IBM Research

Investigating Genome Analysis Pipeline Performance on GATK with Cloud Object Storage

Tatsuhiro Chiba and Takeshi Yoshimura

IBM Research

Agenda

- Background, Motivation and Challenges
- Performance Scalability Analysis
  - GATK with Spark/HDFS
  - GATK with Spark/COS
- Performance and Cost Optimization
- Summary

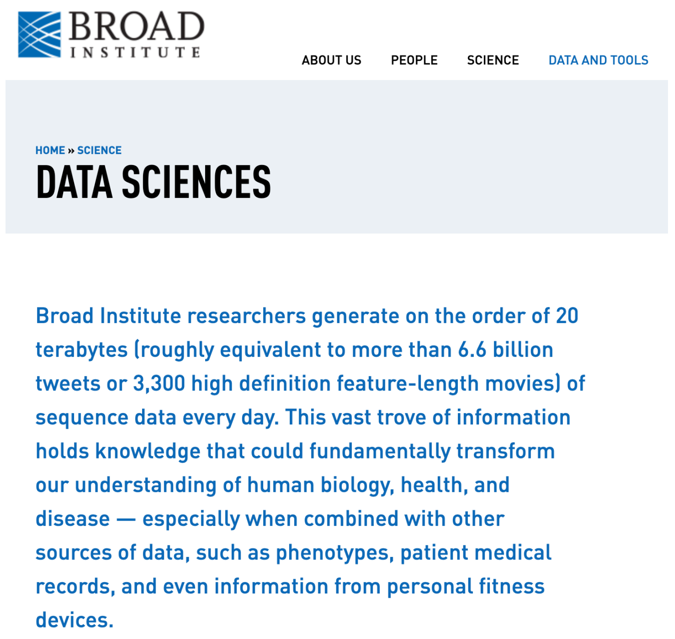
Genome Analysis and GATK

Genome Analysis

- Next Generation Sequencing (NGS) generates tons of genome sequence data
- DNA structure analysis is essential for medical & life scient research
- High scalable system is always required to optimize genome analysis
    1. Accelerating Speed
    2. Reducing Cost

What is GATK?

- most widely used genome analysis toolkit written in Java
- contains many tools and utilities, such as data preprocessing/cleanup, sequence data quality control by recalibration, HaplotypeCaller, etc.
- Users can build their own workflow pipeline to perform variant discovery analysis by combining those tools



BROAD INSTITUTE

ABOUT US PEOPLE SCIENCE DATA AND TOOLS

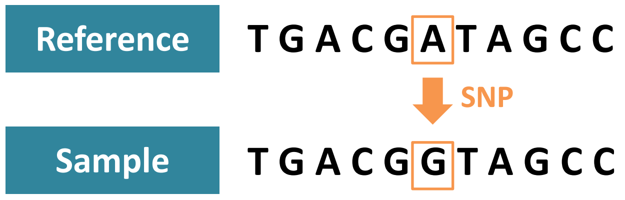
HOME » SCIENCE

DATA SCIENCES

Broad Institute researchers generate on the order of 20 terabytes (roughly equivalent to more than 6.6 billion tweets or 3,300 high definition feature-length movies) of sequence data every day. This vast trove of information holds knowledge that could fundamentally transform our understanding of human biology, health, and disease — especially when combined with other sources of data, such as phenotypes, patient medical records, and even information from personal fitness devices.

https://www.broadinstitute.org/data-sciences

Example of Variant: SNP

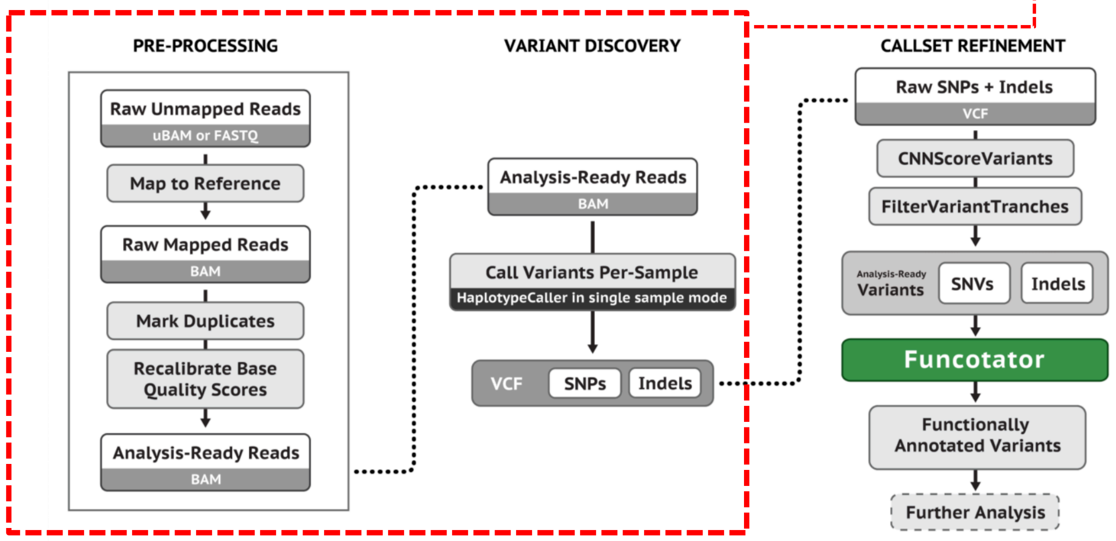


GATK Best Practices and Spark

- GATK Team defines typical variant discovery workflows as **GATK Best Practices**
- GATK leverages Spark to achieve node-level / core-level scalability
- ***ReadsSparkPipeline***, a Spark program in GATK, performs a set of well-defined workflows

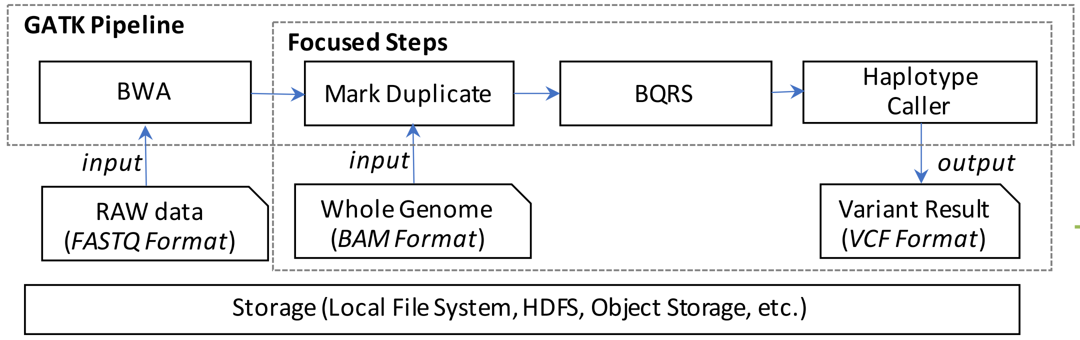
typical pipeline for variant discovery

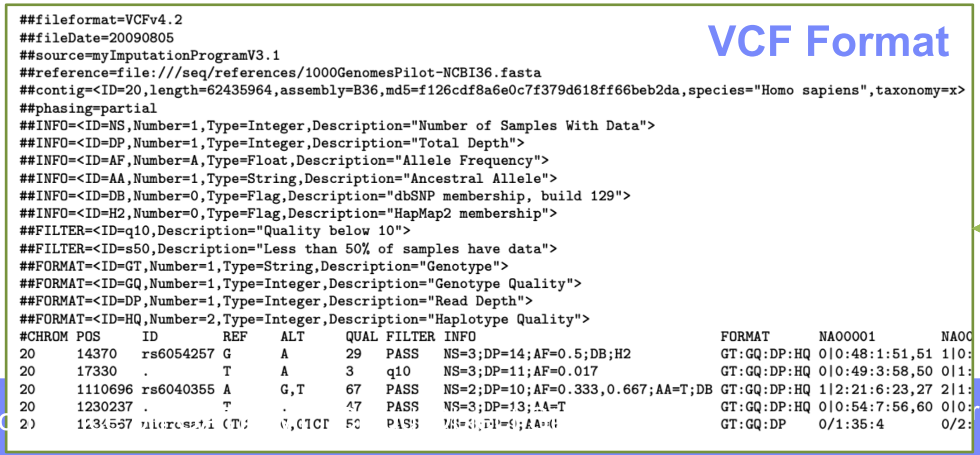
Main steps for Germline Single-Sample Data



https://gatk.broadinstitute.org/hc/en-us/articles/360035535932-Germline-short-variant-discovery-SNPs-Indels-

