Libra: Improved Partitioning Strategies for Massive Comparative Metagenomics Analysis

Illyoung Choi
Dept. of Computer Science
University of Arizona
iychoi@email.arizona.edu

Alise J. Ponsero
Dept. of Agricultural and Biosystems Engineering
University of Arizona
aponsero@email.arizona.edu

Ken Youens-Clark
Dept. of Agricultural and Biosystems Engineering
University of Arizona
kyclark@email.arizona.edu

Matthew Bomhoff
Dept. of Agricultural and Biosystems Engineering
University of Arizona
mbomhoff@email.arizona.edu

Bonnie L. Hurwitz
Dept. of Agricultural and Biosystems Engineering
University of Arizona
bhurwitz@email.arizona.edu

John H. Hartman
Dept. of Computer Science
University of Arizona
jhh@cs.arizona.edu

ABSTRACT

Big-data analytics platforms, such as Hadoop, are appealing for scientific computation because they are ubiquitous, well-supported, and well-understood. Unfortunately, load-balancing is a common challenge of implementing large-scale scientific computing applications on these platforms. In this paper we present the design and implementation of Libra, a Hadoop-based tool for comparative metagenomics (comparing samples of genetic material collected from the environment). We describe the computation that Libra performs and how that computation is implemented using Hadoop tasks, including the techniques used by Libra to ensure that the task workloads are balanced despite non-uniform sample sizes and skewed distributions of genetic material in the samples. On a 10-machine Hadoop cluster Libra can analyze the entire Tara Ocean Viromes of ~4.2 billion reads in fewer than 20 hours.

CCS CONCEPTS

•Information systems → Information systems applications; •Computing methodologies → MapReduce algorithms;

KEYWORDS

comparative genomics, genome distance, metagenomics, k-mer, parallel, task-partitioning

ACM Reference format:

Illyoung Choi, Alise J. Ponsero, Ken Youens-Clark, Matthew Bomhoff, Bonnie L. Hurwitz and John H. Hartman. 2018. Libra: Improved Partitioning Strategies for Massive Comparative Metagenomics Analysis. In *Proceedings of the 9th ACM workshop on Scientific Cloud Computing, Tempe, Arizona, USA, June 11, 2018 (ScienceCloud'18)*, 8 pages.

ScienceCloud'18, June 11, 2018, Tempe, Arizona, USA

1 INTRODUCTION

Microbial communities are comprised of diverse organisms at varied abundances, which change over time due to microbe-microbe interactions and ecosystem processes. Capturing the details of these interactions, however, remains elusive given that the vast majority of microbes cannot be cultured [21,26]. Metagenomics, a method of sequencing microbial DNA/RNA directly from the environment, offers a path forward to analyze the complete genetic repertoire of the microbial community - including both novel and known species. Over the last decade the cost of sequencing has decreased more than a million-fold leading to a rapid influx of metagenomic data from diverse environments, promising insight into novel organisms and ecosystem function. However, this flood of sequencing data has proven difficult to analyze due to its sheer volume.

Hadoop [1] is an implementation of the MapReduce [6] algorithm that has been widely adopted for big-data analytics. MapReduce is a high-level programming abstraction that greatly simplifies the development and deployment of new analytical tools. Programmers implement these tools in terms of "map" and "reduce" functions. Hadoop takes these functions and deploys them across large-scale clusters of machines, allowing the machines to work in concert to process large amounts of data. As a result, programmers do not need specialized training in distributed systems and networking to implement large-scale computations. In addition, Hadoop provides fault-tolerance by reassigning the tasks on a failed machine to other machines. Thus, Hadoop ensures that computation will complete and data are protected even in the event of machine failures.

The success of Hadoop has made it ubiquitous, well-supported, and well-understood. This makes it an appealing platform for scientific computations because researchers can make use of the existing Hadoop infrastructure

and support mechanisms in the cloud, such as Amazon EMR [29]. However, Hadoop was designed for big-data analytics, not for scientific computation, which makes it challenging to use for comparative metagenomics. In particular, some of the techniques Hadoop uses for balancing workloads across tasks require modification.

In this paper we describe the design and implementation of Libra, a tool for performing comparative metagenomics on Hadoop. Libra is capable of performing all-vs-all sequence analysis on large-scale datasets using a variety of similarity/dissimilarity measures that simultaneously consider genetic distance and microbial abundance. Libra provides efficient and scalable computation for comparative metagenomics that can be used to discern global patterns in microbial ecology. Further, Libra's partitioning can be used for load-balancing in any sequence comparison of samples from human health to the environment. On a 10-machine Hadoop cluster Libra can analyze the entire Tara Ocean Viromes of ~4.2 billion reads in fewer than 20 hours.

