DOI 10.12694/scpe.v16i1.1060
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 1, pp. 53–69. http://www.scpe.org

ACCELERATING COMPARATIVE GENOMICS WORKFLOWS IN A DISTRIBUTED ENVIRONMENT WITH OPTIMIZED DATA PARTITIONING AND WORKFLOW FUSION

OLIVIA CHOUDHURY, NICHOLAS L. HAZEKAMP, DOUGLAS THAIN, SCOTT J. EMRICH*

Abstract. The advent of next generation sequencing technology has generated massive amounts of biological data at unprecendented rates. Comparative genomics applications often require compute-intensive tools for subsequent analysis of high throughput data. Although cloud computing infrastructure plays an important role in this respect, the pressure from such computationally expensive tasks can be further alleviated using efficient data partitioning and workflow fusion. Here, we implement a workflow-based model for parallelizing the data-intensive tasks of genome alignment and variant calling with BWA and GATK's HaplotypeCaller. We explore three different approaches of partitioning data, **granularity-based**, **individual-based**, and **alignment-based** , and how each affect the run time. We observe **granularity-based partitioning** for BWA and **alignment-based partitioning** for HaplotypeCaller to be the optimal choices for the pipeline. We further discuss the methods and impact of workflow fusion on performance by considering different levels of fusion and how it affects our results. We identify the various open problems encountered, such as understanding the extent of parallelism, using heterogenous environments without a shared file system, and determining the granularity of inputs, and provide insights into addressing them. Finally, we report significant performance improvements, from 12 days to under 2 hours while running the BWA-GATK pipeline using partitioning and fusion.

Key words: genome alignment, variant calling, workflow fusion, data partitioning, performance

AMS subject classifications. 68M14, 92D20

1. Introduction.

Next generation sequencing (NGS) [35] 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 [29]. 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 [17, 15, 27, 20, 24, 13]. The extent of parallelism that can be achieved by adopting the optimization techniques focused on multicore servers is often limited. For instance, Burrows Wheeler Aligner (BWA) [19] is one of the fastest and most accurate alignment tools currently available. However, for our test data of 100-fold coverage of 50 oak individuals' genomes the multithreading option provided by BWA did not significantly reduce run time. Similarly, GATK's SNP and indel calling walker HaplotypeCaller [8] generates more accurate results compared to the other variant calling tools, but often requires weeks or months to analyze high coverage NGS data. For example, HaplotypeCaller required 12 days to determine variants for our test data, and the in-built options for runtime improvement on a multicore machine also did not considerably reduce run time.

As the underlying algorithms of both tools support coarse-grained parallelism, the trade off between accuracy and speedup can be mitigated by creating a parallelizable workflow [14]. Such a workflow can harness resources from distributed systems by dividing the workload into independent tasks and executing the tasks 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 discussed several open problems that are likely to be encountered while developing such data-intensive applications, particularly understanding the scope of parallelism within an entire pipeline, finding the optimal partitioning value for parallelism, and using a heterogeneous environment without a shared file system.

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

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 optimal granularity value whereas for the latter, we split it based on individual names. We observed that times taken for the alignment procedure in both approaches were similar, but there was an additional cost incurred in the splitting step of the **individual-based** approach. Here, each read from the query had to be compared with the barcode information, which ended up being 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 containing alignment information in BAM format [20]. In addition to that, we adopted a new method, named **alignment-based partitioning**. Here, 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. We then run HaplotypeCaller for a pair of sorted BAM file and its corresponding reference bin. 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 while preserving all required data in smaller reference files. The overall run time of the pipeline for genome mapping and variant calling, which earlier took approximately 12 days, was now drastically reduced to 2 hours.

Makeflow was used as the workflow description language and execution engine. Makeflow allows for the description of static DAG workflows. Work Queue was used as the batch execution engine. It implements the master-worker framework. With Work Queue a determined number of workers can be initialized and used, allowing for caching between jobs on the worker.

After exploring the parallelism within each component of the pipeline, we merged the pipeline into a single workflow to explore the effects of workflow fusion. We define workflow fusion as the process of merging two sequential workflows into a single workflow that takes advantage of caching, elimination of choke points, and higher throughput using stacked partitioning methods. Here, we also explored how different partitioning methods perform within a single fused workflow, and discuss how workflow fusion can apply in data-intensive pipelines often found in bioinformatics.

2. Methods.

2.1. 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. 2.1, after aligning sequences A and B, 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) [3] for data compression, and the second method is based on hashing to speed up alignments. Depending on the genome size, the entire alignment procedure may take several days to compute similarity between two given sequences. For instance, short read alignment tools like MAQ [22] and SOAP [26] typically require more than 5 CPU-months and 3 CPU-years, respectively, for aligning 140 billion base pairs (Bbp) [18].

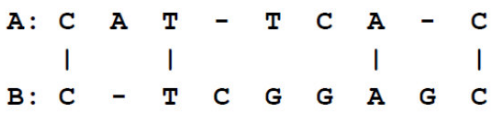

FIG. 2.1. *Genome alignment between sequences A and B showing matches, mismatches, and indels*

One of the widely used alignment tools, Burrows Wheeler Aligner (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 [4] and ABI SOLiD [12]. 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 [20].

2.2. 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. In Fig. 2.1, 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. They 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 data. Popular variant calling tools like SAMtools and MAQ use a Bayesian statistical model to evaluate the posterior probability of genotypes.

GATK employs similar, yet more sophisticated Bayesian algorithms. For our experiments we used GATK's HaplotypeCaller walker because of its high sensitivity. HaplotypeCaller functions by indexing the input set and creating walkers to locate variations between the query and reference. Once a difference is detected, it performs local assemblies to fill gaps or correct mistakes. Though it generates accurate results, it is less frequently used as it takes a long time check more variants, both indels and SNPs, for a given sequence. To remedy the slower performance, we opted for parallel execution of its subtasks.

2.3. 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. As seen in Table 4.5, 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 improve genotype calling by increasing runtime [33], which is a major bottleneck.