A Chain of Spark Jobs in ReadsSparkPipeline


GATK Pipeline
Focused Steps
BWA
Mark Duplicate
BQRS
Haplotype Caller
input
input
output
RAW data (FASTQ Format)
Whole Genome (BAM Format)
Variant Result (VCF Format)
Storage (Local File System, HDFS, Object Storage, etc.)



```
##fileformat=VCFv4.2
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens",taxonomy=x>
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF    ALT     QUAL FILTER INFO                              FORMAT      NA00001        NA00
20     14370   rs6054257 G      A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2           GT:GQ:DP:HQ 0|0:48:1:51,51 1|0:
20     17330   .         T      A       3    q10    NS=3;DP=11;AF=0.017               GT:GQ:DP:HQ 0|0:49:3:58,50 0|1:
20     1110696 rs6040355 A      G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:
20     1230237 .         T      .       47   PASS   NS=3;DP=13;AA=T                   GT:GQ:DP:HQ 0|0:54:7:56,60 0|0:
20     1234567 microsat1 GTC    G,GTCT  50   PASS   NS=3;DP=9;AA=G                    GT:GQ:DP    0/1:35:4       0/2:
```

Migrating Genome Analysis Platform from Local to Clouds

What benefits do we want to achieve after migration?

- Cost Efficiency
- Performance Scalability

What is needed to take full advantage of Cloud Capabilities?

- Decoupling ***compute*** and ***storage*** for Resource Elasticity
  - Storage Elasticity: Cloud Object Storage rather than self-managed local storage
  - Compute Elasticity: Containers rather than self-managed nodes
- Adjusting resource demands dynamically

What problems do we need to solve?

- GATK + Spark reference architecture heavily depends on HDFS (i.e. tightly-coupled)
- Data loading and system setup are not negligible overhead
- Performance characteristics in analysis pipeline are different

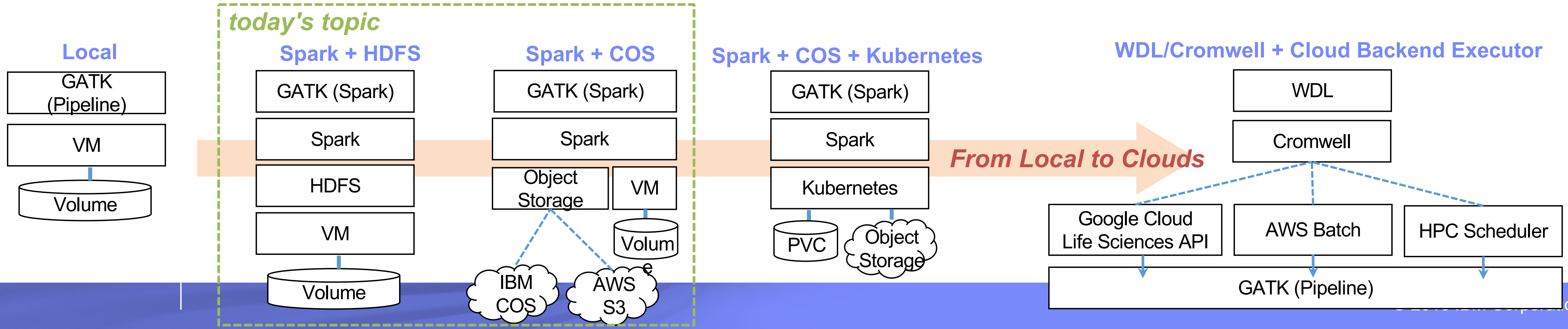
Challenges and Summary of Contributions

Challenges

- reveals **performance characteristics** in Genome analysis pipeline
- decouples compute and **storage** to exploit cloud elasticity
- adjusts resource capacity dynamically based on the pipeline demands

Contributions

- Identifies performance scalability and elasticity issues in Genome analysis pipeline running on GATK with Spark/HDFS
- Provides a new best practice to use **Cloud Object Storage** instead of **HDFS**
- Demonstrates the entire pipeline improvement
  - Performance: **up to 28%** faster
  - Cost: **up to 67%** cost saving



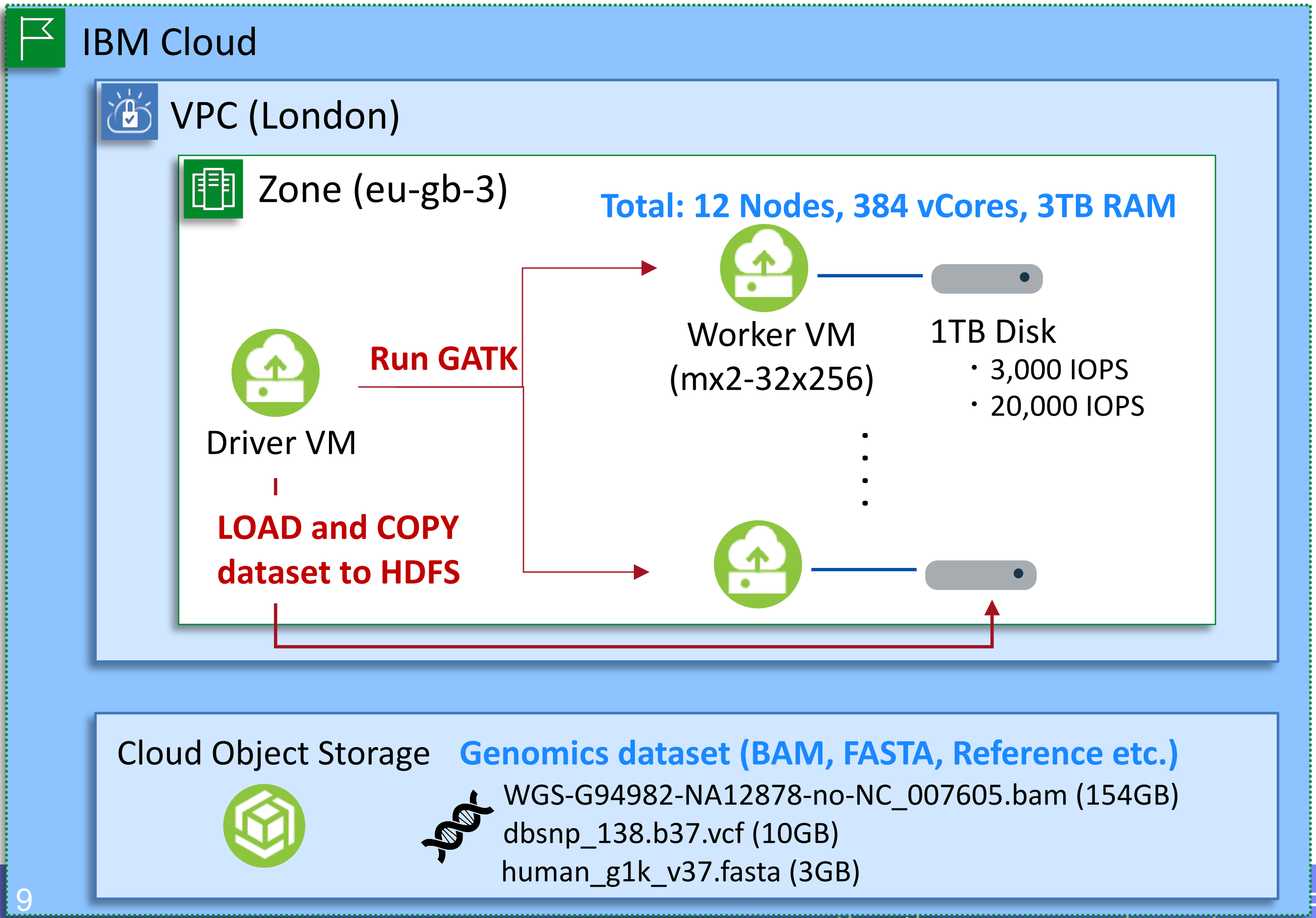
Agenda

- Background, Motivation and Challenges
- Performance Scalability Analysis
  - GATK with Spark/HDFS
  - GATK with Spark/COS
- Performance and Cost Optimization
- Summary

