Accelerating Comparative Genomics Workflows in a Distributed Environment with Optimized Data Partitioning

Olivia Choudhury, Nicholas L. Hazekamp, Douglas Thain, Scott Emrich

***Abstract*—The advent of new sequencing technology has generated massive amounts of biological data at unprecedented rates. High-throughput bioinformatics tools are required to keep pace with this. Here, we implement a workflow-based model for parallelizing the data intensive task of genome alignment and variant calling with BWA and GATK's HaplotypeCaller. We explore different approaches of partitioning data and how each affect the run time. We observe granularity-based partitioning for BWA and alignment-based partitioning for HalotypeCaller to be the optimal choices for the pipeline. We identify the various challenges encountered while developing such an application and provide an insight into addressing them. We report significant performance improvements, from 12 days to 4 hours, while running the BWA-GATK pipeline using 100 nodes for analyzing high-coverage oak tree data.**

I. Introduction

Next generation sequencing (NGS) [1] techniques have had widespread impact in fields such as molecular medicine, evolution, human migration, DNA forensics, and agriculture by inexpensively generating Giga base-pairs (Gbp) per machine day [2]. As sequencing throughput increases, the major research bottlenecks are the time and computational resources demanded for managing, deciphering and analyzing such large-scale biological data. Genome alignment and variant calling are the two most important stages of comparative genomics applications, which often require weeks or months for downstream analysis of NGS data. Thus, developing high throughput workflows for such data-intensive applications is an important and challenging problem in bioinformatics.

In the last few years, many alignment and variant discovery tools have implemented sophisticated algorithms to reduce computational resource requirement and run time for analyzing massive biological data [3–8]. The extent of parallelism that can be achieved by adopting the optimization techniques offered by these tools is often limited. For instance, Burrows Wheeler Aligner (BWA) [9] is one of the fastest and most accurate alignment tools currently available. For our test data of 100-fold coverage of 50 oak individuals' genome, the multithreading option provided by BWA could not cause a significant reduction in run time. On the other hand, GATK's SNP and indel calling walker HaplotypeCaller [10] generates accurate results compared to the other variant calling tools, but often requires time in the order of weeks or months to analyze high coverage NGS data. HaplotypeCaller required 12 days to determine variants for our test data. Again, the in-built options for multithreading, pruning, and downsampling was unable to cause a considerable improvement in run time for our data.

As the underlying algorithms of both the tools support parallelism, the trade off between accuracy and speedup can be mitigated by creating a parallelizable workflow [11]. Such a workflow can harness resources from distributed systems by dividing the workload into independent tasks and executing each task parallely. In this regard, one of the key factors responsible for achieving a considerable speedup is **efficient partitioning of data** at the preliminary stage of the workflow. As BWA and HaplotypeCaller do not provide any method of data partitioning, we explored different ways of doing so, some specific to a particular tool. For each tool, we identified the best partitioning technique and tested the same for the entire pipeline. We also identified the potential challenges that are likely to be encountered while developing such data-intensive applications and proposed solutions for them.

For BWA, we partitioned the data following granularity-based and individual-based approaches. For the former, we split the query file based on a predetermined granularity value whereas for the latter, we split it based on individual names. We observed that although times taken for the alignment procedure in both the approaches were similar, there was an additional cost incurred in the splitting step of individual-based approach. Here, each read of the query data had to be compared with the barcode information to match an individual name, which was time consuming for approximately 140 million reads. Thus, granularity-based approach of data decomposition was efficient for parallelized execution of BWA.

The second stage of the pipeline involved variant calling using HaplotypeCaller. Similar to BWA, we tested granularity-based and individual-based approaches for the input data in sorted BAM format [6]. In addition to that, we adopted a new method, named **alignment-based partitioning**. We first split the reference file into multiple bins, each containing unique contigs. We then split the concatenated SAM output of BWA into smaller files, each having alignment information pertaining to a reference bin. For each approach, we executed HaplotypeCaller for a pair of sorted BAM file and reference file. Contrary to the other two approaches, the alignment-based approach caused a significant run time improvement for HaplotypeCaller as the search space was considerably reduced and the smaller reference files were guaranteed to have the necessary contigs corresponding to the alignment information in each SAM file. The overall run time of the pipeline for genome mapping and variant calling, which earlier took approximately 12 days, was drastically reduced to 4 hours.

Corresponding Author. Email: semrich@nd.edu.
All authors are affiliated with the Department of Computer Science and Engi neering at the University of Notre Dame, Notre Dame, IN, USA.

The rest of the paper is organized as follows. We discuss our adopted methods for this work in Section II. In Section III, we present the results obtained by parallelizing different phases of the pipeline and how each of them scale in a heterogeneous cloud environment. We explore different approaches of data distribution and their effects on run time. Section IV discusses the challenges we came across while developing this data-intensive framework and our proposed solutions. Section V provides an overview of the literature related to our work.

