# Impact of the Networking Infrastructure on the Performance of Variant Calling on Human Genomes in Commodity Clusters

Manas Jyoti Das*
madas@siue.edu
Southern Illinois Univ. Edwardsville
USA

Praveen Rao
praveen.rao@missouri.edu
University of Missouri
USA

Lisong Xu
xu@unl.edu
University of Nebraska-Lincoln
USA

## Abstract

A whole genome sequence of a human can consume gigabytes of storage space. Analyzing a large number of such sequences is a compute and memory intensive process. In this regard, cluster computing has emerged as an attractive solution for large-scale human genome analysis. In this paper, we investigate how the underlying networking infrastructure of a commodity cluster can impact the performance of variant calling, which is a key task to identify variations in a human's genome compared to the reference genome. We measured the performance of variant calling and analyzed the network traffic in 16-node clusters with different hardware and network bandwidth configurations. We observed that by increasing the network link bandwidth, the execution time of variant calling did not improve significantly due to the degree of parallelism that was achievable leading to underutilization of the links. However, with low bandwidth links, data shuffling errors become more likely leading to failures. Furthermore, higher network latency among cluster nodes led to slower execution of variant calling due to lower utilization of the processor cores. By appropriately choosing the network link bandwidth for a cluster, good performance can be achieved while lowering the price of processing genomes especially in a pay-as-you-go pricing model.

## CCS Concepts

• **Computing methodologies**; • **Applied computing**;

## Keywords

Variant calling, human genomes, cluster computing, networking

**ACM Reference Format:**
Manas Jyoti Das, Praveen Rao, and Lisong Xu. 2024. Impact of the Networking Infrastructure on the Performance of Variant Calling on Human Genomes in Commodity Clusters. In *15th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB '24), November 22–25, 2024, Shenzhen, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3698587.3701354

* M.J.D. contributed to this work when he was a postdoc at University of Missouri.

## 1 Introduction

Due to advances in sequencing technologies and lower cost of sequencing[1], whole genome sequencing (WGS) has become economically feasible for use in large-scale genomic studies and clinical practice. It is projected that up to 2 billion human genomes could be sequenced by 2025 [46]. The Sequence Research Archive (SRA) containing petabytes of human genome data, is widely available through commercial cloud providers [21]. Recently, new genome sequencing initiatives [5, 34] were launched for the diagnosis and treatment of life-threatening diseases such as cancer and COVID-19. The *All of Us* Research Program[2] and the UK Biobank[3] host nearly 250,000 and 500,000 whole genome sequences, respectively. Indeed, the volume of genomic data has grown rapidly in recent years.

Human genome sequences are very large in size. The (diploid) human genome contains 6 billion base pairs forming the deoxyribonucleic acid (DNA) [35]. During DNA sequencing, a sample DNA is sliced into short fragments and read as a sequence of bases (a.k.a. *reads*). Due to sequencing errors, a position in the genome is covered by multiple DNA fragments resulting in millions of reads. The coverage of WGS denotes the average number of sequencing reads for each position in the target genome. In essence, a high-coverage whole genome sequence would require 100+ gigabytes (GBs) of storage [17]. These large-sized sequences pose technical challenges for efficient storage, analysis, and data transfer.

*Variant calling* is a key task that is performed to identify variants in a human genome sequence compared to the reference genome. These variants include single nucleotide polymorphisms (SNPs), short insertions/deletions (indels), copy number variation, and other structural variants[4]. Identifying these variants using a variant calling pipeline [22] is critical for assessing an individual's risk to diseases and enabling new innovations in disease diagnosis and treatment. A pipeline consists of several stages including reading a genome sequence, performing alignment of reads with the reference genome, additional pre-processing steps, and finally, invoking a variant caller to produce raw variants. The raw variants are further processed by variant filtering and annotation steps. Variant calling involves computationally intensive tasks and requires significant computing and storage resources to analyze a genome workload.

There is growing demand from hospitals and institutions to process large volumes of genomic data due to *precision medicine*[5]. Cloud computing has become a feasible solution for analyzing genomes due to its elasticity and pay-as-you-go pricing model [23]. In this regard, reducing the cost of analyzing genomes has become a key challenge [28, 32, 45]. In recent years, open source projects

[1] www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost [2] https://allofus.nih.gov [3] https://www.ukbiobank.ac.uk
[4] https://m.ensembl.org/info/genome/variation/prediction/classification.html
[5] www.cancer.gov/publications/dictionaries/cancer-terms/def/precision-medicine

have emerged (e.g., GATK4 [20], ADAM/Cannoli [32, 33]) that employ cluster computing frameworks, namely, Apache Spark [50] and Apache Hadoop [48], to perform variant calling on genomes. Companies such as Microsoft, Databricks, Illumina, Sentieon, and NVIDIA are developing new tools/services to accelerate genome analysis. Thus, there is a growing interest in advancing the state of the art in large-scale human genome sequence analysis [47].

In this paper, we investigate *how the underlying networking infrastructure of a commodity cluster can impact the performance of a variant calling pipeline on a large workload of human genomes*. To the best of our knowledge, none of the prior efforts has thoroughly analyzed the performance of variant calling pipelines for different cluster configurations. Our key contributions are as follows:

• We study the performance of two variant calling pipeline implementations, namely, GATK4 [20] and Adam/Cannoli [32]. Both are open source and designed to run in a commodity cluster using Apache Spark/Hadoop. (GATK4 is a widely adopted software.)

• We evaluated the performance of these pipelines in CloudLab [12], an experimental testbed for cloud computing research available *at no charge*. We employed an asynchronous computing model [41] that enables good cluster utilization on a workload of genomes. We chose different hardware/networking configurations for the clusters and used publicly available WGS data [4].

• Through detailed performance evaluation and network traffic analysis on the chosen dataset, we observed that high bandwidth links (e.g., 10 Gbps) did not significantly improve the execution time of variant calling as the links were underutilized due to the achievable degree of parallelism on the cluster hardware.

• However, lowering the link bandwidth for the cluster (e.g., 500 Mbps) was not a good solution due to Apache Spark shuffle operation errors[6] that occurred at times leading to failed execution of some sequences. Furthermore, higher network latency between cluster nodes increased the pipeline execution time due to lower utilization of the processor cores.

• For users regularly processing genome workloads, we provide a binary-search based approach to select an appropriate cluster configuration. By appropriately choosing the network link bandwidth and ensuring low network latency for a cluster, good performance can be achieved for variant calling while lowering the price especially in a *pay-as-you-go* pricing model.

## 2 Background and Motivation

### 2.1 Overview of Variant Calling Pipelines

Variant calling is a key task performed to identify variants in an individual's genome compared to the reference human genome (e.g., GRCh38 [29]). It is a computationally intensive process and requires significant computing and storage resources. It involves reading ten's of GBs of genome data and writing several large intermediate files. A typical variant calling pipeline for an individual's DNA sample [22] has a set of stages. It starts by reading of raw unmapped reads (e.g., in FASTQ format [49]) output by a sequencer. Using algorithms such as BWA-MEM [24], the alignment of the reads with the reference genome is performed. This produces mapped reads that are stored in a specific format (e.g., BAM format [25]). Duplicate reads in the mapped reads are then marked followed by the execution of base quality score recalibration (BQSR) and local realignment around the indels. BQSR and indel realignment steps are performed to correct sequencing errors and improve accuracy of downstream processing. The last stage involves the execution of a variant calling method. Several variant callers exist today such as FreeBayes [15], GATK HaplotypeCaller [20], DeepVariant [39], and DNAscope [13]. The output file produced by a variant caller contains raw variants and is in the VCF format [14]. The subsequent downstream processing steps include variant filtering and annotation steps on the raw variants. The complexity and variety of pipelines for variant discovery depends on the nature of genomic analysis and the technology used for sequencing [22].

With the availability of Apache Hadoop [48] and Spark [50], several efforts accelerated the DNA variant calling pipelines using these systems. CloudBurst [43], CloudAligner [30], SEAL [38], and BigBWA [1] used Hadoop for speeding up the computationally-intensive alignment stage. Hadoop-BAM was developed to support Hadoop-based parallel I/O [31] for sequencing data. Later, SparkBWA [2] employed Apache Spark to speed up alignment using BWA. Built atop Spark, Cloud Scale-BWAMEM [7] sped up alignment and was adapted for field-programmable gate arrays (FPGAs) [6]. PipeMEM [51] used Spark pipes to speed up alignment using BWA-MEM [24]. Ahmed et. al. [3] used FPGAs to accelerate BWA-MEM [24]; however, they did not use Apache Spark/Hadoop.