GATK Performance Scalability with Spark/HDFS

Performance Analysis at Scale: Spark/HDFS (1/4)

- Built Spark/HDFS cluster on IBM Cloud
  - Spark w/ HDFS (20K-IOPS): Attached 10 IOPS/GB profiled 1TB volume ... (up to 20,000 IOPS)
  - Spark w/ HDFS (3K-IOPS): Attached 3 IOPS/GB profiled 1TB volume ... (up to 3,000 IOPS)
  - Utilized both storage volumes independently to understand *how disk speed makes an impact to the performance*



Software and Spark Configuration

Software

```
GATK:    4.1.7.0 (latest w/ COS support)
Spark:   2.4.5 (latest)
Hadoop:  2.7.7
JVM:     1.8.0_242 (OpenJ9)
```

Spark Config

```
Executors/Node: 4
Cores/Executor: 8
Mem/Executor  : 35GB(heap), 15GB(off-heap)
```

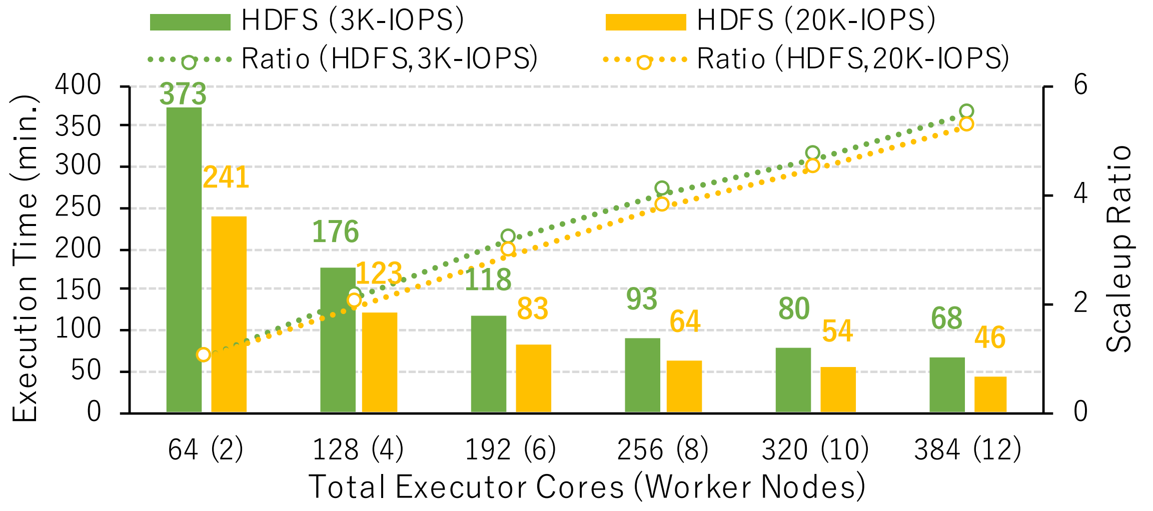
Execution Command on Driver Node

```
$ gatk ReadsPipelineSpark
        -I WGS-G94982-NA12878-no-NC_007605.bam
        -O WGS-G94982-NA12878.vcf
        -R human_g1k_v37.fasta
        --known-sites dbsnp_138.b37.vcf.gz
        -pairHMM AVX_LOGLESS_CACHING_OMP
        --max-reads-per-alignment-start 10
        --java-options "-XX:MaxDirectMemorySize=8589934592"
        --disable-sequence-dictionary-validation true
        -- --spark-runner SPARK --spark-master spark://master:7077
        --executor-cores 8 --num-executors 48 --executor-memory 35g ...
```

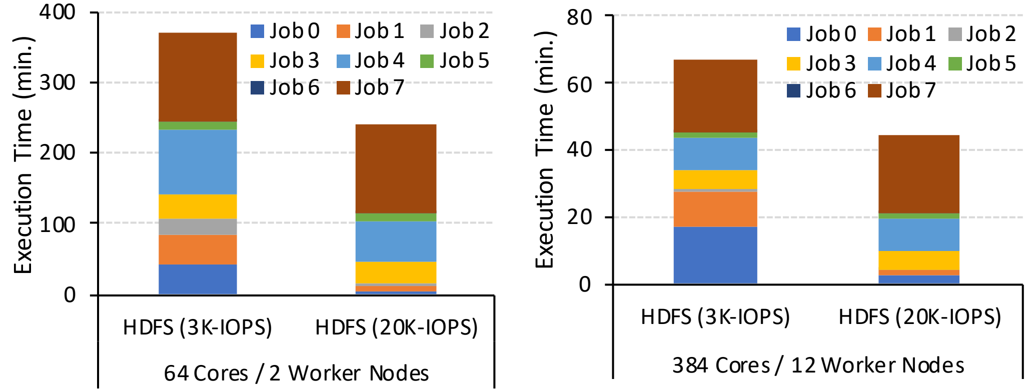
Performance Analysis at Scale: Spark/HDFS (2/4)

- GATK Pipeline has a good scalability (5.5x scaling against 6x resources)
- Based on Job execution time breakdown, 20K-IOPS can reduce Spark Job 0 and 1 time drastically
- Compared to speed up ratio in each job, most of jobs has a good scalability

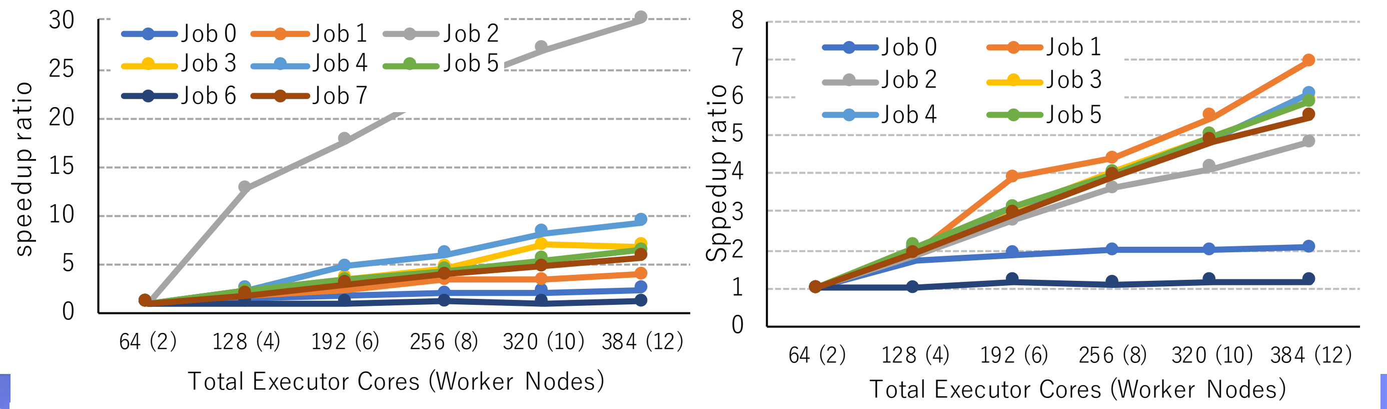
Weak Scaling Performance



Breakdown analysis of Job execution time (2 nodes vs 12 nodes)



Speed up ratio in each job (Left: HDFS w/ 3K-IOPS, Right: HDFS w/ 20K-IOPS)



Performance Analysis at Scale: Spark/HDFS (3/4)

- Job 0 and 1 are disk read heavy (*loading Genome Data from HDFS*)
- Job 3, 4, 5 are are disk write and network heavy (writing intermediate data and shuffling them between nodes)
- Last Job 7 has many shuffle read and CPU intensive (sorting in-memory data and writing a result into HDFS)