II. Method

A. Overview of Genome Alignment and BWA

The millions of short reads generated by next generation sequencing techniques are usually compared to the genome of a model organism, known as the reference genome, for further biological analysis. One accurate means of genome comparison is genome alignment. In Fig. 1, the vertical bars denote the positions of matches whereas the hyphens represent insertions or deletions, commonly known as indels. There are two major categories of mapping algorithms. The first implements the Burrows Wheeler Transform (BWT) [12] for data compression, and the second one is based on hashing to speed up alignments. Depending on the genome size, the entire alignment procedure may take several days to compute. For instance, short read alignment tools like MAQ [13] and SOAP [14] typically require more than 5 CPU-months and 3 CPU-years, respectively, for aligning 140 billion base pairs (Bbp) [15].

```
A: C A T - T C A - C
   |   |       |   |
B: C - T C G G A G C
```

Fig. 1: Genome alignment between sequences A and B showing matches, mismatches, and indels

One of the widely used alignment tools, Burrows Wheeler Alignment (BWA) employs the Burrows Wheeler Transform (BWT) algorithm to align short queries with low error rate as well as long queries with higher error rates. BWA is light-weight, supports paired-end mapping, gapped alignment, and various file formats, like ILLUMINA [16] and ABI SOLiD [17]. The default output format for all its algorithms is SAM (Sequence Alignment Map), which can further be analyzed using the open-source SAMtools package and others to visualize the alignments and call variants [6].

B. Overview of Variant Calling and HaplotypeCaller

Variants are allelic sequences that differ from the reference sequence by a single base pair or a comparatively longer interval. For instance, in Fig. 1, at the fifth position of the aligned sequence there occurs a mismatch or single nucleotide polymorphism (SNP). SNPs can account for the variation in phenotypic traits and disease susceptibility among different individuals. These variants can also serve as markers in genome wide association studies (GWAS) and help in identifying genes associated with various traits or diseases. The performance of a variant calling tool is largely dependent on the quality and coverage of sequencing datasets. Popular variant calling tools like SAMtools and MAQ use a Bayesian statistical model to evaluate the posterior probability of these genotypes.

GATK employs similar, yet more sophisticated Bayesian algorithms. Here we consider its HaplotypeCaller walker which is often preferred over the UnifiedGenotyper tool because of the former's higher sensitivity. HaplotypeCaller functions by indexing the input set and creating walkers to locate variations between the query and reference. Once a difference is detected, HaplotypeCaller performs local assemblies to fill gaps or correct mistakes. Though it generally has more accurate results than UnifiedGenotyper, it is less frequently used because it takes substantially longer to complete. This difference in performance is due to the fact that HaplotypeCaller checks more variants, both indels and SNPs, for a given sequence. To remedy the slower performance, we opted for parallel execution of different subtasks.

C. Framework of the parallelized pipeline

For our active projects, BWA (version 0.7.5) and GATK's (version 2.5-2) SNP and indel calling walker HaplotypeCaller produced the best biological results (unpublished). As seen in TABLE VI, the method of genome alignment, preparation of inputs for subsequent variant calling, and determination of variants using these tools in a sequential order took 12 days to complete. Many newly developed variant detection tools, including GATK's HaplotypeCaller improves genotype calling by increasing runtime [18], which is a major bottleneck.

A possible approach of addressing this problem is to develop efficient algorithms or data structures to reduce the search space during the computationally expensive task of mapping or variant calling [19]. Considering the massive size of NGS data and the considerably high computational resource requirement for their analysis, parallelizing the tools using a workflow-based model that can harness resources from clusters, clouds, and grids is a more tractable solution. We developed workflows to divide the tasks of genome alignment and variant discovery and ran each independent task parallely on remote machines. We used the Makeflow [20] language to define each task along with its dependencies. Makeflow can apply various systems, including multi-core machines, batch systems like the Sun Grid Engine (SGE) [21] and Condor Pool [22] to execute the tasks. For our experiments, we enabled Makeflow to use the Work Queue [23] master-worker framework as an alternative scalable framework to process data across clusters, clouds, and grids. The Work Queue framework used available machines by handling the data transfer and caching of files when running jobs remotely. Each worker executed an assigned task and reported to the master after completion or if there was a discrepancy.

We identified that a key feature for assuring improved run time during parallelization is efficient partitioning of data. We adopted different approaches of data decomposition for BWA and HaplotypeCaller and compared and contrasted each combination using Work Queue-derived resources.

- For **granularity-based partitioning**, we tested the workflow by varying the number of reads contained in each smaller file. We determined the optimal granularity value or partition size to be the one for which

run time was the lowest. The number of partitions or smaller files was proportional to the partition size. For our query data containing approximately 140 million reads, splitting it into smaller chunks such that each contained 200000 reads proved to be the optimal choice. The splitting resulted in the generation of 715 smaller files, which were then analyzed parallely.

- For **individual-based partitioning**, we used coarser granularity by dividing the pooled data into multiple files, each corresponding to an individual. As our data comprised genomic sequences of 50 individuals, we divided it into 50 files based on individual names.