The Broad Institute developed the GATK Best Practices Workflows, which is widely adopted for variant discovery [20]. Halvade [11] parallelized the variant calling pipeline of GATK using MapReduce [10]. Later, GATK4 was released that employed Apache Spark for multithreading and parallelization [20]. NVIDIA developed Parabricks to accelerate GATK pipelines using graphics processing units (GPUs) [37]. Google's DeepVariant [39] used deep learning for variant calling and operated directly on aligned reads. Nothaft et. al. [32, 33] created ADAM/Cannoli to enable the processing of large genomic datasets using Apache Spark's primitives. ADAM supports read transformations and correcting errors in aligned reads. Cannoli leverages pipes and parallelizes the alignment process and variant calling by reusing existing tools.

More recently, Illumina developed the DRAGEN Platform to accelerate the variant calling pipeline using FPGAs [44]. Sentieon [45] developed highly optimized software-based algorithms for variant calling using CPUs for cloud environments. They also developed DNAscope [13], a machine learning-based variant caller.

### 2.2 Motivation

With growing use of cloud computing services for analyzing large genomic datasets [8, 23, 45], the need for efficient cluster computing solutions has increased. Most of the prior work have focused on either (a) developing more accurate variant callers, (b) accelerating different stages of a variant calling pipeline by designing better algorithms, or (c) accelerating the entire pipeline using hardware accelerators and frameworks such as Apache Spark/Hadoop. We believe there is an important gap in the literature yet to be addressed and pose the following motivating questions:

• [Q1] How does the underlying networking infrastructure of a cluster impact the performance of variant calling pipelines due to large data movement for enabling parallel computations?

[6] https://dzone.com/articles/four-common-reasons-for-fetchfailed-exception-in-a

• [Q2] Does the network link bandwidth and network latency of a cluster impact the execution speed of these pipelines?

Hence, the goal of our work is to gain insights to answer the aforementioned questions by observing the performance and analyzing the network traffic generated during variant calling in different cluster settings. While prior work [36] has investigated the impact of CPU, network, and disk I/O on data analytics workloads using Apache Spark, our work focuses on variant calling pipelines that are different from traditional data analytics workloads. Data analytics workloads (expressed in SQL) are focused on select, project, join, ordering, and aggregate operations that are commonly seen in databases and data warehouses. However, genomic data processing, specifically variant calling, invokes alignment algorithms, specialized read processing, and identification of active regions/application of Bayes' rule/Pair Hidden Markov Models [20] for calling the variants. Hence, the two workloads are significantly different. The lessons learned from our work can be beneficial to other workloads with similar compute and I/O patterns.

## 3 Large-Scale Variant Calling in a Cluster

Adam/Cannoli [32] introduced data parallel processing of variant calling pipelines using Apache Spark/Hadoop. However, on a large workload of genomes, ADAM/Cannoli yielded modest cluster utilization [41]. Motivated by these reasons, AVAH (**A**ccelerating **VA**riant Calling on **H**uman Genomes) [41] was developed to accelerate the pipeline execution on the workload by improving the cluster utilization using asynchronous computations and the concept of futures [19]. (Note that recent software systems have successfully employed futures for efficient large-scale distributed execution [27, 42].) More specifically, AVAH distributes the task of executing a variant calling pipeline on input sequences across the cluster nodes. It synergistically combines task parallelism and data parallelism for different stages in the pipeline by launching asynchronous computations. These computations are executed in a sliding window manner on small groups of sequences resulting in improved cluster utilization. AVAH is built atop Apache Spark/Hadoop and can use the APIs of Adam/Cannoli and GATK4. In the reported experiments, AVAH achieved 3×-4× speedup over Adam/Cannoli [41] on a workload of 98 whole genome sequences. (AVAH leveraged the APIs of Adam/Cannoli for data parallelism.)

**Table 1: GATK4's Variant Calling Pipeline Stages**

| Stage | Description |
|---|---|
| 1 | Copy paired-end FASTQ files from HDFS to a local file system, produce an unaligned `.bam`, and store it in HDFS |
| 2 | Align the `.bam` file against a reference genome (e.g., using BWA-MEM) and mark duplicates of mapped reads to produce an aligned `.bam` file, and then store in HDFS |
| 3 | Sort the aligned reads in HDFS to produce a `.bam` file in HDFS |
| 4 | Invoke the variant calling method HaplotypeCaller [20] to produce a `.vcf` file in HDFS |

Consider a single-sample germline variant calling pipeline [22]. Table 1 shows the stages of GATK4 that are invoked sequentially on a human genome. Table 2 shows the stages of ADAM/Cannoli. In AVAH, each stage of a variant calling pipeline is modeled as an atomic task. Tasks are executed as asynchronous computations using futures. Tasks representing the same pipeline stage managed by a Spark executor (on a worker node) are managed by a sliding window approach to control the degree of parallelism.

**Table 2: ADAM/Cannoli's Variant Calling Pipeline**

| Stage | Description |
|---|---|
| 1 | Interleave paired-end FASTQ files in HDFS to produce a `.ifq` file in HDFS |
| 2 | Align the `.ifq` file against a reference genome (e.g., using BWA-MEM) to produce a `.bam` file in HDFS |
| 3 | Using the `.bam` file, sort the aligned reads and mark duplicate reads with BQSR/indel realignment and store the output in HDFS |
| 4 | Invoke the variant calling method Freebayes [15] to produce a `.vcf` file in HDFS |

The genome sequences are assumed to be stored in Hadoop Distributed File System (HDFS). Each genome sequence is identified by a sequence ID. AVAH reads an input file containing a list of sequence IDs and the corresponding sequence sizes as a resilient distributed dataset (RDD). The RDD can be repartitioned for load balancing along with sorting the sequence IDs in each partition by size. Next, AVAH invokes a map operation on each RDD partition along with the appropriate pipeline stage. The map operation is executed on a worker node on the set of sequences identified by the partition. The map operation on all the partitions returns an RDD containing tuples of sequence IDs and status of the execution of a stage on that sequence (i.e., success or failure). Each task/stage on a sequence can be executed in a data parallel manner using either the GATK4-Spark APIs [20] or the ADAM/Cannoli APIs. AVAH chains the map operations that are applied on the partitions for the different pipeline stages; the collect call is executed by the Spark driver at the end of the last stage of the pipeline. This introduces minimal synchronization among the pipeline stages and yields better cluster utilization and faster execution on a workload.

## 4 Methodology

In this section, we present our methodology for performance evaluation including details on the execution model, implementation, experimental setup, traffic measurement, and evaluation metrics.

### 4.1 Execution Model and Testbed Used

In this work, we study the performance of variant calling pipelines in a commodity cluster. As AVAH is designed to achieve good CPU utilization in a cluster for the aforementioned variant calling pipelines (on a large workload of genomes), it is used for the experiments. Furthermore, AVAH executes a genome workload faster than separately executing ADAM/Cannoli or GATK4-Spark APIs on the workload [41]. (We did not use AVAH★ [9] as the tested clusters did not use GPUs.) Hereinafter, we refer to AVAH that uses the GATK4-Spark APIs and the ADAM/Cannoli APIs as AVAH$_G$ and AVAH$_A$, respectively. By using two different pipeline implementations, we aim to gain deeper understanding of how the performance of these pipelines are impacted by a cluster's networking infrastructure.

We ran all of our experiments in CloudLab [12], which is an experimental testbed for cloud computing research available *at no* *charge*. (Note that we did not have usage credits for commercial cloud providers to run many rounds of experiments.) CloudLab has large number of bare metal servers for enabling large-scale experimentation. The servers have CPUs with different number of cores and processing speeds as well as RAM of different sizes. Furthermore, the network bandwidth links for LANs can be configured during cluster setup. In addition, a VLAN (virtual local area network) can be set up across two geographically-separated data centers using Internet2's Advanced Layer 2 Service (AL2S). Hence, CloudLab provides a *highly configurable* environment to study the impact of a cluster's networking infrastructure on variant calling pipelines using large-scale human genome datasets. (Experiments on multiple cloud providers is beyond the scope of this work.)

## 4.2 Implementation

We obtained the code for AVAH$_G$ and AVAH$_A$ published by the original authors on GitHub[7]. AVAH$_A$ was built atop Spark 3.0.0, Scala 2.12.8, and Hadoop 3.2.0. AVAH$_G$ was built atop Spark 2.4.7, Scala 2.11.8, and Hadoop 2.7.6.

