than CRAM compression (e.g., for a fragment of the NA12878 dataset of size 754 MB, the ReduceReads compression took 112 minutes vs. 10 minutes for the CRAM compression).

3.1.3 Cost Optimization by Using Storage Tiering

SeqInCloud uses different storage resources that are available in the HoA 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.

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). MapReduce can directly process the files that are available in blobs, except for the case where a blob is used as an input stream and the record reader seeks a wide offset range. Due to this exception in HoA environment, the blob can only be used in the later stages of the workflow. For better throughput, the blob needs to be provisioned in the same region as the compute nodes.

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. This is done so that the best-suited storage resource based on the requirement of each stage and throughput is chosen for Input/Output. 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).

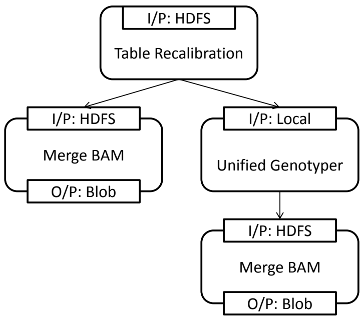

Figure 3: Feasible Input/Output Storage Resource for the "Mix" Mapping.

4 Results and Evaluation

We have evaluated SeqInCloud on a 32-node Azure cluster, where each node is a medium Hadoop on Azure (HoA) 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 [21] in our experiments: 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 used for count covariates stage is dbsnp_135.b37.

4.1 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. 4 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.

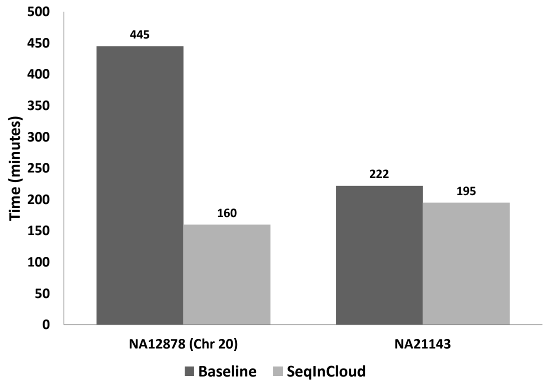

Figure 4: Comparison of Baseline and SeqInCloud Execution Time of Major Workflow Stages for Datasets NA12878 and NA21143.

As discussed in Section 3, the baseline run for the single contig NA12878 dataset utilizes only a single cluster node for the IndelRealigner and CountCovariate stages, thus affecting scalability and accruing usage cost for idle resources. As a result, the run time of SeqInCloud is nearly 2.7-fold faster than the baseline run time for the NA12878 dataset. In the case of the NA21143 dataset, where sequences are aligned to the entire genome, SeqInCloud ran 12% faster than the baseline. The performance improvement here is not as significant because the total number of cluster nodes (32) or map slots (64) is less than the number of contigs (84). We would see an increasing improvement in performance as we keep increasing the number of cluster nodes/map slots beyond 84, which is the baseline upper bound on scalability for the NA21143 dataset.

4.2 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). 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. 5 shows 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.

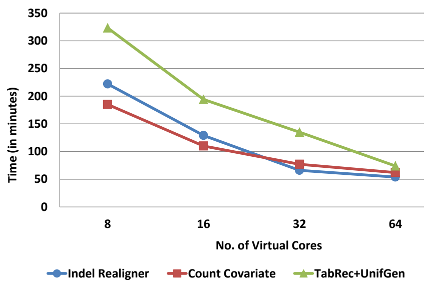

Figure 5: Execution Time of the Major Stages in SeqInCloud for the 24.3 GB NA10847 Dataset.

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 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.

4.3 Evaluation of Storage Savings Due to Compression

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 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%. But, this improvement in the data transfer time comes with an additional overhead of compression from BAM to CRAM at on-premise and decompression from CRAM to lossless BAM at the cloud[4]. 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 to the time taken to run the entire workflow 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.

While using 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 achieved 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 observed an improvement in the workflow performance. Conversely, when the number of on-premise nodes was lesser, we observed a dip in the workflow performance.

Table 1: Performance Improvement and Storage Savings for NA10847, NA21143 and NA12878 Datasets due to Compression using a 14-node On-Premise Hadoop Cluster.

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

Table 1 shows the improvement in workflow performance and storage savings when using the CRAM format instead of the BAM format. Here the number

[4]The compression and decompression is achieved using interfaces from the CRAM toolkit.

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 etc. 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 originial BAM file. As a result, the storage savings also varies across datasets.

4.4 Evaluation of Performance Due to Storage Tiering

We evaluated the performance of SeqInCloud using different combinations of storage resources 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. 6 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.

