2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

A Map-Reduce Framework for Clustering Metagenomes

Zeehasham Rasheed
Department of Computer Science
George Mason University
Fairfax, VA 22030 USA
zrasheed@gmu.edu

Huzefa Rangwala
Department of Computer Science
George Mason University
Fairfax, VA 22030 USA
rangwala@cs.gmu.edu

***Abstract*—The past few years has seen an explosion in the use of sequence technologies for metagenomics i.e., determination of the collective genome of microorganisms co-existing within several environments. In parallel, there has been rapid development of computational tools for the quantification of abundance, diversity and functionality of different species within these communities. Several clustering algorithms (also called binning algorithms) have been developed to categorize similar metagenome sequence reads for efficient post-processing and analysis.**

In this paper we present a distributed algorithm for clustering metagenome sequence reads. The algorithm is implemented within the Map-Reduce based Hadoop platform, and approximates the computation of pairwise sequence similarity with a minwise hashing approach. The algorithm is capable of performing agglomerative hierarchical clustering or a greedy clustering approach and is referred to as `MrMC-MinH`. The key advantage of `MrMC-MinH` is it's ability to handle large volumes of sequence reads obtained from targeted 16S metagenomic or whole metagenomic data. We evaluate the performance of our algorithm on several real and simulated metagenome benchmarks and demonstrate that our approach is computationally efficient, and produces accurate clustering results when evaluated using external ground truth. The source code for `MrMC-MinH` will be made available at the supplementary website[1].

***Keywords*-map-reduce, metagenome clustering, minwise hashing**

I. Introduction

The process of "metagenomics" involves sequencing of the genetic material of organisms co-existing within ecosystems ranging from ocean, soil and human body [1], [2], [3]. Researchers have embarked on various metagenomic-related studies, that range from characterizing the microbial biodiversity within the ocean or understanding the pathogenic role played by microbes within the human gut. Advances in sequencing technologies have allowed this revolution where genomic information can be determined for entire microbial communities, without requiring the separation of individual species.

However, current sequencing technologies do not provide the whole genome for the different organisms, but produce short, contiguous subsequences (referred to as reads) that are fragmented from random positions of the entire genome. Metagenome assembly involves the complex challenge of separating the reads from multiple organisms and stitching together different reads using overall information to produce organism-specific genomes. Other challenges are introduced due to the varying abundance, diversity, complexity, genome lengths of previously uncultured (or never sequenced before) microbes within different communities. Genomic technologies also produce large number of sequence reads, and reads that may have varying error idiosyncracies [4]. As such, the metagenome assembly and analysis problem is complex and challenging [5].

Targeted metagenomics or 16S rRNA gene sequencing provides a first step for the quick and accurate characterization of microbial communities. 16S sequences are marker genes, which exists in most microbial genomes and have a conserved portion for detection (primer development) and a variable portion that allows for categorization within different taxonomical groups [6]. Targeted metagenomics are also effective in detecting species with low abundances. However, they may not be good in discovering unique species (orphans) that have never been sequenced before.

Several algorithms have been developed to analyze targeted metagenomes (16S rRNA marker gene) and whole metagenome samples [4]. Clustering/binning approaches involve the unsupervised grouping of sequences that belong to the same species. Successful grouping of sequence reads has several advantages: (i) it improves the metagenome assembly, (ii) it allows computation of species diversity metrics and (iii) it serves as a pre-processing step by reducing computational complexity within several workflows that analyze only cluster representatives, instead of individual sequences within a sample.

In this paper, we present a new, scalable Map-Reduce based algorithm for metagenome clustering using minwise hashing. We refer to this approach as `MrMC-MinH`, which is an extension of our previously developed, greedy clustering algorithm MC-MinH [7]. The key contributions of this work include: (i) development of the distributed map-reduce based implementation of clustering algorithm and (ii) ability to perform hierarchical agglomerative clustering instead of a greedy clustering approach.

[1] http://www.cs.gmu.edu/~mlbio/MrMC-MinH

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.100

IEEE computer society

MrMC-MinH uses minwise hashing [8] for quickly and accurately computing approximate pairwise sequence similarity (Jaccard similarity) to cluster 16S and whole metagenome sequence reads. Clustering results at different hierarchical taxonomic levels are also produced by setting similarity threshold within a cluster.

We evaluate the performance of MrMC-MinH on two 16S simulated metagenome samples, eight 16S environmental metagenome samples [9] and twelve simulated and real whole metagenome samples [10]. Our comparative study shows the strength of MrMC-MinH in comparison to several other metagenome clustering algorithms when evaluated on the basis of clustering quality and pairwise sequence similarity of sequences within a cluster. Further the source code along with benchmarks is available on the supplementary website; http://www.cs.gmu.edu/~mlbio/MrMC-MinH

II. Related Work

One of the main challenges with current methods for metagenome clustering is the ability to handle large volumes of data while maintaining the quality of clustering. This is particularly an issue for terabyte-scale metagenomics projects. The HiSeq 2000 system can generate up to 0.6 TB of data per run [11].

Among the existing methods, CD-HIT [12] and UCLUST [13] can handle relatively large datasets. UCLUST and CD-HIT are greedy clustering algorithms that achieve computational efficiency by first identifying gapless, high-scoring common subsequences referred to as seeds or hits. Use of seeds reduces the number of needed full length sequence comparisons or alignments. However, UCLUST and CD-HIT are intended for clustering sequences that are highly similar. On the other hand, DOTUR [14], MOTHUR [15] and ESPRIT [16] are exclusively designed for clustering 16S sequences. These methods compute an all-pairwise distance matrix as input and then perform hierarchical clustering. ESPRIT is efficient in comparison to Mothur and DOTUR because it computes k-mer distance for each pair of input sequences, avoiding the expensive global alignment distance calculation. ESPRIT also implements several heuristics to reduce the number of sequence comparisons. In our previous work, we implemented a greedy clustering approach using locality sensitive hashing (LSH) for binning 16S sequences [17], [18].

