Lawrence Berkeley National Laboratory
Recent Work

Title
SpaRC: scalable sequence clustering using Apache Spark.

Permalink
https://escholarship.org/uc/item/7dn7m5rg

Journal
Bioinformatics (Oxford, England), 35(5)

ISSN
1367-4803

Authors
Shi, Lizhen
Meng, Xiandong
Tseng, Elizabeth
et al.

Publication Date
2019-03-01

DOI
10.1093/bioinformatics/bty733

Peer reviewed

OXFORD

Sequence analysis

SpaRC: scalable sequence clustering using Apache Spark

Lizhen Shi[1], Xiandong Meng[2,3], Elizabeth Tseng[4], Michael Mascagni[1] and Zhong Wang 🆔 [2,3,5,*]

[1]Department of Computer Science, School of Computer Science, Florida State University, Tallahassee, FL 32304, USA, [2]US Department of Energy, Joint Genome Institute, Walnut Creek, CA 94598, USA, [3]Environmental Genomics and Systems Biology Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA, [4]Pacific Biosciences Inc, Menlo Park, CA 94025, USA and [5]School of Natural Sciences, University of California at Merced, Merced, CA 95343, USA

*To whom correspondence should be addressed.
Associate Editor: Inanc Birol

Abstract

Motivation: Whole genome shotgun based next-generation transcriptomics and metagenomics studies often generate 100–1000 GB sequence data derived from tens of thousands of different genes or microbial species. Assembly of these data sets requires tradeoffs between scalability and accuracy. Current assembly methods optimized for scalability often sacrifice accuracy and *vice versa*. An ideal solution would both scale and produce optimal accuracy for individual genes or genomes.

Results: Here we describe an Apache Spark-based scalable sequence clustering application, S̲parkR̲eadC̲lust (SpaRC), that partitions reads based on their molecule of origin to enable downstream assembly optimization. SpaRC produces high clustering performance on transcriptomes and metagenomes from both short and long read sequencing technologies. It achieves near-linear scalability with input data size and number of compute nodes. SpaRC can run on both cloud computing and HPC environments without modification while delivering similar performance. Our results demonstrate that SpaRC provides a scalable solution for clustering billions of reads from next-generation sequencing experiments, and Apache Spark represents a cost-effective solution with rapid development/deployment cycles for similar large-scale sequence data analysis problems.

Availability and implementation: https://bitbucket.org/berkeleylab/jgi-sparc

Contact: zhongwang@lbl.gov

1 Introduction

Whole genome shotgun sequencing (WGS) using next-generation sequencing (NGS) technologies followed by *de novo* assembly is a powerful tool for large eukaryote transcriptome analysis [reviewed in (Martin and Wang, 2011)] and analysis of complex microbial communities [reviewed in (Tringe and Rubin, 2005)] without reference genomes. Because of the stochastic sampling associated with WGS and the presence of sequencing errors, it is necessary for the sequenced reads to cover a single gene or a genome many times

(coverage), typically at 30× to 50×, to ensure high quality *de novo* assemblies (Ajay *et al.*, 2011). Unlike in single genome sequencing projects where the majority of the genomic regions are equally represented, in transcriptome and metagenome sequencing projects, different species of transcripts or genomes may have very unequal representation, up to several orders of magnitude (Hughes *et al.*, 2001; Martin *et al.*, 2014). To obtain a good assembly covering low abundance species, much higher sequencing depth is required than in isolate genome projects. As in practice it is difficult to precisely
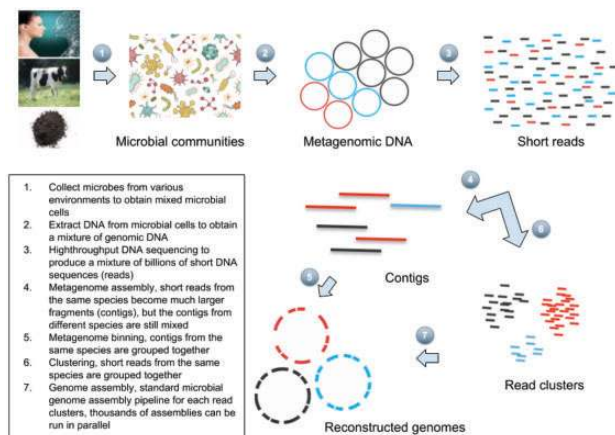
Fig. 1. Illustration of a typical metagenome workflow

