BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 23 2013, pages 3014–3019
doi:10.1093/bioinformatics/btt528

Sequence analysis

Advance Access publication September 10, 2013

BioPig: a Hadoop-based analytic toolkit for large-scale sequence data

Henrik Nordberg[1,2], Karan Bhatia[1,†], Kai Wang[1] and Zhong Wang[1,2,*]

[1]Department of Energy, Joint Genome Institute, Walnut Creek, CA 94598, USA and [2]Genomics Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Associate Editor: Michael Brudno

ABSTRACT

Motivation: The recent revolution in sequencing technologies has led to an exponential growth of sequence data. As a result, most of the current bioinformatics tools become obsolete as they fail to scale with data. To tackle this 'data deluge', here we introduce the BioPig sequence analysis toolkit as one of the solutions that scale to data and computation.

Results: We built BioPig on the Apache's Hadoop MapReduce system and the Pig data flow language. Compared with traditional serial and MPI-based algorithms, BioPig has three major advantages: first, BioPig's programmability greatly reduces development time for parallel bioinformatics applications; second, testing BioPig with up to 500 Gb sequences demonstrates that it scales automatically with size of data; and finally, BioPig can be ported without modification on many Hadoop infrastructures, as tested with Magellan system at National Energy Research Scientific Computing Center and the Amazon Elastic Compute Cloud. In summary, BioPig represents a novel program framework with the potential to greatly accelerate data-intensive bioinformatics analysis.

Availability and implementation: BioPig is released as open-source software under the BSD license at https://sites.google.com/a/lbl.gov/biopig/

Contact: ZhongWang@lbl.gov

Received on February 18, 2013; revised on September 3, 2013; accepted on September 4, 2013

1 INTRODUCTION

Advances in DNA sequencing technologies are enabling new applications ranging from personalized medicine to biofuel development to environmental sampling. Historically, the bottleneck for such applications has been the cost of sequencing—the estimated cost of sequencing the first human genome (3 GB) completed a decade ago is estimated at $2.7 billion (http://www.genome.gov/11006943). Current next-generation sequencing technologies (Metzker, 2010) have extraordinary throughput at a much lower cost, thereby greatly reducing the cost of sequencing a human genome to <$10 000 (http://www.genome.gov/sequencingcosts/, accessed December 2012). The price hit $5495 by the end of 2012 (http://dnadtc.com/products.aspx, accessed December 2012). With the cost of sequencing rapidly dropping, extremely large-scale sequencing projects are emerging, such as the 1000 Genomes Project that aims to study the genomic variation among large populations of humans (1000 Genomes Project Consortium *et al.*, 2010), and the cow rumen deep metagenomes project that aims to discover new biomass degrading enzymes encoded by complex microbial community (Hess *et al.*, 2011). As a result, the rate of growth of sequence data is now outpacing the underlying advances in storage technologies and compute technologies (Moore's law). Taking metagenomic studies as an example, data have grown from 70 Mb (million bases) from termite hindgut (Warnecke *et al.*, 2007) and 800 Mb Tamar wallaby (Pope *et al.*, 2010), to 268 Gb (billion bases) from cow rumen microbiome (Hess *et al.*, 2011) within the past 4 years. The recently published DOE Joint Genome Institute's (JGI) sequencing productivity showed that 30 Tb (trillion bases) of sequences were generated in 2011 alone (http://1.usa.gov/JGI-Progress-2011).

Data analysis at terabase scales requires state-of-the-art parallel computing strategies that are capable of distributing the analysis across thousands of computing elements to achieve scalability and performance. Most of the current bioinformatics analysis tools, however, do not support parallelization. As a result, this exponential data growth has made most of the current bioinformatics analytic tools obsolete because they fail to scale with data, either by taking too much time or too much memory. For example, it would take ~80 CPU years to BLAST the 268 Gb cow rumen metagenome data (Hess *et al.*, 2011) against NCBI non-redundant database. *De novo* assembly of the full dataset by a short read assembler, such as Velvet (Zerbino and Birney, 2008), would require computers with >1 TB RAM and take several weeks to complete. Re-engineering the current bioinformatics tools to fit into parallel programming models requires software engineers with expertise in high-performance computing, and parallel algorithms take significantly longer time to develop than serial algorithms (Fig. 1). In addition, at this scale of computing hardware failures become more frequent, and most bioinformatics software lack robustness so that once they fail they have to be restarted manually. All these challenges contribute to the bottleneck in large-scale sequencing analysis.