In both cases, we assigned the task of running BWA on each pair of small query data and reference data to a work queue worker. At the completion of mapping phase, we added read group and platform information to each output (SAM file) and compressed them into sorted and indexed binary versions (BAM) to be compatible with HaplotypeCaller.

In the next phase of the pipeline, we tested three ways of partitioning mapped data for variant calling.

- For **granularity-based partitioning**, similar to BWA, we split the input data for HaplotypeCaller on the basis of the determined optimal granularity value.
- For **individual-based partitioning**, the mapped data was split based on individual names, resulting in 50 smaller files.
- For the new mechanism, called **alignment-based partitioning**, we split the SAM file based on alignment information. We first split the reference file into multiple smaller files, each having a fixed number of non-overlapping contigs. For each small reference file, we split the pooled SAM file such that the smaller SAM file would contain information of reads that aligned only to the contigs of that particular reference subset.

For each approach, we executed HaplotypeCaller for a pair of smaller sorted BAM file and the reference sequence on a work queue worker. The output of HaplotypeCaller was in VCF format [24]. In the final step of the pipeline, we concatenated individual VCF files into a single VCF file containing variant information of the entire population, that we began the analysis with. As we will discuss in Section III, granularity-based and alignment-based data partitioning were the most efficient approaches for running the parallelized pipeline for BWA and HaplotypeCaller, respectively.

III. Results

For our experiment, we aligned and detected variants for 50 *Quercus rubra* (northern red oak) individuals' ILLUMINA HiSeq RAD [25] data with over 100-fold coverage. The temporary reference sequence for this non-model species contained approximately 400000 individual loci. The system used for these analysis consisted of a cluster of 26 machines, each with 8 cores and 32 GB RAM. To illustrate our base, BWA (single end) required about 4 hours to complete the alignment procedure sequentially, and GATK's HaplotypeCaller required 12 days to detect SNPs and indels. Run times for BWA does not include the time for indexing the reference, which can be done once and re-used for each alignment task. Next, we will discuss different techniques explored for partitioning the data to find an optimal solution.

A. Partitioning

The first approach of data partitioning involved splitting the input file into multiple smaller files based on a granularity value. It allowed the size of the smaller files to vary depending on the number of reads allowed in each file, as specified in the split script. For this particular instance, we tested different sizes and chose 200000 reads in each file, which was within the same order of size as the reference, to be the optimal granularity value. This was known as the granularity-based approach of data decomposition. For this optimal value, the test data created 715 files and tasks, which were then run in parallel.

For the second approach, we adopted individual-based partitioning of data. As mentioned earlier, our test data comprised genomic data of 50 oak individuals. This organization lead to a simple structure of splitting the data based on individual names, which was the basis for individual-based partitioning. This method required comparing each read in the query file with the barcodes provided by RAD data to find the name of the individual to which it belonged.

The above two methods were used to evaluate both BWA and GATK's HaplotypeCaller. Fig. 2 illustrates the workflow of these approaches while parallelizing BWA. The split function refers to the creation of smaller query files based on a granularity value (for granularity-based partitioning) or individual names (for individual-based partitioning). Fig. 3 depicts the workflow for executing parallelized HaplotypeCaller for granularity-based and individual-based approaches.

The third approach, alignment-based partitioning, was applicable only to HaplotypeCaller as it depended on the alignment information obtained from the output of mapping procedure. First, we split the reference sequence into several chunks, each containing unique and contiguous contigs. Once the allocation of contigs to each bin was known, the output of BWA, in concatenated SAM format, was split based on the contigs to which the reads aligned. Each pair of smaller reference file and its corresponding small SAM file was then executed parallely with HaplotypeCaller. Although this implementation was specific to HaplotypeCaller, the use of alignment information in pruning the search space could be adopted in a wide variety of parallelized genomic applications. This method resulted in 10 tasks, involving a more directed searching. Fig. 4 provides a visualization of this approach as implemented in HaplotypeCaller.

B. Tool Improvement

Once a splitting script was created implementing each of the above-mentioned partitioning schemes, BWA and HaplotypeCaller were executed using 100 workers. TABLE I presents the runtimes for BWA and HaplotypeCaller using different data partitioning approaches. For BWA, the alignment and concatenation stages of granularity-based and individual-based approaches required similar time. The step of splitting the query in individual-based partitioning was a bottleneck due to the higher look up time to match an individual's name from

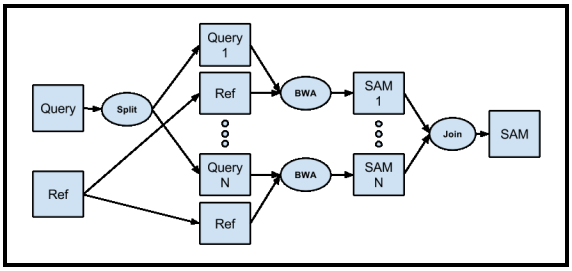

Fig. 2: Framework of granularity-based and individual-based data partitioning approaches in BWA. For granularity-based approach, the query data was split into several smaller files (N=715) based on a granularity value. For individual-based approach, the query file was split into multiple files (N=50) based on individual names. The remaining steps were identical for both approaches.