A possible approach to address this problem is to develop efficient algorithms or data structures to reduce the search space during the computationally expensive task of alignment or variant calling [21]. 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 conceptually divided up genome alignment and variant discovery, and used the Makeflow [38] language to define each subtask along with its dependencies. Makeflow can employ various systems, including multi-core machines, batch systems like the Sun Grid Engine (SGE) [9] and Condor Pool [28] to execute the individual tasks. For our experiments, we enabled Makeflow to use the Work Queue [2] master-worker framework as an alternative scalable framework to process data across clusters, clouds, and grids. The Work Queue framework also handles all data transfers and can cache files when running jobs remotely.

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

- For **granularity-based partitioning** in BWA, we tested the workflow by varying the number of partitions and the size of each partition. We determined the optimal granularity value or partition size to be the one for which run time was the lowest. 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 used for running BWA.
- For **individual-based partitioning** in BWA, 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.

Fig. 2.2 illustrates the workflow of these approaches when applied to 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**). 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 alignment phase, we added read group and platform information to each BWA output (SAM file) and compressed them into their sorted and indexed binary versions (BAM) to be compatible with HaplotypeCaller.

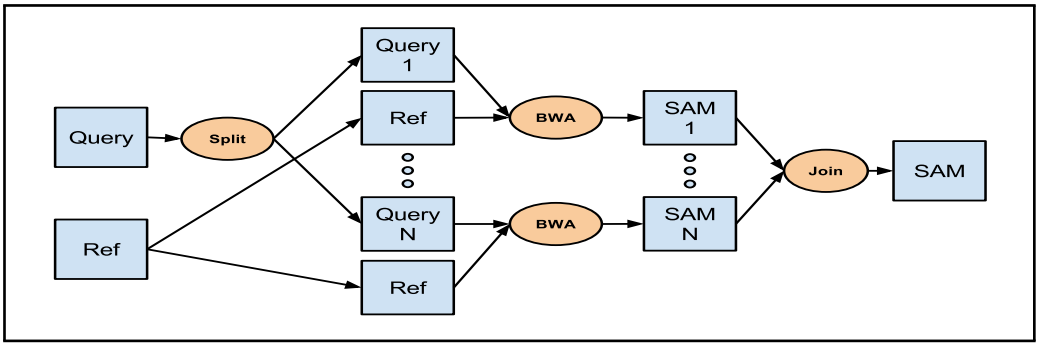

FIG. 2.2. *Framework of* ***granularity-based*** *and* ***individual-based*** *data partitioning approaches in BWA. N=715 for* ***granularity-based*** *and N=50 for* ***individual-based****. At the end, the output files (SAM format) of BWA were joined to form a single file containing all the alignment information.*

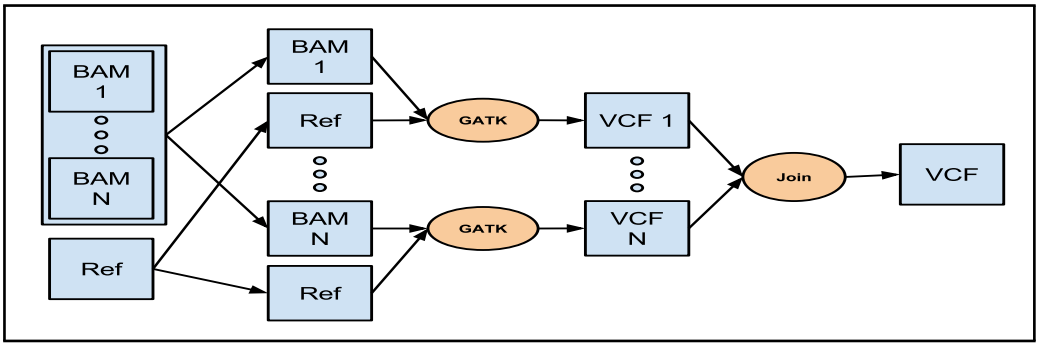

FIG. 2.3. *Framework of* ***granularity-based*** *and* ***individual-based*** *data partitioning approaches in HaplotypeCaller. N=715 for* ***granularity-based*** *and N=50 for* ***individual-based****. The reference file and each sorted and indexed BAM file were sent to a worker for executing GATK's HaplotypeCaller. Outputs of HaplotypeCaller in VCF format were joined to create a single output file.*

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

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

Fig. 2.3 depicts the workflow for executing parallelized HaplotypeCaller using **granularity-based** and **individual-based** approaches. Fig. 2.4 shows **alignment-based** data partitioning implemented in HaplotypeCaller. 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 contained variant information in VCF format [5]. In the final step of the pipeline, we concatenated individual VCF files into a single VCF file containing variants for the entire population, with which we began the analysis. As we will discuss in Section 3, **granularity-based** and **alignment-based** data partitioning were the most efficient approaches for running the parallelized pipeline for BWA and HaplotypeCaller, respectively.

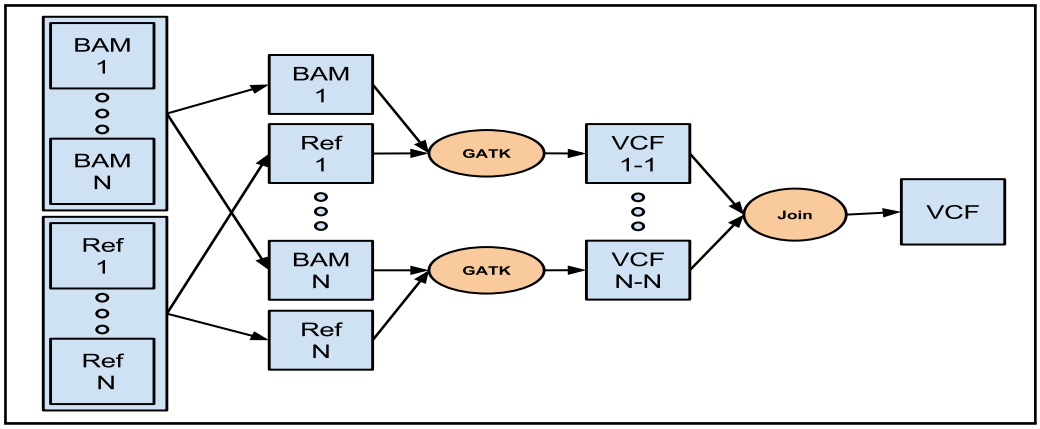

FIG. 2.4. *Framework of* ***alignment-based*** *data partitioning approach in HaplotypeCaller. The reference file was split into bins and the SAM file was split based on the contigs in the bins to which the reads aligned. Each pair of smaller reference bin and its corresponding BAM file were then sent to a worker to run GATK's HaplotypeCaller.*