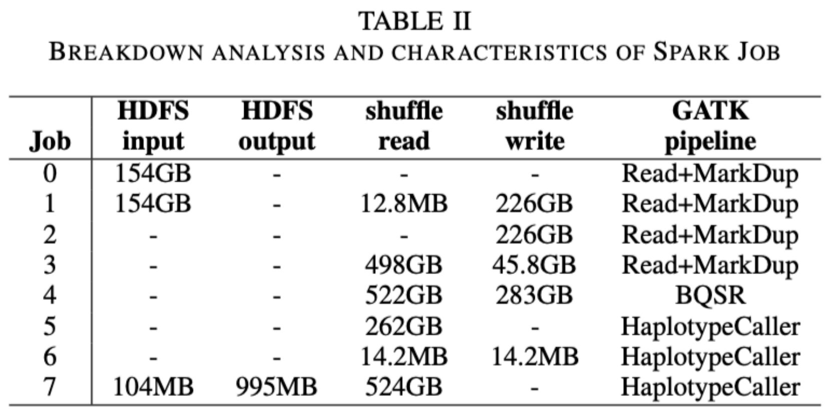
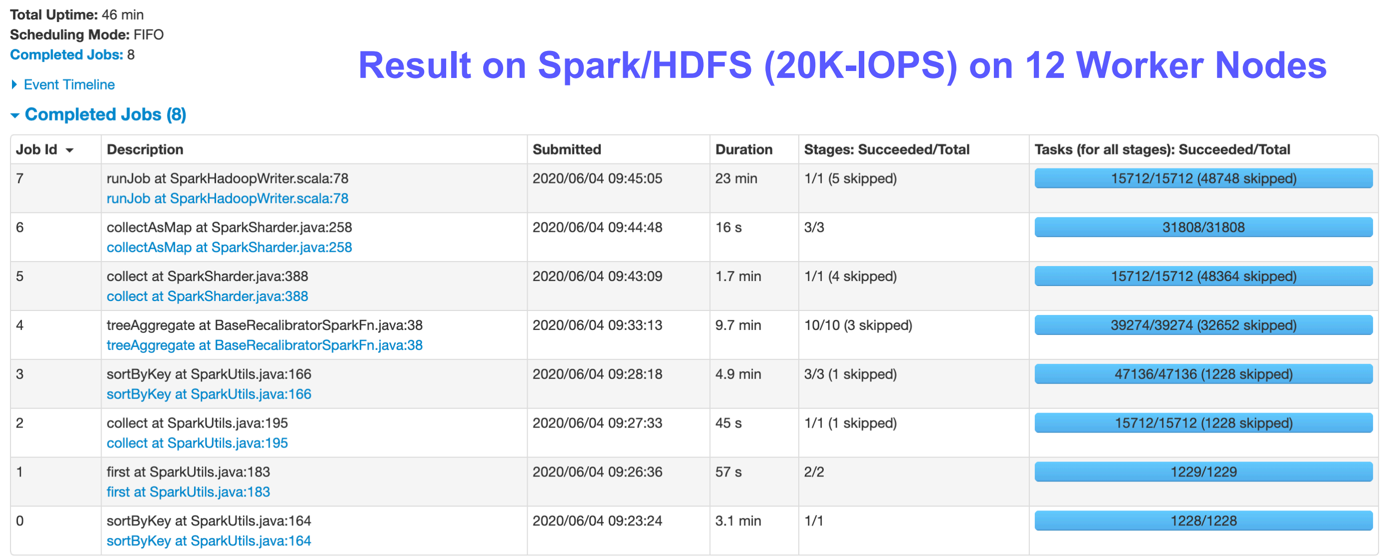


TABLE II
BREAKDOWN ANALYSIS AND CHARACTERISTICS OF SPARK JOB

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

Result on Spark/HDFS (20K-IOPS) on 12 Worker Nodes

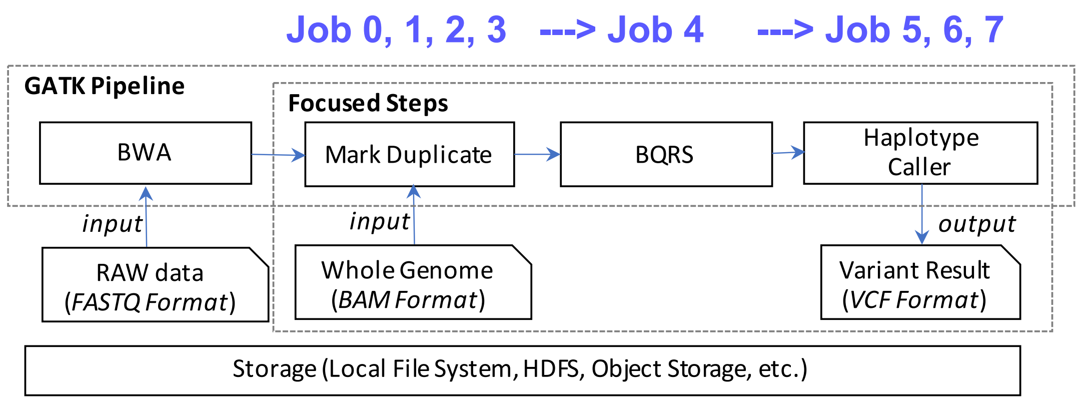


Total Uptime: 46 min
Scheduling Mode: FIFO
Completed Jobs: 8

▸ Event Timeline

▾ Completed Jobs (8)

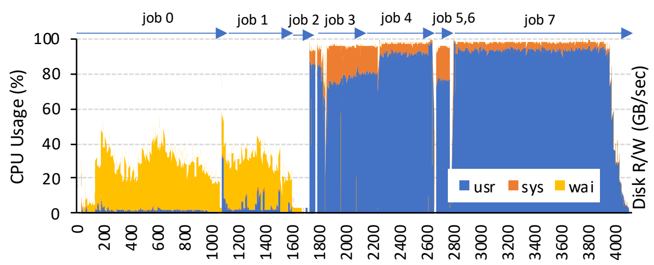
Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
7	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2020/06/04 09:45:05	23 min	1/1 (5 skipped)	15712/15712 (48748 skipped)
6	collectAsMap at SparkSharder.java:258 collectAsMap at SparkSharder.java:258	2020/06/04 09:44:48	16 s	3/3	31808/31808
5	collect at SparkSharder.java:388 collect at SparkSharder.java:388	2020/06/04 09:43:09	1.7 min	1/1 (4 skipped)	15712/15712 (48364 skipped)
4	treeAggregate at BaseRecalibratorSparkFn.java:38 treeAggregate at BaseRecalibratorSparkFn.java:38	2020/06/04 09:33:13	9.7 min	10/10 (3 skipped)	39274/39274 (32652 skipped)
3	sortByKey at SparkUtils.java:166 sortByKey at SparkUtils.java:166	2020/06/04 09:28:18	4.9 min	3/3 (1 skipped)	47136/47136 (1228 skipped)
2	collect at SparkUtils.java:195 collect at SparkUtils.java:195	2020/06/04 09:27:33	45 s	1/1 (1 skipped)	15712/15712 (1228 skipped)
1	first at SparkUtils.java:183 first at SparkUtils.java:183	2020/06/04 09:26:36	57 s	2/2	1229/1229
0	sortByKey at SparkUtils.java:164 sortByKey at SparkUtils.java:164	2020/06/04 09:23:24	3.1 min	1/1	1228/1228

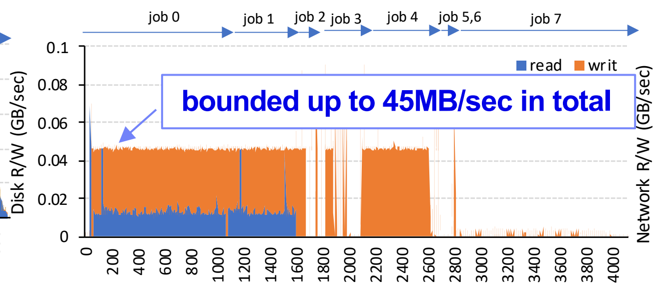

Job 0, 1, 2, 3 ---> Job 4 ---> Job 5, 6, 7
GATK Pipeline
Focused Steps
BWA
Mark Duplicate
BQRS
Haplotype Caller
input
input
output
RAW data (FASTQ Format)
Whole Genome (BAM Format)
Variant Result (VCF Format)
Storage (Local File System, HDFS, Object Storage, etc.)