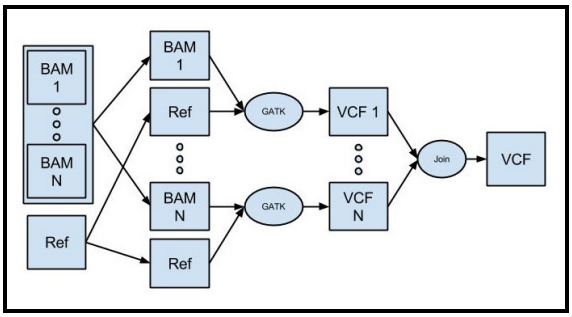

Fig. 3: Framework of granularity-based and individual-based data partitioning approaches in HaplotypeCaller. The SAM file was split (N=50 for individual-based and N=715 for granularity-based) into multiple smaller files. The reference file and each sorted BAM file was sent to a worker for executing HaplotypeCaller. The individual outputs were joined to create a single VCF file.

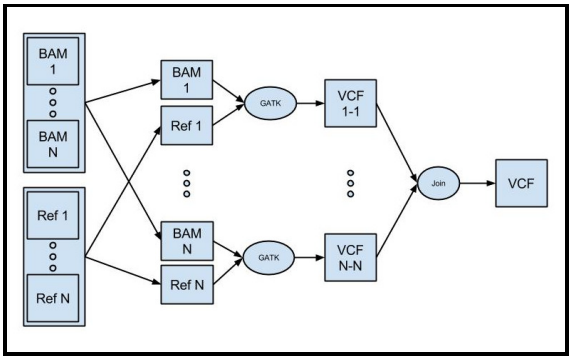

Fig. 4: Framework of alignment-based data partitioning approach in HaplotypeCaller. The reference file was split into bins and the SAM file was split into smaller files based on the contigs in the bins to which the reads aligned. Each pair of smaller reference file and its corresponding sorted BAM file was then sent to a worker to run HaplotypeCaller. The outputs were concatenated into a single VCF file.

barcode information. The granularity-based approach did not involve this searching step and hence was more efficient (TABLE I: shown in bold font). For HaplotypeCaller, preparation of input comprised the times for splitting the concatenated SAM file based on each approach, adding read group information, and converting SAM to sorted and indexed BAM formats. Although there was an additional cost in splitting the SAM file for alignment-based partitioning, the overall runtime was significantly lower (TABLE I: shown in bold font) than the other two approaches.

After inferring granularity-based approach to be the best for BWA, we tested its run time behavior from 2 workers up to 100 workers. Fig. 5 shows how the framework performed with increased number of workers. As more workers were used, the increase in performance diminished rapidly. For this particular data set and machine configuration, we achieved the best performance with 100 workers, after which adding more workers did not improve run time. However, when compared to 20 workers, the 50 and 100 worker iterations had only a minor performance improvement, showing the system did not scale linearly. This was due to the additional time incurred in data transfer, particularly in sending all the query files, reference and index files to each worker, which caused an overhead as we added more workers. TABLE II supports this fact as we can see the amount of data sent and the time required for it increased with more workers. Fig. 6 presents a histogram for running 715 tasks in each stage (bwa aln and samse) of BWA. It gives an overview of the overall behavior of granularity-based BWA tasks when allowed to use 100 workers. Similarly, Fig. 7 provides a visualization of the run time behavior of the same approach for 100 workers. The number of tasks submitted and completed followed a similar pattern until the former reached a plateau denoting submission of all tasks while there were some tasks yet to be complete. The number of tasks running followed a steady line due to the use of a dedicated cluster. We also tested how the framework behaved when doing more work (using the 'k3' and 'k4' flags of 'bwa aln' to increase the run time search space). The parallelized implementation of BWA could handle more load and exhibited similar behavior when subjected to more tasks, as shown in Table III.

As seen in TABLE I, for HaplotypeCaller, alignment-based approach was most efficient. While splitting the SAM file into smaller SAM files, based on the contigs to which the reads aligned, we optimized the method of looking up the reference sequence to detect variants. Instead of searching the entire reference file, we searched through a small section of the reference sequence that was guaranteed to contain contig information for all the alignments in the small SAM file. During splitting of the SAM file we also ignored the entries that did not align to contigs, reducing the number of elements to process. The other two approaches did not have the scope of reducing the search space during variant detection. Also, the overhead of sending the entire reference file to a worker for each task was responsible for increased run times in individual-based and granularity-based approaches of HaplotypeCaller.