Cloud computing has emerged recently as an effective technology to process petabytes of data per day at large Internet companies. MapReduce is a data-parallel framework popularized by Google Inc. to process petabytes of data using large numbers of commodity servers and disks (Dean and Ghemawat, 2008). Hadoop is an open-source implementation of the MapReduce framework and is available through the Apache Software Foundation (http://wiki.apache.org/hadoop). Hadoop uses a

*To whom correspondence should be addressed.
[†]Present address: Amazon Web Services, New York, NY 10019, USA

Programming Model	Serial	Parallel (MPI)	Parallel (BioPig)
Data Size	Small	Big	Big
Programming Complexity	Easy	Difficult	Easy
Developmental Time	Short	Long	Short
Computational Speed	Slow	Fast	Fast

Fig. 1. A comparison between algorithms with serial, MPI and BioPig implementations

distributed file system (HDFS) that brings computation to the data as opposed to moving the data to the computation as is done in traditional computing paradigms. In Hadoop, node-to-node data transfers are minimized, as Hadoop tries its best to perform automatic co-location of data and program on the same node. Furthermore, Hadoop provides robustness through a job handling system that can automatically restart failed jobs. Because of these advantages, Hadoop has been explored by the bioinformatics community (Jourdren *et al*., 2012; Taylor, 2010) in several areas including BLAST (Kolker *et al*., 2011), SNP discovery (Langmead *et al*., 2009; McKenna *et al*., 2010), short read alignment (Nguyen *et al*., 2011; Schatz, 2009) and transcriptome analysis (Langmead *et al*., 2010). In addition, there are solutions that reduce the hurdle to run Hadoop-based sequence analysis applications (Jourdren *et al*., 2012). Using Hadoop requires a good understanding of this framework; however, to break up programs into map and reduce steps, skills in programming languages, such as Java or C++, are also required.

To overcome the programmability limitations of Hadoop, several strategies have been developed. For example, Cascading is an application framework for Java developers by Concurrent Inc. (http://www.cascading.org/), which simplifies the processes to build data flows on Hadoop. Apache's Pig data flow language was developed to enable non-programmer data analysts to develop and run Hadoop programs (http://pig.apache.org/). Somewhat similar in nature to SQL, the Pig language provides primitives for loading, filtering and performing basic calculations over datasets. Pig's infrastructure layer consists of a compiler on the user's client machine that turns the user's Pig Latin programs into sequences of MapReduce programs that run in parallel on the nodes of the Hadoop cluster in a similar fashion to how a database engine generates a logical and physical execution plan for SQL queries. In this article, we describe BioPig, a set of extensions to the Pig language to support large sequence analysis tasks. We will discuss the design principles, give examples on use of this toolkit for specific sequence analysis tasks and compare its performance with alternative solutions on different platforms. There is a similar framework, SeqPig (http://seqpig.sourceforge.net/), developed in parallel to BioPig, which is also based on Hadoop and Pig. We provide a detailed comparison of the two in Section 3.4.

2 METHODS

Using the BioPig modules, we provide a set of scripts that showcase the functionality provided by the framework, as well being useful bioinformatics tools in their own right.

2.1 pigKmer

Given a set of sequences, the pigKmer module computes the frequencies of each kmer and outputs a histogram of the kmer counts. The kmer histogram task is ideally suited for MapReduce, as each kmer count can be generated in parallel without global knowledge of all kmers. The data file is partitioned into a number of blocks and each map task operates on a single block at a time. The grouping operation is naturally implemented in the reduce step where the kmers are sorted and partitioned across the available reducers. Each reducer then counts the number of items in its group. The histogram of the counts is generated in a second MapReduce iteration.

Code Listing 1 shows the BioPig script for this calculation. The script first registers the BioPig functions defined in the jar file on line 1. On line 2, the dataset is read and identified as table named A with four columns: id (sequence id from the header of the FASTA), direction (0 if combined or not paired, 1 or 2 otherwise), sequence (the characters of the read sequence itself) and header (any additional data on the FASTA header). Line 3 generates kmers (20mers in this example) for each read. Lines 4 and 5 group identical kmers and generate a count of each group. Lines 6 and 7 generate bins with counts of groups with the same number of kmers.