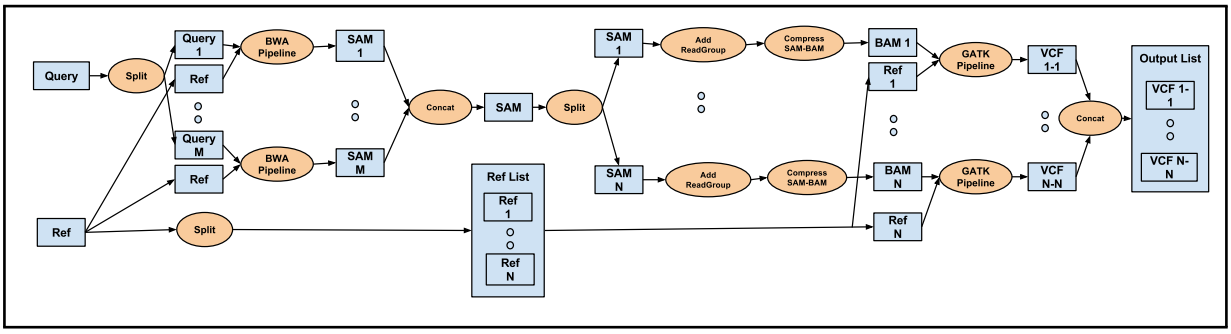

FIG. 2.5. *Framework of the optimized pipeline incorporating* ***granularity-based*** *BWA and* ***alignment-based*** *HaplotypeCaller. It also includes the intermediate stages of adding read groups to SAM files and converting them to their sorted and binary formats.*

2.4. Makeflow and Work Queue. Makeflow is a workflow engine that allows for the expression of static DAG (Directed Acyclic Graphs) workflows. Makeflow uses a straight-forward syntax for describing each task within a workflow. This syntax requires a list of all input files, a list of all output files, and the command to run on these input files. Makeflow requires that all files, both explicit in the command and implicit requirements, are listed. Requiring all files to be mentioned allows the task to be sent to any platform that supports the executable. When the makeflow is executed, only tasks whose required input files exist are sent for execution.

Work Queue is a master-worker job execution engine that is used for this project. With the Makeflow that describes the workflow, execution is started by creating and broadcasting a Work Queue master. This master then waits for worker and distributes jobs as available to keep the workers busy. Workers are created either at a local machine, or by submitting multiple workers at once to a batch system such as Condor or SGE. Once these workers connect, jobs are sent and the files required are transported and cached. This strategy is key as it allows for a file to be sent only once to each worker and then reused for every subsequent job sent to it.

Within a single workflow, the benefits of file reuse are evident, but as soon as the workflow ends the caches are cleared for security and efficient use, as leaving files on remote systems is to be avoided. Therefore, to utilize the cache the workflow must be fused into one workflow, where caches are used consistently within a single workflow.

3. Workflow Fusion. We define workflow fusion as the idea of using information about the software and the computing systems, which are involved in the execution of a workflow, to accelerate a set of sequential but interrelated workflows. These cross workflow optimizations fuse the previously independent workflows into

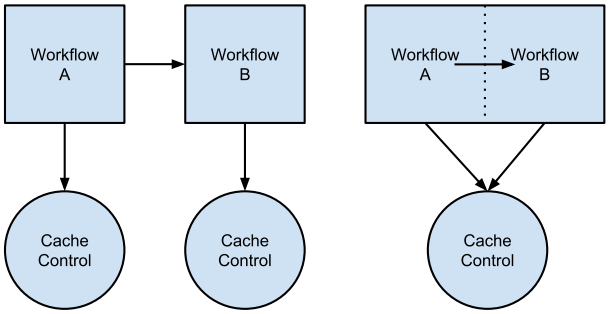

FIG. 3.1. *The figure illustrates how two merged workflows, A and B, utilize the same cache controller. In practice this is done using Work Queue, where all files in a Makeflow are aggressively cached. When separate, the two workflows' caches are independent and can not be used between each other. However, when the control of that cache is shared, previously transferred files can be utilized.*

one. Though this causes less flexibility in how the workflows are used, the improvements can be worthwhile. As we look at ways in which workflow fusion can benefit performance, it can be noted that different methods require different levels of knowledge of the workflows and difficulty in the depth of fusion. We explored full workflow caching, which was done automatically by concatenating makeflow, choke elimination, which required understanding how two workflows can be efficiently partitioned, and full fusion, in which we manually merged two divergent partitioning techniques for acceleration.

3.1. Full Workflow Caching. A basic approach of workflow fusion is full workflow caching. This method relies on a single manager of tasks and data that allows for better data management. Although Work Queue as a system aggressively caches all files that are sent to a worker, no caching occurs between independent workflows. By fusing the workflows, we take full advantage of previously sent data that are useful for multiple steps such as reference information. Fig. 3.1 shows the organization of the cache controller, and how merging them unifies the controller. This is the most basic form of workflow fusion, as all that is needed is to merge two makeflow files together and check that the references and input and output files are consistent to maintain the logical flow of the newly created workflow.

In this form of workflow fusion, we rely on the workflow execution engine for the performance improvement. This method is the easiest to implement, and its benefits can be seen at all levels of workflow fusion, as illustrated in Fig. 3.2. This benefit however, is negated if files are modified between workflows. For example, the base reference is split during the alignment-based partitioning for GATK, and is no longer recognized by Work Queue as a cached file. Originally, the **alignment-based** GATK benefitted by limiting the data that needed to be transferred [32], but with full workflow caching each new smaller piece of the reference is sent and the benefit of the cached reference is lost. Thus, combining two workflows presents challenges when the previously used partitioning scheme may no longer be the optimal solution of the fused workflow.

3.2. Elimination of Choke Points. A choke point is simply a location in the workflow that requires all threads of computation to converge. Similar to a barrier operation in traditional parallel computing, choke points hinder computation by stopping all concurrency until the slowest serial task completes. The effort of eliminating these choke points is the second method of workflow fusion. When eliminating choke points, an effort is made to alleviate traffic where data converges on a single node and is redistributed in some form. Although caching can limit some of the outgoing traffic from later partitioning steps, there is still a lot traffic that is incurred sending files in and out of these choke points.