MetaCluster [19] implements a two-phase (top-down separation and bottom-up merging) approach to cluster metagenome reads or contigs. Clusters are assigned on the basis of k-mer frequency and Spearman distance computation. Other methods, such as sketching techniques [20] and parallel clustering or alignment algorithms [21], [22] are designed to handle large datasets, but are implemented for 16S sequence data. Yang et. al. [23] proposed a Hadoop-based algorithm for the metagenome clustering. They adapted the sketching technique [8] to identify pairwise sequence homologs and produced hierarchical taxonomic clustering results. However, their approach was benchmarked for 16S metagenome sequences only.

Inspired from our previous greedy clustering algorithm called MC-MinH [7], we use minwise hashing [8] to partially offset the time complexity for computing pairwise sequence similarity or alignments. For distributed computing, we implement our algorithm using the Pig scripting language, which provides a high-level platform for developing Map-Reduce applications on Hadoop [24]. The map-reduce paradigm has been drawing attention within the computational biology community [25], [26], [27]. However, majority of these applications are limited to simply distributing the data which is then handled by existing sequential software.

III. Methods

In this section, we present our approach to cluster metagenome sequences using minwise based hashing and agglomerative hierarchical clustering. The hashing approach allows us to reduce the complexity of computing exact pairwise string matching or multiple sequence alignment. We refer to our greedy clustering approach as MrMC-MinH$^{\mathrm{g}}$ and agglomerative hierarchical clustering approach as MrMC-MinH$^{\mathrm{h}}$. We first review the minwise hashing approach and then describe our greedy and agglomerative hierarchical clustering algorithms.

A. Min-Wise Hashing Algorithm

Given a sequence represented as a set of k-mers (words/features), the similarity between two sequences can be defined as the Jaccard similarity between two corresponding sets of k-mers. If sequence s_1 has k-mer set denoted by I_{s_1} and sequence s_2 has k-mer set I_{s_2} then the Jaccard similarity is defined as:

$$sim(s_1, s_2) = \frac{|I_{s_1} \cap I_{s_2}|}{|I_{s_1} \cup I_{s_2}|}. \quad (1)$$

Now, we define the notion of min-wise independent families of permutations. Given $\mathcal{S}_n$ as the set of all independent permutations of $\lfloor n \rfloor = 1 \ldots n$, we define a family of permutations, $F \subseteq \mathcal{S}_n$ to be min-wise-independent, if for any set $X \subseteq \lfloor n \rfloor$ and any $x \in X$, when π is chosen at random in F, we have

$$Pr(min\{\pi(X)\} = \pi(x)) = \frac{1}{|X|} \quad (2)$$

Thus, any element in the set X has an equal probability to become the minimum element of X under the permutation π [8]. Given a random permutation, every element has the same probability of being the smallest element in its subset.

Given input sets of n, k-mers (maximum value of $n = 4^k$) and given a min-wise independent permutation, π chosen randomly from set F, the similarity between two sets (sequences s_1 and s_2) I_{s_1} and I_{s_2} is given by:

$$Pr(min\{\pi(I_{s_1})\} = min\{\pi(I_{s_2})\}) = \frac{|I_{s_1} \cap I_{s_2}|}{|I_{s_1} \cup I_{s_2}|} = sim(s_1, s_2) \quad (3)$$

The probability of having the same minimum value within a permutation for two sets equals the Jaccard similarity between the two sets. As such, we can choose n independent random permutations $\pi_1, \pi_2, \ldots, \pi_n$ and for each sequence s, we store the list:

$$\overline{s} = (min\{\pi_1(I_s)\}, min\{\pi_2(I_s)\}, \ldots, min\{\pi_n(I_s)\}) \quad (4)$$

We can easily estimate the similarity between s_1 and s_2 by performing set intersection and set union of min-wise values in $\overline{s_1}$ and $\overline{s_2}$. The list $\overline{s_1}$ for s_1 is known as the fixed size sketch for representing the sequence.

B. Algorithm Details

For given metagenome sequences represented by k-mer sets, we choose n min-wise independent permutations, represented as $\pi_1, \pi_2, \ldots, \pi_n$. The problem with this approach is that it not feasible to permute a large sequence set. Drawing random permutations is time consuming and practically inefficient. Fortunately, it is possible to simulate the effect of a random permutation by using universal hashing functions. This requires storing few hash values [28]. The standard universal hash function [29] is defined as

$$h_i(x) = (\ (a_i x + b_i) \bmod p\ ) \bmod m, \qquad i = 1, 2, \ldots, n \quad (5)$$

where m is the size of feature set, $p > m$ is a prime number and n is the number of hash functions. The parameters a_i and b_i are chosen uniformly from $\{0, 1, \ldots, p-1\}$. Instead of storing π_i, we now only need to store $2n$ numbers, a and b for each hash function.

In our approach, instead of picking n random permutations, we pick n universal hashing functions $\{h_1, h_2, \ldots, h_n\}$ to approximate the random permutations. To compute the min-wise hash values for a given feature set I, we iterate over all k-mer features x and map them to their hash values $h_i(x)$. We then iterate over all the hash values to find their minimum, which will be the i^{th} min-wise value for that feature set. We formulate the $minHash$ function as the smallest element of a set I under the ordering induced by the universal hashing function h, given by:

$$minHash\ (h(I)) = \arg\min_{x \in I} h(x) \quad (6)$$

Using the min-wise hashing property, the probability of hashing collision for two sets is equal to their Jaccard similarity. For metagenome sequence clustering we are interested in finding those clusters which contain sequences that have similarity greater than some pre-defined threshold θ. Therefore, in our greedy algorithm, we formulate the probability of collision such that if $Pr[minHash\ (h(I_{s_1})) = minHash\ (h(I_{s_2}))] > \theta$, then s_1 and s_2 belong to the same cluster, otherwise a new cluster is created. In case of the hierarchical algorithm, we compute the all pairwise similarity matrix using the input sequence reads with the Jaccard similarity given by Equation 3.