Code Listing 1

```
-- a simple example of pig script to count kmers

1  register /.../biopig-core-0.3.0-job-pig.jar

2  A = load '$input' using gov.jgi.meta.pig.storage.FastaStorage as (id: chararray,
d: int, seq: bytearray, header: chararray);
3  B = foreach A generate flatten(gov.jgi.meta.pig.eval.KmerGenerator(seq, 20))
as (kmer:bytearray);
4  C = group B by kmer parallel $p;
5  D = foreach C generate group, count(B);
6  E = group D by $1 parallel $p;
7  F = foreach E generate group, count(D);
8  store F into '$output';
```

A number of variations of kmer counting are available: count only the number of unique reads that contain the kmer or group kmers within one or two hamming distance (Hamming, 1950).

2.2 pigDuster

pigDuster searches a set of query sequences against a database of known sequences for near exact matches. This application is useful to screen sequence datasets for contaminants or to perform pathogen detection from human sequence data. The application uses shared kmers to find matches to sequences in the dataset. The kmer indices of the known sequences and query sequences are compared by the join function with detect matches.

2.3 pigDereplicator

Raw sequence data often contain reads derived from PCR (Mullis and Faloona, 1987) amplification. But due to small error rates introduced in the sequencing process, the same DNA sequence may not produce identical reads. To remove these artifacts, the dereplication application builds a kmer index by joining the beginning 16 bases of each read of a mate pair. Because the error rate increases as a function of sequencing length in many short read technologies, the first 16 bases typically contain fewer errors. The 32mer is then used to identify near identical read pairs.

Code Listing 2 shows the BioPig code for dereplicating a dataset of reads. For brevity, the comments and headers have been left out. Lines 10 and 11 generate the hash from the first 16 and last 16 characters. Finally, the hash is grouped and all the sequences with a given hash are combined using the CONSENSUS function (for each base position in the sequence, the majority value determined).

Code Listing 2

```
-- a more elaborate pig script example to find all duplicates of a
-- set of sequences within specified Hamming distance
--
-- reads  = the target set of read sequences (hdfs file path)
-- distance = the Hamming distance (int)
-- output = the location for the output (hdfs file path)

1  register /.../biopig-core-1.0.0-job-pig.jar
2  %default distance 0
3  %default p 100

4  define PAIRMERGE gov.jgi.meta.pig.eval.SequencePairMerge();
5  define CONSENSUS gov.jgi.meta.pig.eval.GenerateConsensus();
6  define IDENTITYHASH gov.jgi.meta.pig.eval.IdentityHash();
7  define UNPACK gov.jgi.meta.pig.eval.UnpackSequence();
8  define HAMMINGDISTANCE gov.jgi.meta.pig.eval.HammingDistance();

-- load the target sequences
9  READS = load '$reads' using gov.jgi.meta.pig.storage.FastaStorage as (id:
chararray, d: int, seq: bytearray, header: chararray);

--
-- group the read pairs together by id and filter out any reads that
-- do not have a matching pair.
-- then combine the mate pairs into a single sequence
--
10  GROUPEDREADS  = group READS by id parallel $p;
11  MERGEDREADS   = foreach GROUPEDREADS generate flat-
ten(PAIRMERGE(READS)) as (id: chararray, d: int, seq: bytearray);

-- generate the hash
12  HASH = foreach MERGEDREADS generate
IDENTITYHASH(UNPACK(seq)) as hash, UNPACK(seq) as seq;
13  HASHNEIGHBORS = foreach HASH generate flat-
ten(HAMMINGDISTANCE($0, $distance)) as hash, '0', $1 as seq;

-- now merge all similar reads together
15  E = group HASHNEIGHBORS by $0 parallel $p;
16  F = foreach E generate $0, count($1), CONSENSUS($1);

-- return output
17  store E into '$output';
```

Most of the work in the script is done in the various functions defined in the BioPig library. PAIRMERGE() takes a set of sequences and merges them together into a single sequence; IDENTITYHASH() takes a sequence and returns the hash from the first 16 bases and the last 16 bases; and CONSENSUS() takes a set of sequences and calculates the majority base at each position and returns a new sequence. The sequence data loader used in line 2 also is defined in the BioPig library and supports reading and parsing FASTA-formatted sequence files.