2 SEQUENCING BACKGROUND

Metagenomics consists of comparing DNA samples collected from the environment. DNA is a double helix structure composed of two complementary strands of nucleobases. Each strand is a sequence of four characters, 'A','T','G' and 'C', representing the four nucleobases adenine, thymine, guanine, and cytosine, respectively. 'A'-'T' and 'G'-'C' are paired in the complementary strands. Each strand has an orientation, i.e., '5'-end to 3'-end, and complementary strands have opposite orientations called the *forward* and *reverse-complement*.

The sequence of A, T, G, and C nucleobases in a strand of DNA is read by a sequencing machine and is called a *read*. There is a limit, however, to the number of sequential nucleobases that sequencing technologies can produce at high accuracy, which is currently a few hundred nucleobases. When computing the complete DNA sequence for a particular organism this limit is overcome by producing many overlapping reads of the DNA, then stitching them together to produce the entire DNA sequence. In contrast, in metagenomics the reads are produced from all the DNA in the environmental sample, regardless of the organism to which they belong.

Finally, a common way of analyzing the DNA in a metagenomic sample is through the statistical analysis of *k*-mers. A *k*-mer — also known as an *n*-gram — is a substring of length *k*, typically produced from every offset in a string. In a string in length L, there exists $(L\text{-}k\text{+}1)$ *k*-mers. Also, there can exist at most C^k unique k-mers in a string comprised of C unique characters. *k*-mers are used in many fields, such as computational linguistics, to retrieve similar documents by comparing their *k*-mer compositions. In Libra we use them to compare metagenomic samples.

3 COMPARATIVE METAGENOMICS

Measuring the distance — or similarity/dissimilarity — between samples is an important analysis in metagenomics. By looking at the distance between samples from different times or environments, we find correlations and bring new insights for further research.

Traditionally, the distance between metagenomic samples has been measured by comparing the composition of known organisms in the samples. This method requires mapping the sequence data to known organisms using reference databases. However, this is less effective in large-scale metagenomics studies because a high percentage of sequence data is derived from unknown organisms [12].

As a result, a widely accepted method of comparing genomic signatures compares *k*-mer compositions. The method relies on three core tenets: (*i*) closely related organisms share *k*-mer profiles and cluster together, making taxonomic assignment unnecessary [8,22], (*ii*) *k*-mer abundance is correlated with the abundance of an organism [25], and (*iii*) *k*-mers of sufficient length can be used to distinguish specific organisms [9]. This method offers high performance and precise results compared to traditional methods of analyzing only the known organisms.

The k-mer composition analysis is performed in two steps: *k*-mer counting, and a distance matrix computation. First, the abundance of each *k*-mer is determined in each sample, and indices are constructed to provide an efficient means of subsequently determining the abundance of a particular *k*-mer in a particular sample. Second, the indices are used to compute a distance between every pair of samples, such that the distance is inversely proportional to the similarity between the samples. The distance is typically measured based on the abundance of *k*-mers by using various statistical functions. Libra implements Bray-Curtis, Jensen-Shannon, and Cosine Similarity [4,14,15] as distance metrics.

There are two known difficulties with *k*-mer-based distance computation related to the large number of *k*-mers produced and the pairwise comparisons. First, approximately 500 million *k*-mers are produced in a one GB sample. As a result, today's metagenomic datasets contain hundreds of billions of *k*-mers (see Table 1). Also, in practice, k must be at least 20 base pairs (bp) to provide sufficient accuracy. For example, if k is 20bp, there are 4^{20} possible *k*-mers. Considering the scale of today's metagenomic datasets, such as the Tara Ocean Viromes (TOV) dataset [3] shown in Table 1, processing this enormous number of *k*-mers is challenging and requires massive computation and storage resources.

Second, the number of sample pairs is quadratic in the number of samples. There are $n(n\text{-}1)/2$ pairs of samples in a dataset containing n samples, e.g. there are 903 pairs of samples in the TOV dataset. Independently computing the

distance between each pair of samples is too computationally expensive to be practical.

Table 1: Statistics of the Tara Ocean Viromes (TOV) dataset

Number of samples	43
Size of dataset	551.6 GB
Number of reads	4,194,402,268
Number of base pairs	403,891,365,808
Number of 20-mers	324,197,722,716

4 EXISTING APPROACHES

There are several *k*-mer counting tools available, such as KMC2 [7] and DSK [20]. These tools are optimized to count *k*-mers in a sample efficiently without requiring excessive RAM. They strive to use disks efficiently so as to overcome insufficient RAM, and they compress the *k*-mer data to reduce the storage required. However, these tools are not suitable for large-scale metagenomics because they need an increasingly powerful machine to keep pace with the ever-growing size of metagenomic datasets. Also, these tools merely count *k*-mers, the distance matrix is computed separately.