C. Pipeline Improvement

Based on the aforementioned results, we generated an optimized pipeline for genome alignment and variant calling combining BWA and GATK's HaplotypeCaller, as shown in Fig. 8. TABLE IV presents the average time taken by a task running BWA and Haplotyper. This time represents the

Runtime (mins.)								
Partitioning	**BWA**				**HaplotypeCaller**			
	Split	**Alignment**	**Concatenation**	**Total**	**Preparation of input**	**Variant Calling**	**Concatenation**	**Total**
Granularity	13	21	22	**56**	17	1272	5.5	1294.5
Individual	73	18	20	111	74	1007	9	1090
Alignment	-	-	-	-	158	24.6	4	**186.6**

TABLE I: Comparison of runtimes (in minutes) for different approaches of data partitioning in BWA and HaplotypeCaller using 100 workers. Due to lower splitting time, granularity-based partitioning was the best approach for BWA. For HaplotypeCaller, alignment-based partitioning of data was the most optimal choice due to reduced search space.

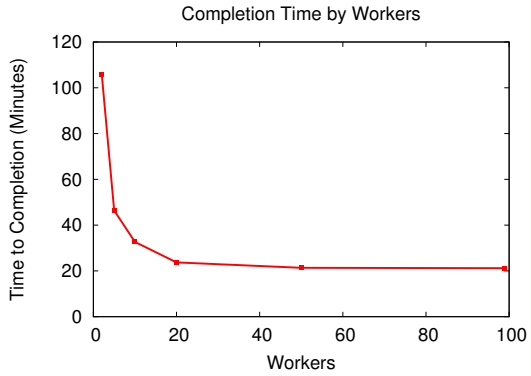

Fig. 5: Runtime of granularity-based BWA for increasing number of workers. The time accounts for the parallelized alignment step of BWA, and does not consider the times for splitting the query sequence or concatenating the outputs. The run time gradually decreases with more workers. For large number of workers the overhead of data transfer hindered further improvement.

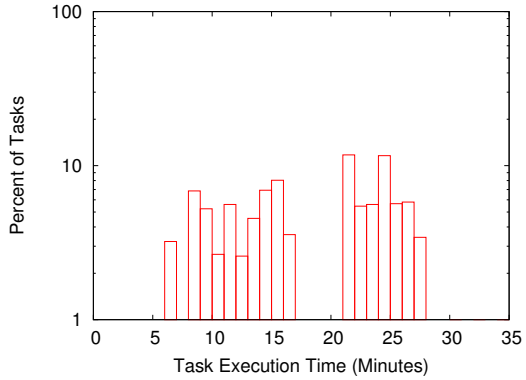

Fig. 6: The histogram for BWA is bimodal due to the alignment (aln) and generation of reads in SAM format (samse) steps. These times were measured for 1430 tasks, where 715 tasks were attributed to each process.

generalized functioning of the pipeline and can be considered to be independent of the underlying batch system. On the other hand, TABLE V gives an insight into the runtimes specific to our pipeline which employed 100 work queue workers in a heterogeneous, distributed system. The table provides a breakdown of run times for individual components of the pipeline. The efficient data partitioning followed by parallelization of tasks reduced the runtimes of each phase, resulting in a considerable overall performance improvement.

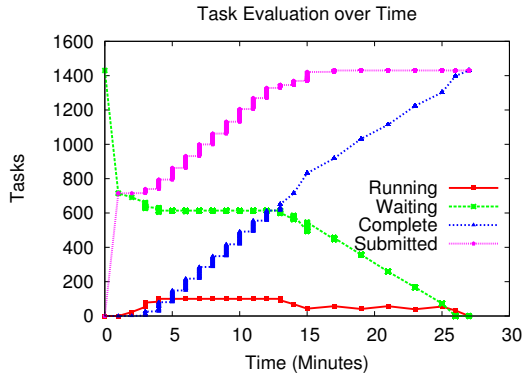

Fig. 7: Run time behavior of granularity-based BWA using 100 workers.

Workers	Total data transferred (Mb)	Total transfer time (s)
2	64266	594
5	65913	593
10	67522	598
20	70350	623
50	74534	754
100	80276	765

TABLE II: Total amount of data transferred and total time taken while using up to 100 workers for granularity-based BWA. It transferred the reference file of size 36.7 MB, 715 small query files, each of size approximately 46 MB and index files of size 80 MB. The amount of data transferred and time taken caused an overhead in the case of more workers.

Runtime (mins.)			
Workers	**-k 2**	**-k 3**	**-k 4**
2	141.9	482.3	576.0
5	74.5	222.9	260.4
10	68.7	123.8	163.7
20	59.0	99.5	115.0
50	57.1	77.6	77.4
100	56.0	64.3	68.33

TABLE III: Run times for granularity-based BWA aln with '-k2', '-k3', and '-k4' options of alignment. For more load (with '-k3' and '-k4'), BWA scaled better with increased number of workers.

Runtime / task (mins.)			
BWA		HaplotypeCaller	
Average	S.D.	Average	S.D.
7.2	3.5	7.5	5.9

TABLE IV: Average time and standard deviation for each task executing BWA and HaplotypeCaller for the parallelized framework. This information indicates the performance of the generalized workflow and can be useful if the framework is later designed to harness other batch systems.