estimate the required sequencing depth without knowing the community structure, sequencing large transcriptomes and complex metagenomes often generates as much data as the budget allows, producing 100–1000 GB of sequence data or more (Howe *et al.*, 2014; Shi *et al.*, 2014). The largest project so far is the Tara Ocean Metagenomics project where 7.2 Tb was generated (Sunagawa *et al.*, 2015).

Since current NGS technologies are not able to read the entire sequence of a genome at once, genomes are broken into small DNA/RNA fragments followed by massive parallel high-throughput sequencing. Figure 1 illustrated a typical work flow to predict microbial genomes from environment samples. Different technologies produce sequence reads that vary in length. For example, Illumina (https://www.illumina.com) technology typically generates about 150 bp per read, while Pacific Biosciences (http://www.pacb.com) reads are 100–60 000 bp. Transcriptome/metagenome assembly from these short fragments is a compute and memory-intensive problem. The problem is further complicated by several factors such as sequencing errors, repetitive elements, and homologous genes shared among related species. For a comprehensive review of *de novo* assembly algorithms please refer to Miller *et al.* (2010).

Assembling these datasets as a whole requires efficient and scalable algorithms, and to achieve this current assemblers use either multiple processes on a shared memory architecture [MetaSPAdes (Nurk *et al.*, 2017), MEGAHIT (Li *et al.*, 2015), etc] or MPI to distribute jobs on a large cluster (Georganas *et al.*, 2015). The shared memory approach is very hard to scale up with exponentially increased NGS data size. In addition, these assemblers try to tackle the problem as a whole and are not able to produce optimized results as different transcripts or genomes may need individualized optimal parameter settings.

Our work was initially inspired by a 'divide-and-conquer' approach presented by DIME (Guo *et al.*, 2015). DIME first clusters reads based on their overlap, then assembles them separately. It was implemented using Apache Hadoop (http://hadoop.apache.org) and, in theory, should scale to large data sets. In practice, however, Hadoop-based implementation has very poor computing efficiency, making it expensive to run on commercial cloud providers such as Amazon Web Services. Further, much larger intermediate files are often generated during the assembly, making it harder to scale.

Apache Spark (http://spark.apache.org) is the most active open big data framework and has overtaken Hadoop in the big data ecosystem due to its fast in-memory computation. Spark has been successfully applied to several genomics problems such as (Bahmani

et al., 2016; de Castro *et al.*, 2017; Klein *et al.*, 2017; Massie *et al.*, 2013; Xu *et al.*, 2016). In this paper we developed a new algorithm called Spark Read Clustering (SpaRC), a generic NGS read clustering algorithm that parallel constructs a read graph and subsequently partitions it. In order to achieve both scalability and accuracy on large datasets we implemented SpaRC using a combination of several tricks: (i) estimation of pairwise sequence similarity (edge weight) by the number of shared k-mers; (ii) down-sampling to control the explosion of data caused by abundant species and noise as data size and complexity grow; (iii) application of a fast overlapping community detection algorithm, Label Propagation Algorithm (LPA) (Raghavan *et al.*, 2007), to efficiently partition the read graph and break partitions containing a mixture of different species due to shared genetic elements. We report clustering accuracy and computing performance on both transcriptomic and metagenomic read clustering with both short read (Illumina) and long read (PacBio) sequencing platforms.

2 Materials and methods

2.1 Algorithm overview
SpaRC is capable of clustering both short and long reads with different parameter settings. It first computes the number of shared k-mers between a pair of reads to approximate their overlap, and then builds an undirected read graph followed by graph partitioning to form clusters. Specifically, it contains four modules: K-mer mapping reads, graph construction and edge reduction, graph partition and sequence retrieval. We describe each of these modules in detail as follows.