Jellyfish [16] is a k-mer counting tool that takes advantage of multi-core machines. The tool is optimized to perform on a multi-core machine using a multi-threaded lock-free algorithm. Jellyfish also compresses the k-mer data to overcome insufficient RAM. However, as with KMC2 and DSK, the tool also is not suitable for large-scale metagenomics because it requires more RAM as the metagenomic datasets grow.

Mash [17] is a distance computation tool. It solves the scalability issue using subsampling. Mash creates smaller *sketches* of the *k*-mer composition of samples using the MinHash algorithm [5], compares these sketches, and computes genetic distance using Jaccard similarity. This greatly improves performance at the expense of accuracy, as the sketches contain only a subset of the *k*-mers and contain no information about their abundance in the samples. Thus, Mash measures genetic distance between samples without considering abundance (dominant vs rare organisms in the sample), which is central to microbial ecology and ecosystem processes.

There are several k-mer counting tools that run on distributed environments, such as Kmerind [18] and Bloomfish [10]. They solve the scalability issue using distributed computing, allowing them to run faster than the most widely accepted tool, Jellyfish. Kmerind achieves high performance using MPI [23] which is a lightweight distributed computing framework. Algorithms based on MPI usually out-perform MapReduce-based algorithms for moderate sizes of data [13]. However, the lack of fault-tolerance makes Kmerid susceptible to failures. Bloomfish improves performance by co-designing the I/O of its base platform, Mimir [11] which is a new implementation of MapReduce over MPI. This makes it unable to use standard MapReduce clusters.

A recently developed distance computation tool, Simka [2] also uses distributed computing. However, Simka has three important limitations. First, tasks for the *k*-mer count are distributed by sample. This can cause workload imbalance when the samples vary in size or there are fewer samples than machines in the cluster. Second, the *k*-mer abundances produced by different machines must be aggregated, involving large amounts of disk I/O. Third, Simka requires specific schedulers, such as the Sun Grid Engine (SGE) [24]. In contrast, Libra makes use of the standard Hadoop infrastructure to distribute tasks across machines and to aggregate the results into the distance matrix.

5 LIBRA OVERVIEW

Libra is implemented in three steps using three different MapReduce jobs — *1*) *k*-mer histogram construction, *2*) inverted index construction (*k*-mer counting), and *3*) distance matrix computation (scoring). Figure 1 shows the Libra workflow.

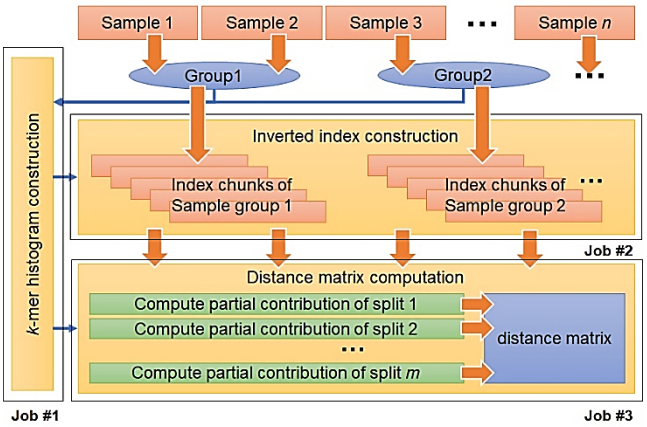

Figure 1: Libra workflow.

In the first step Libra constructs a *k*-mer histogram containing the distribution of *k*-mers for each sample. In the Map phase, a separate Map task is spawned for each data block of the input sample files. The Map tasks calculate the *k*-mer histograms for the blocks in parallel. In the Reduce phase, a Reduce task is spawned to aggregate the partial *k*-mer histograms produced by the Map tasks to produce the complete histogram.

Libra then constructs the inverted index in parallel. In the Map phase, a separate Map task is spawned for every data block in the input sample files. Each Map task generates the *k*-mers from the sequences in its data block, then passes the *k*-mers to the Reduce tasks. In the Reduce phase, the *k*-mer histograms are used to partition the *k*-

mer space. A separate Reduce task is spawned for each partition and in the Shuffle step a custom Partitioner routes the produced *k*-mers to the proper Reduce tasks based on their partitions. Each Reduce task then counts the *k*-mers it receives and produces an index chunk. As a result, each index chunk is stored as a separate file in the Hadoop MapFile [28] format. The MapFile is well-suited for Libra as it stores key-value pairs in key order, allowing for binary search of the keys.

Finally, the distance matrix is computed by distributing the work based on the *k*-mer histograms. The *k*-mer histogram is used to split the *k*-mer space, and a separate Map task is spawned for each split. As a result, each Map task produces an output file containing partial contributions to the distance matrix. Libra merges the partial contributions from the files to produce the complete distance matrix.

