Investigating Genome Analysis Pipeline Performance on GATK with Cloud Object Storage

Tatsuhiro Chiba
IBM Research
19-21, Nihonbashi Hakozaki-cho
Chuo-ku, Tokyo, 103-8510, Japan
chiba@jp.ibm.com

Takeshi Yoshimura
IBM Research
19-21, Nihonbashi Hakozaki-cho
Chuo-ku, Tokyo, 103-8510, Japan
tyos@jp.ibm.com

***Abstract*—Achieving fast, scalable, and cost-effective genome analytics is always important to open up a new frontier in biomedical and life science. Genome Analysis Toolkit (GATK), an industry-standard genome analysis tool, improves its scalability and performance by leveraging Spark and HDFS. Spark with HDFS has been a leading analytics platform in a past few years, however, the system cannot exploit full advantage of cloud elasticity in a recent modern cloud. In this paper we investigate performance characteristics of GATK using Spark with HDFS and identify scalability issues. Based on a quantitative analysis, we introduce a new approach to utilize Cloud Object Storage (COS) in GATK instead of HDFS, which can help decoupling compute and storage. We demonstrate how this approach can contribute to the improvement of the entire pipeline performance and cost saving. As a result, we demonstrate GATK with IBM COS can achieve up to 28% faster than GATK with HDFS. We also show that this approach can achieve up to 67% cost saving in total, which includes the time for data loading and whole pipeline analysis.**

I. Introduction

As an increase of rapid development of Next Generation Sequencing (NGS) technologies, genome analysis has become an emerging research area in bioinfomatics. Thus, this new research capability attracts a lot of interest from bio-scientists who want to perform genome analysis such as Single Nucleotide Polymorphism (SNP) genotyping, genetic variants identification, and so on. While reducing genome sequencing cost with the NGS tools, a huge amount of genome sequencing data has been produced day by day, usually in the range of 100GB. Therefore, it is always important to prepare a system that can handle the immense size of data as effectively as possible because genome analytics requires huge amount of compute and storage resource.

Genome Analysis Toolkit (GATK) [1] is the most popular and widely used open source genome analytics framework developed by Broad Institute. GATK also provides a typical genome variant discovery analysis workflow as a GATK Best practice [2], which combines multiple tasks into a single pipeline, so that various genome analytics tools (BWA and HyplotypeCaller etc.) are integrated with GATK to ensure a genome analysis ecosystem. Recently, GATK has leveraged Spark [3] in order to achieve higher scalability in their analysis. As a result, GATK pipeline can accelerate genome analysis throughput more easily as a whole by taking full advantage of the capability of node-level and core-level parallelism in each tool.

Meanwhile, those genome analysis tools running on Spark assume to utilize Hadoop Distributed File System (HDFS) [4] as an underlying data storage implicitly. It is a natural architectural design, because Spark is a successor to Hadoop and HDFS can effectively manage large data over multiple nodes in a fault-tolerant way. Besides that, the data locality in HDFS helps to co-locate compute and storage, which can exploit those resources as much as possible [5].

Cloud has been widely used in the last decade. Nowadays, several famous cloud platforms such as AWS, Azure, and IBM Cloud have become essential infrastructure to establish large scale system, services, and applications in a scalable and cost-effective way [6]. Cloud brings various benefits such as availability and elasticity to our applications, and those are effective even if the applications do not adopt cloud native principles. In such situation, it is reasonable to move genome analytics platforms from on-premise to clouds in terms of cost efficiency and scalability. However, the current reference architecture of GATK with Spark and HDFS is not ready to adopt recent modern cloud technologies due to the limitation of the analytics system that tightly couples compute and storage.

Thus, there are several challenges to take full advantages of cloud scalability in GATK pipeline. First, it is quite hard to dynamically add or remove storage resource capacity and nodes in HDFS. HDFS balances data pieces across all nodes and then maintains entire file system consistency. This limitation forces us to keep a fixed number of nodes to store dataset even if genome analysis does not require vast storage space. In addition, we must load the entire dataset to HDFS each time we want to adjust an adequate storage resource capacity to the analysis pipeline. This kind of additional data loading overhead also occurs when we launch Spark/HDFS cluster on a cloud from scratch. Second, similar to the concern for the storage elasticity, compute resource elasticity is also crucial for further optimization to utilize resources more efficiently in each stage of the analysis pipeline. Dynamic compute resource allocation based on the analysis demands could help to reduce the waste of unused resources. Third, workload characterization is most important to know the capability of

optimization and also helps us to achieve both storage and compute elasticity in GATK pipeline.

To address these challenges, in this paper, we investigate the performance and workload characteristics of GATK-Spark with HDFS (GATK with HDFS, in short) on a cloud environment to identify potential bottleneck and optimization opportunities for the genome analysis pipeline first. Based on the detailed analysis, we enable Cloud Object Storage (COS) as a replacement for HDFS, which can help to bring elasticity to GATK by decoupling compute and storage. With leveraging Spark with COS in GATK, we provide a new best practice for GATK which introduces storage elasticity into the genome analysis pipeline. We also implement several features in GATK to support object storage access, which can bridge an architectural gap due to the difference between object storage and HDFS. Then, we demonstrate that our approach can contribute to not only performance scalability but also saving cost on the entire pipeline execution while comparing performance of GATK with COS and with HDFS. In a typical variant calling pipeline with whole genome sequencing dataset, we demonstrate that COS achieves a 9% performance improvement than HDFS when attaching a low throughput volume to HDFS, and is 20% worse than HDFS when attaching a high throughput volume to HDFS. We also found a fundamental overhead why the performance drawback exists when utilizing COS instead of HDFS in GATK. By eliminating the performance inhibitor, we finally show that COS achieves a 28% performance improvement or completely same performance than HDFS when attaching slower or faster volumes respectively. As for the cost perspective, we indicate that COS always has good cost performance with HDFS while executing an entire analysis pipeline. The main contributions of this paper are as follows.