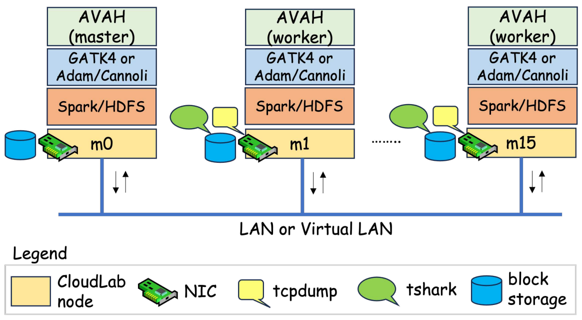


**Figure 1: Architecture for Performance Evaluation**

## 4.3 Cluster Setup

Figure 1 shows the overall architecture for performance evaluation of variant calling on CloudLab. We set up clusters using hardware available in two data centers of CloudLab: *Clemson* and *Wisconsin*. The nodes were physical machines with Intel processors running Ubuntu 18.04 and connected by a Gigabit Ethernet (GbE) network. Local block storage (striped across multiple physical disks) was mounted on each node in order to have ample storage space for HDFS and genome sequences. (For HDFS, the default replication factor of 3 was used.) Note that the cluster nodes were *exclusively* used by us to run AVAH$_G$/AVAH$_A$ without any interference from other jobs or users. Hence, our results can be reproduced in the same cluster settings.[8]

Table 3 shows the hardware details of the different nodes used in the experiments. The *C8220* nodes had two Intel E5-2660 v2 10-core CPUs (2.20 GHz); the *C6320* nodes had two Intel E5-2683 v3 14-core CPUs (2.00 GHz); the *W220g2* nodes had two Intel E5-2660 v3 10-core CPUs (2.60 GHz); and the *W220g5* nodes had two Intel Xeon Silver 4114 10-core CPUs (2.20 GHz). In essence, *C6320* had more cores than the others; *W220g5* was a newer and faster processor than *W220g2* and *C8220*.

[7] https://github.com/MU-Data-Science/EVA [8] When a cluster spans across 2 CloudLab data centers, the results may vary depending on the number of concurrent CloudLab users of AL2S between the data centers.

**Table 3: Node Hardware Description**

| Data Center | Node Type | Cores (Threads) | Main Memory | NIC Type |
|---|---|---|---|---|
| Clemson | C8220 | 20 (40) | 256 GB | 10 GbE |
| Clemson | C6320 | 28 (56) | 256 GB | 10 GbE |
| Wisconsin | W220g2 | 20 (40) | 160 GB | 10 GbE |
| Wisconsin | W220g5 | 20 (40) | 192 GB | 10 GbE |

All Spark jobs in AVAH$_G$ and AVAH$_A$ were executed using Yet Another Resource Negotiator (YARN) [48]. We allocated sufficient memory for YARN containers to successfully execute the pipeline. Note that ADAM/Cannoli runs the tools such as BWA and Freebayes outside of the Java Virtual Machine (JVM) as separate Linux processes. On the other hand, GATK4-Spark runs all the tools inside the JVM. Hence, AVAH$_G$ required higher memory for YARN containers and Spark executors than AVAH$_A$. We chose the memory settings to allow the pipelines to achieve high load average for the chosen cluster. For example, AVAH$_A$ on Clemson was run with *yarn.scheduler.maximum-allocation-mb* to 61,440 MB and *yarn.nodemanager.resource.memory-mb* set to 191,440 MB. AVAH$_G$ on Clemson was run with *yarn.scheduler.maximum-allocation-mb* to 91,440 MB and *yarn.nodemanager.resource.memory-mb* to 220,000 MB. Also, the CPU settings in YARN were chosen based on the hardware. The YARN property for virtual cores was set to 36 for a 40-thread node (i.e., C8220, W220g2, W220g5) and 50 for a 56-thread node (i.e., C6320).

## 4.4 Cluster Configurations

We set up 16-node clusters with different configurations. The *first group* of 8 nodes (named *m0*, ..., *m7*) were of one hardware type, and the *second group* of 8 nodes (named *m8*, ..., *m15*) were of another hardware type. In a *single-site* cluster, both groups belonged to one data center. The cluster nodes were connected using a LAN. In a *multisite* cluster, the two groups spanned across Clemson and Wisconsin data centers. The two groups of nodes were connected by AL2S using a VLAN. The multisite setting is useful *for understanding how network latency (between cluster nodes) can impact the performance of variant calling.*

We configured the maximum network link bandwidth for each cluster setting to 500 Mbps, 1 Gbps, or 10 Gbps. (The maximum transmission unit (MTU) was fixed at 1,500 bytes.) For ease of exposition, we use the notation `X-Y-Z` to refer to a cluster setting, where `X` is the hardware type for the first 8 nodes, `Y` is the hardware type for the remaining 8 nodes, and `Z` is the maximum network link bandwidth between the cluster nodes. For example, `S-C8220-C6320-10G` denotes a *single-site* cluster containing nodes in Clemson of type *C8220* and *C6320* with 10 Gbps link bandwidth. Similarly, `M-C8220-W220g5-1G` denotes a *multisite* cluster (spanning across the Clemson and Wisconsin data centers) with 1 Gbps link bandwidth. CloudLab uses *bandwidth shaping* in case higher bandwidth NICs are selected as part of a cluster setup. Note that specific hardware chosen for different experiments were based on availability of CloudLab resources at the time of experimentation.

**Table 4: Filters Used With tshark**

| Filter ID | Filter |
|---|---|
| $f_1$ | AVG(tcp.analysis.ack_rtt) |
| $f_2$ | MAX(tcp.window_size) |
| $f_3$ | COUNT(tcp.analysis.window_full) |
| $f_4$ | COUNT(tcp.analysis.zero_window) |
| $f_5$ | COUNT(tcp.analysis.lost_segment) |

## 4.5 Publicly Available Genome Dataset

For the experiments, we used 98 whole genome sequences of humans that are publicly available from the 1000 Genomes Project [4]. The total size of these low-coverage (paired-end) sequences was 632 GB (in compressed form). The size of the paired-end sequences ranged from 2.2 GB to 15.4 GB (in compressed form).

## 4.6 Network Traffic and Evaluation Metrics

Using `tcpdump`[9], we collected the network traffic on the network interface used by each cluster node during the execution of a pipeline. Separate PCAP (Packet Capture) files were produced every 30 mins for the packets sent and received on each interface. A snapshot length of 94 bytes was specified for tcpdump to avoid the PCAP files from growing too large. (A merged PCAP file on each worker was typically 30-50 GB in size.) AVAH$_G$ and AVAH$_A$ were run using YARN with the deploy mode as "client"; all the Spark executors were launched on the worker nodes, namely, *m1*, ..., *m15*. Hence, *m0* had very low load average most of the time (< 1.0).

We used `tshark`[10] to analyze the collected packets for data transferred (sent/received), throughput, round-trip time (RTT), and TCP window size. We also analyzed the number of lost segments and window full/zero window events. For TCP/IP metrics, we calculated statistics for 60-second intervals using `"tshark -z io,stat,60,filter"`. We considered the tshark filters shown in Table 4. We also computed the cumulative distribution function (CDF) for certain metrics and report their median/P90/P99 values.

As AVAH$_G$ and AVAH$_A$ used the APIs of existing variant calling pipelines, their accuracy for variant calling is identical to the pipelines GATK4-Spark and ADAM/Cannoli, respectively.

## 5 Performance Evaluation

In this section, we report the performance evaluation results of AVAH$_G$ and AVAH$_A$ for different cluster configurations. (Recall that AVAH$_G$ employed GATK4-Spark APIs, and AVAH$_A$ employed ADAM/Cannoli APIs.) We report the observed bandwidth among nodes, pipeline execution time, CPU load average, and network traffic analysis. We also discuss the *key takeaways* based on our evaluation. Finally, we present a binary search-based approach for a user to choose an appropriate cluster configuration. In the interest of space, we present only the representative results.

## 5.1 Observed Network Bandwidth

To test the available bandwidth between nodes, we used *iPerf* version 2.0.10[11]. The iPerf server was run on node *m0* of a cluster; the iPerf client was run on another node (e.g., *m1*, *m10*). Data was transferred from a client to the server using TCP. The results for a representative set of *single-site* clusters are reported in Table 5. The measured bandwidth was slightly lower than the maximum bandwidth setting for a cluster. The average network latency between two nodes was *0.17 ms*.