6 LOAD BALANCING IN LIBRA

Non-uniform *k*-mer distributions is a major source of task workload imbalance when performing comparative metagenomics on Hadoop. It must be addressed by Libra to ensure balanced workloads among the tasks and avoid bottlenecks.

6.1 *k*-mer counting and partitioning

In the Map phase, each Map task generates *k*-mers from the sequences in its input and for each *k*-mer produces a record containing the *k*-mer, a *FileID,* and the *k*-mer abundance. The *k*-mer is stored in a canonical form — either the forward or the reverse-complement form of the *k*-mer, whichever comes first lexicographically. This is a well-known technique that avoids storing both the forward and reverse-complement versions of the same *k*-mer. The FileID uniquely identifies the sample file and avoids storing the entire filename.

Each Reduce task receives *k*-mer records from the Map task, computes the total abundance of each *k*-mer, and stores the *k*-mers and their abundances in an index file. The Partitioner takes the *k*-mers from the Map tasks and assigns them to the proper Reduce tasks. It is important to partition the *k*-mer space so that the Reduce tasks have balanced workloads. There are several methods of partitioning the *k*-mers, but they may not lead to balanced workloads.

Partitioning by dividing the *k*-mer space into equal ranges is the simplest approach. However, this fixed-range partitioning can cause imbalance in the Reduce tasks because the *k*-mer distributions are not uniform (Figure 2-A). To make matters worse, after converting *k*-mers to their canonical forms, the distribution becomes highly skewed (Figure 2-B).

Another alternative is partitioning *k*-mers by their hash. Hash-based partitioning is a widely-accepted approach for dealing with non-uniform key distribution in Hadoop, however, Libra does not do this because it results in different assignments of *k*-mers to partitions depending on the number of partitions. This makes inverted indices that have different numbers of partitions (e.g., constructed in different runs of Libra) difficult to join during the distance matrix computation, reducing the reusability of the indices. In addition, hashing makes k-mers unordered across index chunks.

Libra's approach to partitioning the *k*-mers is based on variable-size *k*-mer ranges whose size is determined by an approximated *k*-mer distribution under the assumption that workload of a partition is proportional to the number of *k*-mers in it. To construct the approximated *k*-mer distribution, called the *k*-mer histogram, a separate MapReduce job is launched prior to the main Libra computation. In the job, *k*-mers are extracted from samples and canonicalized. However, to reduce the overhead only the first *x* characters of the *k*-mers are used in the histogram. Based on the *k*-mer histogram the *k*-mer space is partitioned to have approximately an equal total number of *k*-mers in each partition.

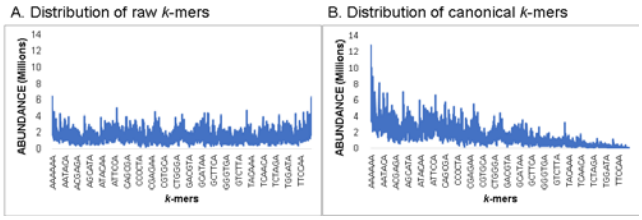

Figure 2: Distribution of *k*-mers in a sample. (Station18_SUR.fa of Tara Ocean Viromes dataset, *k*=20)

We performed an experiment to determine the sensitivity of the partitioning to the length of the prefix, *x*. Figure 3 shows changes of the standard deviation of *k*-mers in partitions (3-A) and the size of histogram (3-B) for different values of *x* (4-8) for *k*=20 using a real sample in the Tara Ocean Viromes (TOV) dataset. Then, the *k*-mer space was divided into 100 partitions based on the histograms, and the *k*-mers in each partition were counted. The workload imbalance was measured by the standard deviation of the number of k-mers in the partitions (3-A). The higher the standard deviation, the more the workloads are imbalanced. The results show that x=6 is sufficient to reduce the workload imbalance. Also, a *k*-mer histogram with x=6 requires only 32KB of space ($4^6 \times 8$ bytes) so that it easily fits in memory. No meaningful runtime difference was observed for different values of *x*. This is because both I/O workloads for reading a large sequence file and computational workloads for producing *k*-mers do not change for different values of *x* – only the number of counters required vary. Using x=6 for *k*=20 also showed good results in another experiment using real samples in the Pacific

Ocean Viromes (POV) dataset [12]. As a result, we chose x=6 in this work.