3 RESULTS

3.1 The BioPig framework and its design principles

The ideal analysis solution should address the three main challenges in data-intensive sequence analysis: (i) scalability: it should scale with data size; (ii) programmability: it should leverage a high-level data flow language that provides abstraction of parallelism details, which enables bioinformatics analysts to focus on analysis over large data sizes without concerns as to parallelization, synchronization or low-level message passing; and (iii) portability: it should be portable without extensive modification to various IT infrastructure. With these design principles in mind, we developed the BioPig data analytic toolkit on top of Apache Hadoop and Pig (Fig. 1).

The resulting BioPig toolkit has both the scalability and robustness offered by Apache Hadoop, which uses the data-parallel methodology of MapReduce to parallelize analysis over many computing cores without significant loss in computing performance, and the programmability and parallel data flow control offered by Pig. In addition, as both Hadoop and Pig are implemented in Java, BioPig inherits their portability.

The BioPig modular implementation consists of a set of libraries, which add an abstraction layer for processing sequence data. Within BioPig's open extensible framework, functions and libraries can be updated and added easily. The first release of BioPig contains three core functional modules. The BioPigIO module reads and writes sequence files in FASTA or FASTQ format. It enables Hadoop to split sequence files automatically and distribute them across many compute nodes. The BioPigAggregation module is a wrapper for common bioinformatics programs, such as BLAST, CAP3, kmerMatch and Velvet. Finally, there is a set of utilities that make working with nucleotide sequences more efficient by using compression and encoding. Table 1 lists the functions provided by the current version of BioPig.

3.2 A simple BioPig application demonstrates scalability, programmability and portability

To systematically evaluate the scalability, programmability and portability of the BioPig framework, we implemented a kmer

Table 1. BioPig functions and programs callable from Pig

Name	Function
BLAST/BLAT	Wrappers for BLAST/BLAT
Cap3, Minimus, Newbler, Velvet	Assembler wrappers
FASTA/FASTQ I/O	FASTA/FASTQ format readers and writers compatible with Hadoop's block I/O
KmerGenerator	Generate Kmers from a sequence
N50	Calculate N50 value for a set of sequences
PackSequence	Store bases in a compact, but still printable, form. Useful to save space and also for debugging
HammingDistance	Implementation of Hamming distance
SequencePairMerge	Merge pairs with the same sequence ID and marked as 0 and 1
SubSequence	Get part of a sequence
GenerateConsensus	Calculate base consensus

counting application (Section 2) to test its performance and compare it with other commonly used serial and parallel methods.

We tested our implementation and report results on two different IT infrastructures:

(1) Magellan system at National Energy Research Scientific Computing Center (NERSC), a 78-node IBM iDataPlex cluster. Each of its nodes has two quad-core Intel Xeon X5550 2.67 GHz processors (8 cores/node) and 24 GB of DDR3 1333 MHz memory.

(2) Amazon Elastic Compute Cloud, a cluster of 15 nodes (1 head node and 14 compute nodes), each node is a cc2.8xlarge instance, which has 60.5 GB memory and 88 Elastic Compute Units (2 x Intel Xeon E5-2670, eight-core).

The size of the dataset we used ranges from the 100 Mb to 500 Gb from the cow rumen metagenomic data (Hess *et al.*, 2011). Kmer counting is an essential component of many bioinformatics methods, such as genome and transcriptome assembly. Even though it is simple to compute, serial programs using kmer counting quickly run out of memory as the size of sequence data increases (Fig. 2). For simplicity, we fixed the size of kmer to 20 (K = 20) and compared the scalability of the BioPig kmer counting program, KmerGenerator, to a serial version, Tallymer (Stefan *et al.*) and an open-source MPI-version (https://github.com/JGI-Bioinformatics/Kmernator).

As shown in Figure 2A, when run on the Magellan Hadoop cluster (NERSC) using 1000 mappers and reducers, respectively, BioPig KmerGenerator scales well from 1 to 500 Gb of input sequences. In contrast, Tallymer ran out of memory on a single node with 48 GB RAM with 1 Gb dataset, and our straightforward implementation of the MPI-version ran out of memory at 50 GB on the same machines tested. We noticed that when the size of the datasets is small (1, 5 and 10 Gb), the KmerGenerator takes about the same time to finish, possibly due to the overhead of the MapReduce framework that BioPig based on. As data sizes reach 10 Gb and higher, however, KmerGenerator scales well with data.