The pseudocode of our approach is described in Algorithm 1 and Algorithm 2. The input parameters include k-mer size k, number of hash functions n and similarity threshold θ. To compute the feature set I for given sequences (line 2), each sequence is converted to a set of fixed length subsequences of length k, called k-mers. This allows both approaches to handle variable length sequences. After the feature sets are computed, we use universal hash functions $h()$ defined in Equation 5 to compute hash value for each element in feature set I. At the end of this procedure, each sequence has n min-wise hash values, where n is the number of hash functions.

1) Greedy Approach: The greedy algorithm follows a step-wise, incremental procedure. The pseudo code is described in Algorithm 1. After computing the minwise values for each sequence, we begin by choosing the first sequence (or any one in the set S) and assign the sequence to the first cluster. Using the list representation (Equation 4), we can compute the Jaccard similarity between the unassigned sequences and the cluster representative. Sequences which have similarity greater than specified threshold θ are assigned to same cluster. We iterate through Steps 5-14 for all sequences in set S until cluster label is assigned to every sequence. When θ is set to 1, the `MrMC-MinH`^{g} algorithm will consider sequences to be similar if and only if all the min-wise values are identical in both sequences. Similarly, when θ is set to 0.95, sequences will go into the same cluster only if 95% of total min-wise values are same in both sequences. Thus lower value of θ allows more sequences to gon into the same cluster, resulting in less number of total clusters.

2) Agglomerative Hierarchical Approach: Instead of following a heuristic approach, `MrMC-MinH`^{h} implements an exhaustive approach for clustering sequences (See Algorithm 2). After computing the minwise values for each sequence, `MrMC-MinH`^{h} computes all pairwise similarity matrix amongst the input sequences. We use Jaccard coefficient as a similarity function which is defined as the size of the intersection divided by the size of the union between minwise values of two feature sets. This similarity matrix is then used to build a dendrogram by iteratively linking each row of matrix according to the linkage policy (single, average or complete) in a bottom up manner. The dendrogram is represented as a series of merge steps for the rows of the similarity matrix, where each row is initially assigned to its own cluster. The similarity threshold θ decides the cutoff level on dendrogram. It produces the clusters such that no pair of sequences within a cluster have less than the pre-defined threshold, θ percent similarity.

C. Overview of `MrMC-MinH`

Figure 1 provides an overview of our approach. Each sequence in fasta format is stored as a HDFS (Hadoop

Algorithm 1 Greedy Clustering

Input: N sequences $S = \{s_1 \dots s_N\}$, each of variable length.
Parameters: k-mer size k, number of hash functions n, similarity threshold θ
Functions: I_s denotes k-mer size feature set for sequence s. $minHash\ (h(I))$ computes the min-wise hash value for I using hash function h .
Output: Cluster Assignments, indexed by different sequences $C[s_1 \dots s_N]$

Initialize array C, $\forall s_i \in S$, $C[s_i] = -1$.
$\forall s \in S$, compute k-mer feature set I_s
$\forall s \in S$, compute $minHash\ (h(I_s))$, where $h_i(x) = ((a_i x + b_i) \bmod p) \bmod m \mid x \in I_s$ and $i = 1, 2, \dots, n$
repeat
  Choose $s_i \in S$, such that $C[s_i] == -1$ i.e., s_i is not assigned
  Assign $C[s_i]$ to a **new** cluster label.
  Remove s_i from S
  for $\forall s_j \in S$ **do**
    / For all s_j in S that are not assigned, try to assign these sequences the same cluster label */*
    if $\lfloor \frac{|minHash(I_{s_i}) \cap minHash(I_{s_j})|}{|minHash(I_{s_i}) \cup minHash(I_{s_j})|} \rfloor \geq \theta$ **then**
      $C[s_j] \leftarrow C[s_i]$
      / s_i and s_j are in the same cluster */*
      Remove s_j from S
    end if
  end for
until S is empty and all sequences are assigned.
return C

Algorithm 2 Agglomerative Hierarchical Clustering

Input: N sequences $S = \{s_1 \dots s_N\}$, each of variable length.
Parameters: k-mer size k, number of hash functions n, similarity threshold θ
Functions: I_s denotes k-mer size feature set for sequence s. $minHash\ (h(I))$ computes the min-wise hash value for I using hash function h .
Output: Cluster Assignments, indexed by different sequences $C[s_1 \dots s_N]$

$\forall s \in S$, compute k-mer feature set I_s
$\forall s \in S$, compute $minHash\ (h(I_s))$, where $h_i(x) = ((a_i x + b_i) \bmod p) \bmod m \mid x \in I_s$ and $i = 1, 2, \dots, n$
$\forall s \in S$, compute all pairwise similarity matrix with respect to $minHash$ values.
Perform Agglomerative Hierarchical Clustering on all pairwise similarity matrix with similarity threshold θ.
return C

Distributed File System) file. Each file is passed to individual mappers to a perform series of operations. First, the DNA alphabets for each sequence are encoded into an integer value. Next, the sequence is transformed into contiguous subsequences or k-mers of fixed size k. After this transformation, each k-mer in a feature set is mapped to hash values using universal hashing functions. Finally, the minimum hash values are chosen as the feature set representing each sequence. These set of minwise hash values are then passed to either of the two different clustering approaches: (i) greedy clustering or (ii) agglomerative hierarchical clustering. For agglomerative hierarchical clustering, the calculation of all pairwise similarity is performed in parallel by performing a row-wise partition. The final clustering output contains cluster label for each sequence and is stored as a HDFS file.