- We characterize a typical genome analysis pipeline on GATK with HDFS, and identify the scalability and elasticity issues in the pipeline.
- We provide a new best practice by enabling COS instead of HDFS, showing that COS scalability and its cost effectiveness, and then explaining the fundamental cause of the overhead introduced by the new best practice utilizing COS.
- We demonstrate how we can overcome the overhead, increasing the entire pipeline performance up to 28% and descreasing the overall costs up to 67%.

II. Background and Related Works

A. Genome Analysis Pipelines

GATK defines and provides a typical set of DNA sequence analysis pipeline as GATK Best Practice [2]. Variant discovery, which identifies genome variants in the DNA, is widely used genomic analysis. Figure 1 explains an overall pipeline defined in GATK, consisting of multiple steps. First step is a data preprocessing to align vast sequences into a reference genome and create a DNA mapping for the further analysis. There exist several fast alignment tools, but Borrows-Wheeler Aligner

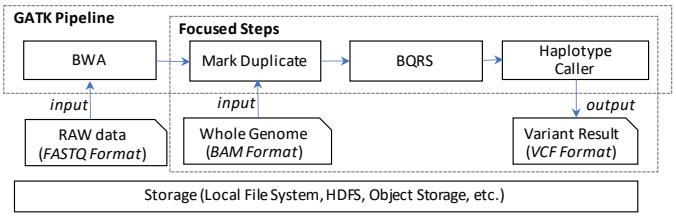

Fig. 1. GATK Pipeline for Genome Variant Discovery

(BWA) is a popular tool based on Borrows-Wheeler Transform (BWT) algorithm. GATK implements BWA-MEM in a first step of the pipeline for this purpose. In this paper, we skipped this step and used a BAM format as an input file. BAM format is compressed binary to represent aligned sequences.

Next step is MarkDupilicates phase, which marks duplicated fragment of sequences by utilizing reference genome data and alignment information. This steps reads a vast amount of genome data while comparing a large number of key value pairs. Then, the third step is Base Quality Core Recalibration (BQSR) phase, which adjusts quality scores for the aligned read dataset by applying a machine learning model to correct systematic technical sequencing machine errors.

After finishing these steps, variant discovery phase, named Haplotype Caller, starts processing as a final step. This phase searches all of the genome variants by comparing well known reference genome variants. To speed up this sequence comparison phase, Haplotype Caller implements PairHMM Forward algorithm, and several accelerator implementations are available to take full advantage of underlying CPU features, such as native SIMD function support and OpenMP multi-threading support. The last stage in this step materializes a final output, VCF file, which contains all of the discovered genome variants with some headers.

B. Analytics Engine and Storage on Cloud

Apache Spark [3] is one of the most widely used analytics engines especially for big data processing. With applying in-memory computing style, Spark has extremely higher throughput than predecessor engines such as Hadoop. Since HDFS [4] has been a primary data lake in on-premise analytics environments in the past decade, the combination of Spark and HDFS still remains as a first choice even if we move into a cloud, because this architecture is stable and does not require additional learning cost. Spark itself is flexible in a backend storage for reading/writing data, so it can work well in not only HDFS, but also with other types of storage such as RDB, key-value store, and object storage.

Cloud object storage such as AWS S3, IBM Cloud Object Storage (COS), and Google Cloud Storage (GCS) provides high capacity, reliable, and cost effective managed service to users and applications. Cloud object storage was basically suitable for the purpose of storing large amount of data, so it was not supposed to be a replacement file system for applications, because disk bandwidth was always higher than network capacity, and also disk access latency is lower than network latency. By enabling high network capacity in a

cloud, however, instances and services within a cloud can achieve higher throughput than ever before. This capability makes us rethink system architectural design regarding how much performance impact we can achieve in analytics platform [7]. In terms of storage scalability, object storage will be an essential piece of modern cloud which can help to decouple compute and storage while maintaining or improving performance. Of course, object storage is not a POSIX file system, but a storage service that can be accessible through RESTful API; therefore, applications must take into consideration the difference between object storage and file storage.

C. Related Works

A couple of earlier works have been studied before to address scalability problems and workload characterization of genome analysis pipeline. SparkGA [8] is an Apache Spark based framework for a DNA analysis pipeline, which introduces an optimal parallel implementation of the analysis with input load balancing and maximizing resource usage. A successor of the work, SparkGA2 [9] improves the efficiency of data access on HDFS and reduces memory footprint with an optimized compression method for intermediate data. CloudGT [10] proposes a Parquet columnar based optimization to improve IO performance in several genome analysis pipelines using Spark. Costa et al. [11] investigate performance characteristics of genome analysis pipeline on GATK using Spark, and introduce JVM and Spark configuration level tuning to accelerate performance. Doppio [12] proposes a disk IO aware Spark analytical model to estimate Spark application behavior. It utilizes GATK analysis pipeline as a representative workload of this work. However, all of the previous works assume HDFS as an underlying primary storage.

