MapReducing a Genomic Sequencing Workflow

Luca Pireddu
CRS4
Pula, CA, Italy
luca.pireddu@crs4.it

Simone Leo
CRS4
Pula, CA, Italy
simone.leo@crs4.it

Gianluigi Zanetti
CRS4
Pula, CA, Italy
gianluigi.zanetti@crs4.it

ABSTRACT

Modern DNA sequencing machines have opened the flood gates of whole genome data; and the current processing techniques are being washed away. Medium-sized sequencing laboratories can produce 4-5 TB of data per week that need to be post-processed. Unfortunately, they are often still using ad hoc scripts and shared storage volumes to handle the data, resulting in low scalability and reliability problems. We present a MapReduce workflow that harnesses Hadoop to post-process the data produced by deep sequencing machines. The workflow takes the output of the sequencing machines, performs short read mapping with a novel parallel version of the popular BWA aligner, and removes duplicate reads—two thirds of the entire processing workflow. Our experiments show that it provides a scalable solution with a significantly improved throughput over its predecessor. It also greatly reduces the amount of operator attention necessary to run the analyses thanks to the robust platform that Hadoop provides. The workflow is going into production use at the CRS4 Sequencing and Genotyping Platform.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed Programming*

General Terms

Algorithms, Performance, Reliability

Keywords

Next-generation Sequencing, Sequence Alignment, MapReduce

1. INTRODUCTION

Advancements in DNA sequencing methods [22] have made reading human and other DNA faster, simpler and cheaper opening the doors to very large scale DNA sequencing projects [4], and bringing systematic whole genome sequencing to the capabilities of medium size laboratories. As an example, the CRS4 Sequencing and Genotyping Platform (CSGP) is involved in the deep sequencing of hundreds of individuals in the context of two large scale studies on auto-immune diseases [25] and longevity [24]. With regards to the former, the genetics of type 1 diabetes (T1D) and multiple sclerosis (MS) is studied by comparing the genomes of more than 4,000 affected individuals and 2,000 healthy volunteers from Sardinia, an area with one of the highest incidences of these diseases world-wide and representing one of the main reservoirs of ancient genetic variation in Europe. The latter is characterized by studies at the population level for evidence of genetic influences on a group of traits expected to be linked to longevity ranging from arterial stiffness to positive emotions. This is one of the largest family-based studies and one of the most characterized samples, clinically and genetically, worldwide.

The CSGP, equipped with three Illumina HiSeq 2000 sequencing machines and two Illumina GAIIx, can achieve sequencing rates of close to 200 Gbases per day, for a total of 4-5 TB of data each week. The raw data produced consists of pairs of short fragments of DNA called *reads*, each—with current technology and biochemistry—about 100 bases long. Before being useful for analysis, these fragments need to be post-processed and, with such high data flow rates, the computational load on the post-processing pipeline is sufficient to overwhelm traditional analysis workflows that rely on serial designs or simplistic parallelization efforts based on the coordination of job submissions to batch queue systems. Work has been done to provide scaling solutions (for example, see Schatz [26], Langmead et al [12], or Matsunaga et al [20]), but it is currently limited to only short read lengths (order of 30 bases) and it does not integrate multiple post-processing steps to provide a single, scalable and robust solution.

The goal of this article is to present how one can replace "traditional" deep sequencing processing workflows with MapReduce applications thus dramatically improving the pipeline's scalability and robustness. Specifically, we discuss the original workflow used at CSGP and then present a new MapReduce [3] workflow that replaces its first half, with a plan to substitute it entirely in the near future. The MapReduce workflow performs read alignment and duplicate read removal—typically the first steps in a DNA sequencing workflow. It directly processes the output of the Illumina sequencing machines without any conversion, and provides good scaling characteristics. The core component of the pipeline is a novel version of the read alignment BWA tool [16] which, to the best of our knowledge, is the first distributed implementation. This new workflow is currently going into production use at the CSGP.

The rest of this article is structured as follows. Section 2 provides the background information regarding DNA sequencing and computational post-processing steps. It then describes the original workflow used at CRS4, and the problems that affect it. Section 3 describes the MapReduce solution presented by this paper, and leads into its evaluation in Section 4. The related work in this area is then delineated in Section 5.