[9] https://www.tcpdump.org [10] https://www.wireshark.org [11] https://iperf.fr

**Table 5: iPerf Results for a Single-Site Cluster**

| Cluster Configuration | Server-Client | Data Transfer | Measured Bandwidth |
|---|---|---|---|
| S-C8220-C6320-500M | *m0-m5* | 569 MB | 476 Mbps |
| S-C8220-C6320-500M | *m0-m13* | 572 MB | 478 Mbps |
| S-C6320-C6320-1G | *m0-m1* | 1.11 GB | 950 Mbps |
| S-C6320-C6320-1G | *m0-m10* | 1.08 GB | 929 Mbps |
| S-C8220-C6320-10G | *m0-m1* | 11.0 GB | 9.41 Gbps |
| S-C8220-C6320-10G | *m0-m10* | 11.0 GB | 9.41 Gbps |

Another representative set of test results for multisite clusters are reported in Table 6. The average network latency between a Clemson node and a Wisconsin node was *25.86 ms*. As expected, this was much higher than in a single-site cluster. Also, the inter-data center measured bandwidth was lower than the maximum bandwidth setting for each cluster. It was noticeably lower for M-C8220-W220g5-1G and M-C6320-W220g5-10G. These situations are beyond our control as AL2S is a shared resource across CloudLab users. On a positive note, this enables us to study how network bandwidth/latency impacts the performance of variant calling.

**Table 6: IPerf Results for a Multisite Cluster**

| Cluster Configuration | Server-Client | Data Transfer | Measured Bandwidth |
|---|---|---|---|
| M-C8220-W220g5-500M | *m0-m1* | 572 MB | 478 Mbps |
| M-C8220-W220g5-500M | *m0-m10* | 551 MB | 461 Mbps |
| M-C8220-W220g5-1G | *m0-m1* | 1.12 GB | 956 Mbps |
| M-C8220-W220g5-1G | *m0-m10* | 680 MB | 569 Mbps |
| M-C6320-W220g5-10G | *m0-m1* | 10.9 GB | 9.39 Gbps |
| M-C6320-W220g5-10G | *m0-m10* | 803 MB | 674 Mbps |

## 5.2 Total Execution Time

**Table 7: Total Execution Time of AVAH$_G$**

| Cluster Configuration | Time Taken |
|---|---|
| S-C8220-C6320-500M | 29.96 hr |
| S-C6320-C6320-1G | 28.46 hr |
| S-C8220-C6320-10G | 27.09 hr |
| M-C6320-W220g5-500M | 38.99 hr |
| M-C6320-W220g5-1G | 36.68 hr |
| M-C6320-W220g5-10G | 35.14 hr |

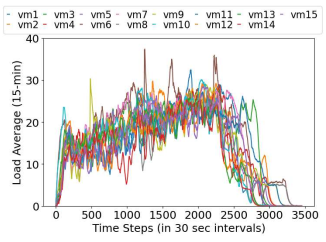
We report the total *wall-clock* time required to successfully execute AVAH$_G$ and AVAH$_A$ for different cluster configurations. Table 7 shows the total execution time of AVAH$_G$ for single-site and multisite clusters. Table 8 shows the total execution time of AVAH$_A$

**Table 8: Total Execution Time of AVAH$_A$**

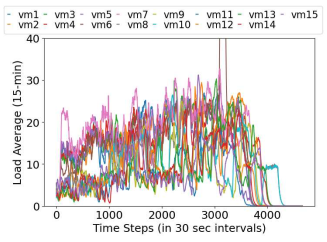
| Cluster Configuration | Time Taken |
|---|---|
| S-C8220-C6320-1G | 31.88 hr |
| S-C8220-C6320-10G | 28.73 hr |
| M-C8220-W220g5-500M | 32.24 hr |
| M-C8220-W220g5-1G | 32.45 hr |
| M-C6320-W220g2-10G | 33.13 hr |

for single-site and multisite clusters. One may wonder if higher link bandwidth can lead to faster execution of AVAH$_G$/AVAH$_A$. On the contrary, we observed that the total execution time *improved/varied marginally* for both AVAH$_G$ and AVAH$_A$ with increase in the maximum network link bandwidth. We observed these trends for both single-site and multisite clusters. It is interesting to observe that both AVAH$_G$ and AVAH$_A$ were slower in a multisite cluster than in a single-site cluster. In fact, AVAH$_G$ was slower by a bigger margin in a multisite cluster than in a single-site cluster. This is because when AVAH$_G$ was executed, the latency between nodes across the two data centers was *significantly higher* than between nodes in a single-site setting. (More analysis on network latency will be provided in Section 5.4.) Also, as reported in Tables 5 and 6, the measured bandwidth between nodes across the two data centers was significantly lower than in a single-site setting.

Further investigation of the CPU load average showed that some of the cluster nodes (running the Spark workers) in *Wisconsin* were underutilized leading to slower execution. (The Spark master was running in *Clemson.*) As a representative case, we present the cluster utilization of AVAH$_G$. Figure 2(a) shows the 15-minute load average (measured every 30 seconds on the Spark worker nodes) for a single-site setting. Figure 2(b) shows the 15-minute load average for a multisite setting. Clearly, some of the cluster nodes in the multisite setting were underutilized due to higher network latency across the Clemson and Wisconsin sites. Similar cluster utilization trend was observed for AVAH$_A$ in a multisite setting.



(a) S-C8220-C6320-10G



(b) M-C6320-W220g5-10G

**Figure 2: AVAH$_G$: Load Average of Cluster Nodes**

Next, we state two key takeaways.

**Takeway 1**: For the tested workload of genomes, a cluster with high network link bandwidth (i.e., 10 Gbps) provided marginal benefit in improving the execution speed of the variant calling pipelines. A cluster with moderate network link bandwidth (i.e., 1 Gbps) was sufficient to execute the pipelines with comparable performance. Hence, higher bandwidth links may not always be necessary to achieve good performance for variant calling.

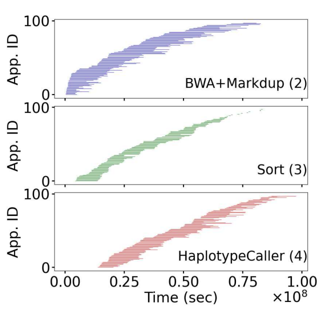
**Takeway 2**: Higher network latency between cluster nodes, which was observed in a multisite setting, caused slower execution of the pipelines. Further analysis showed that some of the processor cores had lower utilization in a multisite setting compared to a single-site setting. Hence, low latency links would enable better performance for variant calling.

One may wonder if lower bandwidth links are beneficial. Unfortunately, for 500 Mbps link bandwidth, we observed that Spark shuffle operation errors were more likely to happen leading to failed execution on some sequences. Hence, a lower link bandwidth setting may not be ideal for executing AVAH$_G$/AVAH$_A$. Note that the results reported for the 500 Mbps link bandwidth setting were for successful execution of the pipelines, i.e., no failures occurred.

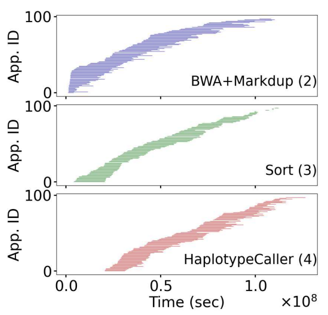
We also increased the TCP buffer sizes to 64 MB and re-ran the pipelines. However, this change yielded marginal improvement in the total processing time (less than 5% improvement) implying that the CPU speed is a bottleneck.

## 5.3 Task Parallelism During Pipeline Execution

During execution of AVAH$_G$/AVAH$_A$, we logged the Apache Spark events to better understand when different Spark applications (or jobs) were scheduled. (Note that these pipelines execute each variant calling stage as a Spark application/job invoked via `spark-submit`.) Initially, AVAH$_G$/AVAH$_A$ is first involved as a Spark application; later, more Spark applications are launched as the pipeline progresses.) For AVAH$_G$, the last three stages described in Table 1 are shown in Figure 3(a-b) for single-site and multisite clusters with 10 Gbps link bandwidth. For AVAH$_A$, the four stages described in Table 2 are shown in Appendix (Figure 8(a-b)) for single-site and multisite clusters with 10 Gbps link bandwidth. In each subplot, a line indicates the start time and end time of a Spark application for a sequence. As there were 98 sequences (and no failures occurred during the execution), there are 98 lines in each subplot denoting 98 Spark applications. The first variant calling stage of AVAH$_G$ for each sequence (described in Table 1) was executed as part of the main Spark application, and hence is not shown separately. The plots show *the inherent task parallelism* across different variant calling stages that enables high cluster utilization and fast execution of the pipelines [41]. For other link bandwidth settings (e.g., 500 Mbps, 1 Gbps), the observed trends were similar.