Several works have investigated how big data analytics platforms can utilize object store for their workloads. SwiftAnalytics [13] provides a system that considers data access locality with placement control to leverage OpenStack Swift object storage system for Spark. To easily and transparently access data in object store from Spark application, many storage connectors are available in Spark, such as AWS S3A connector, IBM COS connector (i.e. Stocator [14]), OpenStack Swift Connector, and so on. Since they ensure their connectors over Hadoop File System APIs, application can transparently access objects without any modifications. However, Hadoop File System APIs are originally designed for HDFS, so some APIs do not work well with object storage.

III. Performance Analysis at Scale

In this section, we first evaluate genome analysis pipeline performance on GATK4 using Spark with HDFS (GATK with HDFS, in short) to get a better understanding of the issues on a typical Spark Hadoop analytics framework in a cloud. We also reveal its performance characteristics and scalability with detailed Spark application and system metrics. Based on the results, we identity what challenges are remaining when we run genome analysis pipeline to leverage modern cloud scalability as much as possible.

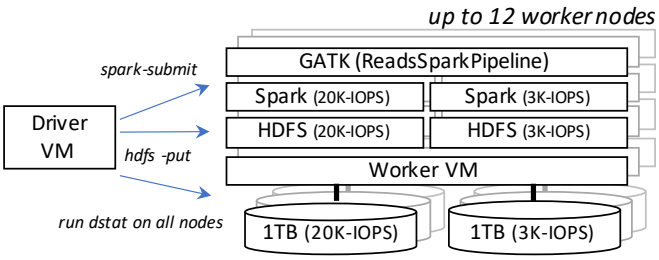

Fig. 2. Architectural overview of GATK-Spark with HDFS on Cloud

A. Experiment Settings

Cloud Environment: We set up a Spark and Hadoop cluster on IBM Cloud Virtual Private Cloud (VPC) environment. According to the cluster virtual server configuration, we launched up to twelve mx2-32x256 instances on the same availability zone, each equipped with 32 vCPU cores and 256 GB RAM. We choose Ubuntu 18.04.1 LTS (kernel: 4.15.0-42-generic) for the operating system. These instances are connected with up to 16 Gbps network. We also attached two 1-TB block storages with different IOPS profiles for each node: 3-IOPS/GB and 10-IOPS/GB. As for the storage profile, we can select several storage IOPS profiles based on our workload requirements. Actual storage performance with storage IOPS profiles depend on volume size and profile limitation, so that a 1-TB volume with 3-IOPS/GB has up to 3,000 IOPS (3K-IOPS) and a 1-TB volume with 10-IOPS/GB has up to 20,000 IOPS (20K-IOPS) respectively. We formatted the volumes with XFS. These disks are utilized by not only HDFS, but also Spark executors, because Spark requires local temporary storage to store intermediate data for shuffling over nodes. Virtual server settings and specifictions are summarized in Table I.

Software Stack: We installed GATK 4.1.4.1, Spark 2.4.5, Hadoop 2.7.7, and OpenJ9 JVM 1.8.0_242-b08 on the cluster. Software configuration tuning is always crucial to leverage application performance as much as possible. In terms of Spark, Hadoop, and underlying JVM parameter tuning, several works have been already studied best configuration practice [15][11]. Configuration tuning itself is out-of-scope in this paper, so we borrowed knowledge on tuning from references. As discussed in those papers, we managed multiple Spark executors in each node; we launched four executors with 8 vCPUs, 35 GB heap, and 15 GB off-heap in each. Thus, almost all compute and memory resources are reserved by Spark executors. We applied default configurations to HDFS, like three replicas, 128 MB block size, and so on.

Genome Dataset and GATK Pipeline with Spark: We prepare an open whole genome (WGS) dataset obtained from GATK, originally coming from the 1,000 genome project [16]. We used WGS-G94982-NA12878-no-NC_007605.bam as input, which has 154 GB in total. GATK also provides a reference benchmarking pipeline from aligned reads to variant calling for Spark, named ReadsPipelineSpark, in their repository. Based on the reference runner scripts for the WGS dataset, we upgraded pairHMM option to AVX_LOGLESS_CACHING_OMP to accelerate Haplotype

TABLE I
CLUSTER AND NODE CONFIGURATION ON CLOUD

Region	Nodes	OS	Profile	vCPUs	Memory	Network	Block Storage (IOPS)
London (eu-gb)	12	Ubuntu 18.04.1	mx2-32x256	32	256 GiB	16 Gbps	1TB (3K-IOPS) and 1TB (20K-IOPS)

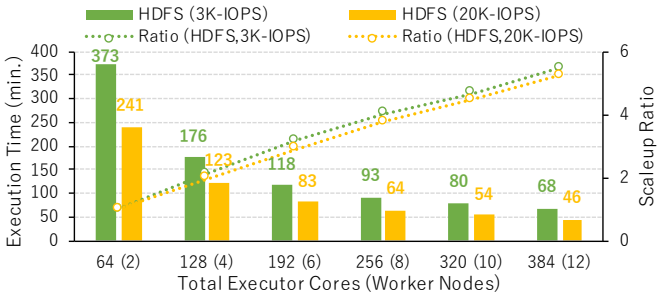

Fig. 3. Weak Scaling Performance on GATK with HDFS

Caller more with SIMD instructions and OpenMP multi-threading.