2. SEQUENCING WORKFLOW

Several steps are required to read an organism's DNA, beginning with a biological sample in the wet laboratory, and finishing with in silico post-processing. While the biochemical sequencing procedure may vary substantially between different sequencing technologies [29], the analysis varies much less. In short, the sequencing procedure consists of the following steps:

1. Read DNA fragments;
2. Map fragments to reference genome;
3. Detect and remove duplicate reads;
4. Recalibrate base quality scores.

We briefly summarize the biochemical procedure in the first step, and then focus on the computational aspects of the workflow.

2.1 Read DNA fragments

To begin, a sample of DNA is acquired from living tissue, such as peripheral blood. The sampled cells are then broken apart, first mechanically and then chemically, so that the DNA material can be extracted and isolated from other cell material. Once the DNA is isolated, the biochemical procedure may vary for different sequencing technologies. Here we treat briefly how to sequence with the Illumina technology, but we suggest Shendure and Li [29] for a more thorough introduction.

In these platforms adapter sequences, ligated to both ends of the DNA molecule, are bound to a glass surface (flow cell) coated with complementary oligonucleotides. This is followed by solid-phase DNA amplification, where several million double-stranded DNA fragments, typically 300 to 500 bases long, are generated in each of the eight channels (or *lanes*) of the flow cell. Finally, sequencing-by-synthesis [2] is performed: each cycle of the sequencing reaction occurs simultaneously for all clusters in the presence of all four nucleotides, each labeled with a different fluorescent dye. Bases are identified one at a time by capturing the emitted fluorescence from each DNA cluster in subsequent sub-regions, or *tiles*, of the flow channel. The actual reads are obtained by first processing the images to extract spot location, intensity and background noise, and then by analysing, tile by tile, the pertaining stacks of images obtained during the single base sequencing reaction cycles to extract the actual sequences. The latest Illumina sequencers perform image analysis internally and output location-intensity files that are subsequently converted to sequence files in the qseq format [9].

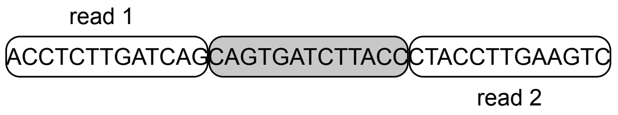

Figure 1: The relationship between paired reads and the whole DNA fragment. The gray section in the middle is the unread section between the two reads.

Each fragment is first read from one end and then from the other, producing a total of two *reads*. With the HiSeq 2000 the read length is typically about 100 bases, meaning that for each DNA fragment of 300–500 nucleotides we have read the first and the last 100 or so, leaving an unread section in between.

From a single run the sequencing machine can produce 800 GB of data, consisting of billions of records—two for each fragment—each containing four pieces of data: a key identifying a DNA fragment; the read number (one or two); the DNA sequence read; the quality score for each base in the DNA sequence, which estimates the probability of a reading error at each base.

From this point the work shifts to the computational domain. The data need to be processed before it can be used in bioinformatic analysis. The type of post-processing performed depends on the intent of the sequencing study. Our target case is the resequencing of the genomes by read pair alignment on a reference sequence, so that it may then be used for Genome Wide Association Studies (GWAS) and SNP analysis, among others [32].

2.2 Map fragments to reference genome

The work in the wet laboratory results in reading a genome that has been cut into random pieces. However, for our target analyses the genomic sequence must be reconstructed by determining the original locations of the fragments. For this purpose, a reference genome is chosen—i.e., a previously sequenced genome of the same species. Then, each fragment is aligned to the reference, generally by finding the substring of the reference with the shortest edit distance from the fragment. When aligning read pairs, the fact that the distance between the two fragments can be estimated is used to direct the alignment process to choose a position where both reads can be aligned within a statistically reasonable distance from each other. This alignment process is known as *Short Read Alignment*. Several software tools are available to perform this step, such as BWA [16] and Bowtie [13].

2.3 Detect and remove duplicate reads

Identical fragments may be read by the sequencing process. Some duplicates derive from preparation of the sample, while others are optical duplicates where the sequencing machine erroneously reads one point as two. These duplicate reads must be eliminated to avoid introducing statistical biases into the data. Although the duplicate reads may not be exactly identical because of reading errors, they are generally very similar and thus tend to align to the same location. Therefore, the elimination strategy is to assume that two fragments or pairs that align to the exact same position on the reference are duplicate reads, and the one with the worse read quality is eliminated.