Table VI compares the times when running the individual tools sequentially to that when run in our optimized pipeline with 100 workers in a cloud environment.

IV. Challenges

Understanding the extent of parallelism: One of the most important factors while developing a parallel application is understanding if the underlying algorithm exhibits potential for parallelism. In each step of this pipeline, individual tasks were independent, without one task having to wait for the completion of another, but the data decomposition was critical. In this context, the commands written in the Makeflow script represented the order of execution of tasks as a directed acyclic graph (DAG) [26] but additional work was needed to do it in the best manner. As shown in our results, adding more workers does not guarantee reduced run time. Resource contention, lack of sufficient tasks to exploit the availability of all workers, few tasks with abnormally long run time, network traffic, and higher data transfer time often cause an overhead in such data-intensive parallel applications.

Using a heterogeneous environment without shared file system: The procedure of data transfer while executing tasks in a distributed environment becomes challenging in the absence of a shared file system. The use of work queue workers mitigates this problem as they solely depend on the files transferred by the masters, as specified in the makeflow rules. For even larger data-intensive applications that require various clusters to scale up their performance, a hierarchical implementation of work queue comprising a multi-slot worker and a foreman [27] can reduce the overhead of data transfer, causing an improvement in performance. For our application, because the shared data (reference file) was comparatively smaller than unique data (sequence/BAM), we did not benefit from these features, but others may.

Determining the granularity value: One of the important challenges in developing parallel programs is determining the optimal granularity to balance the times spent on communication, particularly the overhead due to file transfer, and computation. Earlier studies [28] have shown that a significant improvement in performance can be achieved if the number of tasks and size of each task can be optimized. This trade off is based on the underlying algorithm as well as the configuration of the machine on which the program runs. For a heterogeneous system like the Condor pool, care should be taken while determining the optimal value of granularity; however we have shown our framework is consistent under load and will use the maximum parallelism available.

Connecting to the master programs: We recognize that a generic master-worker framework should be secure, specially while dealing with sensitive patient data. These challenges can be addressed by using the Catalog server, which advertises the master processes along with their project names, port numbers, resource requirements. The workers connect to the masters by matching the requirements. This connection can be password protected to ensure secure data transfer between the masters and the workers.

Maintaining workers: Although our scalable application was tested with Condor, it can use any other batch system to generate workers. If a worker in the pool failed or was evicted, an advantage of the pool is that a new worker could be started and the task reassigned, increasing fault tolerance without affecting available parallelism.

V. Related Work

Genome mapping and assembly have traditionally been refactored using MPI (Message Passing Interface) [29] or MapReduce [30] frameworks. ClustalW-MPI [31] is a distributed version of the multiple sequence aligner ClustalW [32] that uses MPI to run on distributed memory systems. CloudBurst [33] is an early application that implements the seed-and-extend genome mapping algorithm on a Hadoop [34] based MapReduce framework. Although CloudBurst can align millions of reads in a reasonable time, it takes longer to process billions of reads when compared to a multi-core version of another cloud-based variant calling tool Crossbow [35]. Further, CloudBurst identifies all possible alignments in a read, which is often not a requirement in current comparative genomics projects. Crossbow employs Hadoop to parallelize Bowtie for alignment and SOAPsnp [36] for variant calling. It does not natively support gapped alignment. SEAL [37] is another scalable aligner based on BWA and Picard that uses the MapReduce framework for parallelization of tasks.

Although the MPI framework allows execution of parallel programs, it requires complicated software development for refactoring tools. With Hadoop's MapReduce-based parallelization, it is not easy to tune the parameters for granularity, as was required in our experiments. Hadoop implements its own method of mapping tasks and reducing them on completion. Also, our application was capable of dynamically scaling up or down the workers as needed, which is difficult to implement in Hadoop's MapReduce-based framework. Finally, unlike Hadoop, our developed framework could harness resources from a heterogeneous system. Our system supported these features by implementing the Makeflow language and Work Queue master-worker framework to generate a parallelized workflow for comparative genomics applications. In this paper, we also analyzed the effect of different data partitioning approaches on the runtime of mapping and variant calling tools. Although we considered BWA and HaplotypeCaller as a test case, the approaches of data decomposition presented in this paper are generic and can be implemented to improve the performance of other bioinformatics tools. Particularly, the tools whose underlying algorithm or data structure is similar to our test tools, can potentially benefit from our proposed approach of optimized data partitioning. Moreover, the discussion on the possible barriers and the solutions to overcome them is useful for adapting parallelized workflows on ad hoc clouds. This in turn can aid the development of