Figure 2 shows an overall architecture of GATK benchmarking environment on a cloud. We prepared a driver node used to submit GATK jobs and load data into HDFS. In addition, the driver node manages dstat based monitoring tool to capture various kind of system-wide metrics such as CPU, memory, network, and disk usage while running benchmark pipeline on all nodes. To evaluate how disk speed impacts performance, we manage two set of Spark and HDFS which can utilize those two types of disks separately as mentioned in the Cloud Environment section. Therefore, 3K-IOPS or 20K-IOPS annotation mean that both Spark and HDFS only manipulate 1TB 3K-IOPS or 20K-IOPS disk respectively while running a benchmark.

B. GATK Workload Scalability

First we evaluate scalability of the ReadSparkPipeline workload while changing the number of nodes and total cores. Figure 3 shows a weak scaling result with two to twelve nodes. As shown in Figure 3, GATK ReadsSparkPipeline has good scalability and achieved around 5.5x speedup in twelve nodes compared to the performance in two nodes as a whole. Focusing on the result in twelve nodes that have 384 vCPUs and 3TB RAM in total, we finally achieved 68 minutes and 46 minutes to complete a full analytics pipeline with 3K-IOPS and 20K-IOPS disk respectively.

Next we perform breakdown analysis on the GATK pipeline. Table II describes characteristics of each Spark job, including input/output data size, shuffle read/write size, and the category this job corresponds to in GATK pipeline. Although it depends on the version of GATK and Spark, the version of GATK we used in this paper translates an entire pipeline shown in the Figure 1 into eight spark jobs. Each job has different characteristics: read IO heavy, network heavy, shuffle read/write heavy, and so on. The first four jobs represent Mark Duplicate with Aligned Reads. WGS genome data is loaded from HDFS in the first job, then the data is consumed in the following three jobs with shuffling. The fifth job represents BQSR in the pipeline. It performs a significant amount of shuffle read and write in the computation. The last three jobs represent HaplotypeCaller, which finally writes variant result into HDFS with many data shuffles. (a) and (b) in Figure 4 show a breakdown analysis of how much time is spent in each job. Both represent the result on two and twelve nodes with 3K-IOPS and 20K-IOPS disk. We can observe several characteristics with the breakdown analysis. First, disk IO performance of jobs 0 and 1 are dominant. Although job 1 also reads the same data that job 0 has read, job 1 benefits from Spark in-memory architecture; job 1 finishes quite faster than job 0 because data is cached in memory as a file cache. Moreover, job 7 is dominant in all of the entire pipelines with the increase of nodes.

As for (c) and (d) in Figure 4, they describe how much each job scales when increasing nodes and cores. We can observe here that the scaling characteristics are also different in each job and analytics pipeline stage. Regarding the first two jobs related to MarkDuplicate, their scalabilities are bounded by performance of disk IO and memory respectively. The following two jobs, jobs 2 and 3 have good scaling because they are network and memory intensive. Job 4, BQSR in a pipeline, has also great scalability; it achieved 10x and 6x scaling with 3K and 20K IOPS HDFS respectively. Job 7, which is the last stage in HaplotypeCaller, accounts for half of the time in the pipeline when relaxing a disk performance constraint with higher throughput disk.

Then we study pipeline characteristics from resource usage perspective. Figures 5 show each resource usage (i.e. CPU, Disk, Memory, and Network) focusing on a node when running benchmark over twelve nodes. (a) of the figures represents the metrics in 3K-IOPS and (b) in 20K-IOPS respectively. Each resource usage graph also has a map onto the Spark jobs where they run. In 3K-IOPS case while running jobs 0 and 1, disk read/write bandwidth reached up to the limit, around 45 MB/sec in total. In contrast, the 20K-IOPS case utilized around 300 MB/sec and it still has a bandwidth capacity up to the 20K-IOPS limitation. We can see the limitation from how much wait time accounts for in CPU usage graph as well; Utilizing 3K-IOPS accounts for 20-40% in wait while running jobs 0 and 1, but 20K-IOPS does not. We can also observe an interesting characteristics in disk usage; read operation happens only in job 0 and 1. In other words, the later jobs do not read data from HDFS, but read from shuffle write data stored in memory as a file cache. As shown in the memory usage graphs, file cache occupies over half of memory in both scenarios. Job 3, the final phase in MarkDuplicate, starts utilizing all resources evenly, but especially consumes a huge amount of heap memory and network for shuffling

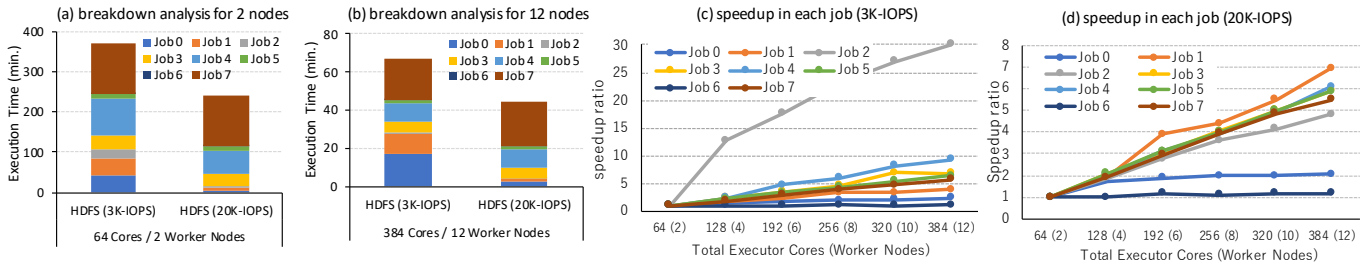

Fig. 4. Breakdown analysis of job execution time (a and b) and speedup ratio in each job (c and d)