2.4 Recalibrate base quality scores

The final step in the DNA sequencing workflow is to adjust the quality scores of the detected bases to take into account several factors in addition to the optical quality measures used by the sequencing machine. In particular, the probability of error for a base is affected by several factors, such as: the position within the read (errors are more likely towards the end of a fragment); the preceding and current nucleotide observed (some combinations are more prone to errors); the probability of mismatching the reference genome (some locations are unlikely to vary, while others are—i.e. Single Nucleotide Polimorphims (SNPs)). After this step the data are ready to be distributed and used in bioinformatics analysis.

2.5 Complete workflow

The computational workflow described in Algorithm 1 is a prototypical pipeline and implements the three computational steps previously described. Its implementation at CSGP leverages several open source tools to perform the various computations at each step (see Algorithm 1 and Table 1 for details). Where off-the-shelf tools are not already available, custom programs or scripts are written. For instance, this is the case for the two steps Qseq2Fastq (file format conversion) and IlluminaQual2SangerQual (quality score conversion). Each step reads one or more input files and writes one or more output files. Generally data are shared through a centralized file system exported via NFS.

Algorithm 1: Summary of serial computational workflow at CSGP.

```
Data: qseq output from sequencer
Result: Recalibrated BAM file containing the aligned
        read
foreach $lane \in flowcell$ do
    foreach $read \in 1, 2$ do
        Qseq2Fastq($lane$, $read$);
        IlluminaQual2SangerQual($lane$, $read$);
        BWA align($lane$, $read$);
    end
    BWA pair_align($lane$);
    SamTools sort($lane$);
end
SamTools merge($flowcell$);
Picard rmdup($flowcell$);
$table \leftarrow$ GATK recalibration_table($flowcell$);
GATK recalibrate($flowcell$, $table$);
```

Where easily done (e.g. the *foreach* loops), the computation can be parallelized by submitting concurrent jobs to the cluster queue system. Nevertheless, this approach has difficulty scaling to a large numbers of nodes for a number of reasons. First of all, the shared file system becomes a bottleneck as the number of nodes writing to it increase. Personal experience showed that even state-of-the-art parallel file systems such as Lustre [28] have problems withstanding this type of workload on more than 16 nodes concurrently. Secondly, by parallelizing the work to be done only on the lane and read dimensions, the number of possible concurrent processes is limited. Specifically, to process the output of a run on the Illumina HiSeq 2000, which uses 8-lane flow cells, a maximum of 16 concurrent processes can be launched, and this only in the early phases of the workflow. Moreover, later in the pipeline the work is constrained to a single data flow per sample. Admittedly, more elaborate parallelization strategies could be implemented, but such strategies would translate to implementing a new distributed computing framework when a production-level, open source option such as Hadoop [7] is already available. Finally, even if the workflow was designed to scale up to a large number of nodes, the increased probability of job failure would make the process too difficult to manage.

The result is a high-maintenance workflow that requires constant attention by its operators, and an expensive high-performance storage system to operate; and notwithstanding the investment and effort, it still only achieves modest performance levels.

Step	Program
Qseq2Fastq	custom script
IlluminaQual2SangerQual	custom script
Read alignment	BWA [16]
Sorting	SamTools [17]
Duplicates removal	Picard [23],
Recalibration	GATK [21]

Table 1: Programs used to implement Algorithm 1.

3. MAPREDUCE WORKFLOW

The MapReduce workflow has been developed as a solution to the problems affecting the serial pipeline described in Section 2. It leverages the Hadoop implementation of MapReduce to: distribute the I/O load eliminating the bottleneck that was the shared filesystem; provide high scalability through improved parallelism; improve reliability by resisting node failures and transient events such as peaks in cluster load. The current version of the new workflow runs the read mapping and the duplicate removal steps in two MapReduce iterations.