1) Map-Reduce Implementation using Pig Script: We implement our Map-Reduce clustering framework using Pig scripting language and Java. Pig is an apache project developed for Hadoop to execute Map-Reduce jobs in an easy way. It comprises of scripts for executing queries which runs a Hadoop job in the background. Pig scripts also contain UDFs (User Defined Functions) to implement custom functions. Algorithm 3 shows the scripts used in our implementation. Step 1 loads the fasta file from HDFS (Hadoop Distributed File System) using UDF FastaStorage which is implemented in Java. In Step 2, UDF StringGenerator maps the DNA alphabets into integer value. Step 3 calculates the k-mers for each sequence with size $KMER by calling UDF TranslateToKmer. Similarly, Step 4 computes the minwise hash values using UDF CalculateMinwiseHash with parameters $NUMHASH and $DIV, where $DIV is a prime number greater than size of feature set and $NUMHASH is the number of hash functions. In Step 5 which uses UDF CalculatePairwiseSimilarity, we see that

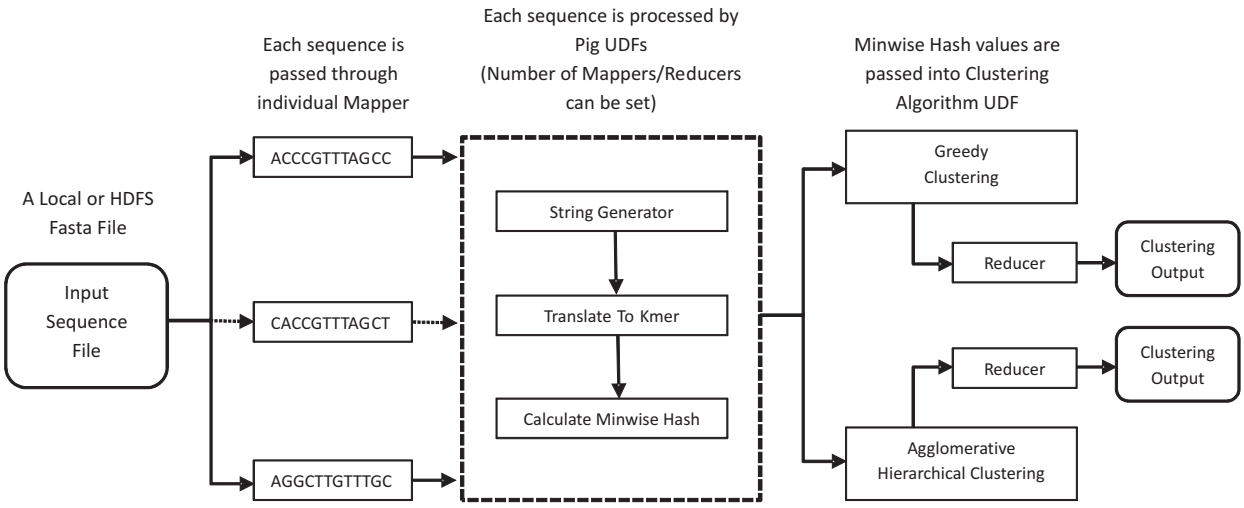

Figure 1. MrMC-MinH Overview and Implementation.

pairwise similarity calculation is performed in parallel by sending each record with the group of sequences. From Step 6 to Step 9, agglomerative hierarchical clustering and greedy clustering are performed using UDFs AgglomerativeHierarchicalClustering and GreedyClustering, respectively. For UDF AgglomerativeHierarchicalClustering, parameter $LINK gives a choice of selecting single-linkage, average-linkage or complete-linkage method for hierarchical clustering, whereas parameter $CUTOFF specifies the cluster similarity threshold at which two clusters are merged together in a hierarchy. For UDF GreedyClustering, $CUTOFF is required to put similar sequences in a cluster until all the sequences are clustered. In the end, results are stored in HDFS and can be accessed using Hadoop commands. The keyword FOREACH ensures that every operation is performed parallel on each sequence to provide distributed computing.

Algorithm 3 Hadoop Map-Reduce Implementation using Pig Script

```
1: A = LOAD '$INPUT' using FastaStorage as (readid:chararray, d:int, seq:bytearray, header:chararray);
2: B = FOREACH A GENERATE FLATTEN (StringGenerator(seq, readid)) as (seq:chararray, seqid:chararray);
3: C = FOREACH B GENERATE FLATTEN (TranslateToKmer(seq, seqid, $KMER)) as (seqkmer:long, seqid2:chararray);
4: E = FOREACH C GENERATE FLATTEN (CalculateMinwiseHash(seqkmer, seqid2, $NUMHASH, $DIV)) as (minwise:long, seqid3:chararray);
5: F = FOREACH E GENERATE FLATTEN (minwise), FLATTEN (seqid3);
6: I = GROUP F ALL;
7: J = FOREACH F GENERATE FLATTEN (CalculatePairwiseSimilarity(minwise, I.F)) as (similaritymatrix: double);
8: K = FOREACH J GENERATE FLATTEN (AgglomerativeHierarchicalClustering(similaritymatrix, $LINK, $NUMHASH, $CUTOFF)) as (clusterlabel: int);
9: L = FOREACH I GENERATE FLATTEN (GreedyClustering(I.F, $NUMHASH, $CUTOFF)) as (clusterlabel: int);
10: STORE K INTO '$OUTPUT1';
11: STORE L INTO '$OUTPUT2';
```

IV. Experimental Evaluation

A. Dataset Description

We evaluate the performance of our algorithm on both, the 16S and whole metagenome sequence datasets.

1) 16S Simulated metagenome samples: The simulated data contains 345,000 short sequences, generated by pyrosequencing two PCR amplicon libraries. The libraries contains 43 known 16S rRNA gene fragments using the Roche GS20 system. Furthermore, simulated data is improvised by incorporating some sequencing errors to test the clustering results. This simulated data is originally used by Huse et. al. [30].

2) 16S Environmental samples: This dataset comprises of eight seawater samples taken from a study by Sogin et. al. [9]. For each sample, a massively parallel DNA sequencing technology [31] (454/Roche) is used to efficiently increase the number of sequenced PCR amplicons. These samples provides a global in-depth description of the diversity of microbes and their relative abundance in the ocean. Samples contain unequal length sequences with average sequence length of 60 bp. The description of these datasets are shown in Table I.

Table I
ENVIRONMENTAL DNA SAMPLES.