TABLE II
BREAKDOWN ANALYSIS AND CHARACTERISTICS OF SPARK JOB

Job	HDFS input	HDFS output	shuffle read	shuffle write	GATK pipeline
0	154GB	-	-	-	Read+MarkDup
1	154GB	-	12.8MB	226GB	Read+MarkDup
2	-	-	-	226GB	Read+MarkDup
3	-	-	498GB	45.8GB	Read+MarkDup
4	-	-	522GB	283GB	BQSR
5	-	-	262GB	-	HaplotypeCaller
6	-	-	14.2MB	14.2MB	HaplotypeCaller
7	104MB	995MB	524GB	-	HaplotypeCaller

data between executors. Job 4 has similar characteristics to job 3. Job 7, the last pipeline in HaplotypeCaller, does not read/write data from/to disk but highly utilizes CPU and network instead. In summary, MarkDuplicate is categorized as disk I/O intensive, BQSR as disk and network intensive, and HaplotypeCaller as CPU and network intensive.

C. Analytics Infrastructure at Rutime

Next we evaluate performance from a different angle; we investigate how much time is required to set up a genome analysis system on a cloud, such as instance start-up, software installation, and genome data loading time to HDFS. Most previous works assume that Spark and Hadoop are already available, but it is important to keep minimizing resource usage on a cloud in terms of cost reduction [6]. An ideal situation is that we construct genome analysis pipeline at runtime, and then deallocate the system after finishing all pipelines. So we evaluate an analytics system setup time from scratch to understand how practical it is.

Table III shows how much time is spent in each phase. We used Terraform to set up infrastructure, which can help provisioning volumes and instances easily on a cloud. We also prepared a VM image that has already included Spark and Hadoop jars. The data loaded into HDFS is stored in local on the driver node. As shown in the table, provisioning infrastructure does not take so much time, so it is trivial compared to the entire pipeline computation time. However, data loading time is quite large. It takes about 30 minutes in our test. This cost is dominant in the entire pipeline, since it takes 45 minutes to finish the computation pipeline on a twelve node cluster with 20K-IOPS disk as shown in figure 3. Although the transfer time and speed depend on the disk or network bandwidth, it might not be negligible if we copy

TABLE III
SYSTEM SETUP TIME

	create volumes	create instances	load data into HDFS
elapsed time	56 sec	2.5 mins	30 mins

vast mounts of genome data into HDFS every time when we deploy a new system at runtime.

IV. CHALLENGES AND APPROACHES

In this section, we summarize what challenges still remain to be optimized in genome analysis pipeline on a cloud, based on the results and workload characteristics shown in the previous section. We also explain what approaches we can use do tackle those remaining challenges, especially for relying on a strategy of decoupling compute and storage.

A. Storage Elasticity

Challenge: As shown in the figure 4 and table II, each pipeline has different resource usage patterns and characteristics, such as CPU-intensive or data-intensive, and this tendency often causes resource waste. We hypothoesize that by modeling the pipelines we can flexibly utilize resources. But with the current analytics system, it would be difficult to achieve this degree of elasticity, because a typical analytics system (i.e. Spark with HDFS) requires tightly coupled compute with storage. This collocation concept is always effective to achieve the best performance with an on-premise system, however, it is difficult to scale compute and storage independently. To take full advantage of modern cloud elasticity, genome analysis pipeline should be also decoupled from compute and storage to reduce cost and achieve high scalability.

Moreover, data copying is another potential overhead as shown in the table III. If we continuously execute genome analysis pipelines on an analytics system for an extended period, data loading time to HDFS might be negligible. However, HDFS architecture does not expect to scale or descale underlying nodes frequently so we cannot avoid restructuring HDFS. Otherwise we might accept consuming unnecessary resources if workload size and demands are changing.

Approach: Object storage architecture can overcome the limit of storage scalability. In addition, we can delegate storage durability and availability to cloud. Even though object storage is not a file system, it would be applicable if it can reduce cost and achieve sufficient performance compared to HDFS.