Because choke point elimination strives to take advantage of files already distributed, it is important to understand how to convert the results of the previous workflow into the inputs of the next workflow. Such transformations are crucial, since they can be applied directly to the data. A clear example of a transformation is the sorting of aligned results prior to starting variant calling. Prior to fusion, all of these processes were sequentially completed, but performing a transform as part of the prior step allows avoiding expensive traffic

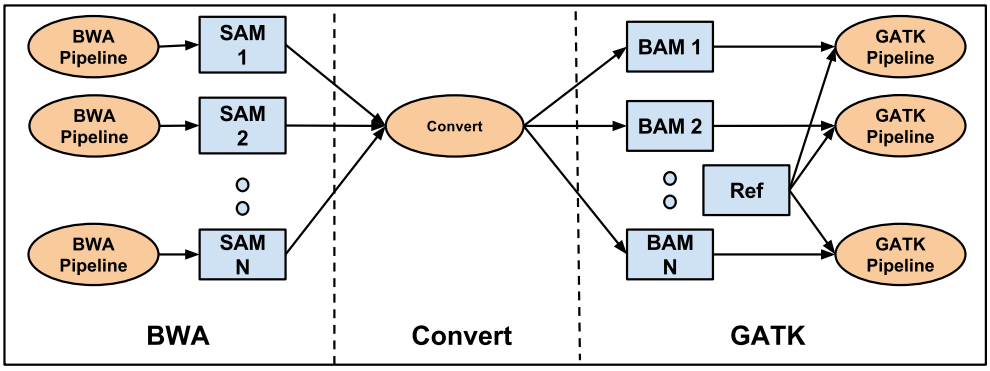

FIG. 3.2. *Framework of the Cache Fused Concept. The pipeline comprises the BWA step, intermediate conversion steps for adding read groups, converting SAM files to their sorted and indexed formats, and the GATK step. The reference used by GATK is the same as that used in BWA, allowing for it to be cached at the worker and not sent as additional traffic.*

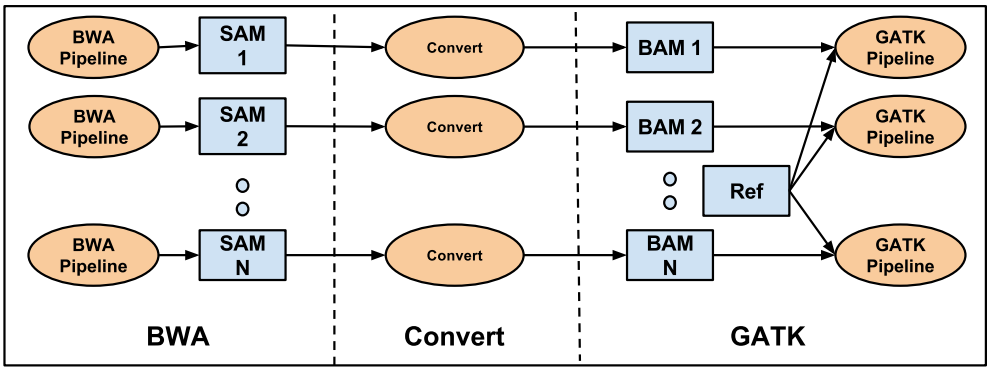

FIG. 3.3. *Framework of the Choke Elimination Concept. As can be seen here, the removal of intermediate data choke points allows computational threads to progress further through the workflow without needing to wait for slower threads to finish and communicate with the master. This lowers the amount of data the master is required to deliver at once and allows it the transfers to be offset based on when they arrive.*

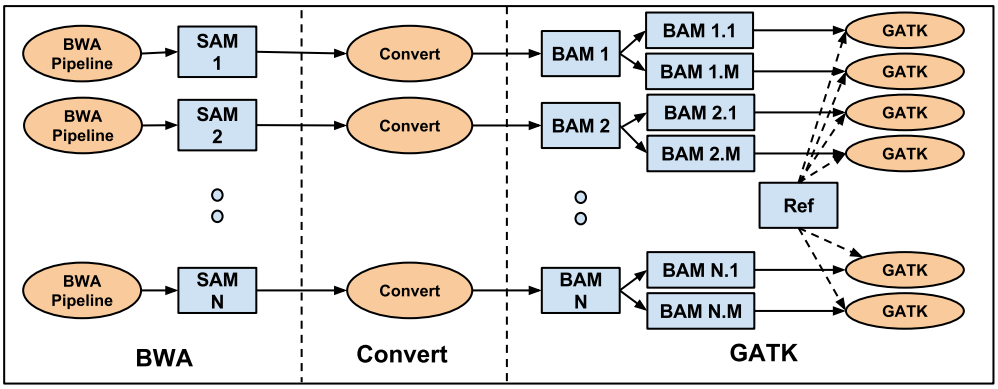

FIG. 3.4. *Framework of the Merged Partition Concept. As can be seen here, the conversion step appears prior to the mapping, but could go either before or after depending on what state the inputs need to be. It is also useful to note how, though as simple map is used here, any process that relates two partitioning methods could be added as long as it does not require coalescing the full data set.*

to converge the data on one system.

This is done most clearly by utilizing the partitions of the previous workflow as the partitions of the subsequent workflow, as shown in Fig. 3.3. Using established partitioning eliminates intermediate partition transfers, and allows independent partitions to continue executing without waiting for slower branches. The cost of this approach is that both workflows must be able to utilize this partition with acceptable performance. If the two different workflows do not share a common suitable partitioning scheme, we can utilize the information about the workflows to potentially find a more appropriate hybrid method.

3.3. Merged Partitioning. Finding a hybrid partitioning scheme requires knowledge about how different workflows interact and therefore how they could be optimally partitioned. Unfortunately, it is not always obvious how to do such divisions. For example, alignment-based partitioning is best for GATK but this requires performing alignments using BWA. One compromise is to maintain BWA partitions utilizing **individual-based** partitioing in BWA, and then partioning each individual utilizing the **alignment-based**. Although this is a basic method to merge partitioning schemes, the idea can be extend to any method that relates one set of inputs to another, as shown in Fig. 3.4.

When successful, applying these ideas to a set of workflows alleviates the time and stress on the system, while allowing faster machines to process more without having to wait and proceed in lock step with remaining tasks.