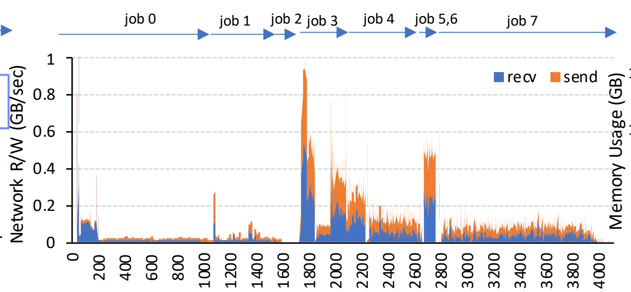
Performance Analysis at Scale: Spark/HDFS (4/4)

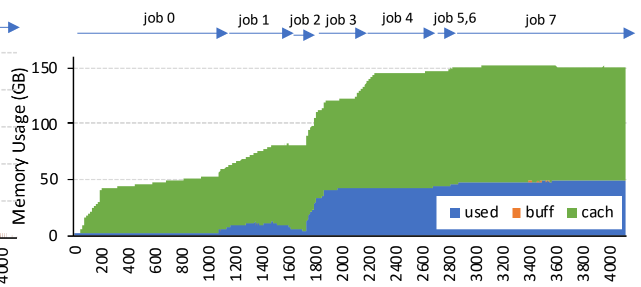
- In 3K-IOPS, disk r/w bandwidth is bounded up to 45 MB/sec in total
- Disk read happens in Job 0 and 1 only
- Almost all shuffled data resides in memory as file cache

Resource Usage on HDFS (3K-IOPS)

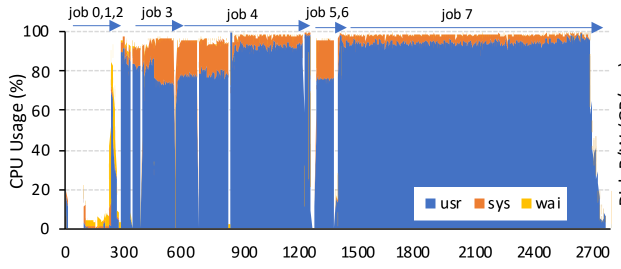


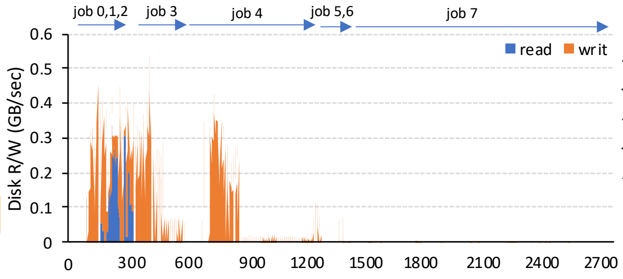


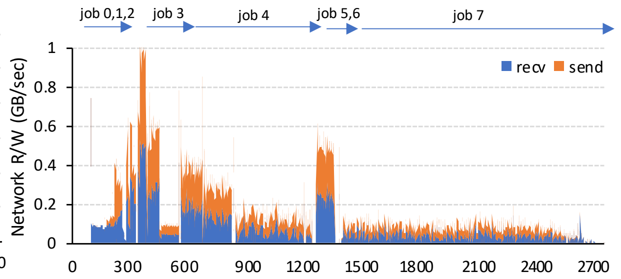


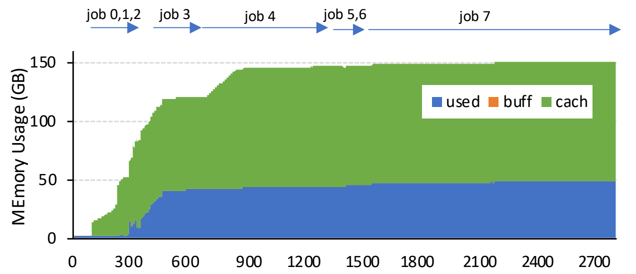


Resource Usage on HDFS (20K-IOPS)









What Challenges Still Exist?

Storage Elasticity

- HDFS (20K-IOPS) is quite faster, but spark jobs does **not always require high-throughput disk**
- Hard to **resize HDFS capacity/nodes**, and need to keep paying high cost even if not required
- load time (**copying data to HDFS**) is not negligible
- Cloud Object Storage (COS), such as AWS S3 & IBM Cloud Object Storage, has a capability to overcome the limit of storage scalability → ***Can we utilize COS instead of HDFS for GATK?***

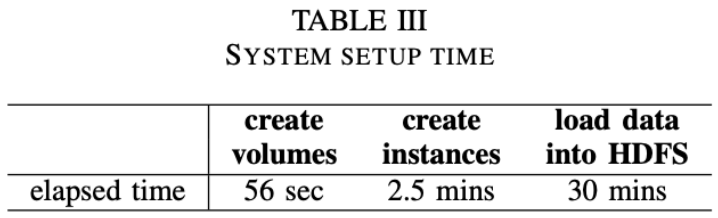


TABLE III
SYSTEM SETUP TIME

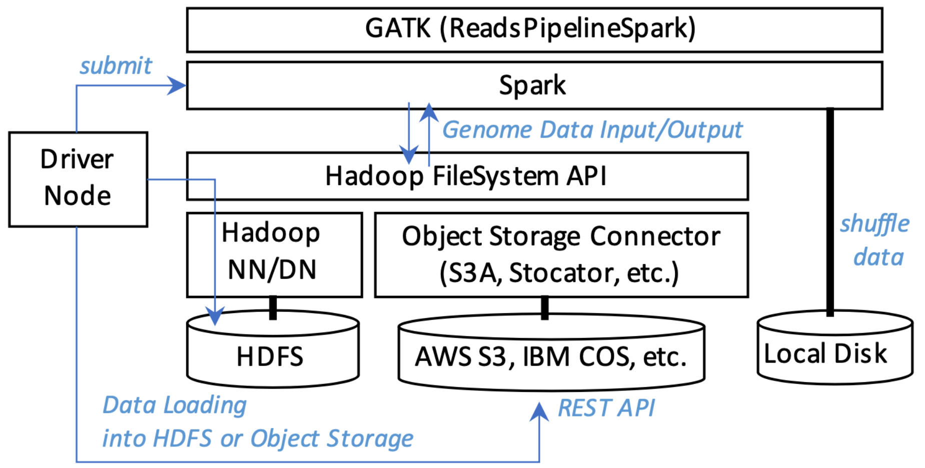
	create volumes	create instances	load data into HDFS
elapsed time	56 sec	2.5 mins	30 mins

HDFS vs. Cloud Object Storage

- POSIX File System vs REST API Based Storage
- Performance depends on **DISK bandwidth vs. Network bandwidth**

Architecture Overview

- Spark (Connector) can access COS via Hadoop FileSystem API
- Data load time (COS): only once even if resizing compute resource
- Data load time (HDFS): every time if resizing compute resource