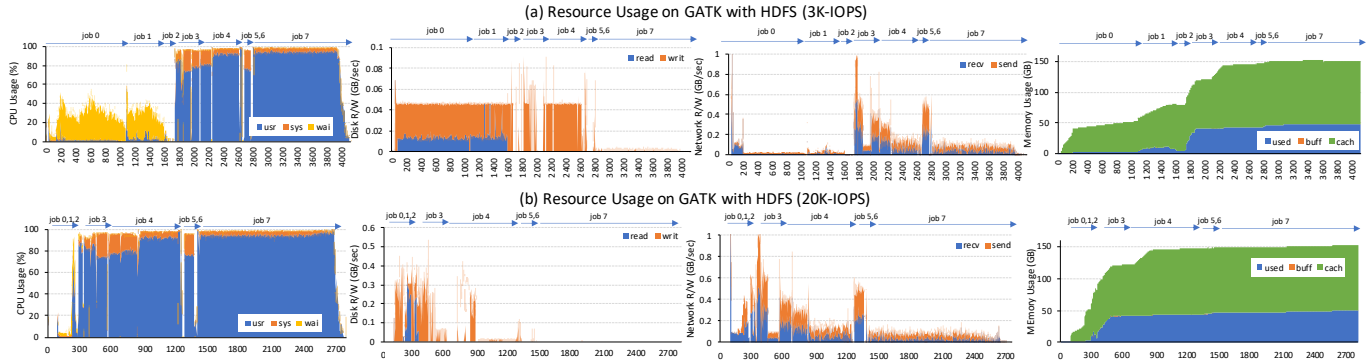

Fig. 5. CPU, Disk, Network, and Memory Resource Usage. (a): on GATK with HDFS (3K-IOPS) and (b): on GATK with HDFS (20K-IOPS)

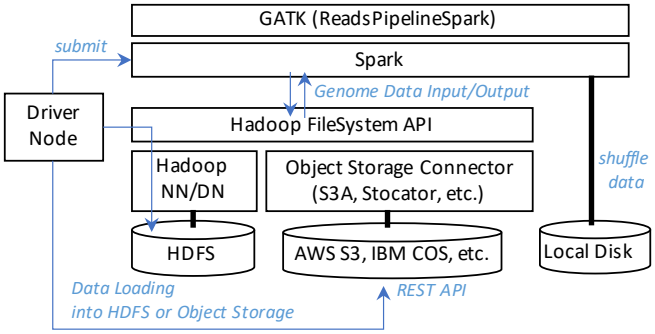

Fig. 6. Overall Design for GATK with Cloud Object Storage

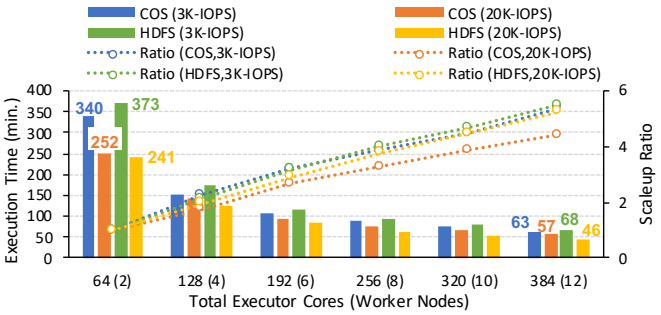

Fig. 7. GATK Scalability Comparison: COS vs. HDFS

B. Compute Elasticity

Challenge: Besides storage elasticity, compute elasticity might help to dynamically adjust the resource usage to our genome workloads. As shown in the Figure 4, required resources and core scalability characteristics are different in each job. Similar to the storage elasticity, we might be able to reduce unnecessary compute resource as well. This result indicates that GATK pipelines have a potential capability to accept different size of resources for each execution stage.

Approach: Execution runtime and framework supports are mandatory to schedule resource dynamically. Spark provides a dynamic resource allocation feature to adjust resource usage that our workloads consume. Based on resource request or remove policy, Spark scheduler increases or decreases additional executors. On the other hand, many applications including GATK are containerized recently, so it is natural to consider running on an orchestration framework such as Kubernetes. Cloud workflow engines (such as Kubeflow and Argo) are intended to manage data pipeline in a Kubernetes native way. Thus, it is important to redesign an overall pipeline and think how we can accelerate pipeline performance on these systems with minimal resources. We do not have enough space to discuss compute elasticity, so we only focus on storage elasticity here, but we plan to integrate it with GATK as a future work.

C. Design and Implementation

Figure 6 describes a new overall design for GATK integrating with COS to achieve cloud elasticity. Instead of HDFS, we enhanced GATK to read/write genome dataset from/to COS. To realize this design, we enable Stocator [14] in Spark. Since GATK and required libraries that heavily depend on HDFS, we modified them to access objects from the GATK analysis pipeline.

V. Performance Evaluation

A. Scalability: GATK with Cloud Object Storage

First we evaluate performance scalability of GATK with COS while changing the number of nodes and cores similar to GATK with HDFS. As we discussed in the section III-A, we still need to utilize local disk even in a COS scenario because Spark requires it to manage shuffle data. Thus, we prepare two evaluation scenarios in COS as well as in HDFS; COS (3K-IOPS) utilizing 3K-IOPS disk for Spark shuffle and COS (20K-IOPS) utilizing 20K-IOPS disk. Unlike previous experiment where disk bandwidth is shared between Spark shuffle and HDFS I/O, the disk bandwidth is almost entirely used by Spark shuffle.