2.2 K-mer mapping reads (KMR)
Given a set of sequence reads, K-mer mapping reads (KMR) splits them into k-mers according to a pre-defined k-mer length and only keeps distinct k-mers for each read. KMR keeps track each k-mer and the reads containing it. The length of k-mer (k) is a parameter to control the sensitivity and specificity of read overlap detection. Shorter k-mers result in more sensitivity but less specificity and vice versa. The ideal k-mer size depends on the sequence platform, the read depth and sequence complexity.

In general, k-mers appearing in only a single read are derived from either sequencing errors or rare molecules. While they represent a large fraction of the total k-mers, they are not useful for computing read overlap, therefore they are filtered out. k-mers with a very high count may derive from very abundant species or repetitive elements. Because they can greatly increase the number of edges (see blow) while contributing to more noise, here we manually cap k-mer count to control the amount of computation and noise derived from repetitive elements. Clustering very abundant species should not be affected as long as the number is sufficiently large (200 by default). KMR allows users to specify customized filtering criteria (*min_kmer_count* and *max_kmer_count*) to adjust clustering sensitivity and specificity.

2.3 Graph construction and edge reduction (GCER)
Graph construction and edge reduction (GCER) constructs a read graph in which a node is a read and an edge links two nodes if they share k-mers. As discussed above, some nodes, if derived from repetitive elements or genes conserved among species or contamination, can have extremely high edge count (degree). Besides the *max_kmer_count* to reduce the total number of edges produced, here we also set the maximum degrees of any vertex in a graph

(*max_degree*) as a parameter to further reduce the amount of computation during the graph partition step (below).

After all the vertices and edges are generated, GCER merges edges having the same source and destination, and filters out edges having fewer shared k-mers than specified by parameter *min_shared_kmers* as these are likely caused by noise.

2.4 Graph partition (LPA)

This step partitions the above read graph into clusters. Since repetitive elements and homologous genetic elements are shared between different molecules/genomes thus creates 'overlap communities', we implemented LPA for its capability of overlapping community detection and near-linear execute time (Raghavan *et al.*, 2007). SpaRC also provides another iterative graph partition algorithm, connected components (CCs), it might be useful where the overall sequencing depth is very low, or there are very few repetitive genetic contents.

2.5 Sequence retrieval (AddSeq)

In the above modules reads are represented by numeric IDs to save memory and storage. Once the clusters are formed, AddSeq retrieves the sequences and format them for downstream parallel assembly process.

2.6 Algorithms

1 For each read r in the read set R:
2   Generate distinct k-mer-read tuples
3
4 Group the tuples by k-mer and generate k-mer-reads pairs (KR)
5 Filter KR by only keeping pairs overlapping between *min_kmer_count* and *max_kmer_count*
6
7 For each list of read in KR:
8   Generate pairwise edges (reads as nodes)
9
10 For each node in the edges:
11   If the node degree > *max_degree*, sample *max_degree* edges
12
13 Count distinct edges and generate edge-count pairs (EC)
14 Filter EC to only keep pairs whose count is more than *min_shared_kmers*
15 Generate graph g_0 with the edges in EC
16
17 If clustering algorithm $\bar{A}$ is CC:
18   Generate the CCs of g_0.
19   For each CC, add the connected
20   Component to the set of read clusters Ω.
21 else if $\bar{A}$ is LPA:
22   Run label propagation step for m iterations
23   Group the nodes (reads) by its labels
24   For each reads group, add the group to Ω

2.7 Hardware and software environment

SpaRC was implemented in Scala (Scala 2.11.8). To assess its adaptivity in different hardware and software environments, we conducted experiments on two cloud clusters, 20 nodes on Open Telekom Cloud (https://cloud.telekom.de/en/infrastructure/open-telekom-cloud) and Amazon's Elastic MapReduce (EMR, emr-5.9.0) and one 8 nodes HPC cluster at the Pittsburgh Supercomputing Center (Nystrom *et al.*, 2015). On all three clusters, one node is used as the master and all other nodes are used as workers. Configuration details are shown in Table 1.