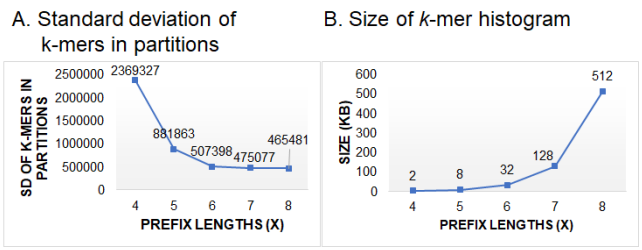

Figure 3: Changes of the standard deviation of *k*-mers in partitions and the size of *k*-mer histogram for different *k*-mer prefix lengths. (Station18_SUR.fa of Tara Ocean Viromes dataset, *k*=20)

6.2 Distance matrix computation (scoring)

To compute the distance matrix, Libra uses a vector space model [20] that produces a vector for each sample. Each dimension of a vector corresponds to a unique *k*-mer, and its value is the weight given to the corresponding *k*-mer in the scoring function. The weight is calculated from the *k*-mer abundance in the inverted indices which uses functions such as logarithmic weighting to dampen the effect of high-frequency *k*-mers. The distance matrix is then computed using various distance functions such as Bray-Curtis, Jensen-Shannon, and Cosine Similarity.

For example, Cosine Similarity determines the similarity between the two samples from the cosine of the angle between their vectors. When the angle is zero (i.e., the vectors are identical except for their magnitude) the cosine is one and less than one otherwise. Therefore, the distance between two samples is 1-*similarity*.

Cosine Similarity has two features that make it useful in comparative metagenomics. First, it has high parallelism. The distance can be efficiently computed in parallel by dividing the vectors into ranges of dimensions. The contributions of each range are computed in parallel, and the contributions are merged in a post-processing step into the distance matrix. Second, the size of a sample does not alter the distance. This means that samples with different sequencing depths can be compared without requiring normalization.

The distance between two samples requires multiplying the weights at each dimension. Therefore, only shared *k*-mers (i.e., those with non-zero abundance in both samples) contribute to the final score, making efficient determination of shared *k*-mers important for high performance. In Libra, a sort-merge join is used to detect shared *k*-mers. Because the inverted indices have entries already in lexicographic order by *k*-mer, the sort-merge join can be performed efficiently.

The same histogram-based *k*-mer range partitioning is used to split inverted indices so that they have approximately equal number of *k*-mers in each split (Figure 4), under the assumption that the workload is proportional to the number of *k*-mers in the split. In fact, the workload is proportional to the number of unique *k*-mers in the split; therefore, it would be more accurate to use the distribution of unique *k*-mers. However, the construction of a unique *k*-mer histogram requires significant computation, and the total *k*-mer histogram works well enough.

A split (p) is assigned for the same *k*-mer range over all inverted indices. Also, a split can span multiple chunks in an inverted index depending on the *k*-mer distribution of the sample. Nevertheless, the input-splitting guarantees that a *k*-mer shared between samples always occurs in the same split because inverted indices have their *k*-mers in lexicographic order.

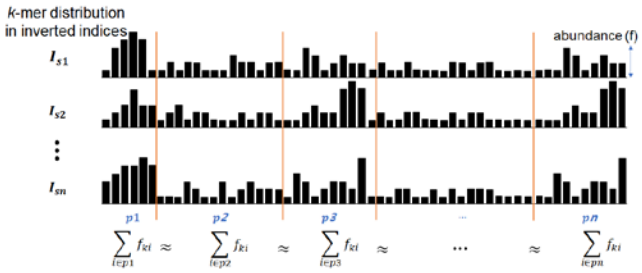

Figure 4: Histogram-based input-splitting in distance matrix computation.

Libra computes the similarity between every pair of samples in the inverted indices. The sort-merge join allows shared *k*-mers to be found in linear time in the total size of the samples. For those samples that share a particular *k*-mer, the contributions to the final score are computed between every pair of samples. This requires quadratic time in the number of samples that share the *k*-mer. While this is potentially a large overhead, in practice it has negligible impact on overall performance because relatively few samples share a *k*-mer and the pairwise computation consists of simple multiplication.

7 RESULTS

7.1 Experimental setup

We benchmarked Libra on a Hadoop cluster consisting of 10 physical machines (9 MapReduce worker machines). Each machine contains 2 Intel Xeon X5670 2.93GHz CPUs (each with 6 cores), 128 GB of RAM, and is configured to run a maximum of 7 YARN containers simultaneously each with 10 GB of RAM. The remaining system resources are reserved for the operating system and other Hadoop services.