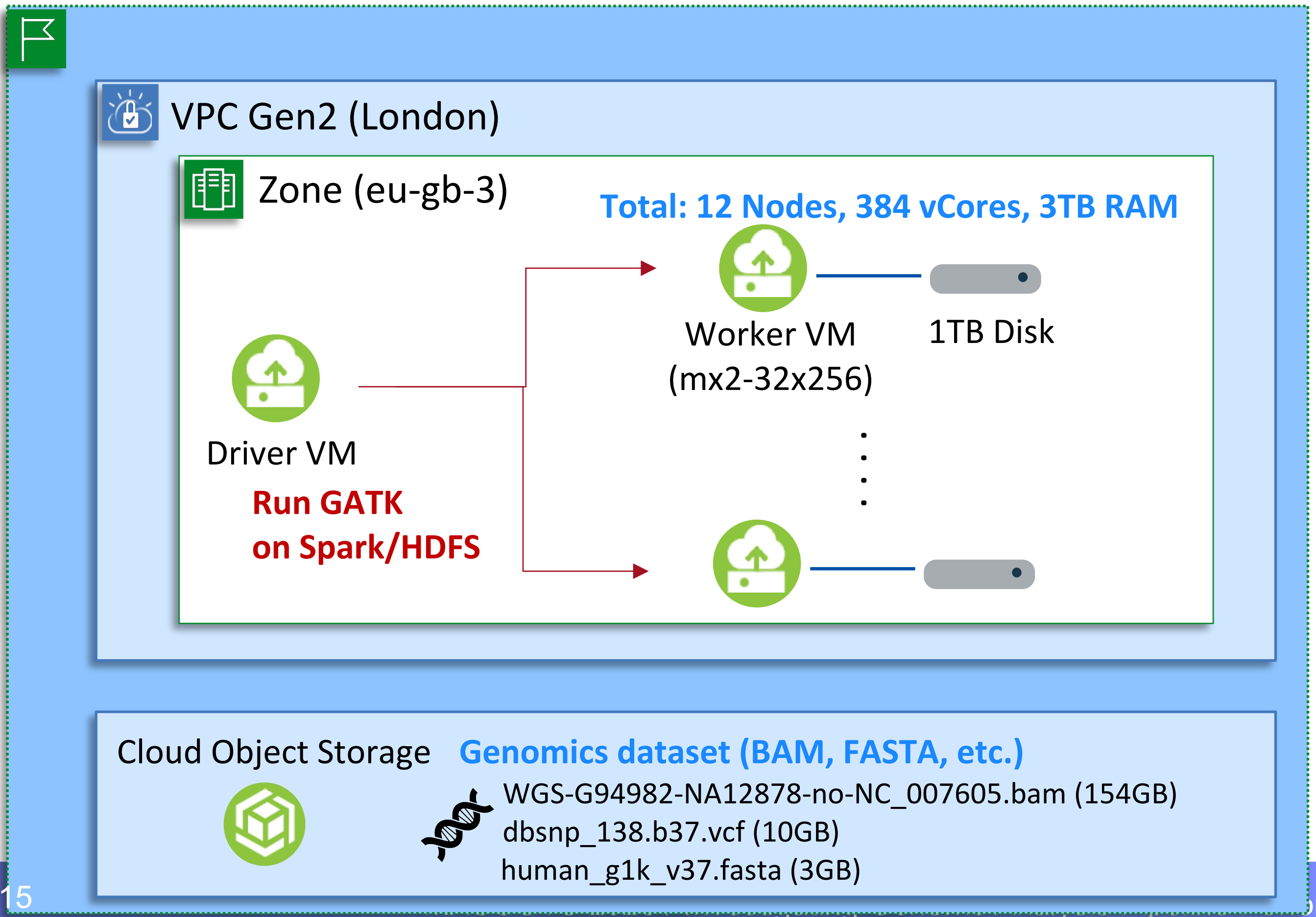


GATK Performance Scalability with Spark/COS

Experiment Settings: GATK on Spark with COS vs. HDFS

- Read and write genomics dataset not from/to **HDFS** but from/to **COS** directly
- Modified GATK to use cos://bucket/object, and integrated with Stocator (Spark Connector for COS)
- Compared systems: Spark w/ HDFS (3K-IOPS), w/ HDFS (20K-IOPS), w/ COS (3K-IOPS), and w/ COS (20K-IOPS)

VPC Architecture



WGS-G94982-NA12878-no-NC_007605.bam (154GB)
dbsnp_138.b37.vcf (10GB)
human_g1k_v37.fasta (3GB)

Software and Spark Configuration

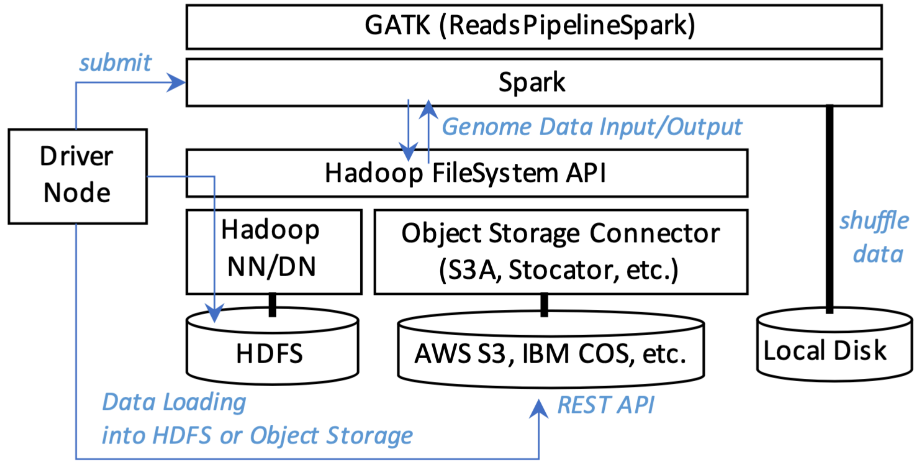
Software

```
GATK:   4.1.7.0 (latest w/ COS support)
Spark:  2.4.5 (latest)
Hadoop: 2.7.7
JVM:    1.8.0_242 (OpenJ9)
```

Spark Config

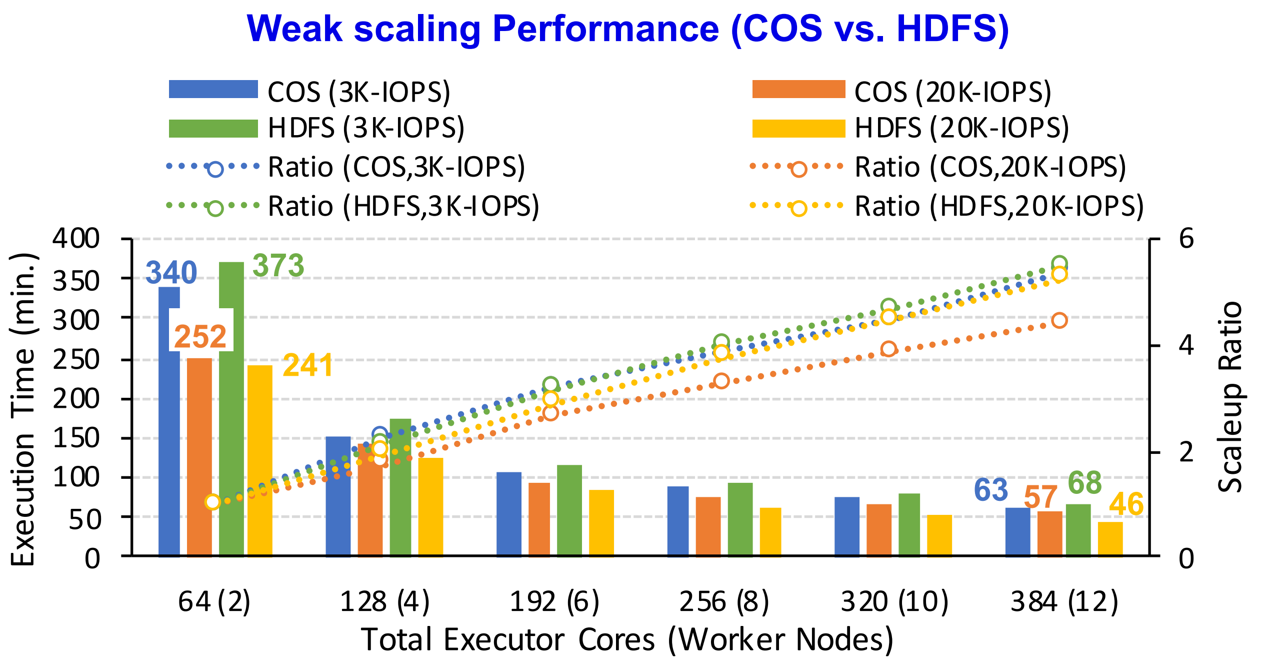
```
Executors/Node: 4
Cores/Executor: 8
Mem/Executor  : 35GB(heap), 15GB(off-heap)
```