3.4. Issues that may arise with workflow fusion. Though performance is often improved by melding multiple workflows together, there are several pitfalls. The first is that as the workflows become more intertwined, debugging becomes more difficult to monitor, understand, and fix. Undetected failures in different states produce incorrect output, which could have occurred at any of the fused processes. This issue can be exacerbated when workflows operate at different stages concurrently. For this reason a clear understanding of what errors may occur within the different processes and how to handle them is paramount. The second pitfall is some methods of partitioning, or combinations of partitioning methods, may not accelerate the workflow or be feasible due to dependencies (e.g., alignment-based GATK drawn from BWA output). When this occurs more time must be spent understanding and writing appropriate transformations.

4. Results. For our experiment, we aligned and detected variants for 50 *Quercus rubra* (northern red oak) individuals' ILLUMINA HiSeq RAD [30] data with over 100-fold coverage. The reference sequence for this non-model species contained approximately 400000 individual loci. The system used for analysis consisted of a cluster of 26 machines, each with 8 cores and 32 GB RAM. This allowed us to use up to 208 (26×8) workers in the cluster. To illustrate our base run time, 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 data to find an optimal solution.

4.1. Partition. Prior to running the experiments, it was important to understand the impact the selected partitioning method would have on the overall runtime.

The performance of the algorithm with each partitioning method is important, but if several methods perform similarly within a process the deciding factor is the impact of the method used on the data organization. As such, we developed several parsing scripts in Perl that allowed us to partition the data using a common language for comparison. To find an accurate time for each, the scripts were run 5 times with the reported run time being the average of those runs.

First we explored the two methods that were discussed for BWA. Granularity-based partitioning performed well, by only requiring 13:23 minutes to parse the entire data set into 715 partitions. This was expected, as the partitioning done here was superficial and only relied on the number of reads that were to be placed in each file. The second method, **individual-based** was expected to be considerably slower as it needed to map its barcodes to individual names for subsequent splitting. However, **individual-based** partitioning required only 18:39 minutes to create 50 partitions. A comparison of results can be seen in Table 4.1.

Next, the different partitioning methods that were used in GATK were explored. Granularity-based approach required 4 hours 52 minutes to divide the aligned data from BWA up to 2000 partitions. **Individual-**

TABLE 4.1

Comparison of runtimes for different approaches of data partitioning in BWA. Due to lower splitting time, ***granularity-based partitioning*** *was the best approach for BWA. However, the two approaches' similar times make them good candidates for the pipeline.*

BWA Runtime (DD:HH:MM:SS)				
Partitioning	**Split**	**Align**	**Concat**	**Total**
Individual	18:39	17:58	22:26	59:23
Granular	13:23	21:11	21:10	**55:44**

TABLE 4.2

Granularity-based *BWA requires less time when more workers are in use. For the test data set (Data 1), due to the overhead of additional data transfer, the system could not scale well beyond 20 workers. This should not be considered as a shortcoming of the framework, which achieved higher speedup and efficiency with more workers when tested on a larger query data set (Data 2) comprising 60 individuals.*

Runtime (HH:MM:SS)						
Workers	**Data 1**			**Data 2**		
	Time	**Speedup**	**Efficiency**	**Time**	**Speedup**	**Efficiency**
2	3:01:57	1.94	0.97	8:46:13	1.80	0.90
5	1:14:30	4.77	0.95	3:33:45	4.45	0.89
10	1:08:42	5.19	0.51	1:48:10	8.78	0.87
20	59:00	5.98	0.29	55:34	17.23	0.86
50	57:06	6.19	0.12	33:17	28.27	0.57
100	56:00	6.30	0.06	20:52	47.41	0.47

based required 27:35 minutes and produced 50 partitions, for 50 individuals. The following partitioning approach was based on **alignment-based partitioning** that was discussed earlier. **Alignment-based** splits the reference and entire SAM query file into 10 partitions based on alignment information. This method required 15:41 minutes to perform. A comparison of results can be seen in Table 4.3.

4.2. 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 4.1 presents the runtimes for BWA 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 barcode information. The **granularity-based** approach did not involve this searching step and hence was more efficient (Table 4.1: 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 4.3: 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. 4.1 shows how the framework performed with increased number of workers. As more workers were used, the increase in performance diminished rapidly. For the test 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. Fig. 4.1 supports this fact as we can see the amount of data sent and the time required for it increased with more workers. Fig. 4.3 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. 4.4 provides a visualization of the run

TABLE 4.3

Comparison of runtimes for different approaches of data partitioning using GATK's HaplotypeCaller. Due to overall shorter coupling time ***alignment-based partitioning*** *was the most optimal choice. The HaplotypeCaller phase was also fastest due to reduced search space by limiting the number of loci in the reference at each partition. The quick HaplotypeCaller performance for granularity, similarly, comes from limited loci available during variant calling.*

GATK Runtime (DDd HH:MM:SS)					
Partitioning	**Coupling**		**Variant Calling**	**Concat**	**Total**
	Split	**Other**			
Individual	27:35	3:59:10	2d 5:18:23	9:00	2d 9:54:08
Granular	4:52:34	7:06:25	41:51	5:30	12:46:20
Alignment	15:41	1:03:47	24:36	4:00	**1:48:04**

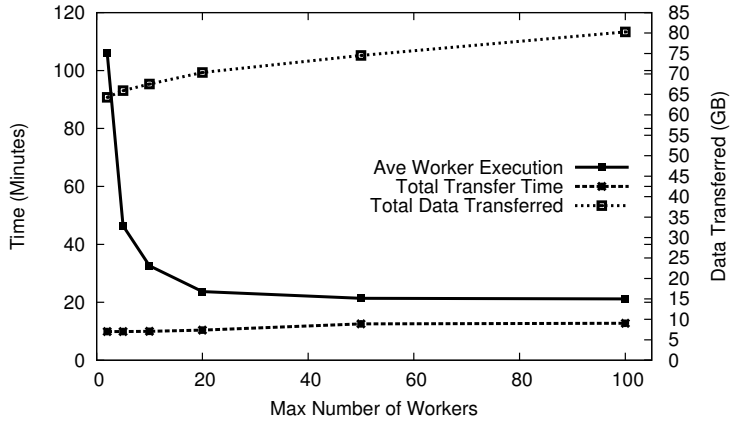

FIG. 4.1. *Runtime of* ***granularity-based*** *BWA for increasing number of workers. The time accounts for the parallelized alignment step of BWA, and does not consider the times for splitting the query sequence or concatenating the outputs. The run time gradually decreases with more workers. For large number of workers the overhead of data transfer hinders further improvement.*

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 the framework for another data set (Data 2 in Table 4.2) having a larger unique query data. Fig. 4.2 shows the transfer of necessary files from the master to the workers over the network. This feature is particularly useful in an environment without a shared file system, as in our case. The transferred shared data can be cached at the workers to be re-used by the workers.