To demonstrate the scale and performance of Libra, we analyzed 43 Tara Ocean Viromes (TOV) from the 2009-

2011 Expedition [3], representing 26 sites, 43 samples, 4.2 billion reads, and 324 billion 20-mers (Table 1).

To demonstrate Libra's load-balancing, we used a subset of the Tara Ocean Viromes (TOV) that consists of 10 random samples (119.2 GB in total). The subset is used because the full dataset could not be processed with fixed-range partitioning, which caused the intermediate output of some partitions to exceed the disk capacity. We made a change to MapReduce job configuration to facilitate accurate measurements. During the inverted index construction we configured Hadoop to not run the Reduce tasks until the Map tasks completed (i.e., Reduce tasks could not overlap Map tasks). By default, Hadoop will start some Reduce tasks before the Map tasks complete, causing the Reduce tasks to wait for input and disturbing our measurements.

In all experiments we configured Libra to have 256 partitions. We chose this number to balance the overhead of creating Map and Reduce processes for each partition against the overhead of sorting each partition in the Shuffle step. Too many partitions cause excessive task creation overhead, whereas too few cause excessive sorting overhead. As a result, 256 Reduce tasks were created during the inverted index construction, 256 index chunks were produced, and 256 map tasks were created during the distance matrix computation. We performed the same benchmark with the same samples three times and averaged the results.

7.2 Workload distribution during the inverted index construction

Figure 5 shows the durations of Reduce tasks during the inverted index construction using different partitioning schemes. With histogram-based partitioning 100% of the Reduce tasks completed in less than 1100 seconds, and 84% of the tasks completed in 700 to 900 seconds. In comparison, fixed-range partitioning had much higher variance, with 70% of the tasks finishing in less than 900 seconds, but with a long tail out to 4800 seconds. This variance leads to the larger run times of the overall computation.

Figure 6 shows the sizes of the index chunks produced by the Reduce tasks during inverted index construction. With histogram-based partitioning most chunks are in the range 1100-1300 MB, whereas with fixed-range partitioning the chunk sizes are more widely distributed. This leads to the long tail of task durations seen with fixed-range partitioning.

7.3 Workload distribution in the distance matrix computation

Figure 7 shows the durations of Map tasks in the distance matrix computation using different input splitting algorithms. With histogram-based input-splitting most of the Map tasks completed within ±60 seconds of the average while the fixed-range partitioning showed imbalanced durations between Map tasks.

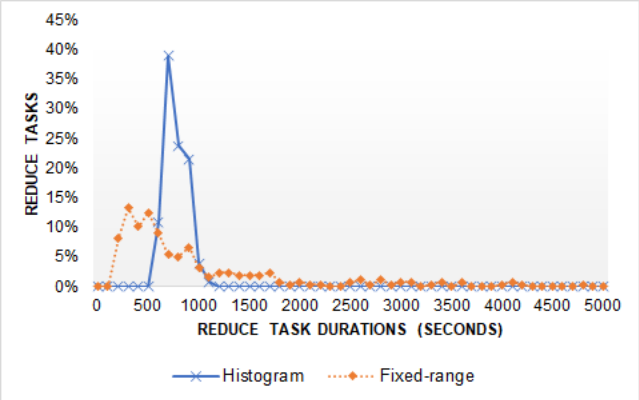

Figure 5: Reduce task durations during inverted index construction.

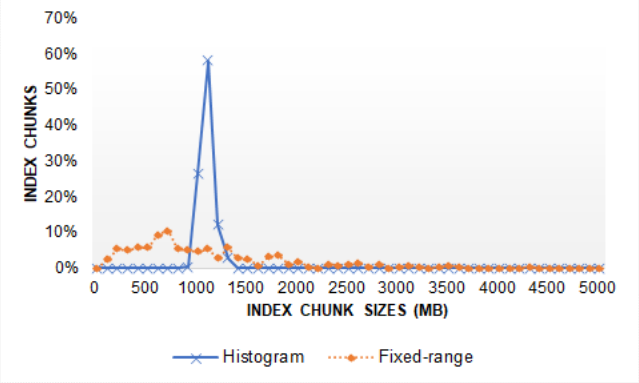

Figure 6: Index chunk sizes.

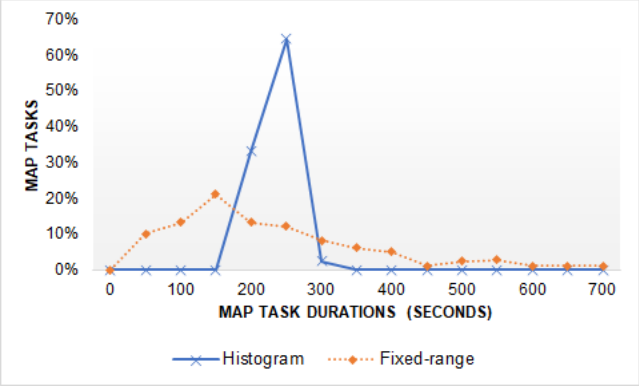

Figure 7: Map task durations during the distance matrix computation.