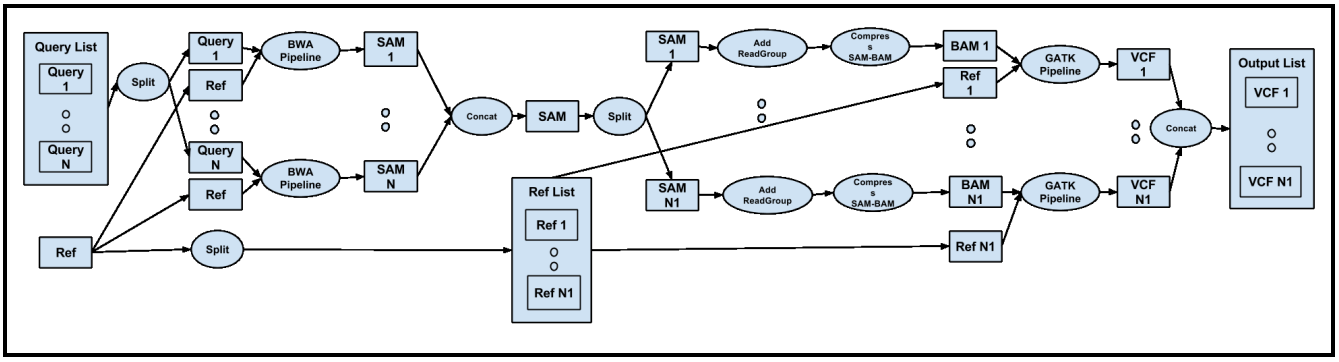

Fig. 8: Framework of the optimized pipeline incorporating granularity-based BWA and alignment-based HaplotypeCaller.

Runtime (mins.)									
Split Query	**Parallelized BWA**	**Concat SAM**	**Split Ref**	**Split SAM**	**Add RGs**	**SAM to BAM**	**Parallelized GATK**	**Concat VCF**	**Total**
13	21	22	3	107	11	40	24	4	4.08 hrs.

TABLE V: Run times of individual steps of the optimized pipeline using 100 workers. Splitting of data in BWA follows granularity-based partitioning and that for HaplotypeCaller implements alignment-based partitioning of data.

Runtime			
	BWA	**HaplotypeCaller**	**Pipeline**
Sequential	4 hrs. 04 mins.	12 days	12 days
Parallel	0 hr. 56 mins.	0 hr. 24 mins.	4 hrs. 05 mins.

TABLE VI: Comparison of run times for sequential and parallel execution of the pipeline. The total time for running the pipeline also includes the time required for preparing the input files for Haplotypecaller, like addition of read groups and conversion of SAM file to sorted and indexed BAM formats. The run time for the parallelized execution represents the times taken by the optimized pipeline for 100 workers.

efficient bioinformatics applications that can harness resources from a heterogeneous, distributed environment.

VI. Conclusion

We addressed the problem of performance improvement in a pipeline for genome mapping and subsequent variant discovery by developing a workflow to run parallely on a heterogeneous, distributed system. We identified that one of the important factors to enable efficient parallelism is optimized partitioning of data. In this regard, we discussed different techniques of data decomposition, namely granularity-based, individual-based, and alignment-based partitioning. We considered BWA for genome alignment and GATK's HaplotypeCaller for SNP and indel calling as our test tools. We tested both the tools for all the approaches of data partitioning and concluded that granularity-based partitioning was the best approach for BWA. Unlike individual-based partitioning, this method did not require the overhead of comparing each read in the query with the barcode information to find a matching individual name. For HaplotypeCaller, alignment-based approach was proved to be the most efficient technique. By splitting the reference file and the SAM file accordingly, it reduced the search space to a considerable extent, which in turn lowered the runtime from days to hours. After selecting the optimized data partitioning approach for each tool, we combined them into a pipeline framework that could efficiently analyze NGS data in a substantially reduced time. For the test dataset comprising genomic data of 50 oak individuals, our optimized pipeline required approximately 4 hours when run parallely with optimized number of workers, in this case 100 workers, as opposed to 12 days, when the tools were run sequentially. We further discussed the potential challenges that are likely to be encountered while parallelizing such data-intensive applications and proposed a solution to each.

Acknowledgment

The authors would like to thank Jeanne Romero-Severson for providing the Illumina data used to test our framework. This work was supported in part by National Institutes of Health/National Institute for Allergy and Infectious Diseases (grant number HHSN272200900039C) to SJE and National Science Foundation grant SI2-SSE (number 1148330) to DT.

References

[1] J. Reis-Filho, "Next-generation sequencing," *Breast Cancer Res*, vol. 11, no. Suppl 3, p. S12, 2009.

[2] M. Metzker, "Sequencing technologiesthe next generation," *Nature Reviews Genetics*, vol. 11, no. 1, pp. 31–46, 2009.

[3] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg *et al.*, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome," *Genome Biol*, vol. 10, no. 3, p. R25, 2009.

[4] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.

[5] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang, "Soap2: an improved ultrafast tool for

short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, 2009.

[6] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin *et al.*, "The sequence alignment/map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[7] R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang, "SNP detection for massively parallel whole-genome resequencing," *Genome research*, vol. 19, no. 6, pp. 1124–1132, 2009.