As seen in Table 4.3, for HaplotypeCaller, the **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 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.

4.3. Pipeline Improvement. Based on the aforementioned results, we generated an optimized pipeline for genome alignment and variant calling combining **granularity-based** BWA and **alignment-based** HaplotypeCaller, as shown in Fig. 2.5. Fig. 4.3 presents the average time taken by a task running BWA and Haplotyper. This time represents the generalized functioning of the pipeline and is independent of the underly-

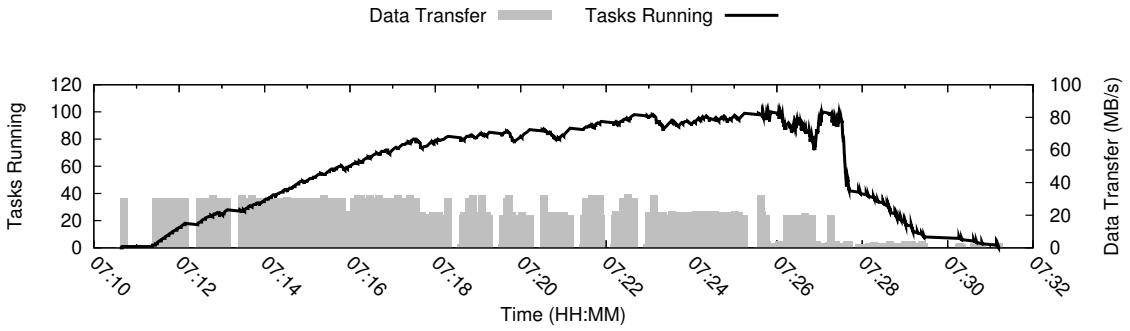

FIG. 4.2. *Mechanism of data transfer for an environment without a shared file system. The thick line denotes the number of tasks executing during the given timeline for* ***granularity-based*** *BWA. The gray bars show data transfer from the master to worker nodes. The left axis measures the number of tasks running whereas the right axis measures the rate of data transfer in MB/s.*

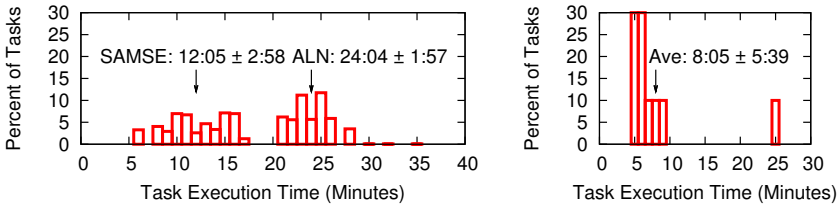

FIG. 4.3. *The histogram for BWA (left) is bimodal due to the alignment (aln) and generation of reads in SAM format (samse) steps. These times were measured for 1430 tasks, where 715 tasks were attributed to each process.The histogram for GATK (right) only contains 10 tasks, which are tightly grouped, except for a single heavy outlier.*

ing batch system. On the other hand, Table 4.4 provides the runtimes specific to our pipeline which employed 100 work queue workers in a heterogeneous, distributed system. It shows the 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 an overall runtime improvement from approximately 12 days to under 3 hours.

4.4. Workflow Fusion. Table 4.5 compares the times when running the individual tools sequentially to that when run in our optimized pipeline as well as the workflow fusion-based pipeline.

The first method that we performed was a full cache fused workflow. This was done by concatenating the Makeflow files for each workflow together. Following this, all intermediate conversion processes were added, and the fused makeflow was run utilizing Work Queue as before. We were able to acheive a result of 2 hours and 42 minutes, as shown in Table 4.5. This is not surprising as it is close to the seperate disjoint pieces of the workflow, and used **individual-based** BWA followed by **alignment-based** HaplotypeCaller. This result verifies concept, and does not hinder the performance of the workflow.

The second method implemented was the choke point elimination by matching partitioning types. For this we utilized the makeflows written using **individual-based partitioning** for both BWA and GATK. For the intermediate processes, the concatenation of BWA and splitting of GATK were ommitted and the intermediate steps of adding read groups and converting to sorted and indexed binary formats were applied to each individual. This method completed in almost 2 and a half days,(Table 4.5) which is on the same scale as the BWA and GATK seperate makeflow using **individual-based partitioning**. **Individual-based paritioning** was used, as **granularity-based** is conceptually the same from BWA to GATK, but split into different partitions due to the different organization of FASTQ and SAM files. This implementation improved the coupling of the workflow by eliminating the join-split sequence used in the pipeline and cache fused. However, the results are much slower, due to GATK's performance on this partition method. The intermediate steps while faster, are dominated by GATK's slower perfomance using **individual-based partitioning**.

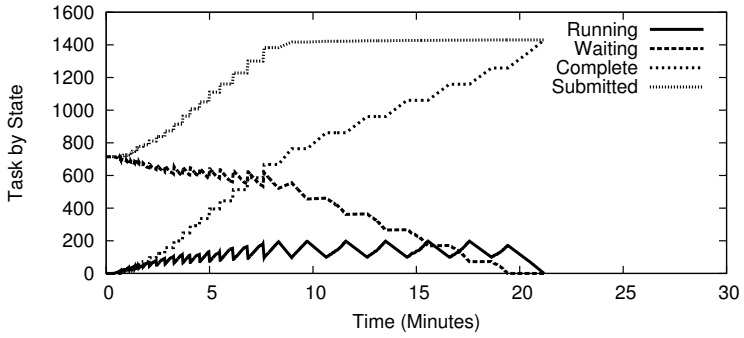

FIG. 4.4. *Run time behavior of* ***granularity-based*** *BWA using 100 workers. The lowest line represents the currently running tasks, the jagged appearance is due to the manner in which tasks are distributed using Work Queue. While distributing tasks, finished tasks wait to be collected, they are then collected and tasks are sent out. This waiting causes the jaggedness.*