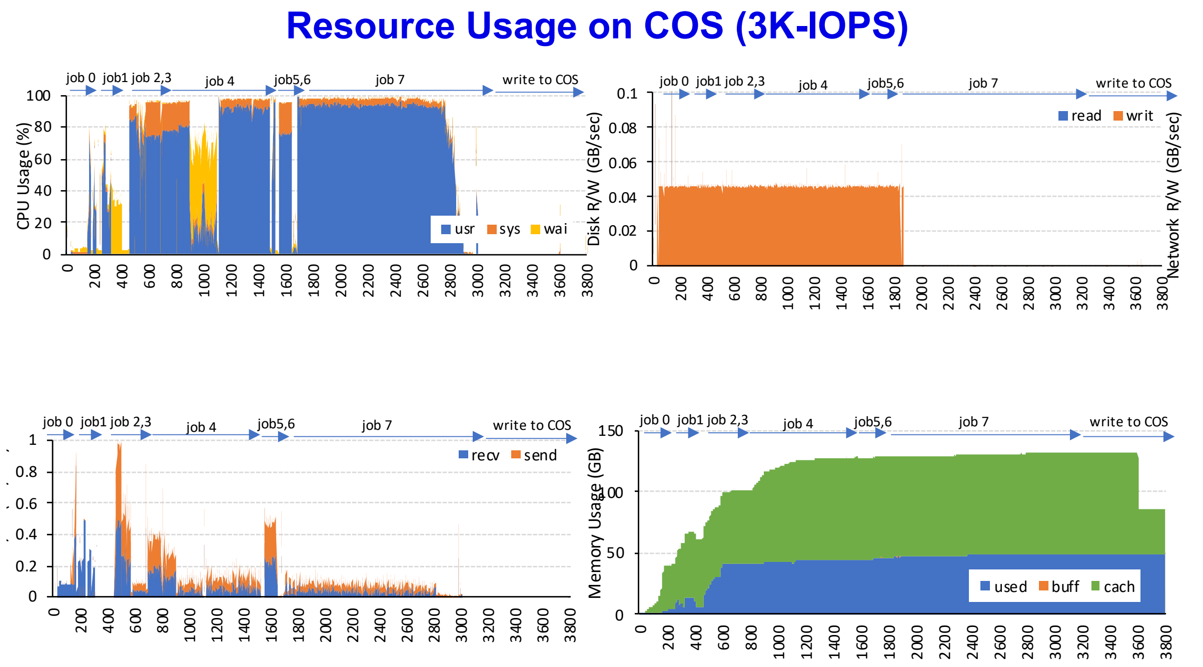
Software Stack Overview



Weak Scaling Performance – COS vs. HDFS

- GATK Pipeline has a good scalability in both cases basically (achieved 5.5x scaling against 6x resources)
- COS (20K-IOPS) case is slightly worse scaling than other three (explain it later)
- As for resource usage on Spark w/ COS, disk bandwidth is consumed only by shuffle write
- Instead, Spark w/ COS can highly utilize network capacity

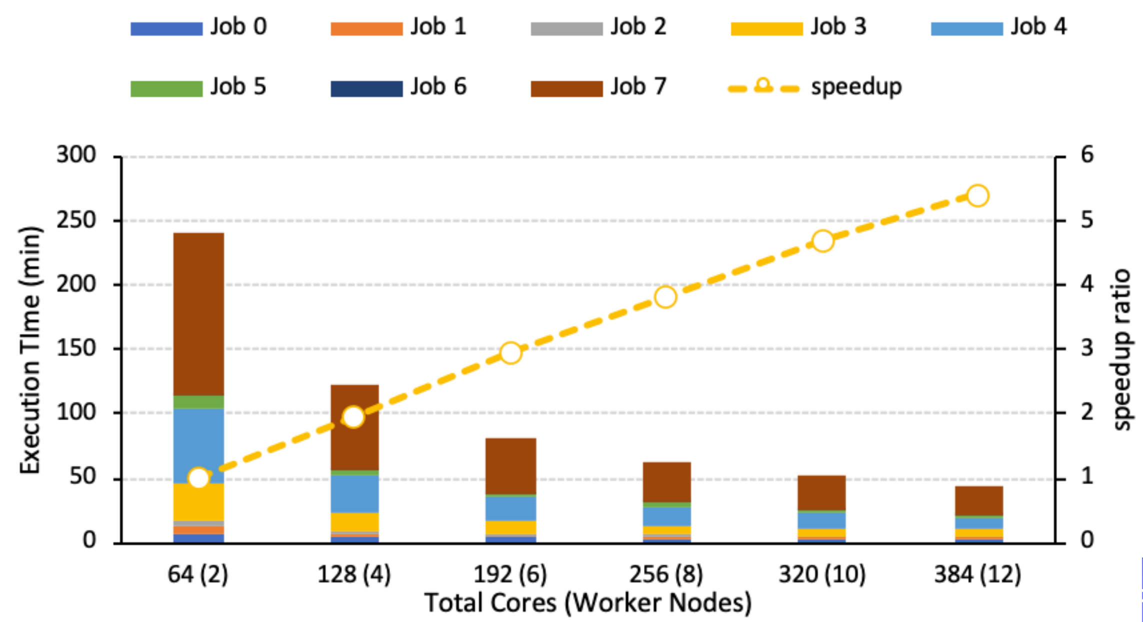




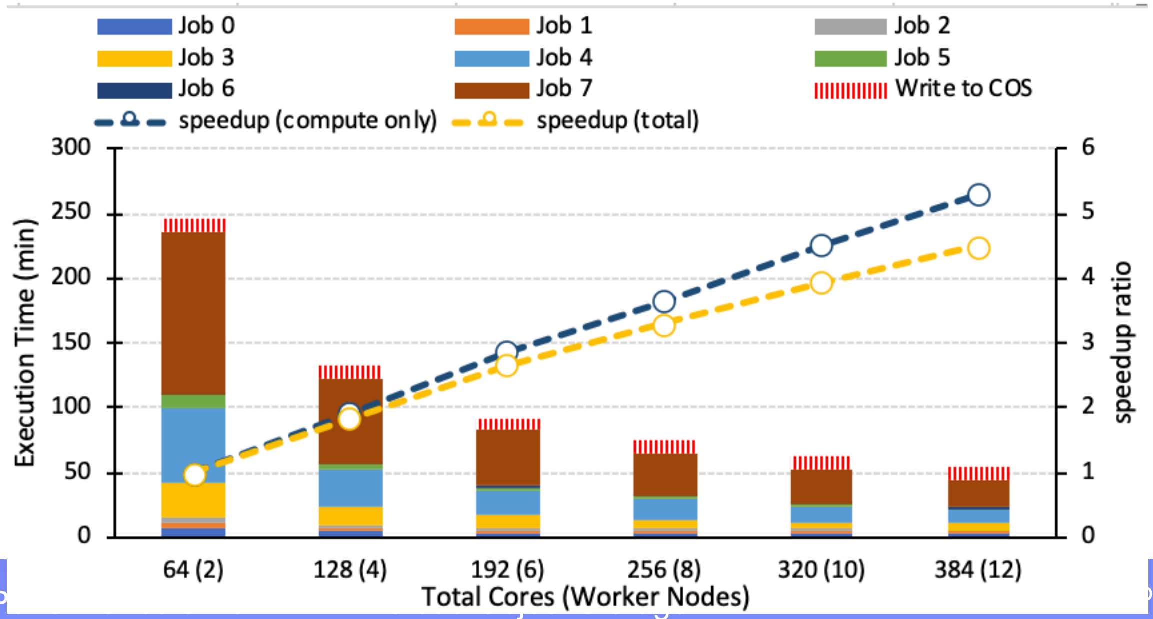
Why GATK with COS does not scale well in many nodes?

- GATK with COS has an additional cost to write a result (VCF file)
  - Takes around 10 mins for finalizing 1GB output file onto COS, depending on the result size
- **Why HDFS does not have the overhead, but COS has?**
  - Difference in the supported file system operations between HDFS and COS
  - HDFS supports (logical) *concat operation* on the file system inside, but COS does not
- **GATK explicitly calls concat operation in the finalization phase**
  - HDFS can complete concat operation without any copies (just logically move it on HDFS)
  - Object Storage connector cannot support *concat operation* directly (several copies happen between systems

GATK with HDFS (20K-IOPS)



GATK with COS (20K-IOPS)



Performance and Cost Optimization for GATK with Spark/COS

Protocol Detail and Optimization in Finalization Phase

HDFS

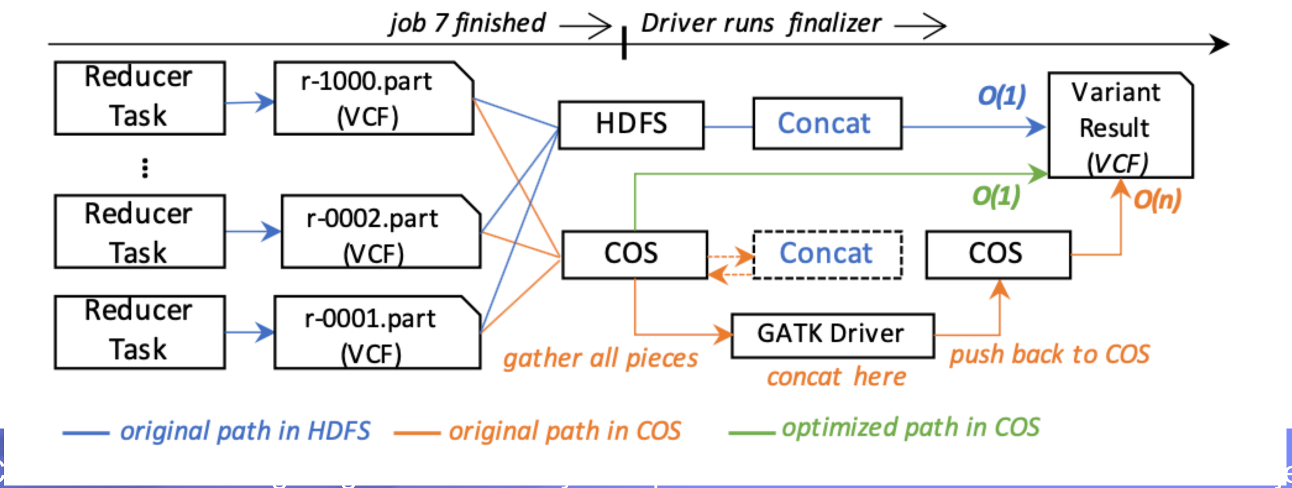
- each reducer tasks store the parts of files into HDFS
- Main program calls Concat Hadoop FileSystem API → just move and set a representative name to VCF file