SID	Site	La°N, Lo°W	Dep	T	Reads
53R	Labrador seawater	58.300,-29.133	1,400	3.5	11218
55R	Oxygen minimum	58.300,-29.133	500	7.1	8680
112R	Lower deep water	50.400,-25.000	4,121	2.3	11132
115R	Oxygen minimum	50.400,-25.000	550	7.0	13441
137	Labrador seawater	60.900,-38.516	1,710	3.0	12259
138	Labrador seawater	60.900,-38.516	710	3.5	11554
FS312	Bag City	45.916,-129.983	1,529	31.2	52569
FS396	Marker 52	45.943,-129.985	1,537	24.4	73657

Description of the samples used in this paper. These samples are collected from North Atlantic Deep Water and Axial Seamount, Juan de Fuca Ridge. La is the latitude, Lo is the longitude, Dep is the depth in meters, T is the temperature in °C and Reads are the number of sequences in a sample.

3) Simulated and Real Whole metagenome sequences: For whole metagenome sequences, we have 14 simulated samples with varying proportions of microbes and 1 real metagenome sample. For each sample, reads from multiple genomes are pooled to model the challenges of metagenome sequencing such as varying number of species, relative abundance, phylogenetic diversity and the differences in GC content between genomes. These datasets are taken from a previous studies by Chatterji et al. [10]. Furthermore, we test our method on a publicly available metagenome sample R1 that contains sequence reads obtained from gut of the glassy-winged sharpshooter *Homalodisca coagulata* (insect). All samples are described in Table II.

B. Performance Metrics

We evaluate the performance of our algorithm with different metrics and criteria. Taxonomic class labels for each sequence are taken as ground truth for the simulated datasets. Using these labels, we determine a weighted cluster accuracy. Each cluster is designated by class/genera based on the most frequent class in the cluster, and then the accuracy is evaluated by computing the percent of correctly assigned sequences with respect to the designated class. The reported accuracy is averaged across all clusters, weighted by the number of sequences in each cluster. This is denoted by "W.Acc" in this paper. We also compute the sequence similarity within the clusters for each clustering solution. Typically, sequence similarity is evaluated by performing a pairwise alignment operation (i.e., best arrangement of DNA nucleotides (characters) to identify regions of similarity between sequences). Global alignments attempt to align every character between the sequences and local alignments find the best sub-regions of similar characters [32]. Generally, a good sequence clustering solution should produce sequences within a cluster that are similar to each other. We report only the average global sequence alignment similarity (weighted by number of sequences in a cluster). This quantity is referred to as "W.Sim" in this paper. We report results for clusters having number of sequences greater than 50.

C. Hardware and Software Details

The `MrMC-MinH` algorithm is implemented using the Pig scripting language and Java. We evaluated our approach on the Amazon Elastic Map-Reduce Hadoop environment. Particularly, we used the M1 Large instance type (7.5 GiB memory, 4 EC2 Compute Units, 850 GB instance storage) nodes, and increase the number of nodes of same type for our experiments evaluating speedup.

V. RESULTS AND DISCUSSION

We perform a comprehensive set of experiments to evaluate the performance of our algorithm on simulated and real metagenome datasets. Our benchmarking focuses on the quality of clustering evaluated using external ground truth, computational complexity (run time) and pairwise sequence similarity of sequences within a cluster.

A. Algorithm Evaluation

We compare the clustering performance of the greedy clustering (described in Algorithm 1) and the agglomerative hierarchical clustering (described in Algorithm 2). The results are shown in Table III, benchmarked on the simulated and real whole metagenome sequence read sets. We observe from Table III that the hierarchical clustering approach outperforms the greedy clustering in terms of clustering accuracy and pairwise similarity. The hierarchical clustering involves computing an all-pairwise sequence similarity (using minwise hashing), which will better capture the grouping among the input sequence reads. On the other hand, the greedy clustering is faster because it iteratively computes similarities for a shrinking input set. For samples S1-S10, the run time performance for `MrMC-MinH`^{h} averages about 4m 20s. This is because the number of input sequence reads are the same in these 10 samples and the cost of computing the all pairwise similarity is the most time consuming and identical for the 10 samples. These runs were benchmarked on the Amazon cluster using 8 nodes of type described in Section IV-C.

B. Scalability and Speed Up

We assess the scalability of `MrMC-MinH` algorithm by varying the number of computing nodes and input sizes. We specifically, report the run time results for the hierarchical clustering algorithm by varying the number of computing nodes from 2 to 12, and changing the number of input sequence reads from 1000 to 10 million. The reads were obtained from benchmark S1 (whole metagenome sequences) shown in Table II. We observe in Figure 2, that increasing the number of input nodes leads to a decrease in the run times, as more parallelism is supported by a larger input size. For

Table II
WHOLE METAGENOMIC SEQUENCE READS.

SID	Species	Ratio	Taxonomic Difference	# Cluster	# Reads
S1	Bacillus halodurans [0.44], Bacillus subtilis [0.44]	1:1	Species	2	49998
S2	Gluconobacter oxydans [0.61], Granulobacter bethesdensis [0.59]	1:1	Genus	2	49998
S3	Escherichia coli [0.51], Yersinia pestis [0.48]	1:1	Genus	2	49998
S4	Rhodopirellula baltica [0.55], Blastopirellula marina [0.57]	1:1	Genus	2	49998
S5	Bacillus anthracis [0.35], Listeria monocytogenes [0.38]	1:2	Family	2	49998
S6	Methanocaldococcus jannaschii [0.31], Methanococcus mariplaudis [0.33]	1:1	Family	2	49998
S7	Thermofilum pendens [0.58], Pyrobaculum aerophilum [0.51]	1:1	Family	2	49998
S8	Gluconobacter oxydans [0.61], Rhodospirillum rubrum [0.65]	1:1	Order	2	49998
S9	Gluconobacter oxydans [0.61], Granulobacter bethesdensis [0.59], Nitrobacter hamburgensis [0.62]	1:1:8	Family,Order	3	49996
S10	Escherichia coli [0.51], Pseudomonas putida [0.62], Bacillus anthracis [0.35]	1:1:8	Order, Phylum	3	49996
S11	Gluconobacter oxydans [0.61], Granulobacter bethesdensis [0.59], Nitrobacter hamburgensis [0.62], Rhodospirillum rubrum [0.65]	1:1:4:4	Family, Order	4	99998
S12	Escherichia coli [0.51], Pseudomonas putida [0.62], Thermofilum pendens [0.58], Pyrobaculum aerophilum [0.51], Bacillus anthracis [0.35], Bacillus subtilis [0.44]	1:1:1:1:2:14	Species, Order, Family, Phylum, Kingdom	6	99994
S13	Acinetobacter baumannii SDF, Pseudomonas entomophila L48	1:1	-	2	4000
S14	Ehrlichia ruminantium Gardel, Anaplasma centrale Israel, Neorickettsia sennetsu Miyayama	1:1:1	-	3	6000
R1	Glassy-winged sharpshooter endosymbionts	-	-	-	7137