Table 1. Configuration for OTC, AWS EMR and Bridges

	OTC	AWS EMR	Bridges
# of cores/node	8	8	28
Memory/node	64	61	128
Storage/node	500 GB SSD	160 GB SSD	8TB HDD
Ethernet	1 Gbps	10 Gbps	Omni Path
Spark version	2.1.1	2.2.0	2.1.0
Hadoop version	2.7.3	2.7.3	2.7.2
Cluster mode	Standalone	YARN	YARN
# of executors/node	3	3	4
Driver memory	55 GB	40 GB	55 GB
Driver cores	5	5	5
Memory/executor	18 GB	16 GB	27 GB
Cores/executor	2	2	3
HDFS Block size	32 MB	32 MB	32 MB

2.8 Datasets

To systematically test the scalability and accuracy of SpaRC we prepared several real world test datasets (Table 2). A maize sequence dataset, we generated previously from Martin *et al.* (2014), and the Cow Rumen metagenome dataset (Hess *et al.*, 2011), from which we generated subsets of 1–100 GB in fastq, for testing scalability. Two simulated metagenome datasets, including a mock dataset containing 26 genomes and the CAMI2 simulated human microbiome datasets (https://data.cami-challenge.org/participate), are used to verify accuracy. Three long read transcriptome datasets were provided by PacBio. The datasets are described in details in the Results section, here we simply summarize their metrics.

3 Results

3.1 SpaRC clustering accuracy

In order to measure the clustering accuracy of SpaRC, we used two sets of data with 'known answers' and ran SpaRC to obtain clusters.

The first dataset is a Human Alzheimer whole brain transcriptome sequenced by PacBio, consisting of 1 107 889 full-length transcript sequences. Transcript sequences were clustered using an isoform-level clustering algorithm (Gordon *et al.*, 2015), then consensus sequences were mapped to the human genome to identify source loci. Reads originating from clusters in which the mapped genomic locations overlap by at least 1 bp are considered to be from the same loci. This is the theoretical limit for overlap-based clustering algorithms. The second dataset is two million Illumina short metagenome reads (150 bp) sampled from a mock microbial community consisting of 26 genomes described previously (Singer *et al.*, 2016). Clusters are defined similarly as above for the PacBio transcriptome dataset.

By comparing the SpaRC clusters to 'known answers' in the above two datasets, we measured SpaRC's cluster purity and completeness. Here cluster purity is defined as the percentage of reads belonging to the dominant known cluster for each SpaRC cluster, and completeness is defined as the maximum percentage of reads from a known cluster captured by a SpaRC cluster. It is worth noting that measured cluster completeness will underestimate true cluster completeness as the 'known answers' are overestimates as described above. As the sensitivity of overlap detection is heavily influenced by the read length, in the Illumina metagenome dataset we joined the reads in a pair that is pair-end sequenced to double the read length for clustering.

Table 2. Metrics of test datasets used in this work

Dataset	# Species	Read length (bp)	Size (GB)
PacBio1 (Human Alzheimer brain transcriptome)	High	300–30 816	3.8
PacBio2 (Human UHRR + synthetic RNA, 2Cell)	High	62–14 621	1
PacBio3 (Human UHRR + synthetic RNA, 3Cell)	High	54–14 833	1.8
Maize transcriptome	High	151×2	4
Mock metagenome	Low (26)	$(90–150) \times 2$	15
Cow Rumen metagenome	Medium	100×2	100
CAMI2 simulated metagenome	High	150×2	100

In both experiments SpaRC clustered the majority of the reads (PacBio: 82.65%, Ilumina: 98.3%), and generated very pure clusters (Fig. 2A and E). For the impure read clusters in both datasets, contamination appears to be relatively low, as purity increases with cluster size (Fig. 2B and F).

Clustering long reads achieved much higher completeness than clustering short reads, with many more clusters that have completeness $\geq 90\%$ (84.88%, $n = 9578$, Fig. 2C), comparing to short read clusters (37.19%, $n = 37\,879$, Fig. 2G). For the transcriptome dataset, the completeness improves as cluster size increases, suggesting more copies of a transcript increases the chances of finding overlap k-mers. For the metagenome dataset, as the genome copy number is fixed in that dataset, larger clusters translate into larger regions, and they are more easily broken into smaller clusters presumably uneven coverage. Therefore, the overall completeness drops as cluster size increases.