COS (Original)

- each reducer tasks store the parts of files into COS
- Main program calls Concat Hadoop FileSystem API → NotSupportedOperation Exeception
- As an exception handling, main driver gathers all pieces locally, merges them, and stores it back to COS

COS (Opt)

- each reducer tasks store the parts of files into COS
- Main program calls Concat Hadoop FileSystem API → implemented a dummy concat operation
- delegates concat task to client (VCF reader)
- constantly eliminate data sink time (i.e. 10 mins) → COS (3K-IOPS) is **up to 28% faster** than HDFS (3K-IOPS)



Price/Performance Comparison: COS vs. HDFS

Solid Line:

- Comparing computation pipeline time in COS (Original) with HDFS
- COS (Original) always achieves better cost performance than HDFS
- 3K-IOPS is **15 – 60 % better** cost performance than 20K-IOPS

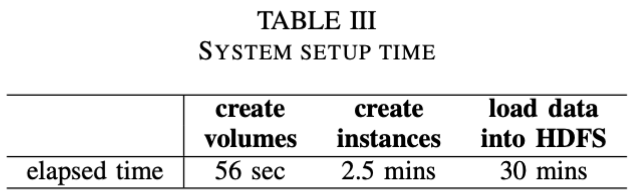
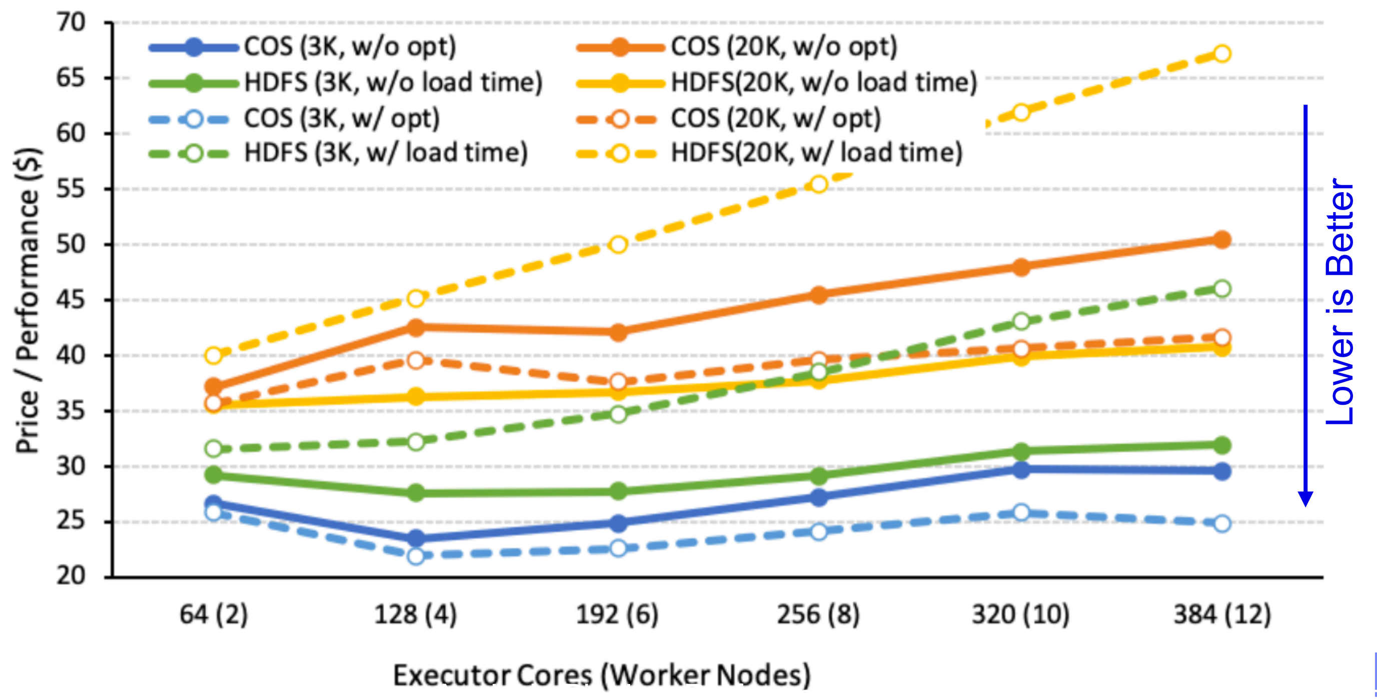


TABLE III
SYSTEM SETUP TIME

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

Dotted Line:

- Comparing whole clock time in COS (Opt) with HDFS, which includes data loading time (i.e. 30 mins)
- Achieved **up to 67% cost saving** with COS (Opt) w/ 3K-IOPS, and up to 61% w/ 20K-IOPS



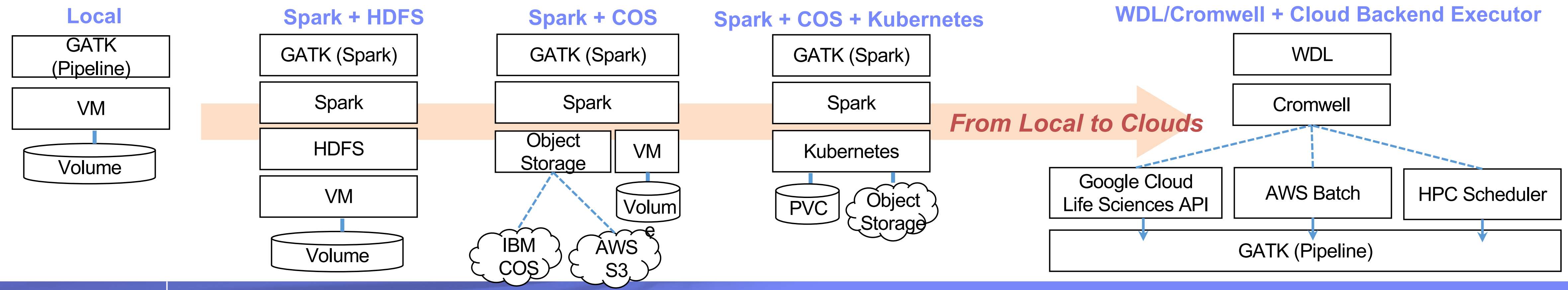
Conclusion

Summary

- Identified performance scalability and elasticity issues in Genome analysis pipeline running on GATK with Spark/HDFS
- Provided a new best practice to use **Cloud Object Storage** instead of **HDFS**
- Demonstrated the entire pipeline improvement
  - Performance: **up to 28%** faster
  - Cost: **up to 67%** cost saving

Next Steps

- Demonstrates compute elasticity with container & Kubernetes
- Applies our investigation results and optimization to GATK + Cromwell


Local
GATK (Pipeline)
VM
Volume
Spark + HDFS
GATK (Spark)
Spark
HDFS
VM
Volume
Spark + COS
GATK (Spark)
Spark
Object Storage
VM
Volume
IBM COS
AWS S3
Spark + COS + Kubernetes
GATK (Spark)
Spark
Kubernetes
PVC
Object Storage
From Local to Clouds
WDL/Cromwell + Cloud Backend Executor
WDL
Cromwell
Google Cloud Life Sciences API
AWS Batch
HPC Scheduler
GATK (Pipeline)