(a) S-C8220-C6320-10G



(b) M-C6320-W220g5-10G

**Figure 3: Spark Events for AVAH$_G$ (Stages 2-4)**

## 5.4 Network Traffic Analysis

Next, we report the analysis of the collected network traffic.

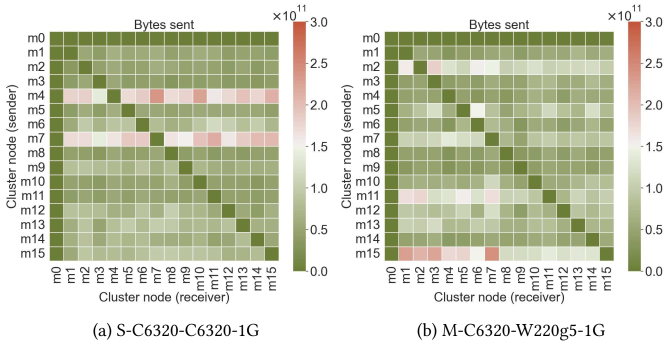


**Figure 4: IP Traffic for AVAH$_G$**

### 5.4.1 *Analysis of IP Traffic Between Cluster Nodes.*

We first analyzed the IP traffic between cluster nodes during the execution of AVAH$_G$. We ran "`tshark -z conv,ip`" on the PCAP files collected on the nodes *m1-m15*. This computes the IP data (in bytes) sent from one IP address (sender) to another (receiver). Figure 4 shows the heatmaps for AVAH$_G$ in single-site and multisite cluster configurations with 1 Gbps link bandwidth. We observed that very few nodes tend to send more data between each other (denoted by red shade) than others. Similar trends were observed for other cluster configurations. In all cases, *m0* (the master node) did not exchange much data with other nodes due to the YARN/Spark settings. Similar trends were observed for AVAH$_A$ and representative results are shown in Figure 9 (see Appendix).

### 5.4.2 *Analysis of TCP Traffic Between HDFS Nodes.*

In HDFS, DataNodes that ran on *m1-m15* were responsible for managing file data blocks and transferring data between a reader/writer and HDFS. (The NameNode that ran on *m0* was responsible for file system metadata and did not generate much traffic.) We computed the total amount of data transferred between the HDFS DataNodes during the execution of AVAH$_G$. We ran "`tshark -z endpoints,tcp`" on the collected PCAP files and computed the traffic sent/received on the DataNode server port in HDFS.

Figure 5 shows the results for AVAH$_G$ in terms of total data sent and total data received by the different HDFS DataNodes for different cluster configurations. We observed that each DataNode (shown on the x-axis) approximately sent (or received) the same amount of data during variant calling (upper point trend in each plot). The lower point trends show how much data was sent (or received) by others DataNode to (or from) a DataNode (shown on the x-axis). Overall, the TCP traffic seemed to be balanced across the DataNodes. Similar trends were observed for other bandwidth settings. Whereas for AVAH$_A$, the received traffic was slightly higher than the sent traffic as shown in Figure 10 (see Appendix). We attribute this to the difference in the pipeline stages of AVAH$_A$ and AVAH$_G$ and their design and implementation.

### 5.4.3 *Analysis of IP Throughput.*

We analyzed the throughput of IP packets sent/received by cluster nodes *m1-m15* during the execution AVAH$_G$/AVAH$_A$. On the collected PCAP files, we ran "`tshark -z io,stat,60,ip`". Figure 6 shows the throughput (per minute) for a single-site and a multisite setting when executing AVAH$_G$. For 10 Gbps links (Figures 6(b) and 6(d)), we observed that the links were underutilized for both single-site and multisite settings. (A 100% utilization would be 75 GB/min for 10 Gbps links and 7.5 GB/min for 1 Gbps links.) We attribute this to the *achievable degree of parallelism* as the number of YARN containers that could be executed were limited by the amount of RAM and number of CPU cores in a cluster. Having more RAM and cores would increase the degree of parallelism and can ultimately improve the network bandwidth utilization. Similar trends of link bandwidth underutilization were observed when executing AVAH$_A$. (See Figure 11 in Appendix.)

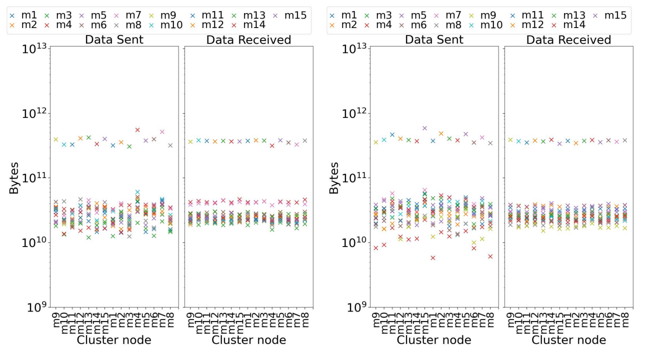


**Figure 5: Traffic Between HDFS DataNodes for AVAH$_G$**

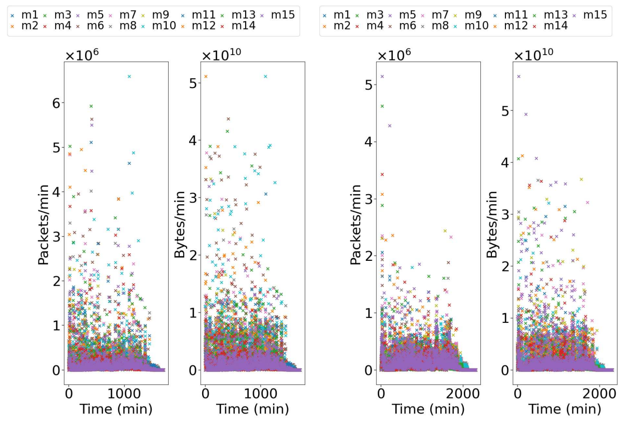


**Figure 6: AVAH$_G$: IP Throughput**

One observation is that the IP traffic patterns were different for AVAH$_G$ and AVAH$_A$. In AVAH$_A$, the interleaving of FASTQ files (in Stage 1) was done by reading and writing data blocks to HDFS. Hence, AVAH$_A$ had a spike in the IP traffic when the pipeline began execution. For AVAH$_G$, the FASTQ files were copied from HDFS to a node, combined locally to create a `.bam` file, and then copied back to HDFS for the next stage. We did this because GATK4-Spark only supports local FASTQ files for `.bam` file creation.

Based on the results, we state the following takeaway:

**Takeway 3**: The underutilization of the network link bandwidth during execution of AVAH$_G$/AVAH$_A$ can be attributed to the available cluster hardware resources such as CPU cores and RAM. More cores and larger RAM size would allow AVAH$_G$ to launch higher number of YARN containers leading to higher degree of parallelism in executing genome sequences. We believe this would generate more network I/O, thereby improving the link utilization.

*5.4.4 Analysis of Network Latency.* To better understand the latency of data transfer during the execution of AVAH$_G$/AVAH$_A$, we analyzed the average RTT for the TCP packets in 60-second intervals. We ran "`tshark -z io,stat,60`" with filter $f_1$ (shown in Table 4) on the collected PCAP files. We analyzed the aggregated traffic across all the worker nodes (*m1-m15*) by computing the median, P90, and P99 avg. RTT during the pipeline execution. The results for AVAH$_G$ and AVAH$_A$ are reported in Table 9. We observed the avg. RTT percentile values were higher for multisite clusters compared to single-site clusters for the same bandwidth setting. Also the clusters with 500 Mbps link bandwidth had higher latencies (compared to clusters with 1 Gbps/10 Gbps links) implying that such low bandwidth links are not ideal for executing AVAH$_G$ as this would increase the likelihood of observing Spark shuffle operation errors. As reported earlier, higher latencies in the multisite setting resulted in lower cluster utilization on a subset of cluster nodes leading to slower execution of the pipelines.