5 Conclusion

In this paper, we present SeqInCloud, our highly scalable implementation of a popular genome analysis pipeline based on GATK, on the Windows Azure platform. We evaluate the strong-scaling behavior of SeqInCloud by varying the number of virtual cores from 8 to 64 and observe that SeqInCloud scales nearly linearly. SeqInCloud optimizes network and storage costs with the help of a compressed sequence format, i.e., CRAM. It also optimizes the I/O throughput by intelligently mapping various data onto different storage resources on Azure according to their characteristics.

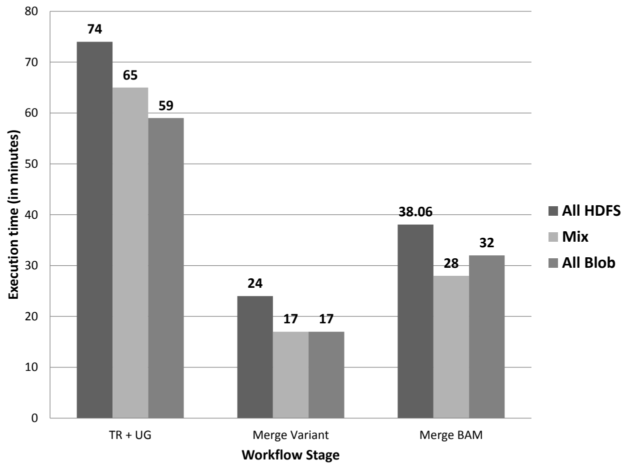

Figure 6: Execution Time of "All HDFS", "Mix" and "All Blob" Mappings.

SeqInCloud is easy to configure and does not require installation of any additional packages. In the future, we plan to provide flexibility for the users to customize the workflow. We intend to bundle SeqInCloud as a virtual machine image and offer it to the community via public cloud storage services like Azure Blob.

Acknowledgments

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. We thank NSF for funding this project and Microsoft for providing us the hardware to run our experiments and for their timely help in resolving technical issues faced by us.

References

[1] Scott D. Kahn, "On the Future of Genomic Data," in *SCIENCE*, vol. 331, 2011, pp. 728–729.

[2] Heng Li *et al.*, "Fast and accurate long-read alignment with BurrowsWheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2009.

[3] McKenna A *et al.*, "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data." *Genome Research*, vol. 20, pp. 1297–1303, 2010.

[4] DePristo M *et al.*, "A framework for variation discovery and genotyping using next-generation DNA sequencing data," *Nature Genetics*, vol. 43, no. 5, pp. 491–498, 2011.

[5] Microsoft, "Windows Azure." [Online]. Available: www.windowsazure.com/en-us

[6] Luca Pireddu *et al.*, "MapReducing a genomic sequencing workflow," in *MapReduce '11 Proceedings of the second international workshop on MapReduce and its applications*. ACM, 2011, pp. 67–74.

[7] Hugo Y K Lam *et al.*, "Detecting and annotating genetic variations using the HugeSeq pipeline," *Nature Biotechnology*, vol. 30, no. 3, pp. 226–229, 2012.

[8] Broad Institute, "GATK Queue." [Online]. Available: http://gatkforums.broadinstitute.org/discussion/1306/overview-of-queue

[9] The Apache Software Foundation, "Hadoop." [Online]. Available: http://hadoop.apache.org

[10] Heng Li *et al.*, "The Sequence Alignment/Map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[11] Markus Hsi-Yang Fritz *et al.*, "Efficient storage of high throughput DNA sequencing data using reference-based compression," *Genome Research*, vol. 21, pp. 734–740, 2011.

[12] Ben Langmead *et al.*, "Searching for SNPs with cloud computing," *Genome Biology*, vol. 10, no. 11, p. R134, 2009.

[13] Ben Langmead *et al.*, "Cloud-scale RNA-sequencing differential expression analysis with Myrna," *Genome Biology*, vol. 11, no. 8, p. R83, 2010.

[14] Langmead B *et al.*, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, 2009.

[15] Li R *et al.*, "SNP detection for massively parallel whole-genome resequencing," *Genome Research*, vol. 19, no. 6, pp. 1124–32, 2009.

[16] Luca Pireddu *et al.*, "SEAL: a distributed short read mapping and duplicate removal tool," *Bioinformatics*, vol. 27, no. 15, pp. 2159–2160, 2011.

[17] Maria Fischer *et al.*, "SIMPLEX: Cloud-Enabled Pipeline for the Comprehensive Analysis of Exome Sequencing Data," *PLoS ONE*, vol. 7, no. 8, 2012.

[18] Dong Xie, "Bioinformatics On Windows." [Online]. Available: http://bow.codeplex.com/releases

[19] Matti Niemenmaa *et al.*, "Hadoop-BAM: directly manipulating next generation sequencing data in the cloud," *Bioinformatics*, vol. 28, no. 6, pp. 876–877, 2012.

[20] European Nucleotide Archive, "CRAM Toolkit." [Online]. Available: http://www.ebi.ac.uk/ena/about/cram_toolkit

[21] The 1000 Genomes Project Consortium, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, no. 7, pp. 1061–1073, 2010.