Algorithm 2: Map and reduce functions for the pair reads step

```
$map(id, read, seq, qual) \rightarrow (id : read, (seq, qual))$

$reduce(id, [(seq1, qual1), (seq2, qual2)]) \rightarrow$
                                   $cat(id, seq1, qual1, seq2, qual2)$
```

3.1 MR Step 1: pair reads

As previously mentioned, the input records produced by the sequencing process identify the DNA fragment from which they were read. However, the first and second reads of the fragment are in two different records, which are not output together. Therefore, the first step of the workflow processes the original data to group both reads of the same fragment into the same record. The *map* step produces a composite key containing the fragment id, and the read number. As customary, the complete key is used for sorting, but, the map output is grouped only by fragment id, ignoring the read number by using custom partitioning and grouping functions. As a result, each invocation of the *reduce* step receives exactly two values, already ordered. Thus the reducer only has to form and emit the appropriate output record containing the fragment id, and the sequences and base qualities of both DNA fragments.

3.2 MR Step 2: read alignment and duplicate removal

Given the paired sequences output by the first phase, the second MapReduce iteration in the workflow performs steps 3 and 4 outlined in Sections 2.2 and 2.3: paired read alignment to the reference genome, and duplicate read removal.

3.2.1 Map: read alignment

The map step of the algorithm performs the read alignment. Rather than implementing a read aligner from scratch, we integrate BWA [16] into our tool. BWA is a command-line tool that, in its original form, does not lend itself easily to alternative workflows. We refactored its functionality into a new library, libbwa, which is described in Section 3.3. Libbwa is mainly written in C, and it provides a high-level CPython interface—i.e., the standard C implementation of Python. To take advantage of this feature, and avoid working with the Java Native Interface (JNI), the mapper is written in Python, and integrates into the Hadoop framework using Pydoop [15] (see Section 3.4).

For each pair of reads, the aligner produces a pair of read alignment records. Sometimes a suitable alignment position cannot be found for a read; the aligner flags these records as “unmapped”. On the other hand, if a probable alignment position is found the alignment record indicates the read’s most probable position in the genome (chromosome, coordinate within the chromosome, and whether the read is on the forward or reverse DNA strand). The alignment also provides a quality score that estimates its probability of error, as well as other details such as edit distance and edit operations (match/delete/insert/substitute) required to match the read fragment to the relevant section in the reference sequence.

Unmapped reads are eliminated, as are reads whose alignment quality is below a user-defined threshold. This filtering process leaves some aligned reads unpaired. Conversely, good alignments are serialized into a Protobuf [6] message and emitted to the MapReduce framework using as a key the precise location to which they are aligned. Aligned pairs therefore result in two keys—one for each read—while lone reads only result in one.

The mapping tasks are very memory intensive, principally because the current BWA implementation needs to load the entire reference and reference index into memory, at a cost of about 4 GB of RAM per task. With the addition of the memory required for the computations and the MapReduce framework, each mapping task currently requires about 6 GB of RAM. This requirement limits the number of concurrent mapping tasks that can be run on each node to only two at this time. We configured Hadoop’s Capacity Scheduler to enforce this limit. To alleviate this impediment, we have made some of the alignment phases multi-threaded. Furthermore, we are taking steps to resolve this problem altogether by having all mappers on the same node share a single reference structure in memory.

3.2.2 Reduce: duplicate removal

The Picard rmdup program to be replaced defines two pairs as duplicates if their alignment positions on the reference genome are identical, both for their first and second reads. Likewise, lone reads are considered duplicates if they are aligned to the same position. When a set of duplicate pairs is found, only the one with the highest average base quality is kept; the rest are discarded as duplicates. Moreover, when a lone read is aligned to the same position as a paired read, the lone one is discarded. If on the other hand only lone reads are found at a specific position then, as for pairs, only the one with the highest average base quality is kept.

The reducer in this MapReduce workflow implements these same rules, following the logic presented in Algorithm 3. In it rmdup implements the duplicate criteria as described above.

Algorithm 3: Remove duplicates reducer algorithm

```
Input: key ← (chromosome, position, strand)
       values ← list(pairs, reads)
position ← key;
pairs ← [ p | p ∈ values, paired(p) ] ;
if $pairs \neq \emptyset$ then
    paired_reads ← true;
end
left_pairs ← [ p | p ∈ pairs, position = left_coord(p) ] ;
foreach $p \in$ rmdup(left_pairs) do
    emit(p)
end
if $paired\_reads \neq true$ then
    unpaired ← [ u | u ∈ values, paired(u) = false ] ;
    foreach $u \in$ rmdup(unpaired) do
        emit(u)
    end
end
```

3.3 libbwa

BWA is an open source short read alignment tool, released under the GPLv3. It is the tool selected for the original sequencing pipeline at the CSGP because it supports gapped alignments, which is essential for aligning long reads (order of 100 bases). BWA is designed as a command-line program where each step of the alignment algorithm is implemented by a sub-command. A full alignment run would include several steps:

- **index:** create an index over the reference sequence(s);
- **aln:** find the suffix array (SA) coordinates of the input reads (see Li and Durbin [16] for details);
- **samse/sampe:** convert SA coordinates to chromosomal coordinates and generate alignment records in the SAM format [17].