We also tested whether or not completeness is worse if the read pairs in the short read dataset are not joined. This indeed is the case, as clusters that have completeness $\geq 90\%$ is decreased to 6.08% and more small clusters are produced ($n = 42\,181$).

3.2 Accuracy comparison with alternative solutions

3.2.1 Long reads
To assess whether using SpaRC improves recovery of known synthetic spike-in transcripts (synthetic SIRV transcripts) in the PacBio human data, clustering results from SpaRC were compared with minimap-based (Li, 2016) clustering results and run through the PacBio Iso-Seq clustering pipeline (https://github.com/PacificBiosciences/IsoSeq_SA3nUP). The results (Table 3) from SpaRC show comparable results with slightly improved recovery of the synthetic spike-in transcripts (more true positives) and slightly reduced artifacts (less false positives). The difference seems to be more pronounced when sequence depth is lower (PacBio2).

3.2.2 Short reads
For accuracy comparison on short reads we used a synthetic metagenome dataset derived from the Critical Assessment of Metagenome Interpretation (CAMI) project, the first-ever community-organized benchmark for evaluating computational tools for metagenomes (Sczyrba et al., 2017). CAMI2 consists of 49 human microbiome samples from gastrointestinal tract, oral cavity, airways, skin and urogenital tract. We selected a subset of the simulated Illumina datasets (sample no. 2–9, 14), and the total size of these 9 samples is 100 GB in Fastq format.

We compared SpaRC with several alternative clustering tools, including MC-MinH (Rasheed and Rangwala, 2013), MetaCluster (Wang et al., 2012), bwtCluster (Alanko et al., 2017) and Eigengenomes (LSA) (Cleary et al., 2015), for clustering accuracy on the above CAMI2 dataset. MC-MinH is designed to cluster 16S

ribosomal sequences. The software was implemented as a single threaded program with quadratic complexity. MetaCluster 5.0 uses a two-round approach to cluster metagenomic reads. It first separates the reads into high abundance and low abundance groups, then it uses different k-mers to cluster each group. BwtCluster is a space-efficient clustering algorithm. It first uses connect component to cluster the reads into small clusters and then uses k-means to further cluster them into bigger ones. LSA is a scalable partitioning approach that clusters raw reads based on abundance covariation among many metagenome samples.

We were not able to run MC-MinH and MetaCluster because the two solutions are not scalable to the 100 GB dataset. For a fair comparison with the bwtCluster, we only ran the first round (precluster) because a reasonable number of cluster has to be set for the k-means in the second round. LSA requires a threshold parameter (0–1) to determine the number of clusters formed. We tuned this parameter as recommended and found that 0.95 gave the best result for this dataset.

Table 4 shows the comparison result for clustering the CAMI2 dataset. SpaRC and bwtCluster form many small clusters with relatively high purity and low completeness. In contrast, LSA produces very few clusters with high completeness but very low purity. Comparing with bwtCluster, SpaRC clusters more reads, produces many more clusters, achieves higher cluster purity but suffers from lower completeness. If the goal is to pre-cluster metagenome short reads, SpaRC results seem to be more desirable as higher purity means few errors will be carried over to the next step.

BwtCluster was implemented in C++ for speed and also showed efficient memory requirement, about $3\times$ of the input data size. However, the traversal of the suffix-link tree in bwtCluster was not capable of scaling well with more than four cores as mentioned in the paper (Alanko et al., 2017), and it is a monolithic program and cannot utilize multiple nodes, so its scalability should be limited. In contrast, SpaRC as a distributed program designed to scale up to datasets of terabytes or more, requires much more resources for its scalability and fault tolerance.

3.3 Data complexity has a major effect on SpaRC execution time
As introduced above, SpaRC consists of four steps: KMR, GCER, LPA and AddSeq. To measure the computing efficiency of each step on data with different complexity (number of species, see Table 2), we ran SpaRC against Human Alzheimer transcriptome, Maize transcriptome and Cow Rumen metagenome with the same data size (4 GB) on OTC computing environment.