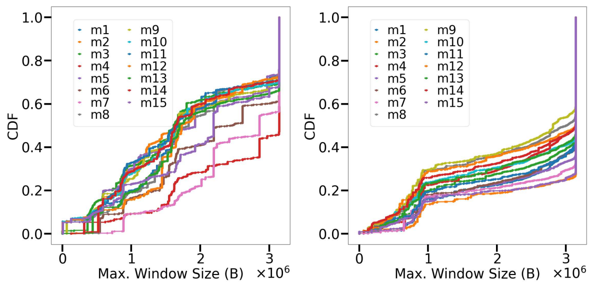
**Table 9: Avg. RTT (Aggregated Across All Workers)**

| Cluster Configuration | Median Avg. RTT | P90 Avg. RTT | P99 Avg. RTT |
|---|---|---|---|
| **AVAH$_G$** | | | |
| S-C8220-C6320-500M | 4.08 ms | 13.26 ms | 41.28 ms |
| S-C6320-C6320-1G | 2.47 ms | 11.03 ms | 18.33 ms |
| S-C8220-C6320-10G | 2.11 ms | 10.67 ms | 17.48 ms |
| M-C6320-W220g5-500M | 13.18 ms | 23.80 ms | 44.54 ms |
| M-C6320-W220g5-1G | 10.78 ms | 20.68 ms | 28.90 ms |
| M-C6320-W220g5-10G | 12.07 ms | 21.86 ms | 31.20 ms |
| **AVAH$_A$** | | | |
| S-C8220-C6320-1G | 8.44 ms | 13.88 ms | 21.35 ms |
| S-C8220-C6320-10G | 2.88 ms | 7.61 ms | 14.72 ms |
| M-C8220-W220g5-500M | 10.37 ms | 17.18 ms | 27.23 ms |
| M-C8220-W220g5-1G | 7.78 ms | 13.67 ms | 20.16 ms |
| M-C6320-W220g2-10G | 4.23 ms | 7.86 ms | 16.01 ms |

Based on the latencies observed in different cluster settings, we state the following takeaway:

**Takeway 4**: It is not a good idea to use low network bandwidth links (i.e., 500 Mbps). This is because the likelihood of Spark shuffle operation errors increases due to higher RTTs leading to failures in processing sequences. Re-execution of failed sequences would ultimately increase the total processing time of the pipelines.

*5.4.5 Analysis of TCP Window Size.* To better understand the TCP traffic pattern/congestion for different cluster configurations, we analyzed the maximum TCP window size for every 60-second interval.



(a) S-C6320-C6320-1G (b) M-C6320-W220g5-1G

**Figure 7: AVAH$_G$: CDF of Max. TCP Window Size**

We ran "`tshark -z io,stat,60`" with filter $f_2$ (shown in Table 4) on the collected PCAP files. (TCP Cubic [18] was the congestion control algorithm used by the cluster nodes.) We report the CDF of the maximum TCP window size (in bytes) computed every minute for each cluster node during the execution of AVAH$_G$/AVAH$_A$. Figure 7 shows the CDF of max. TCP window size for AVAH$_G$. Figure 12 (see Appendix) shows the CDF of max. TCP window size for AVAH$_A$. The nodes were more likely to use a larger TCP window size in a single-site setting than in a multisite setting. These plots clearly show a difference in the traffic patterns produced by AVAH$_G$ and AVAH$_A$. When executing AVAH$_A$, the cluster nodes were more likely to use the max. TCP window size (i.e., 3,145,728 bytes).

We also analyzed the number of *window full* and *zero window* events (for every 60-second intervals) in the TCP traffic for AVAH$_G$ and AVAH$_A$. We ran "`tshark -z io,stat,60`" with filters $f_3 - f_4$ (shown in Table 4) on the collected PCAP files. We analyzed the aggregated traffic across all the worker nodes (*m1-m15*) by computing the P90/P99 values for the number of events during a pipeline execution. The results are reported in Table 10. We observed that these events occurred more frequently as the link bandwidth increases implying that *the CPU speed is the bottleneck.*

**Table 10: Window Full/Zero Window Events (All Workers)**

| Cluster Configuration | Window Full | | Zero Window | |
|---|---|---|---|---|
| | P90 Count | P99 Count | P90 Count | P99 Count |
| **AVAH$_G$** | | | | |
| S-C8220-C6320-500M | 3,173 | 7,030 | 1,947 | 3,287 |
| S-C6320-C6320-1G | 3,805 | 11,111 | 2,441 | 6,288 |
| S-C8220-C6320-10G | 10,265 | 42,711 | 4,162 | 12,527 |
| M-C6320-W220g5-500M | 5,637 | 13,320 | 1,492 | 2,628 |
| M-C6320-W220g5-1G | 6,370 | 15,298 | 1,885 | 3,600 |
| M-C6320-W220g5-10G | 8,456 | 21,063 | 2,262 | 6,049 |
| **AVAH$_A$** | | | | |
| S-C8220-C6320-1G | 5,873 | 20,100 | 3,769 | 10,483 |
| M-C8220-W220g5-500M | 2,867 | 6,187 | 2,207 | 5,062 |
| M-C8220-W220g5-1G | 3,532 | 7,578 | 2,488 | 5,680 |
| M-C6320-W220g2-10G | 5,649 | 11,725 | 4,507 | 9,133 |

*5.4.6 Analysis of TCP Lost Segments.* Finally, we ran "`tshark -z io,stat,60`" with filters $f_5$ (shown in Table 4) on the collected

PCAP files. We computed the ratio of lost segments in a 60-second interval and the total number of packets sent/received in that interval. We observed that the network was highly reliable with a typical loss rate of below 1% for different settings.

## 5.5 Recommendations for Users

**Algorithm 1** Instance Type Selection for Variant Calling

```
Require: ($I_1$, $I_2$, ..., $I_n$) - sorted list of instances; $T(> 0)$ - user specified threshold
Ensure: $I_{i_{best}}$ - the best instance
 1: Run AVAH$_G$ (or AVAH$_A$) on the $m$-node cluster of $I_n$ node type using the user's representative workload of human genomes; allocate 90% of the RAM/cores per node to YARN
 2: Let $t_{best}$ denote the execution time of AVAH$_G$ (or AVAH$_A$)
 3: Let $i_{best} \leftarrow n$
 4: $low \leftarrow 1$, $high \leftarrow n$
 5: while $high - low > 0$ do
 6:     $mid \leftarrow \lfloor \frac{high+low}{2} \rfloor$
 7:     Run AVAH$_G$ (or AVAH$_A$) on the $m$-node cluster of $I_{mid}$ node type using the user's sample workload of human genomes; allocate 90% of the RAM/cores to YARN
 8:     Let $t_{mid}$ denote the execution time of AVAH$_G$ (or AVAH$_A$)
 9:     if [(Spark shuffle errors or Java heap space errors are observed leading to failures in generating VCF files) OR ($t_{mid} > T \times t_{best}$)] then
10:         $low \leftarrow mid$
11:     else
12:         $high \leftarrow mid$; $t_{best} \leftarrow t_{mid}$; $i_{best} \leftarrow mid$
13: return $I_{i_{best}}$
```

For users who regularly process genome workloads, we present a binary search-based approach for choosing an appropriate cluster configuration (focused on network link bandwidth/latency). For example, Amazon Web Services (AWS)[12] provides several general purpose instances with different network performance (e.g., 5 Gbps, 10 Gbps, 12.5 Gbps, 25 Gbps, 50 Gbps). It is challenging to figure out *a priori* which instance type would work best for a representative genome workload; trying out every instance is also a tedious task.

Suppose there are $n$ instances to choose from to set up an $m$-node cluster. Suppose each instance is represented by its attributes network link bandwidth ($b$), network latency ($l$), instance's RAM size ($r$), and number of cores in an instance ($c$). Let ($I_1$, $I_2$, ..., $I_n$) denote the sorted list of instances by the composite key $< b, \frac{1}{l}, c, r >$. We assume instance $I_n$ always leads to successful execution of the genome workloads. Algorithm 1 shows the steps involved. First, a cluster with $I_n$ instance type is used to run a representative workload of genomes to compute the initial best time (Lines 1-3). The binary search process starts by selecting an instance in the middle of the sorted list and tests on a cluster with that instance type (Lines 6-8). If the performance degrades significantly (based on a user-specified threshold) or Spark shuffle/Java heap errors occur, then the search proceeds to the right half of the list. Otherwise, the left half of the list is searched next and the new best timing/instance are noted. (See Lines 9-12.)

[12] https://aws.amazon.com/ec2/instance-types

## 5.6 Discussion

CloudLab is representative of a commodity cluster in a data center where the nodes are connected by high-speed networking. In contrast to a commercial cloud that typically provides virtual machines, our experiments were conducted on baremetal servers in CloudLab. The processors on CloudLab are likely to be older than those available today in a commercial cloud. Also, today cloud providers offer compute-optimized nodes and enhanced networking services for compute-intensive and latency-sensitive applications albeit for a cost. A multisite setting can be useful when resources across data centers are aggregated for solving a data intensive problem. Via AL2S, we can better understand the impact of network latency and bandwidth on variant calling given shared users. Overall, CloudLab is a valuable resource for our study as our experiments would have cost us thousands of dollars in a commercial cloud.