Each stage reads its input from and writes its output to the local file system. Unfortunately, in the original source code the logic for these I/O activities is mixed with the alignment-related logic, which makes it impossible to reuse the same functions to perform alignments outside of the data flow scheme originally envisioned by the BWA authors.

```
align = BwaAligner()
align.nthreads = 8
align.hit_visitor = SamEmitter()
align.reference = sys.argv[1]
del sys.argv[1]

for line in fileinput.input():
  align.load_pair_record(
      line.rstrip("\r\n").split("\t"))
align.run_alignment()
```

Figure 2: Align read pairs with libbwa in Python

To adapt BWA to the new requirement of working within a MapReduce framework, we refactored the BWA logic to separate different responsibilities, in fact changing the key entry routines to take pre-loaded data structures as parameters—as opposed to file pointers, for instance. We added functions to facilitate the creation of these library-specific data structures from generic arrays of character strings, which can be read from any source in batches or one by one. The C library is built into a shared object which can be linked into any application.

Finally, the low level C library is wrapped in a higher level, object oriented Python interface that allows reads to be aligned in just a few lines of Python code (see Figure 2 for an example).

3.4 Pydoop

Pydoop is a Python API for Hadoop MapReduce and the Hadoop Distributed File System (HDFS) that allows object-oriented MapReduce programming and full access to HDFS. It has several advantages over the other main solutions for Python Hadoop programming: with respect to Jython, Pydoop has the advantage of being a CPython package, which means that users have access to all Python libraries, either built-in or third-party, including any C/C++ extensions; with respect to Hadoop Streaming, the main advantages are its object-oriented API, access to a wider range of components (e.g., the RecordReader) and better performance [15].

Given the existence of the Python bindings for libbwa, Pydoop provided a natural means to integrate libbwa and Hadoop, thus enabling the MapReduce version of the genomic sequencing workflow with a reduced programming effort when compared to the implementation of new JNI bindings for libbwa. Pydoop works through Hadoop pipes, and thus incurs an extra overhead when compared to a JNI solution that integrates directly into the framework. However, since the run times of the map tasks are largely dominated by the computation of the alignments we consider the overhead relatively negligible.

4. EVALUATION

We evaluate the performance of the MapReduce workflow as compared to the performance of a baseline serial workflow, and to the actual workflow implementation that was used at the CSGP. In addition, the scalability characteristics of the MapReduce workflow with respect to input size and number of compute nodes are evaluated. All experiments are performed on a homogeneous computation cluster. Each node runs CentOS release 5.2 and is equipped with: two Intel® Xeon® CPUs @ 2.83 GHz with four cores and 6 MB of cache each (8 cores total); 16 GB of RAM; two 250 GB SATA hard disks, one of which has been used for HDFS storage (the other one is reserved for the OS); Gigabit Ethernet NIC. These machines are part of the same computing cluster used for production sequence analyses.

Dataset	No. tiles	No. pairs	Size (GB)
Dataset B1	1	3255065 ($3.3 \cdot 10^6$)	1.4
Dataset B2	5	17397564 ($1.7 \cdot 10^7$)	7.7
Dataset B3	10	35679581 ($3.6 \cdot 10^7$)	15.7

Table 2: Baseline input dataset sizes

Dataset	No. lanes	No. pairs	Size (GB)
Dataset MR1	1	$1.2 \cdot 10^8$	51
Dataset MR2	2	$2.3 \cdot 10^8$	102
Dataset MR3	3	$3.3 \cdot 10^8$	147

Table 3: MapReduce workflow evaluation: input data sizes

4.1 Baseline

The baseline workflow implements the serial workflow presented in Section 2, up to and including the Picard rmdup step. Its only parallelization is in the BWA align step, since the aln subcommand of BWA is multi-threaded, and thus would be easily exploitable by any scientist with a workstation. On the contrary, parallelizing the rest of the workflow would require a more significant effort and some programming expertise.