TABLE 4.4

Run times of individual steps of the optimized pipeline using 100 workers. For BWA, the query data is split based on ***granularity-based partitioning****. The outputs of BWA, in SAM format, are then concatenated. This is followed by the coupling stage which includes splitting the reference and concatenated SAM file based on* ***alignment-based partitioning****, adding read groups (RGs), and converting SAMs to their sorted, indexed binary versions (BAMs). GATK's HaplotypeCaller runs on the BAM files and reference bins to generate VCF files, which are finally concatenated to contain variant information for the entire dataset. It is important to note that conversion from SAM to BAM is the most expensive step. We address this problem using Workflow Fusion.*

Runtime (MM:SS)		
BWA	**Split Query**	13:23
	Parallelized BWA	21:11
	Concat SAM	21:10
Coupling	**Split Ref and SAM**	15:41
	Add RGs	11:10
	SAM to BAM	52:37
GATK	**Parallelized GATK**	24:36
	Concat VCF	4:00
Total		2:43:48

The final method implemented was the full partition merging of BWA and GATK. Beginning with the makeflow for **individual-based** BWA, the concatenation step was removed, and each individual was then split using the alignment-based method. Following the splitting for **alignment-based**, each new split had the intermediate processes performed. This added a significant amount of tasks, with only a small amount of additional overhead, as the inputs were smaller. This method was able to complete in 1 hour and 55 minutes (Table 4.5). This was an improvement over the best results we were able to obtain utilizing the seperate makeflows, and was a result of caching and higher number of available tasks. Specifically, nodes that took longer now did not slow down the workflow. This higher level of granularity and better partitioning scheme selection allowed for runtime improvement. The performance of the entire pipeline was improved by increasing the concurrency and allowing coupling steps to be completed more quickly, as seen in Fig. 4.5. In Fig. 4.6, the significant differences between the coupling steps of cache fused and partition fused are evident.

The interesting information that arises when using different workflow fusion techniques is the intermediate steps. The three different methods that were outlined, do not necessarily apply well to all workflows. This current workflow is a perfect example, because the overall runtime for choke elimination is considerably slower than the pipeline described without fusion, the intermediate steps were faster. This can be seen in Table 4.6. It is also interesting to note that additional splitting step for partition fused did not slow down the entire execution more than it increased performance. However, when choke elimination was applied in conjuction with partition

TABLE 4.5

Comparison of run times for sequential, parallel, and workflow fusion execution of the pipeline. CF stands for Cache Fused, CE stands for Choke Elimination, and PF stands for Partition Fused. The run time for the parallelized execution represents the times taken by the optimized pipeline for 100 workers. The intermediate coupling column describes the time spent transforming data from BWA for GATK. The fused workflows were run using 100 workers, as can be seen by the conversion times workflow fusion mitigates conversion and spends more time in computation.

Runtime (HH:MM:SS)				
	BWA	**Coupling**	**GATK**	**Total**
Sequential	4:04:00	6:50:41	12d 00:00:00	12d10:54:41
Parallel	55:44	1:23:28	24:36	2:43:48
CF	2:42:49			2:42:49
CF + CE	2d 8:43:46			2d 8:43:46
CF + CE + PF	1:55:37			1:55:37

TABLE 4.6

Comparison of time spent in the coupling stages between BWA and GATK using different levels of workflow fusion. It is interesting to note that time spent in partition fused splitting the data further increased the remaining intermediate steps enough to mitigate the cost.

Coupling Runtime (HH:MM:SS)				
	Concat	**Split**	**Convert**	**Total**
CF	9:11	15:41	1:42:09	2:07:01
CF + CE	-	-	1:07:00	1:07:00
CF + CE + PF	-	18:52	39:36	58:28

fusion the resultant workflow performed well and aided the effect of partition fusion.

While applying these different optimization methods, we can clearly see that these methods can be applied in a cumulative manner to the workflow. In fact, the three methods that were outlined have a hierarchy, with the base method of cache fusion is built on by choke elimination, and choke elimination is used with partition fusion. This set of methods is a preliminary look at methods of workflow fusion, and could likely be expanded into different operations that could be either cumulative, as the methods presented, or mutually exclusive from one or more methods due to workflow structure. Exploring workflow fusion with other workflow pipelines would facilitate better understanding of workflow fusion methods.

Using this set of experiments, we were able to outline and show a set of operations for workflow fusion and how they applied to a genomic pipeline. We were able to achieve a 40% improvement in performance from the base pipeline. This improvement warrants further exploration of workflow fusion, and understanding how different operations used in fusion affect the performance of a process. Though this concept was applied to a genomic pipeline in this paper, the concepts discussed are applicable to different sectors of research.

5. Open Problems. 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) [11] 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 times, network traffic, and higher data transfer times often cause an overhead, especially 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 (Fig. 4.2), 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

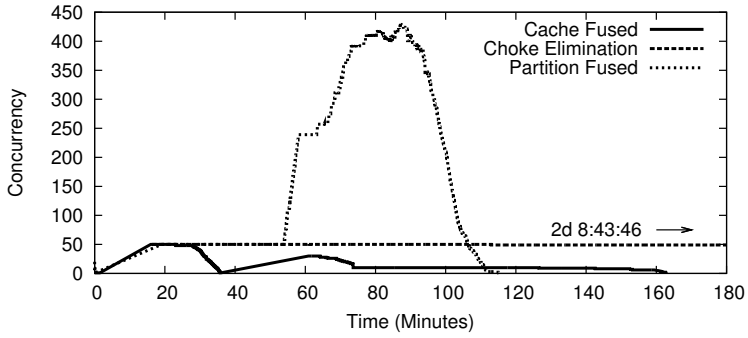

FIG. 4.5. *Comparison of available concurrency for Partition Fused, Choke Elimination, and Cache Fused workflows. Available concurrency consists of all tasks that are ready to be run and either waiting or running. As can be seen by the significant size of the Partition Fused line, the increase in partition promotes a significantly higher level of concurrency. Also, though Choke Elimination had more available concurrency, the partitioning method used caused slower performance.*