Figure 7 shows core scalability on COS and HDFS, and plots speedup ratio with twelve nodes. In the case of two nodes, the entire pipeline takes 340 minutes, 373 minutes, 252 minutes, and 241 minutes on COS (3K-IOPS), HDFS (3K-IOPS), COS (20K-IOPS) and HDFS (20K-IOPS) respectively. In the 3K-IOPS scenario, COS is always around 10% faster than HDFS, and the execution time with twelve nodes was 63

Fig. 8. CPU, Disk, Network, and Memory Resource Usage on GATK with COS (3K-IOPS)

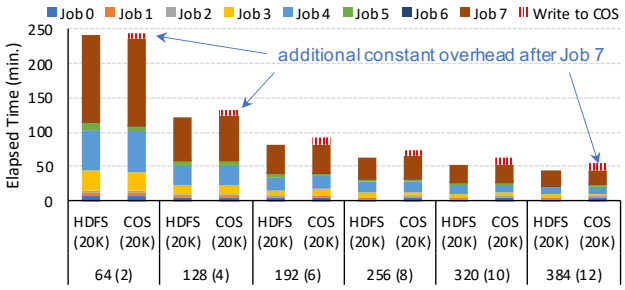

Fig. 9. Comparison of Job Breakdown analysis in GATK with HDFS (20K-IOPS) and COS (20K-IOPS)

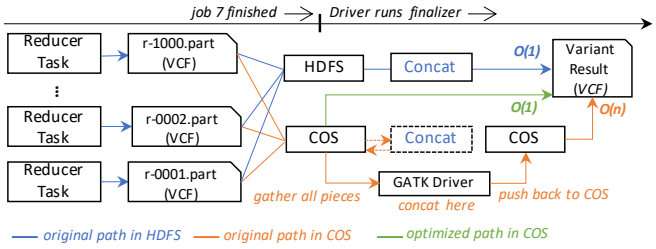

Fig. 10. Comparison of Concat Operation Flow After Finishing Job 7

and 68 minutes in COS and HDFS. On the other hand, COS is always slightly worse than HDFS in the 20K-IOPS scenario; the performance gap between them was 5% in two nodes test, but it accounted for 24% in twelve nodes test, and the elapsed time in COS and HDFS were 57 and 46 minutes. We will discuss why this drawback exists later.

As for the resource usage shown in Figure 8, we can see several notable characteristics compared to the result with HDFS. For example, disk bandwidth is consumed only by shuffle write. Additionally, jobs 0 and 1 finished within 7 minutes even though it took 28 minutes with 3K-IOPS HDFS. That is because network throughput is much larger than disk IO throughput.

B. Optimization for GATK with Cloud Object Storage

Next, we investigate why this slowness is noticeable with COS (20K-IOPS). Figure 9 shows the breakdown analysis that shows how much time each job spends on the scaling test with a 20K IOPS disk. As shown in the graph, Spark job performance is almost the same between HDFS and COS, but COS has an additional part, writing to COS phase. This additional overhead takes around 10 minutes to save a final VCF file output into COS. This is a constant that does not depend on the node scale but on the finalized data size. As a result, the speedup ratio becomes gradually worse in the twelve node test since this constant cost is relatively dominant in the full computation, even though the pipeline computation part itself is competitive with HDFS.

Why does COS have this overhead when HDFS does not? The reason comes from the difference in the supported Hadoop FileSystem API between HDFS and COS. Figure 10 shows a diagram and operational flow of the final phase in a variant searching pipeline. Blue, orange, and green lines represent original flow using HDFS, original flow using COS, and optimized flow using COS without any concat operation respectively. The last stage of job 7 manages a large number of reducer tasks that generate a piece of the final VCF file, and then persist them into HDFS or COS. After finishing job 7, the GATK main driver explicitly calls a concat operation in Hadoop FileSystem API which merges them into a single file. HDFS implements all operations including concat, which can complete the concat operation without any copies inside or outside the cluster. An HDFS client has the capability to read these pieces as a file at runtime, so it does not need to physically merge them into one. As a consequence, this concat operation finishes immediately in HDFS. On the other hand, an object storage connector to COS does not implement the concat operation in its Hadoop compatible file system. Although the object storage connector imitates a Hadoop FileSystem, it is essentially not easy to support all APIs due to the difference of backend storage implementation and a general limitation existing in S3 compatible object storage system. As a result, the GATK main driver performs error handling; it copies all pieces into the local driver first, merges them into a final VCF file, then pushes it back to an object in COS. Thus, this additional constant data copying overhead in the finalization phase always exists in GATK with COS.

We have several possible approaches to eliminate this overhead. One approach is to modify GATK internal code to stop calling concat. Another approach is for the object storage connector to simply pass through concat operations without raising errors. In both approaches, a client must understand how to read these pieces, but it can suppress unnecessary data copy. We also implemented code to skip this file merge process to demonstrate how to reduce this overhead. As a result, by using all of these techniques, the performance of COS tests can be uniformly reduced by 10 minutes in the final sink phase; COS (3K-IOPS) is 28% faster than HDFS (3K-IOPS), and COS (20K-IOPS) shows almost same performance in HDFS (20K-IOPS).