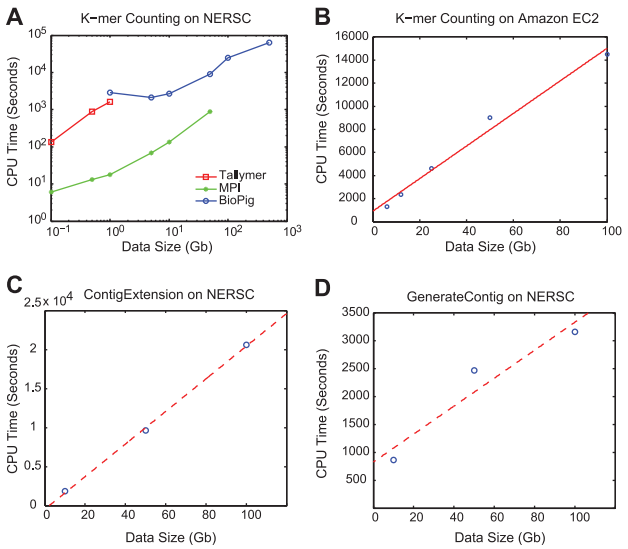

Fig. 2. Scalability of BioPig over increasing size of input sequencing data. (**A**) A comparison of the scalability of kmer counting program implemented with BioPig, serial programming or MPI programming models. All three programs were run on the same machines. (**B**) Kmer counting using Amazon EC2. (**C** and **D**) The scalability of ContigExtension (C) and GenerateContig on NERSC (D), the dotted lines are generated by linear regression

To investigate whether or not other applications based on BioPig also scale with data, we tested ContigExtention and GenerateContig on increasing data sizes (10, 50 and 100 Gb). ContigExtention script (Code Listing 3) iteratively searches for short reads that map to the ends of a long sequence (contig) based on kmer matching and assembles the matching reads with the long sequence to extend it. The algorithm is linear, and we observed near linear performance with data (Fig. 2C). GenerateContig script consists of two steps: it starts by running BLAST to find short reads matching a long sequence, and subsequently assembles them into contigs by calling Velvet. The first step is linear in relation to input size, whereas the second assembly step is not. As a result, the overall performance of the script over data displays a nonlinear scaling pattern (Fig. 2D).

Code Listing 3

```
#!/usr/bin/python
from org.apache.pig.scripting import *
indexFile = 'index.data'
contigFile = 'contigs.data'
p = 100
Pig.fs("rmr output")
PP = Pig.compile("""
register /…/biopig-core-1.0.0-job.jar;
-- load the target sequences
 readindex = load '$indexFile' using PigStorage as (seq: bytearray, kmer:
bytearray);
-- load the contigs, create the index
 contigs = load '$contigFile' using PigStorage as (geneid: chararray, seq:
chararray);
 contigindex = foreach contigs generate geneid,
FLATTEN(gov.jgi.meta.pig.eval.KmerGenerator(seq, 20)) as
                                                    (kmer:bytearray);
-- join reads with the contigs database
 j = join readindex by kmer, contigindex by kmer PARALLEL $p;
 k = foreach j generate contigindex::geneid as contigid,
gov.jgi.meta.pig.eval.UnpackSequence(readindex::seq) as readseq;
 kk = distinct k PARALLEL $p;
 l = group kk by contigid PARALLEL $p;
 m = foreach l {
   a = $1.$1;
   generate $0, a;
 }
-- join the contigid back with the contigs
 n = join contigs by geneid, m by $0 PARALLEL $p;
 contigs = foreach n generate $0 as geneid,
gov.jgi.meta.pig.aggregate.ExtendContigWithCap3($1, $3) as res:(seq:chararray,
val:int);
 split contigs into notextended if res.val==0, contigs if res.val==1;
 contigs = foreach contigs generate $0 as geneid, res.seq as seq;
 store contigs into 'output/step-$i';
 store notextended into 'output/data-$i';
 """)
 for i in range(50):
 stats = PP.bind().runSingle()
 if not stats.isSuccessful():
 break;
 if ( stats.getNumberRecords('output/step-'+str(i)) <= 0):
   break;
 else:
   contigFile = 'output/step-'+str(i)
```

KmerGenerator does kmer counting with just eight lines of code (Code Listing 1). To achieve the same task, Tallymer and the MPI-version use thousands of lines of code. This suggests BioPig has excellent programmability and can greatly reduce the amount of software development time. This programmability feature is particularly attractive to next-generation sequencing data analysis, as the sequencing technology rapidly changes and software constantly needs to be updated.