In terms of scaling, we expect to see similar trends in network traffic with increasing number of the genomes for a given cluster. This is because the number of YARN containers that can execute concurrently depends on the amount of RAM and number of cores in the cluster.

In our study, we did not use the recent human pangenome reference [26], which is known to improve read mapping and variant calling performance [16]. We did not measure disk I/O nor consider data locality in our experiments. Data locality can improve performance especially in a multisite setting. We also did not consider security and privacy concerns of genome data processing. A recent work showed that SmartNICs can be used to perform secure variant calling [40]. This work could be extended to perform secure variant calling in a cluster.

## 6 Conclusion

In this paper, we studied the impact of a cluster's network infrastructure on the performance of variant calling over human genomes. We evaluated the performance of two open-source variant calling pipelines and analyzed the network traffic generated during execution for different cluster configurations (e.g., single-site, multisite). We observed that higher bandwidth links may not always lead to faster execution of variant calling pipelines and can be underutilized due to the achievable degree of parallelism. Higher network latency among cluster nodes can lead to slower execution of the pipelines due to lower utilization of the processor cores. Hence, by choosing an appropriate network link bandwidth and ensuring low network latency, good performance can be achieved for variant calling on a large workload of human genomes. In the future, we would like to analyze the network traffic during variant calling on high-coverage sequences with different cluster configurations especially in the presence of hardware accelerators.

## Acknowledgment

This work was supported by the National Science Foundation Under Grant Nos. 2034247 and 2201583. We thank the anonymous reviewers for their insightful comments and suggestions. We also thank Arun Zachariah for his assistance.

## References

[1] J. M. Abuin, J. C. Pichel, T. F. Pena, and J. Amigo. 2015. BigBWA: Approaching the Burrows-Wheeler Aligner to Big Data Technologies. *Bioinformatics* 31, 24 (2015), 4003–4005.

[2] J. M. Abuin, J. C. Pichel, T. F. Pena, and J. Amigo. 2016. SparkBWA: Speeding up the Alignment of High-Throughput DNA Sequencing Data. *PLoS ONE* 11, 5 (2016).

[3] Nauman Ahmed, Vlad-Mihai Sima, Ernst Houtgast, Koen Bertels, and Zaid Al-Ars. 2015. Heterogeneous Hardware/Software Acceleration of the BWA-MEM DNA Alignment Algorithm. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* 240–246.

[4] Adam Auton and et.al. 2015. A Global Reference for Human Genetic Variation. *Nature* 526, 7571 (2015), 68–74.

[5] Jean-Laurent Casanova, Helen C Su, and the COVID Human Genetic Effort. 2020. A Global Effort to Define the Human Genetics of Protective Immunity to SARS-CoV-2 Infection. *Cell* 181, 6 (2020), 1194–1199.

[6] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. 2016. When Apache Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration. In *Proc. of the 8th USENIX Conference on Hot Topics in Cloud Computing* (Denver, CO). 64–70.

[7] J. Cong, Jie Lei, Sen Li, Myron Peto, P. Spellman, Peng Wei, and Peipei Zhou. 2015. CS-BWAMEM: A Fast and Scalable Read Aligner at the Cloud Scale for Whole Genome Sequencing. In *High Throughput Sequencing Algorithms and Applications (HITSEQ).*

[8] Jacklyn M Dahlquist, Sarah C Nelson, and Stephanie M Fullerton. 2023. Cloud-based Biomedical Data Storage and Analysis for Genomic Research: Landscape Analysis of Data Governance in Emerging NIH-Supported Platforms. *Human Genetics and Genomics Advances* 4, 3 (2023).

[9] Manas Das, Khawar Shehzad, and Praveen Rao. 2023. Efficient Variant Calling on Human Genome Sequences Using a GPU-Enabled Commodity Cluster. In *Proc. of 32nd ACM Intl. Conf. on Information and Knowledge Management (CIKM).* 3843–3848.

[10] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of 6th OSDI Conference.* 137–150.

[11] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier. 2015. Halvade: Scalable Sequence Analysis with MapReduce. *Bioinformatics* 31, 15 (2015), 2482–2488.

[12] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (Renton, WA). 1–14.

[13] Donald Freed, Renke Pan, Haodong Chen, Zhipan Li, Jinnan Hu, and Rafael Aldana. 2022. DNAscope: High Accuracy Small Variant Calling Using Machine Learning. *bioRxiv:2022.05.20.492556* (2022).

[14] GA4GH. 2021. The Variant Call Format (VCF) Version 4.2 Specification. https://samtools.github.io/hts-specs/VCFv4.2.pdf.

[15] Erik Garrison and Gabor Marth. 2012. Haplotype-Based Variant Detection From Short-Read Sequencing. arXiv:1207.3907

[16] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, Benedict Paten, and Richard Durbin. 2018. Variation Graph Toolkit Improves Read Mapping by Representing Genetic Variation in the Reference. *Nature Biotechnology* 36, 9 (2018), 875–879.

[17] Sara Goodwin, John D McPherson, and W Richard McCombie. 2016. Coming of Age: Ten Years of Next-Generation Sequencing Technologies. *Nature Reviews Genetics* 17, 6 (2016), 333–351.

[18] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (jul 2008), 64–74.

[19] Robert H. Halstead. 1985. MULTILISP: A Language for Concurrent Symbolic Computation. *ACM TOPLAS* 7, 4 (1985), 501–538.

[20] Broad Institute. 2023. GATK4. https://gatk.broadinstitute.org.

[21] Kenneth Katz, Oleg Shutov, Richard Lapoint, Michael Kimelman, J Rodney Brister, and Christopher O'Sullivan. 2021. The Sequence Read Archive: A Decade More of Explosive Growth. *Nucleic Acids Research* 50, D1 (11 2021), D387–D390.

[22] Daniel C. Koboldt. 2020. Best Practices for Variant Calling in Clinical Sequencing. *Genome Medicine* 12, 1 (2020), 91.

[23] Ben Langmead and Abhinav Nellore. 2018. Cloud computing for genomic data analysis and collaboration. *Nature Rev. Genetics* 19, 4 (2018), 208–219.

[24] Heng Li. 2013. Aligning Sequence Reads, Clone Sequences and Assembly Contigs With BWA-MEM. *arXiv e-prints* (2013), arXiv:1303.3997. arXiv:1303.3997

[25] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. 2009. The Sequence Alignment/Map Format and SAMtools. *Bioinformatics* 25, 16 (2009), 2078–2079.

[26] Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, Glenn Hickey, Shuangjia Lu, Julian K Lucas, Jean Monlong, Haley J Abel, et al. 2023. A Draft Human Pangenome Reference. *Nature* 617, 7960 (2023), 312–324.

[27] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proc. of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA). 561–577.

[28] Paul Muir, Shantao Li, Shaoke Lou, Daifeng Wang, Daniel J. Spakowicz, Leonidas Salichos, Jing Zhang, George M. Weinstock, Farren Isaacs, Joel Rozowsky, and Mark Gerstein. 2016. The Real Cost of Sequencing: Scaling Computation to Keep Pace with Data Generation. *Genome Biology* 17, 1 (2016), 53.

[29] NCBI. 2013. Genome Reference Consortium Human Build 38. https://www.ncbi.nlm.nih.gov/assembly/.

[30] T. Nguyen, W. Shi, and D Ruden. 2011. CloudAligner: A Fast and Full-Featured MapReduce Based Tool for Sequence Mapping. *BMC Research Notes* 4, 1 (2011), 171.

[31] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemela, E. Korpelainen, and K. Heljanko. 2012. Hadoop-BAM: Directly Manipulating Next Generation Sequencing Data in the Cloud. *Bioinformatics* 28, 6 (2012), 876–877.

[32] Frank A. Nothaft. 2017. *Scalable Systems and Algorithms for Genomic Variant Analysis.* Ph. D. Dissertation. UC Berkeley, ProQuest.

[33] Frank Austin Nothaft, Matt Massie, Timothy Danford, Zhao Zhang, Uri Laserson, Carl Yeksigian, Jey Kottalam, Arun Ahuja, Jeff Hammerbacher, Michael D. Linderman, Michael J. Franklin, Anthony D. Joseph, and David A. Patterson. 2015. Rethinking Data-Intensive Science Using Scalable Analytics Systems. In *Proc. of the 2015 ACM SIGMOD Conference* (Victoria, Australia). 631–646.