The baseline tests are run on a single computation node. A General Parallel File System (GPFS) volume on an 8-disk array is used to store all the input, intermediate, and output data used by the baseline workflow. We tested the baseline set-up with varying input sizes, as summarized in Table 2. We measure runtime and compute throughput for each input size. Each experiment is run three times. Average values are reported.

4.2 MapReduce Workflow

The MapReduce workflow is tested with a varying input size of 1, 2, and 3 HiSeq 2000 lanes (as detailed in Table 3), where each lane contains 64 tiles. The MapReduce workflow is also tested with a varying cluster size of 16, 32, 64, and 96 compute nodes.

The experimental procedure is as follows. A Hadoop cluster, version 0.20.2, of the required size plus two nodes is allocated. The two extra nodes are used for the job tracker and the name node services, respectively; the remaining nodes each run a task tracker and a data node service. The input data and a tarball of the reference sequence and its index are copied onto the HDFS volume. All experiments for the selected cluster size are run.

Each combination of cluster size and input size is run 4 times. Usually the first run incurs a penalty due to a one-time set cost of about 6-8 minutes to copy the reference sequence to all the compute nodes; on that basis, the first time is dropped. The runtime of each workflow phase, as well as the entire workflow is measured, and the throughput is computed. The average of the results in runs 2-4 are reported.

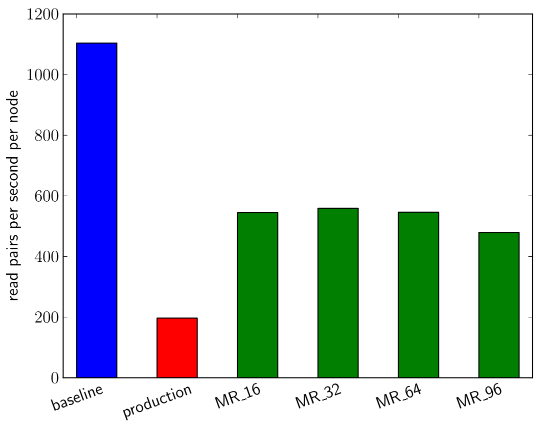

Figure 3: Throughput per node for the various test scenarios. The baseline has been run on a 10-tile data set, while the production and the MapReduce pipelines have been run on a 3-lane data set. MapReduce results, represented by columns 3 through 6, are shown for increasing cluster sizes.

Scenario	No. nodes	Runtime (h)
Production	16	29.1
MapReduce	16	10.5
MapReduce	32	5.1
MapReduce	64	2.6
MapReduce	96	2.0

Table 4: Wall clock times for the different test scenarios using Dataset MR3.

4.3 Results and discussion

Figure 3 shows the throughput per node for the various test scenarios, measured in read pairs per second per node. The first column corresponds to the baseline, run on a 10-tile data set. The baseline, run on a single node, serves as a benchmark against which the distributed workflows are compared. The second column refers to the production workflow, run on a 3-lane data set and using 16 cluster nodes, i.e., one per HiSeq 2000 output file (see section 2.5). As shown in the diagram, the production workflow achieves a throughput of less than one fifth that of the baseline: as discussed previously, this is mostly due to the limited bandwidth of the shared file system and to the fact that only the BWA align step is multi-threaded. The MapReduce workflow—columns 3 through 6, referring to the 3-lane data set—has a throughput of about one half that of the baseline, which starts to decay for cluster sizes above 64 nodes.

Figure 4 shows how throughput per node varies with cluster size, for data set sizes of 1, 2 and 3 lanes. In the ideal case of perfectly linear scalability, all curves would be flat; in practice, as the number of nodes increases, infrastructural overhead becomes more and more relevant, disrupting performance. This effect is sharper in the case of smaller data sets, where the overhead cost is more significant with respect to the computation. In the 2-lane and 3-lane cases, normalized throughput reaches a saturation point between 16 and 64 nodes: after that, it is subject to diminishing returns, i.e., adding nodes yields increasingly lower throughput per node. The 1-lane curve, on the other hand, is already beyond saturation at 16 nodes.