To test the portability of BioPig, we evaluated KmerGenerator function on Amazon Elastic Compute Cloud (Amazon EC2). Porting BioPig to Amazon was simply a matter of uploading the BioPig core JAR (Java ARchive) file, along with the bioinformatics tools called from BioPig code. Without any change to the underlying code, KmerGenerator was successfully run on both platforms, demonstrating the portability of the BioPig framework. Furthermore, we observed scalability on EC2 similarly as on the Magellan system with various sizes of sequence datasets (Fig. 2B).

3.3 Embedding BioPig into other programming languages

BioPig has the flexibility to be embedded into other languages, such as Python or JavaScript, to achieve the types of control flows such as loops and branches that are not currently available in the Pig language. This flexibility greatly extends the usefulness of the BioPig toolkit.

Code Listing 3 illustrates a Python script with BioPig embedded. This contig extension algorithm greedily extends by iteratively searching for reads that can extend the contig followed by extending the contigs via assembling the reads at each end. In each iteration, the Cap3 assembler is used to extend the contigs, of which only those contigs being extended go into the next iteration. The loop will stop when there are no contigs left to be extended or when it reaches a predefined number of steps.

3.4 BioPig and related MapReduce-based frameworks

Several computational frameworks have been recently developed for bioinformatics. Despite that they all aim to scale analysis to big data by implementing existing algorithms onto MapReduce/Hadoop, they differ significantly on the specific algorithms implemented (Table 2). Therefore, it is not straightforward to compare the performance of BioPig with these frameworks. The only exception is SeqPig (http://seqpig.sourceforge.net/). SeqPig and BioPig are two independent projects, and they share some similarities. First, they both extend Pig, and therefore have the same programming syntax. Second, there are a few similar functions (such as sequence import and export). Because they are based on the same framework (Hadoop and Pig), when run in the same hardware environment, they are expected to have similar run time performance. The difference between the two projects lies in their user-defined functions. BioPig includes several kmer-based applications that are not available in SeqPig (Table 2). It also provides wrappers to run many frequently used bioinformatics applications such as BLAST, Velvet and CAP3. In contrast, SeqPig mainly implements functions of Picard (http://picard.sourceforge.net) and SamTools (Li *et al.*, 2009). SeqPig and BioPig can be installed side-by-side on the same Hadoop and Pig cluster. From the same Pig script, it is straightforward to call both BioPig and SeqPig functions.

Table 2 provides a comparison of BioPig with these related frameworks.

4 DISCUSSION

In this work, we present a solution for improved processing of large sequence datasets in many bioinformatics applications. Using only a few core modules, we believe we have demonstrated the usefulness of this toolkit, while its modular design should enable many similar applications that fit in the MapReduce framework. BioPig has several advantages over alternative parallel programming paradigm: it is easy to program; it is scalable to process large datasets (with 500 Gb being the largest one tested); and it is generically portable to several examples of Hadoop infrastructure, including Amazon EC2, without modification.

We also noticed several limitations of BioPig, most of which likely derived from Hadoop itself. For example, it is slower than handcrafted MPI solutions. This is due to both the latency of Hadoop's initialization and the fact that generic MapReduce algorithms are not optimized for specific problems. For big datasets this limitation may not be a problem, as the time spent on data analysis far exceeds the cost for the start-up latency. Recently, the Hadoop community has started to address this problem. Certain commercial implementations of MapReduce, such as IBM's Symphony product, have been developed to reduce Hadoop's start-up latency. Another promising solution is SPARK, which can speed up Hadoop applications 100 times by using a low latency, in memory cluster computing (Zaharia *et al.*, 2012).