C. Cost: Price Per Performance

Finally, we compare actual costs between COS and HDFS in our scenarios. Figure 11 compares costs while changing

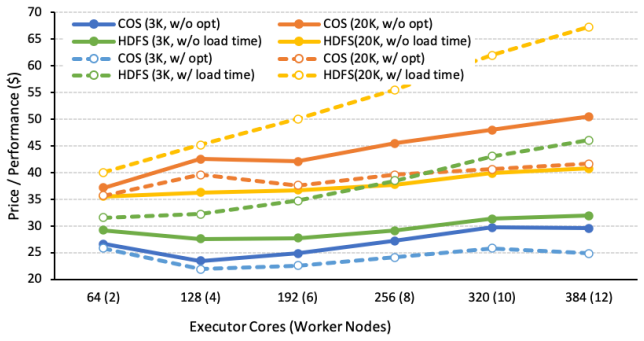

Fig. 11. Price/Performance Comparison: COS vs. HDFS

node scaling and disk configurations. We calculate total cost based on a public price list and then divide it by elapsed time. The First scenario focuses on pipeline computation time only, represented with solid lines. COS always achieves better cost performance than HDFS with 3K-IOPS disk. While utilizing 20K-IOPS disk, HDFS outperforms COS due to the difference in elapsed time for entire pipeline execution. Comparing 3K-IOPS with 20K-IOPS, cost performance in 3K-IOPS is 15 - 60% better than 20K-IOPS. This is because 20K-IOPS is 10x more expensive than 3K-IOPS.

The Next scenario compares COS with HDFS when eliminating a concat overhead in OCS and loading all data into HDFS, represented with dotted lines. If we start from data loading phase into HDFS, analytics pipeline must wait to complete all data transfer. Moreover, we cannot load data before starting the cluster. Therefore, we appended the additional transfer time (i.e. 30 minutes) to the elapsed time in HDFS result. In COS result, we can manage data transfer task independently, so we do not account the cost here. In addition, COS results include the optimization which removes an additional data sink overhead (i.e. 10 minutes). In a such situation, as a result, it can achieve up to 67% cost savings with 3K-IOPS disk, and up to 61% cost savings with 20K-IOPS disk on twelve nodes.

VI. Conclusion

In this paper we investigate the performance characteristics of GATK using Spark with HDFS and identify scalability issues on a modern cloud. Based on a quantitative analysis, we introduce a new approach to utilize cloud object storage in GATK instead of HDFS, which helps decouple compute and storage. We demonstrate how this approach contributes to performance scalability and cost saving in a cloud. We also reveal an existing overhead when utilizing cloud object storage in current GATK. By mitigating this performance issue, we finally confirm GATK using COS can achieve a 28% performance improvement over than HDFS while using a slower but inexpensive disk, and completely the same performance with HDFS using a faster but more expensive disk. Moreover, we show that it can achieve up to 67% cost savings to complete all genome analysis pipeline including data loading time into HDFS.

References

[1] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo, "The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, vol. 20, no. 9, pp. 1297–1303, Jul. 2010.

[2] G. A. Auwera, M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, A. Levy-Moonshine, T. Jordan, K. Shakir, D. Roazen, J. Thibault, E. Banks, K. V. Garimella, D. Altshuler, S. Gabriel, and M. A. DePristo, "From FastQ data to high-confidence variant calls: The genome analysis toolkit best practices pipeline," *Current Protocols in Bioinformatics*, vol. 43, no. 1, Oct. 2013.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '10)*. USENIX Association, 2010, pp. 10–10.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[5] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares, and Xiao Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010, pp. 1–9.

[6] S. Chaisiri, R. Kaewpuang, B. Lee, and D. Niyato, "Cost minimization for provisioning virtual servers in amazon elastic compute cloud," in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2011, pp. 85–95.

[7] A. Trivedi, P. Stuedi, J. Pfefferle, R. Stoica, B. Metzler, I. Koltsidas, and N. Ioannou, "On the [ir]relevance of network performance for data processing," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. USENIX Association, Jun. 2016.

[8] H. Mushtaq, F. Liu, C. Costa, G. Liu, P. Hofstee, and Z. Al-Ars, "SparkGA: A spark framework for cost effective, fast and accurate DNA analysis at scale," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*, ser. ACM-BCB '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 148–157. [Online]. Available: https://doi.org/10.1145/3107411.3107438

[9] H. Mushtaq, N. Ahmed, and Z. Al-Ars, "SparkGA2: Production-quality memory-efficient apache spark based genome analysis framework," *PLOS ONE*, vol. 14, no. 12, p. e0224784, Dec. 2019.

[10] A. Xiao, S. Dong, C. Liu, L. Zhang, and Z. Wu, "Cloudgt: A high performance genome analysis toolkit leveraging pipeline optimization on spark," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2018, pp. 1343–1350.

[11] C. H. A. Costa, C. Misale, F. Liu, M. Silva, H. Franke, P. Crumley, and B. D'Amora, "Optimization of genomics analysis pipeline for scalable performance in a cloud environment," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2018, pp. 1147–1154.

[12] P. Zhou, Z. Ruan, Z. Fang, M. Shand, D. Roazen, and J. Cong, "Doppio: I/o-aware performance analysis, modeling and optimization for in-memory computing framework," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 22–32.

[13] L. Rupprecht, R. Zhang, B. Owen, P. Pietzuch, and D. Hildebrand, "SwiftAnalytics: Optimizing object storage for big data analytics," in *2017 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, Apr. 2017.

[14] G. Vernik, M. Factor, E. K. Kolodner, P. Michiardi, E. Ofer, and F. Pace, "Stocator: Providing high performance and fault tolerance for apache spark over object storage," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2018, pp. 462–471.

[15] T. Chiba and T. Onodera, "Workload characterization and optimization of tpc-h queries on apache spark," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 112–121.

[16] "A global reference for human genetic variation," *Nature*, vol. 526, no. 7571, pp. 68–74, Sep. 2015.