The above results show that Libra's histogram-based load-balancing is superior to fixed-range load balancing, and importantly reduced the overall run time. Table 2

shows that the using *k*-mer histograms to load-balance improved overall execution time by more than 10%, even when accounting for the additional time required to construct the histogram.

Table 2: Comparison of elapsed times for the different load-balancing approaches (σ=standard deviation).

Step	Histogram	Fixed-range
k-mer histogram construction	0:08:20 (σ=0:00:08)	N/A
Inverted index construction	2:38:52 (σ=0:00:38)	3:03:34 (σ=0:06:15)
Distance matrix computation	0:15:36 (σ=0:00:06)	0:17:14 (σ=0:00:03)
Total	3:02:48 (σ=0:00:27)	3:20:47 (σ=0:06:11)

7.4 Complete Tara Ocean Viromes analysis

The complete analysis of ~4.2 billion reads in the Tara Ocean Viromes (TOV) dataset finished in just under 20 hours (Table 3). The inverted index construction (*k*-mer counting) consumed the most time. This is because the shuffle step involves transferring more than 4.7 TB between the Map and Reduce tasks (Table 4, Reduce Shuffle). By comparison, once the inverted indices are constructed, computing the distance matrix (43×43) for the full Tara Ocean Viromes dataset required less than 2 hours.

Table 3: Elapsed times for Libra based on the full Tara Ocean Viromes (TOV) dataset (σ=standard deviation)

Step	Elapsed Time
k-mer histogram construction	0:40:52 (σ=0:04:16)
Inverted index construction	17:29:18 (σ=0:30:34)
Distance matrix computation	1:45:45 (σ=0:44:51)
Total	19:55:55 (σ=1:09:30)

8 DISCUSSION

Spark [27] is increasingly popular for large-scale data analysis because of its outstanding performance caused by keeping all data in memory. Partitioning strategies used in Libra can be easily ported to Spark because both Hadoop and Spark have similar partitioning interfaces. With our partitioning strategies, Resilient Distributed Datasets (RDDs) can be partitioned and distributed over a Spark cluster evenly. This leads Spark operations (i.e., distance matrix computation) to perform efficiently by balancing workloads. Keeping all data in memory eliminates the disk I/O for intermediate data (Table 4 Reduce Shuffle). Nevertheless, we think Hadoop is still an attractive platform because it doesn't have the same memory requirements as Spark, and therefore doesn't require as many machines. Performing the analysis using Spark wasn't feasible on the hardware we had available, and the problem will only get worse as the datasets get larger.

Table 4: I/O during inverted index construction of the full Tara Ocean Viromes (TOV) dataset

I/O Type	Size
HDFS Read	552.76 GB
HDFS Write	1323.85 GB
Reduce Shuffle	4761.20 GB

9 CONCLUSION

Scientific computing for biology is increasingly driven by large-scale next generation sequencing datasets. It is now possible to generate, aggregate, archive, and share datasets that are terabytes and even petabytes in size. Libra is a Hadoop-based tool for performing comparative metagenomics. Libra uses Hadoop because it is ubiquitous, well-supported, and well-understood. However, load-balancing is a common challenge when using Hadoop for large-scale scientific computing, caused by task durations that are not proportional to the input data size. To solve this problem, Libra uses histogram-based load-balancing that pre-computes a histogram of the *k*-mers present in the input samples. This histogram is used to balance loads in subsequent phases of the computation, and more than offsets the cost of constructing the histogram. Using histogram-based load-balancing, Libra is able to improve the overall execution time to perform comparative metagenomics on a subset of the Tara Ocean Viromes samples by 10% versus a fixed-range load-balancing scheme.

ACKNOWLEDGMENTS

We thank Binil Benjamin, Russell Lewis and Hurwitz Lab for comments on the implementation of the algorithm and feedback on the manuscript. We thank staff at the University of Arizona Information Technology Services for access and system support for the Hadoop cluster. This research was funded in part by NSF grant OAR-1640775.

REFERENCES

[1] The Apache Software Foundation, "Apache Hadoop," Retrieved from *http://hadoop.apache.org/*.

[2] Gaëtan Benoit, Pierre Peterlongo, Mahendra Mariadassou, et al., "Multiple comparative metagenomics using multiset k-mer counting," *PeerJ Comput. Sci.*, 2, p. 21, 2016.

[3] Jennifer R. Brum, J. Cesar Ignacio-Espinoza, Simon Roux, et al., "Patterns and ecological drivers of ocean viral communities," *Science*, 348(6237), p. 1261498, 2015.

[4] Anne Chao, Robin L. Chazdon, Robert K. Colwell, et al., "Abundance-based similarity indices and their estimation when there are unseen species in samples," *Biometrics*,

62(2), pp. 361-371, 2006.