Another issue is computing resource demand. When dealing with huge datasets, BioPig shifts the need from expensive resources such as large memory ($\geq$1TB RAM) machines and/or parallel programming expertise, to large disk space on commodity hardware. For example, a kmer/read index in BioPig needs

Table 2. BioPig compared with other related frameworks

Framework	Programming models	Bioinformatics applications
BioPig	Pig/Java/Hadoop	FASTA I/O; several kmer-based algorithms; wrapper for BLAST and short read assemblers
Biodoop	Python/Hadoop	FASTA I/O; sequence alignment and alignment manipulation; wrapper for BLAST (Leo *et al.*, 2009)
GATK	Java/MapReduce	Short read alignment, variant calling
Hadoop-BAM	Java/Hadoop	BAM (Binary Alignment/Map) I/O; alignment manipulation functions based on Picard API (Niemenmaa *et al.*, 2012)
SeqPig	Pig/Java/Hadoop	FASTA, FASTQ, BAM I/O; alignment-related functions; sequence statistics

lots of disk space, with the largest index taking up 15 TB of disk space for the 500 Gb dataset. This trade-off is usually a favored option, as RAM is more expensive than hard disk. Therefore, processing large datasets with BioPig requires a stable Hadoop environment with fast interconnects and plentiful disk space. One might reduce the requirement by keeping fewer copies of the files in the HDFS system, which is a trade-off for redundancy.

BioPig is built on MapReduce and Hadoop, so algorithms that do not run well on MapReduce will not run well with BioPig either. For example, large short read assembly algorithms involve large graph processing. Currently, they require message-passing and have not yet been implemented on MapReduce.

ACKNOWLEDGEMENTS

The authors would like to thank Shane Cannon for providing technical support for Hadoop clusters, Rob Egan for providing MPI-based kmer counting program and Nicole Johnson for helpful edits and suggestions.

Funding: Department of Energy Joint Genome Institute was supported in part by the Office of Science of the U.S. Department of Energy under (Contract No. DE-AC02-05CH112 and DE-AC02-05CH11231) (cow rumen metagenomics data analysis and informatics).

Conflicts of Interest: none declared.

REFERENCES

Dean,J. and Ghemawat,S. (2008) MapReduce: simplified data processing on large clusters. *Commun. ACM*, **51**, 107–113.

1000 Genomes Project Consortium. *et al.* (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.

Hamming,R.W. (1950) Error detecting and error correcting codes. *AT&T Tech. J.*, **29**, 147–160.

Hess,M. *et al.* (2011) Metagenomic discovery of biomass-degrading genes and genomes from cow rumen. *Science*, **331**, 463–467.

Jourdren,L. *et al.* (2012) Eoulsan: a cloud computing-based framework facilitating high throughput sequencing analyses. *Bioinformatics*, **28**, 1542–1543.

Kolker,N. *et al.* (2011) Classifying proteins into functional groups based on all-versus-all BLAST of 10 million proteins. *Omics*, **15**, 513–521.

Langmead,B. *et al.* (2009) Searching for SNPs with cloud computing. *Genome Biol.*, **10**, R134.

Langmead,B. *et al.* (2010) Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biol.*, **11**, R83.

Leo,S. *et al.* (2009) Biodoop: bioinformatics on hadoop. In: *Parallel Processing Workshops, 2009. ICPPW'09. International Conference on IEEE*. Vienna, Austria, pp. 415–422.

Li,H. *et al.* (2009) The sequence alignment/map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

McKenna,A. *et al.* (2010) The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**, 1297–1303.

Metzker,M.L. (2010) Sequencing technologies - the next generation. *Nat. Rev. Genet.*, **11**, 31–46.

Mullis,K.B. and Faloona,F.A. (1987) Specific synthesis of DNA *in vitro* via a polymerase-catalyzed chain-reaction. *Method Enzymol.*, **155**, 335–350.

Nguyen,T. *et al.* (2011) CloudAligner: a fast and full-featured MapReduce based tool for sequence mapping. *BMC Res. Notes*, **4**, 171.

Niemenmaa,M. *et al.* (2012) Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, **28**, 876–877.

Pope,P.B. *et al.* (2010) Adaptation to herbivory by the Tammar wallaby includes bacterial and glycoside hydrolase profiles different from other herbivores. *Proc. Natl Acad. Sci. USA*, **107**, 14793–14798.

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

Stefan,K. *et al.* (2008) A new method to compute K-mer frequencies and its application to annotate large repetitive plant genomes. *BMC Genomics*, **9**, 1471–2164.

Taylor,R.C. (2010) An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, **11** (**Suppl. 12**), S1.

Warnecke,F. *et al.* (2007) Metagenomic and functional analysis of hindgut microbiota of a wood-feeding higher termite. *Nature*, **450**, 560–565.

Zaharia,M. *et al.* (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, San Jose, CA, p. 2.

Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–829.