[34] All of Us Research Program Investigators. 2019. The "All of Us" Research Program. *New England Journal of Medicine* 381, 7 (2019), 668–676.

[35] National Research Council Committee on Mapping and Sequencing the Human Genome. 1988. *Mapping and Sequencing the Human Genome.* National Academies Press.

[36] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making Sense of Performance in Data Analytics Frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15).* 293–307.

[37] Kyle A O'Connell, Zelaikha B Yosufzai, Ross A Campbell, Collin J Lobb, Haley T Engelken, Laura M Gorrell, Thad B Carlson, Josh J Catana, Dina Mikdadi, Vivien R Bonazzi, et al. 2023. Accelerating Genomic Workflows Using NVIDIA Parabricks. *BMC Bioinformatics* 24, 1 (2023), 221.

[38] L. Pireddu, S. Leo, and G. Zanetti. 2011. SEAL: A Distributed Short Read Mapping and Duplicate Removal Tool. *Bioinformatics* 27, 15 (2011), 2159–2160.

[39] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T Afshar, Sam Gross, Lizzie Dorfman, Cory McLean, and DePristo Mark. 2018. A Universal SNP and Small-Indel Variant Caller Using Deep Neural Networks. *Nature Biotechnology* 36, 10 (2018), 983–987.

[40] Praveen Rao and Khawar Shehzad. 2024. A Technique for Secure Variant Calling on Human Genome Sequences Using SmartNICs. In *Proc. of the 17th IEEE International Conference on Cloud Computing (CLOUD 2024).* 1–8.

[41] Praveen Rao, Arun Zachariah, Deepthi Rao, Peter Tonellato, Wesley Warren, and Eduardo Simoes. 2021. Accelerating Variant Calling on Human Genomes Using a Commodity Cluster. In *Proc. of 30th ACM International Conference on Information and Knowledge Management (CIKM).* 3388–3392.

[42] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proc. of the 14th Python in Science Conference*, Kathryn Huff and James Bergstra (Eds.). 130 – 136.

[43] Michael C. Schatz. 2009. CloudBurst: Highly Sensitive Read Mapping with MapReduce. *Bioinformatics* 25, 11 (2009), 1363–1369.

[44] Konrad Scheffler, Severine Catreux, Taylor O'Connell, Heejoon Jo, Varun Jain, Theo Heyns, Jeffrey Yuan, Lisa Murray, James Han, and Rami Mehio. 2023. Somatic small-variant calling methods in Illumina DRAGEN Secondary Analysis. *bioRxiv 2023.03.23.534011* (2023).

[45] Sentieon. 2023. https://www.nature.com/articles/d44224-023-00020-w.

[46] Zachary D. Stephens, Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. 2015. Big Data: Astronomical or Genomical? *PLOS Biology* 13, 7 (2015), 1–11.

[47] Tomoya Tanjo, Yosuke Kawai, Katsushi Tokunaga, Osamu Ogasawara, and Masao Nagasaki. 2021. Practical guide for managing large-scale human genome data in research. *Journal of Human Genetics* 66, 1 (2021), 39–52.

[48] Tom White. 2009. *Hadoop: The Definitive Guide* (1st ed.). O'Reilly Media, Inc.

[49] Wikipedia. 2000. FASTQ Format. https://en.wikipedia.org/wiki/FASTQ_format.

[50] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proc. of 2nd USENIX Conference on Hot Topics in Cloud Computing.* 1–7.

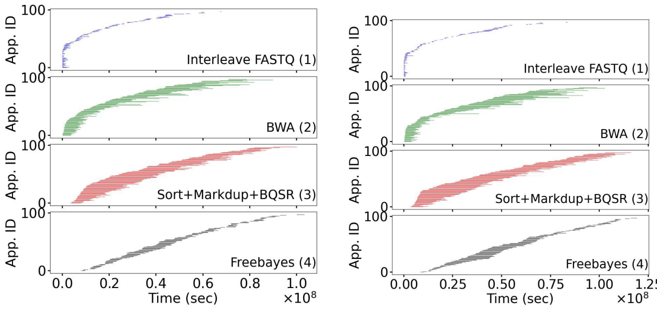
[51] Lingqi Zhang, Cheng Liu, and Shoubin Dong. 2019. PipeMEM: A Framework to Speed Up BWA-MEM in Spark with Low Overhead. *Genes* 10, 11 (2019).

## Appendix

In the interest of space, some of the AVAH$_A$ results are shown in this Appendix.

## Task Parallelism During Pipeline Execution

Figure 8 shows the Spark events for AVAH$_A$.

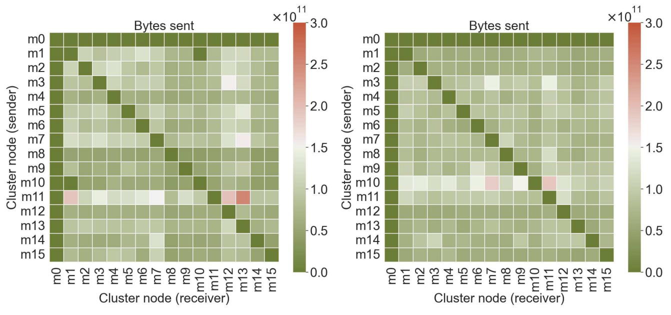


(a) S-C8220-C6320-10G (b) M-C6320-W220g2-10G

**Figure 8: Spark Events for AVAH$_A$ (Stages 1-4)**

## Analysis of IP Traffic Between Cluster Nodes

Figure 9 shows the IP traffic between cluster nodes for AVAH$_A$.

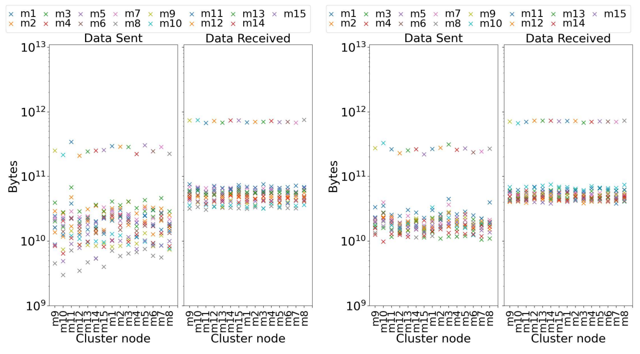


(a) S-C8220-C6320-1G (b) M-C8220-W220g5-1G

**Figure 9: IP Traffic for AVAH$_A$**

## Analysis of TCP Traffic Between HDFS Nodes

Figure 10 shows the TCP traffic between HDFS nodes.

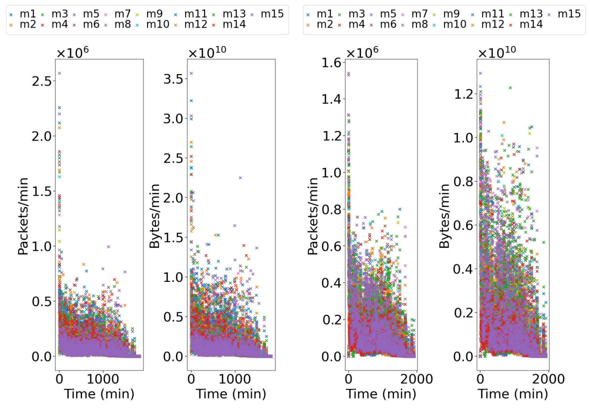


(a) S-C8220-C6320-1G (b) M-C8220-W220g5-1G

**Figure 10: Traffic Between HDFS DataNodes for AVAH$_A$**

## Analysis of IP Throughput

Figure 11 shows the IP throughput for AVAH$_A$.

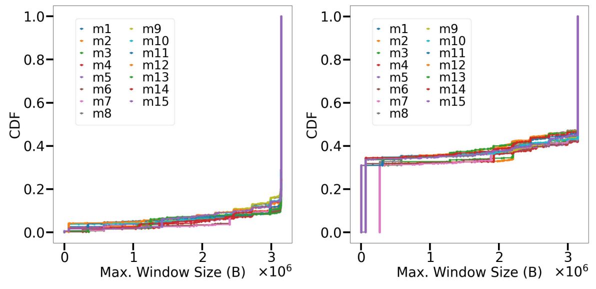


(a) S-C8220-C6320-10G (b) M-C6320-W220g2-10G

**Figure 11: AVAH$_A$: IP Throughput**

## Analysis of TCP Window Size

Figure 12 shows the CDF of max. TCP window size for AVAH$_A$.



(a) S-C8220-c6320-1G (b) M-C8220-W220g5-1G

**Figure 12: AVAH$_A$: CDF of Max. TCP Window Size**