Simulated (S) and Real (R) metagenome shotgun datasets. Each dataset has a unique SID for reference. The GC content of each genome is written in [] brackets. S1-S12 are obtained from the work by Chatterji et al. [10]

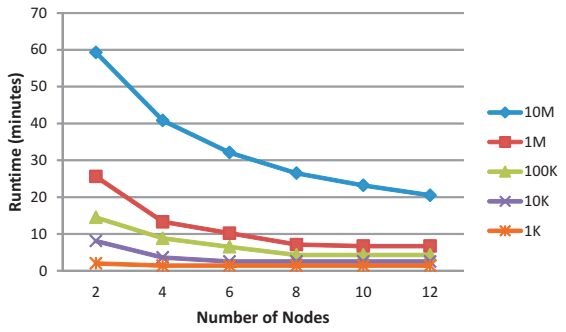

Figure 2. Runtime (in minutes) with respect to number of nodes and number of reads.

the 10 million sequence benchmark, we can further reduce the run time by introducing more nodes. On the other hand, for the smallest input size of 1000 sequences, we observe that there is no effect on run time of increasing the number of nodes. This is because few nodes are sufficient for small input size and therefore, using large number of nodes has no effect on the system performance.

C. Comparative Performance

1) 16S Simulated dataset:: Table IV shows the performance of different clustering algorithms for the 16S simulated set derived from 43 distinct genomes. We report results for reads that have less than 3% and 5% error. From Table IV, we observe that for both, the 3% and 5% sequencing error benchmarks, all algorithms either underestimate or overestimate the number of species. At 5% error, number of clusters produced by MrMC-MinH^{h} (47), MrMC-MinH^{g} (37), MC-LSH (41) and CD-HIT (47) are closer to ground truth. We also report the weighted pairwise similarity within the clusters, and observe that MrMC-MinH^{h} shows promising weighted similarity results with less number of clusters. Different number of clusters with similar weighted sequence similarity is due to the presence of single sequence clusters which are not included in calculating weighted sequence similarity.

2) 16S Environmental dataset:: In Table V, we report the results of several clustering methods on 16S environmental samples. We can observe that MrMC-MinH^{h} outperforms other methods by producing similar weighted similarity (W.Sim) with less number of clusters. For example, Mothur and DOTUR use the same all pairwise similarity in their

Table III
CLUSTERING PERFORMANCE ON SIMULATED AND REAL WHOLE METAGENOME READS.

	MrMC-MinH^h Agglomerative Hie. Clustering				MrMC-MinH^g Greedy Clustering				MetaCluster			
SID	# Cluster	W.Acc	W.Sim	Time	# Cluster	W.Acc	W.Sim	Time	# Cluster	W.Acc	W.Sim	Time
S1	12	**90.42**	**59.20**	4m 25s	7	86.98	55.14	**2m 35s**	5	87.46	50.92	11m 25s
S2	10	**88.05**	**57.35**	4m 18s	6	85.70	54.05	**2m 32s**	7	85.19	50.20	12m 10s
S3	11	**85.16**	**60.08**	4m 20s	8	81.27	55.36	**2m 35s**	7	80.84	51.85	12m 48s
S4	8	**93.72**	**61.44**	4m 15s	3	90.16	57.40	**2m 08s**	3	92.34	54.30	11m 33s
S5	9	**84.10**	**58.60**	4m 17s	4	81.25	53.71	**2m 10s**	5	79.69	50.71	12m 18s
S6	8	**98.64**	**57.40**	4m 15s	7	98.14	56.92	**2m 25s**	8	99.12	56.67	13m 17s
S7	5	**97.50**	**58.17**	4m 13s	3	97.08	56.60	**2m 04s**	4	98.44	55.18	12m 51s
S8	13	**96.22**	**59.25**	4m 30s	6	93.65	55.78	**2m 25s**	5	92.29	53.05	13m 04s
S9	6	**89.45**	**60.32**	4m 15s	4	86.20	56.62	**2m 04s**	4	85.20	50.41	12m 28s
S10	7	**98.06**	**57.78**	4m 16s	5	97.44	56.90	**2m 16s**	7	98.63	54.66	13m 08s
S11	9	**91.88**	**58.23**	8m 35s	7	88.52	53.10	**4m 15s**	9	87.37	51.10	30m 25s
S12	15	**97.54**	**60.56**	8m 44s	10	95.32	57.45	**4m 25**	12	96.47	53.94	32m 10s
R1	7	-	**59.18**	2m 30s	4	-	55.80	**1m 08s**	3	-	50.45	6m 10s

All experiments are carried out with 5 k-mer and 100 hash functions. Number of clusters (# Cluster), weighted cluster accuracy in % (W.Acc), weighted sequence similarity (W.Sim) in % and Running Time in minutes or seconds (Time) are the performance metrics. The numbers in bold indicate that MrMC-MinH^h algorithm performs better than MrMC-MinH^g and MetaCluster. R1 represents the real metagenome sample (7137 sequences). S1-S12 represent simulated metagenome samples (each sample S1-S9 contains about 50,000 sequences of 1000 basepairs whereas each sample S11-12 contains about 100,000 sequences of 1000 basepairs). Clustering results are trimmed after applying threshold on number of clusters.