[5] O. Chum, J. Philbin, and A. Zisserman, "Near Duplicate Image Detection: min-Hash and tf-idf Weighting," *BMVC*, 2008.

[6] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, 51(1), pp. 107-113, 2008.

[7] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, et al., "KMC 2: fast and resource-frugal k-mer counting," *Bioinformatics*, 31(10), pp. 1569-1576, 2015.

[8] Veronika B. Dubinkina, Dmitry S. Ischenko, Vladimir I. Ulyantsev, et al., "Assessment of k-mer spectrum applicability for metagenomic dissimilarity analysis," *BMC bioinformatics*, 17, p. 38, 2016.

[9] Yuriy Fofanov, Yi Luo, Charles Katili, et al., "How independent are the appearances of n-mers in different genomes?," *Bioinformatics*, 20(15), pp. 2421-2428, 2004.

[10] Tao Gao, Yanfei Guo, Yanjie Wei, et al., "Bloomfish: A Highly Scalable Distributed K-mer Counting Framework," In *Proceedings of the IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2017.

[11] Tao Gao, Yanfei Guo, Boyu Zhang, et al., "Mimir: Memory-Efficient and Scalable MapReduce for Large Supercomputing Systems," In *Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1098-1108, 2017.

[12] Bonnie L. Hurwitz and Matthew B. Sullivan, "The Pacific Ocean virome (POV): a marine viral metagenomic dataset and associated protein clusters for quantitative viral ecology," *PLoS ONE*, 8(2), 2013.

[13] Sol Ji Kang, Sang Yeon Lee, and Keon Myung Lee, "Performance Comparison of OpenMP, MPI, and Mapreduce in Practical Problems," *Adv. MultiMedia*, 2015.

[14] Bo Liao, Cheng Zeng, F. Q. Li, et al., "Analysis of similarity/dissimilarity of DNA sequences based on dual nucleotides," *MATCH Commun. Math. Comput. Chem.*, 59(2008), pp. 647-652, 2008.

[15] Wentian Li, "The Measure of Compositional Heterogeneity in DNA Sequences Is Related to Measures of Complexity," *Complexity*, 3(2), pp. 33-38, 1997.

[16] Guillaume Marçais and Carl Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics*, 27(6), pp. 764-770, 2011.

[17] Brian D. Ondov, Todd J. Treangen, Pall Melsted, et al., "Mash: fast genome and metagenome distance estimation using MinHash," *Genome Biol.*, 17, 2016.

[18] Tony Pan, Patrick Flick, Chirag Jain, et al., "Kmerind: A Flexible Parallel Library for K-mer Indexing of Biological Sequences on Distributed Memory Systems," *IEEE/ACM Trans. Comput. Biol. Bioinform.,* 2017.

[19] Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi, "DSK: k-mer counting with very low memory usage," *Bioinformatics*, 29(5), pp. 652-653, 2013.

[20] G. Salton, A. Wong, and C. S. Yang, "A Vector Space Model for Automatic Indexing," *Commun. ACM*, 18(11), pp. 613-620, 1975.

[21] Shinichi Sunagawa, Luis Pedro Coelho, Samuel Chaffron, et al., "Structure and function of the global ocean microbiome," *Science*, 348(6237), p. 1261359, 2015.

[22] H. Teeling, J. Waldmann, T. Lombardot, et al., "TETRA: a web-service and a stand-alone program for the analysis and comparison of tetranucleotide usage patterns in DNA sequences," *BMC bioinformatics*, 5(1), p. 163, 2004.

[23] David W. Walker and Jack J. Dongarra, "MPI: A standard message passing interface," *Supercomputer*, 12, pp. 56-68, 1996.

[24] Wolfgang Gentzsch, "Sun Grid Engine: towards creating a compute power grid," In *Proceedings of the first IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 35–36, 2001.

[25] Yu-Wei Wu and Yuzhen Ye, "A novel abundance-based algorithm for binning metagenomic sequences using l-tuples," *J. Comput. Biol.*, 18(3), pp. 523-534, 2011.

[26] S. Yooseph, G. Sutton, D. B. Rusch, et al., "The Sorcerer II Global Ocean Sampling expedition: expanding the universe of protein families," *PLoS Biol.*, 5(3), 2007.

[27] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, et al., "Spark: Cluster computing with working sets," *HotCloud*, 10(10-10), p. 95, 2010.

[28] Jon Zuanich, "Hadoop I/O: Sequence, Map, Set, Array, BloomMap Files," Retrieved from *http://blog.cloudera.com/blog/2011/01/hadoop-io-sequence-map-set-array-bloommap-files/*.

[29] Amazon Web Services, Inc., "Amazon EMR," Retrieved from *https://aws.amazon.com/emr/?nc1=h_ls*.