[8] D. C. Koboldt, K. Chen, T. Wylie, D. E. Larson, M. D. McLellan, E. R. Mardis, G. M. Weinstock, R. K. Wilson, and L. Ding, "Varscan: variant detection in massively parallel sequencing of individual and pooled samples," *Bioinformatics*, vol. 25, no. 17, pp. 2283–2285, 2009.

[9] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[10] M. DePristo, E. Banks, R. Poplin, K. Garimella, J. Maguire, C. Hartl, A. Philippakis, G. del Angel, M. Rivas, M. Hanna *et al.*, "A framework for variation discovery and genotyping using next-generation DNA sequencing data," *Nature genetics*, vol. 43, no. 5, pp. 491–498, 2011.

[11] I. Lanc, P. Bui, D. Thain, and S. Emrich, "Adapting bioinformatics applications for heterogeneous systems: a case study," *Concurrency and Computation: Practice and Experience*, 2012.

[12] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," 1994.

[13] H. Li, J. Ruan, and R. Durbin, "Mapping short dna sequencing reads and calling variants using mapping quality scores," *Genome research*, vol. 18, no. 11, pp. 1851–1858, 2008.

[14] R. Li, Y. Li, K. Kristiansen, and J. Wang, "Soap: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, 2008.

[15] T. J. Ley, E. R. Mardis, L. Ding, B. Fulton, M. D. McLellan, K. Chen, D. Dooling, B. H. Dunford-Shore, S. McGrath, M. Hickenbotham *et al.*, "Dna sequencing of a cytogenetically normal acute myeloid leukaemia genome," *Nature*, vol. 456, no. 7218, pp. 66–72, 2008.

[16] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants," *Nucleic acids research*, vol. 38, no. 6, pp. 1767–1771, 2010.

[17] O. Harismendy, P. C. Ng, R. L. Strausberg, X. Wang, T. B. Stockwell, K. Y. Beeson, N. J. Schork, S. S. Murray, E. J. Topol, S. Levy *et al.*, "Evaluation of next generation sequencing platforms for population targeted sequencing studies," *Genome Biol*, vol. 10, no. 3, p. R32, 2009.

[18] R. Nielsen, J. S. Paul, A. Albrechtsen, and Y. S. Song, "Genotype and snp calling from next-generation sequencing data," *Nature Reviews Genetics*, vol. 12, no. 6, pp. 443–451, 2011.

[19] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.

[20] L. Yu, C. Moretti, A. Thrasher, S. Emrich, K. Judd, and D. Thain, "Harnessing parallelism in multicore clusters with the all-pairs, wavefront, and makeflow abstractions," *Cluster Computing*, vol. 13, no. 3, pp. 243–256, 2010.

[21] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*. IEEE, 2001, pp. 35–36.

[22] M. Litzkow, M. Livny, and M. Mutka, "Condor-a hunter of idle workstations," in *Distributed Computing Systems, 1988., 8th International Conference on*. IEEE, 1988, pp. 104–111.

[23] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain, "Work queue+ python: A framework for scalable scientific ensemble applications," in *Workshop on Python for High Performance and Scientific Computing at SC11*, 2011.

[24] O. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry *et al.*, "The variant call format and vcftools," *Bioinformatics*, vol. 27, no. 15, pp. 2156–2158, 2011.

[25] M. R. Miller, J. P. Dunham, A. Amores, W. A. Cresko, and E. A. Johnson, "Rapid and cost-effective polymorphism identification and genotyping using restriction site associated dna (rad) markers," *Genome Research*, vol. 17, no. 2, pp. 240–248, 2007.

[26] S. Handley, "On the use of a directed acyclic graph to represent a population of computer programs," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. IEEE, 1994, pp. 154–159.

[27] M. Albrecht, D. Rajan, and D. Thain, "Making work queue cluster-friendly for data intensive scientific applications," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–8.

[28] C. Moretti, J. Bulosan, D. Thain, and P. J. Flynn, "All-pairs: An abstraction for data-intensive cloud computing," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–11.

[29] A. Darling, L. Carey, and W.-C. Feng, "The design, implementation, and evaluation of mpiBLAST," *Proceedings of ClusterWorld*, vol. 2003, 2003.

[30] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[31] K.-B. Li, "ClustalW-MPI: Clustalw analysis using distributed and parallel computing," *Bioinformatics*, vol. 19, no. 12, pp. 1585–1586, 2003.

[32] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic acids research*, vol. 22, no. 22, pp. 4673–4680, 1994.

[33] M. Schatz, "Cloudburst: highly sensitive read mapping with mapreduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[34] A. Hadoop, "Hadoop," 2009.

[35] B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg, "Searching for SNPs with cloud computing," *Genome Biol*, vol. 10, no. 11, p. R134, 2009.

[36] R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang, "Snp detection for massively parallel whole-genome resequencing," *Genome research*, vol. 19, no. 6,

pp. 1124–1132, 2009.
[37] L. Pireddu, S. Leo, and G. Zanetti, "SEAL: a distributed short read mapping and duplicate removal tool," *Bioinformatics*, vol. 27, no. 15, pp. 2159–2160, 2011.