Table IV
CLUSTERING RESULTS ON 16S SIMULATED DATASET

	Dataset with 3% error		Dataset with 5% error	
Method	# Cluster	W.Sim	# Cluster	W.Sim
MrMC-MinH^h	53	97.80	**47**	**95.46**
MrMC-MinH^g	39	97.70	**37**	95.18
MC-LSH	47	97.70	**41**	95.20
UCLUST	91	97.74	53	95.20
CD-HIT	108	97.74	**47**	95.20
ESPRIT	180	97.78	86	95.20
DOTUR	210	97.80	135	95.46
Mothur	214	97.80	138	95.46

These datasets contain sequence reads upto 3% and 5% errors with respect to reference 16S rRNA sequences. Number of clusters (# Cluster) and weighted sequence similarity (W.Sim) in % are the performance metrics. The numbers in bold indicate that result produced by an algorithm is closer to the ground truth and performs better than other algorithms.

computation and produce large number of clusters as compared to MrMC-MinH^h.

3) Whole Metagenome Reads:: Table III compares the clustering results for MrMC-MinH^h, MrMC-MinH^g and MetaCluster on simulated and real whole metagenome sequence read samples. For datasets S1-S12, we have ground truth labels to compute the clustering accuracy based on species distribution. We see that MrMC-MinH^h outperforms both MrMC-MinH^g and MetaCluster in weighted sequence similarity across all samples. We also observe that MrMC-MinH^h performs better in terms of weighted cluster accuracy. For real metagenome sample R1, MrMC-MinH^h outperforms MrMC-MinH^g and MetaCluster respect to weighted sequence similarity. There was no ground truth available for this sample to compute the clustering accuracy metric.

VI. CONCLUSION

In this work, we present an efficient clustering algorithm, MrMC-MinH for metagenome analysis. Our algorithm uses the minwise hashing approach to approximate the computation of pairwise similarity and incorporates an agglomerative hierarchical clustering method to group metagenome sequences. MrMC-MinH is developed on the widely available Hadoop platform and uses Pig scripting language to create Map-Reduce framework for distributed computation. It can easily handle large datasets that are currently produced by the second and third generation sequencing technologies for metagenome studies. We evaluate MrMC-MinH on both, the 16S and whole metagenome datasets and demonstrate that MrMC-MinH is computationally efficient and accurate in comparison to state-of-the-art clustering algorithms. The source code along with benchmarks is available at the supplementary website.

ACKNOWLEDGMENT

Support by NSF Career to HR, IIS 1252318.

REFERENCES

[1] S.R. Gill, M. Pop, R.T. DeBoy, P.B. Eckburg, P.J. Turnbaugh, B.S. Samuel, J.I. Gordon, D.A. Relman, C.M. Fraser-Liggett,

Table V
CLUSTERING RESULTS ON 16S ENVIRONMENTAL SETS.

Approach	Metric	53R	55R	112R	115R	137	138	F312(t)	F396(t)
MrMC-MinH^h	# Cluster	1180	1205	1694	1217	1020	1054	1983	1360
	W.Sim	**96.95**	**94.06**	**91.33**	**93.50**	**95.86**	**93.10**	**91.25**	**91.40**
	Time (s)	8.4	8.9	9.5	9.7	8.8	8.5	5.4	5.4
MrMC-MinH^g	# Cluster	1165	1077	1634	1156	1020	1042	1965	1340
	W.Sim	96.90	92.45	91.18	93.33	95.86	93.10	91.25	91.16
	Time (s)	2.5	2.1	3.1	3.1	2.0	2.3	2.1	2.1
MC-LSH	# Cluster	1172	1199	1795	1205	1041	1072	1965	1363
	W.Sim	96.90	93.12	91.33	93.50	95.86	93.10	91.25	91.40
	Time (s)	161.0	183.0	317.0	188.0	172.0	175.0	112.0	101.0
UCLUST	# Cluster	1062	992	1561	1071	900	923	1965	1346
	W.Sim	96.67	91.67	91.02	93.33	93.50	92.82	91.25	91.16
	Time (s)	2.6	2.2	3.1	3.3	2.2	2.5	2.0	2.0
CD-HIT	# Cluster	824	716	1196	820	712	725	1626	945
	W.Sim	92.56	90.80	90.61	93.33	91.82	90.16	89.70	88.45
	Time (s)	3.6	3.1	3.9	3.8	3.2	3.1	3.8	2.5
ESPRIT	# Cluster	940	859	1361	970	818	832	1936	1280
	W.Sim	93.12	91.35	90.88	93.33	91.82	90.16	91.20	89.50
	Time (s)	283.0	266.0	537.0	348.0	280.0	296.0	205.0	192.0
DOTUR	# Cluster	1241	1258	1854	1279	1096	1121	2012	1381
	W.Sim	96.95	94.06	91.33	93.50	95.86	93.10	91.25	91.40
	Time (s)	5129.0	3511.0	5567.0	9237.0	6563.0	5618.0	1990.0	1457.0
Mothur	# Cluster	1238	1256	1853	1278	1094	1119	2008	1372
	W.Sim	96.95	94.06	91.33	93.50	95.86	93.10	91.25	91.40
	Time (s)	10130.0	5940.0	12303.0	13501.0	12861.0	12310.0	2224.0	1583.0

For MrMC-MinH^h and MrMC-MinH^g, all experiments are carried out with 15 k-mer and 50 hash functions. Number of clusters (# Cluster), Weighted Similarity in % (W.Sim) and Running Time in seconds (Time) are the performance metrics. The numbers in bold indicate that an algorithm performs better than other methods. Similarity threshold of 95% is used for all the methods and Weighted Similarity is calculated separately to keep it consistent for all methods. F312(t) and F396(t) are trimmed down to smaller number of sequences because DOTUR and Mothur were unable to run on original dataset. MrMC-MinH^h and MrMC-MinH^g work for both original and trimmed datasets.

and K.E. Nelson. Metagenomic analysis of the human distal gut microbiome. *science*, 312(5778):1355–1359, 2006.