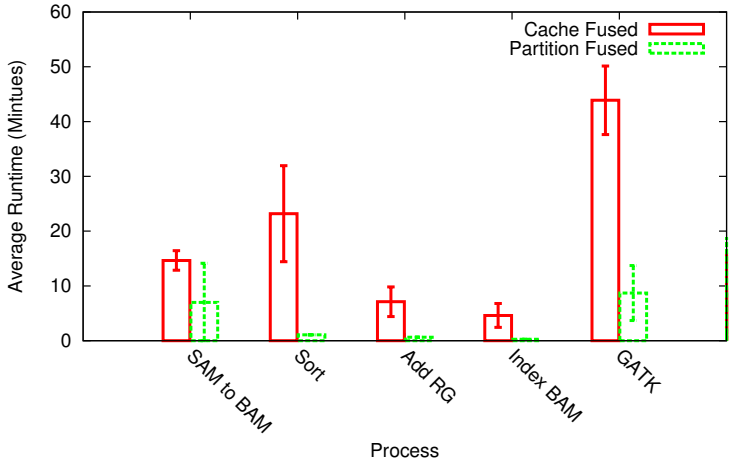

FIG. 4.6. *Comparison of average runtimes for the set of coupling processes in cache fused and partition fused workflows. Clearly, the time spent both in intermediate steps as well as GATK itself was longer per node in cache fused than in partition fused. It is important to note that the number of partitions was 10 for cache fused and 500 for partition fused.*

a multi-slot worker and a foreman [1] 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 [31] 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.

6. Related Work. Genome mapping and assembly have traditionally been refa- ctored using MPI (Message Passing Interface) [6] or MapReduce [7] frameworks. ClustalW-MPI [23] is a version of the multiple sequence aligner ClustalW [37] that uses MPI to run on distributed memory systems. CloudBurst [36] is an early application that implements the seed-and-extend genome mapping algorithm on a Hadoop [10] 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 [16]. 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 [25] for variant calling. It does not natively support gapped alignment. SEAL [34] 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 test cases, the approaches of data partitioning 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. We observed that our optimized pipeline could further benefit from workflow fusion which merges diverse sequential workflows for optimization. We explained the impact of caching, eliminating choke points, and stacked partitioning methods on performance while implementing workflow fusion. Again, this idea is highly applicable in developing parallelized workflows for any data-intensive application. 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 efficient bioinformatics applications that can harness resources from a heterogeneous, distributed environment.

7. Conclusion. We significantly improved the performance of a variant discovery bioinformatics pipeline by developing a new 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. Following our exploration of the pipeline, we applied the concepts discussed in workflow fusion to our optimized workflow. After applying all of the concepts, we were able to achieve an execution time of just under 2 hours. 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] 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.

[2] 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.

[3] M. Burrows and D. Wheeler, *A block-sorting lossless data compression algorithm*, (1994).

[4] 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, 38 (2010), pp. 1767–1771.

[5] 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, 27 (2011), pp. 2156–2158.

[6] A. Darling, L. Carey, and W.-C. Feng, *The design, implementation, and evaluation of mpiBLAST*, Proceedings of ClusterWorld, 2003 (2003).

[7] J. Dean and S. Ghemawat, *Mapreduce: simplified data processing on large clusters*, Communications of the ACM, 51 (2008), pp. 107–113.

[8] 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, 43 (2011), pp. 491–498.

[9] 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.

[10] A. Hadoop, *Hadoop*, 2009.

[11] 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.

[12] 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, 10 (2009), p. R32.

[13] 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, 25 (2009), pp. 2283–2285.

[14] 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).

[15] B. Langmead and S. L. Salzberg, *Fast gapped-read alignment with bowtie 2*, Nature methods, 9 (2012), pp. 357–359.

[16] B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg, *Searching for SNPs with cloud computing*, Genome Biol, 10 (2009), p. R134.

[17] 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, 10 (2009), p. R25.

[18] 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, 456 (2008), pp. 66–72.

[19] H. Li and R. Durbin, *Fast and accurate short read alignment with burrows–wheeler transform*, Bioinformatics, 25 (2009), pp. 1754–1760.

[20] 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, 25 (2009), pp. 2078–2079.

[21] H. Li and N. Homer, *A survey of sequence alignment algorithms for next-generation sequencing*, Briefings in bioinformatics, 11 (2010), pp. 473–483.

[22] H. Li, J. Ruan, and R. Durbin, *Mapping short dna sequencing reads and calling variants using mapping quality scores*, Genome research, 18 (2008), pp. 1851–1858.

[23] K.-B. Li, *ClustalW-MPI: Clustalw analysis using distributed and parallel computing*, Bioinformatics, 19 (2003), pp. 1585–1586.

[24] 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, 19 (2009), pp. 1124–1132.

[25] 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, 19 (2009), pp. 1124–1132.

[26] R. Li, Y. Li, K. Kristiansen, and J. Wang, *Soap: short oligonucleotide alignment program*, Bioinformatics, 24 (2008), pp. 713–714.

[27] 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, 25 (2009), pp. 1966–1967.

[28] 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.

[29] M. Metzker, *Sequencing technologiesthe next generation*, Nature Reviews Genetics, 11 (2009), pp. 31–46.

[30] 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, 17 (2007), pp. 240–248.

[31] 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.

[32] O. Choudhury, N. Hazekamp, D. Thain, and S. Emrich, *Accelerating Comparative Genomics Workflows in a Distributed Environment with Optimized Data Partitioning*, C4Bio Workshop at CCGrid, 2014
[33] R. Nielsen, J. S. Paul, A. Albrechtsen, and Y. S. Song, *Genotype and snp calling from next-generation sequencing data*, Nature Reviews Genetics, 12 (2011), pp. 443–451.
[34] L. Pireddu, S. Leo, and G. Zanetti, *SEAL: a distributed short read mapping and duplicate removal tool*, Bioinformatics, 27 (2011), pp. 2159–2160.
[35] J. Reis-Filho, *Next-generation sequencing*, Breast Cancer Res, 11 (2009), p. S12.
[36] M. Schatz, *Cloudburst: highly sensitive read mapping with mapreduce*, Bioinformatics, 25 (2009), pp. 1363–1369.
[37] 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, 22 (1994), pp. 4673–4680.
[38] 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, 13 (2010), pp. 243–256.

Edited by: Jesus Carretero
Received: August 31, 2014
Accepted: December 1, 2014