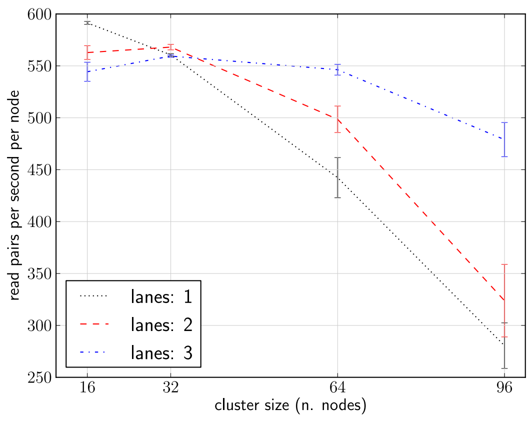

Figure 4: Throughput per node of the MapReduce workflow with respect to cluster size. An ideal system would produce a flat line. Note that in the 2- and 3-lane cases the throughput per node reaches a saturation point between 16 and 64 nodes. On the other hand, the 1-lane curve is already beyond saturation at 16 nodes.

5. RELATED WORK

Usage of MapReduce for bioinformatics applications started in late 2008 [5, 20] and has since continued to increase [30]. Although the model has been successfully applied to a number of "classic" applications, such as BLAST and GSEA [14, 20], the area of most advancement is deep sequencing data analysis [10–12, 21, 26, 27], where several useful Hadoop-based tools have been developed, most notably Crossbow [12] for SNP discovery, and Myrna [11] for RNA-Seq [31] differential expression analysis.

Bowtie [13], like BWA, is a short read aligner. Both programs are based on a highly memory-efficient data structure [18]. However, currently only BWA supports gapped alignments [16], which is a required feature for aligning long reads such as the ones produced by the HiSeq 2000.

Crossbow uses Bowtie to align reads to a reference genome in the map phase, groups alignments by genomic region and identifies SNPs with SOAPsnp [19] in the reduce phase. Myrna's workflow consists of several separate MapReduce steps, most of them map-only or reduce-only (i.e., where the map function is the identity function), that lead from input reads to a series of reports on differentially expressed genes. Crossbow and Myrna use Hadoop Streaming [8], with mappers and reducers implemented as Perl scripts that, in most cases, act as wrappers for binary executables (e.g., Bowtie) or R/Bioconductor scripts. Both applications can also run on a single node (without Hadoop) or on Amazon Elastic MapReduce [1], which however may not be an attractive option for a genome sequencing application since data transfer costs and time would be quite significant.

The Genome Analysis Toolkit (GATK) [21] is a MapReduce-like programming framework that provides a set of data access patterns commonly used by sequencing data analysis programs. GATK defines a programming contract where components are organized into data producers, or traversals, and data consumers, or walkers (i.e., analysis modules).

There are also other deep sequencing data analysis tools that make use of MapReduce. Quake [10] is a tool to detect and correct sequencing errors, while Contrail [27] runs on Hadoop to perform *de novo* assembly. The former employs a modified version of the classic word count application to count all k-mers (size k subsets) in a genome, while the latter uses Hadoop to build and transform very large graphs (billions of nodes). In addition, the one reported here is a MapReduce version of BWA, and to the best of our knowledge it is the first.

6. CONCLUSIONS AND FUTURE WORK

The MapReduce workflow presented in this article provides a new way to perform reference alignment and duplicate removal of genomic sequencing data. The empirical results show its ability to scale, handle large inputs, and take advantage of computing resources. In addition, thanks to the Hadoop framework, it provides a robust solution that can withstand node failures and transient cluster problems, in addition to eliminating the bottleneck and single point of failure represented by centralized storage. Furthermore, since it is based on Hadoop, a user could choose to run it on a cloud service such as the one offered by Amazon, although due to the sheer size of the input and output data for this task we feel that at this point in time the cost and time required to transfer the data would make it impractical.

This new workflow is going into production use at the CSGP, where runs with 8 lanes of input ($9 \cdot 10^8$ read pairs) have been processed in less than 4 hours on 128 nodes. We plan on further improving the performance of the tool, in particular by implementing the genome reference in shared memory. A preliminary implementation of this change has resulted in throughputs of up to 1100 read pairs/s/node by allowing us to run almost one mapper per core on our cluster. Subsequently, we will implement the base quality recalibration step, thus moving the entire workflow into Hadoop and HDFS.