[2] H.N. Poinar, C. Schwarz, J. Qi, B. Shapiro, R.D.E. MacPhee, B. Buigues, A. Tikhonov, D.H. Huson, L.P. Tomsho, A. Auch, et al. Metagenomics to paleogenomics: large-scale sequencing of mammoth dna. *science*, 311(5759):392–394, 2006.

[3] J.C. Venter, K. Remington, J.F. Heidelberg, A.L. Halpern, D. Rusch, J.A. Eisen, D. Wu, I. Paulsen, K.E. Nelson, W. Nelson, et al. Environmental genome shotgun sequencing of the sargasso sea. *science*, 304(5667):66–74, 2004.

[4] P. Hugenholtz and G.W. Tyson. Microbiology: metagenomics. *Nature*, 455(7212):481–483, 2008.

[5] A. Charuvaka and H. Rangwala. Evaluation of short read metagenomic assembly. *BMC genomics*, 12(Suppl 2):S8, 2011.

[6] J.F. Petrosino, S. Highlander, R.A. Luna, R.A. Gibbs, and J. Versalovic. Metagenomic pyrosequencing and microbial identification. *Clinical chemistry*, 55(5):856–866, 2009.

[7] Zeehasham Rasheed and Huzefa Rangwala. Mc-minh: Metagenome clustering using minwise based hashing. In *SIAM International Conference in Data Mining*, Austin, TX, 05 2013. SIAM.

[8] A.Z. Broder, M. Charikar, A.M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336. ACM, 1998.

[9] M.L. Sogin, H.G. Morrison, J.A. Huber, D.M. Welch, S.M. Huse, P.R. Neal, J.M. Arrieta, and G.J. Herndl. Microbial diversity in the deep sea and the underexplored rare biosphere. *Proceedings of the National Academy of Sciences*, 103(32):12115–12120, 2006.

[10] S. Chatterji, I. Yamazaki, Z. Bai, and J. Eisen. Compostbin: A dna composition-based algorithm for binning environmental shotgun reads. In *Research in Computational Molecular Biology*, pages 17–28. Springer, 2008.

[11] M. Margulies, T.P. Jarvie, J.R. Knight, and J.F. Simons. The 454 life sciences picoliter sequencing system. *Perspectives in Bioanalysis*, 2:153–186, 2007.

[12] W. Li and A. Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[13] R.C. Edgar. Search and clustering orders of magnitude faster than blast. *Bioinformatics*, 26(19):2460–2461, 2010.

[14] P.D. Schloss and J. Handelsman. Introducing dotur, a computer program for defining operational taxonomic units and estimating species richness. *Applied and environmental microbiology*, 71(3):1501–1506, 2005.

[15] P.D. Schloss, S.L. Westcott, T. Ryabin, J.R. Hall, M. Hartmann, E.B. Hollister, R.A. Lesniewski, B.B. Oakley, D.H. Parks, C.J. Robinson, et al. Introducing mothur: open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Applied and environmental microbiology*, 75(23):7537–7541, 2009.

[16] Y. Sun, Y. Cai, L. Liu, F. Yu, M.L. Farrell, W. McKendree, and W. Farmerie. Esprit: estimating species richness using large collections of 16s rrna pyrosequences. *Nucleic Acids Research*, 37(10):e76–e76, 2009.

[17] Zeehasham Rasheed, Huzefa Rangwala, and Daniel Barbara. LSH-Div:species diversity estimation using locality sensitive hashing. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Philadelphia, USA, 10 2012. IEEE.

[18] Zeehasham Rasheed, Huzefa Rangwala, and Daniel Barbara. Efficient clustering of metagenomic sequences using locality sensitive hashing. In *SIAM International Conference in Data Mining*, pages 1023–1034, Anaheim, CA, 04 2012. SIAM.

[19] B. Yang, Y. Peng, H. Leung, SM Yiu, J. Qin, R. Li, and F.Y.L. Chin. Metacluster: unsupervised binning of environmental genomic fragments and taxonomic annotation. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, pages 170–179. ACM, 2010.

[20] M. Cameron, Y. Bernstein, and H.E. Williams. Clustered sequence representation for fast homology search. *Journal of Computational Biology*, 14(5):594–614, 2007.

[21] A. Kalyanaraman, S. Aluru, S. Kothari, and V. Brendel. Efficient clustering of large est data sets on parallel computers. *Nucleic Acids Research*, 31(11):2963–2974, 2003.

[22] J. Zola, X. Yang, S. Rospondek, and S. Aluru. Parallel t-coffee: A parallel multiple sequence aligner. *Proceedings of ISCA PDCS-2007*, pages 248–253, 2007.

[23] X. Yang, J. Zola, and S. Aluru. Parallel metagenomic sequence clustering via sketching and maximal quasi-clique enumeration on map-reduce clouds. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1223–1233. IEEE, 2011.

[24] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[25] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 20(9):1297–1303, 2010.

[26] M.C. Schatz, B. Langmead, and S.L. Salzberg. Cloud computing and the dna data race. *Nature biotechnology*, 28(7):691, 2010.

[27] D.P. Wall, P. Kudtarkar, V.A. Fusaro, R. Pivovarov, P. Patil, and P.J. Tonellato. Cloud computing for comparative genomics. *BMC bioinformatics*, 11(1):259, 2010.

[28] P. Li, A. Shrivastava, and C.A. Konig. Gpu-based minwise hashing: Gpu-based minwise hashing. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 565–566. ACM, 2012.

[29] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

[30] S.M. Huse, J.A. Huber, H.G. Morrison, M.L. Sogin, D.M. Welch, et al. Accuracy and quality of massively parallel dna pyrosequencing. *Genome Biol*, 8(7):R143, 2007.

[31] M. Margulies, M. Egholm, W.E. Altman, S. Attiya, J.S. Bader, L.A. Bemben, J. Berka, M.S. Braverman, Y.J. Chen, Z. Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

[32] X. Huang. On global sequence alignment. *Computer applications in the biosciences: CABIOS*, 10(3):227–235, 1994.