Finally, in the near future, we plan on releasing under an open source license the code to both the workflow and libbwa (Pydoop has already been released).

7. ACKNOWLEDGMENTS

We would like to thank our colleagues Michele Muggiri and Carlo Podda for helping us get access to the computing resources needed to run our experiments, and helping resolve any system-related problems we had. We would also like to thank Frédéric Reinier and Riccardo Berutti for being so helpful in getting the runtime data for the original workflow implementation and providing much user feedback.

8. REFERENCES

[1] Amazon elastic MapReduce. `http://aws.amazon.com/elasticmapreduce`.

[2] D. R. Bentley. Whole-genome re-sequencing. *Current opinion in genetics & development*, 16(6):545–552, 2006.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI '04: 6th Symposium on Operating Systems Design and Implementation*, 2004.

[4] R. M. Durbin, D. L. Altshuler, G. R. Abecasis, et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, 2010.

[5] M. Gaggero, S. Leo, S. Manca, F. Santoni, O. Schiaratura, and G. Zanetti. Parallelizing bioinformatics applications with MapReduce. *CCA-08: Cloud Computing and its Applications*, 2008.

[6] Google Protobuf: protocol buffers. `http://code.google.com/p/protobuf/`.

[7] Hadoop. `http://hadoop.apache.org`.

[8] Hadoop streaming. `http://hadoop.apache.org/common/docs/r0.20.0/streaming.html`.

[9] Illumina, Inc. *Sequencing Analysis Software User Guide For Pipeline Version 1.4 and CASAVA Version 1.0*. 9885 Towne Centre Drive, San Diego, CA 92121 USA, 2009.

[10] D. R. Kelley, M. C. Schatz, and S. L. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11(11):116, 2010.

[11] B. Langmead, K. D. Hansen, and J. T. Leek. Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biology*, 11(8):83, 2010.

[12] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg. Searching for SNPs with cloud computing. *Genome Biology*, 10(11):134, 2009.

[13] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):25, 2009.

[14] S. Leo, F. Santoni, and G. Zanetti. Biodoop: bioinformatics on Hadoop. In *The 38th International Conference on Parallel Processing Workshops (ICPPW 2009)*, pages 415–422, 2009.

[15] S. Leo and G. Zanetti. Pydoop: a Python MapReduce and HDFS API for Hadoop. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 819–825, 2010.

[16] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[17] H. Li, B. Handsaker, A. Wysoker, et al. The sequence alignment/map (SAM) format and SAMtools. *Bioinformatics*, 25:2078–2079, 2009.

[18] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–483, 2010.

[19] R. Li, Y. Li, X. Fang, et al. SNP detection for massively parallel whole-genome resequencing. *Genome Research*, 19(6):1124–32, Jun 2009.

[20] A. Matsunaga, M. Tsugawa, and J. Fortes. Cloudblast: combining MapReduce and virtualization on distributed resources for bioinformatics applications. In *Fourth IEEE International Conference on eScience*, pages 222–229, 2008.

[21] A. McKenna, M. Hanna, E. Banks, et al. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, 2010.

[22] M. L. Metzker. Sequencing technologies — the next generation. *Nat Rev Genet*, 11(1):31–46, Jan 2010.

[23] Picard. `http://picard.sourceforge.net`.

[24] G. Pilia, W. Chen, A. Scuteri, et al. Heritability of cardiovascular and personality traits in 6,148 sardinians. *PLoS Genet.*, 2(8):25, 2006.

[25] S. Sanna, M. Pitzalis, M. Zoledziewska, et al. Variants within the immunoregulatory CBLB gene are associated with multiple sclerosis. *Nat Genet.*, 42(6):495–7, 2010.

[26] M. C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[27] M. C. Schatz, D. Sommer, D. R. Kelley, and M. Pop. Contrail: Assembly of large genomes using cloud computing. `http://contrail-bio.sourceforge.net`.

[28] P. Schwan. Lustre: building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, 2003.

[29] J. Shendure and H. Ji. Next-generation DNA sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008.

[30] R. Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11(Suppl 12):1, 2010.

[31] Z. Wang, M. Gerstein, and M. Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nat Rev Genet*, 10(1):57–63, Jan 2009.

[32] L. Y., W. C, S. S., and A. G.R. Genotype imputation. *Annu Rev Genomics Hum Genet.*, 10:387–406, 2009.