We found different datasets gave rise to very different overall execution time (Fig. 3), with PacBio transcriptome being the longest, about twice as long as Illumina transcriptome, and three times as long as Illumina metagenome. Execution time for each of the four
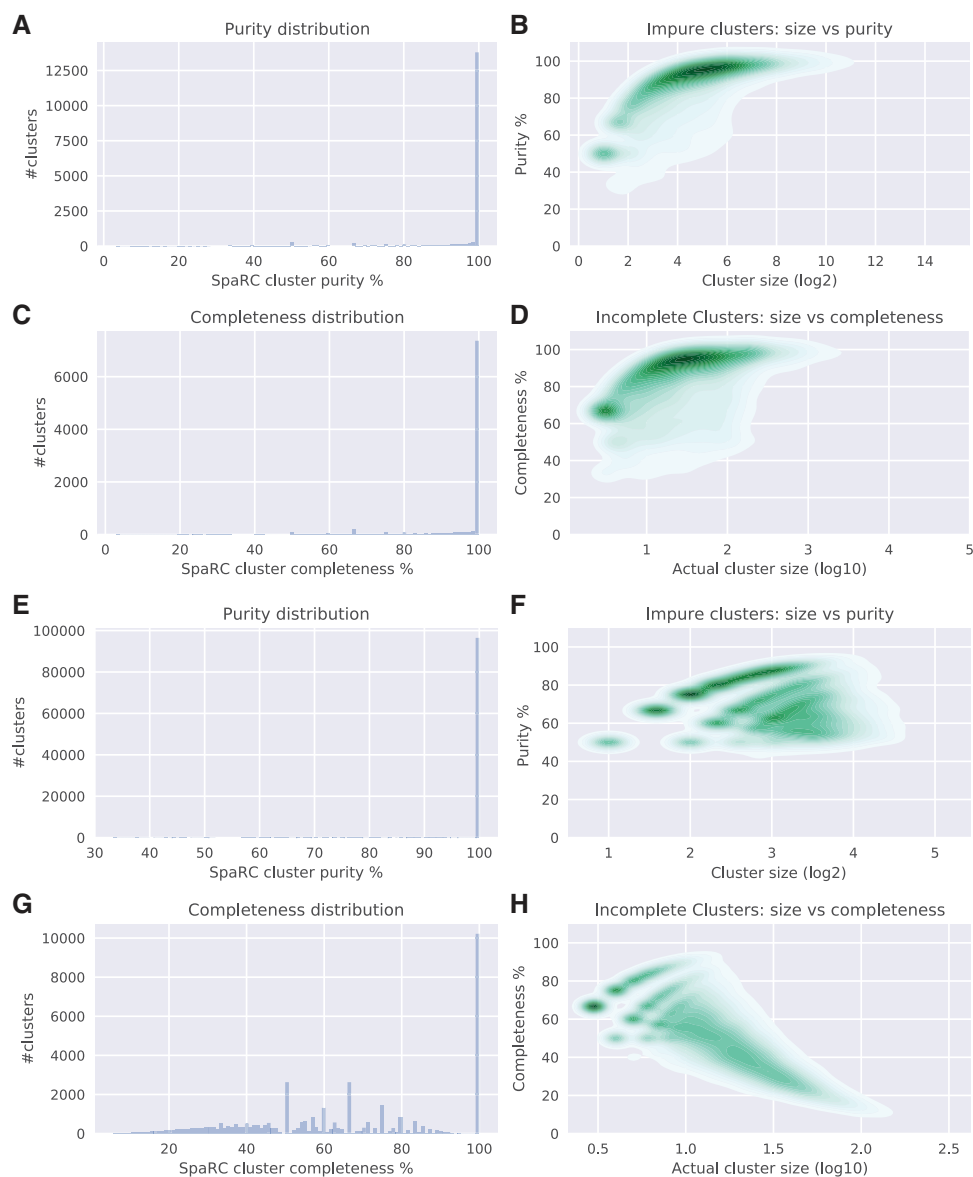
Fig. 2. SpaRC's clustering performance on long and short reads: (**A–D**) SpaRC clustering results on a PacBio transcriptome dataset from a Human Alzheimer whole brain sample and (**E–H**) on a Illumina metagenome dataset sequenced from a mock microbial community consisting of 26 genomes. Cluster purity is measured as the percentage of reads belonging to the dominant known cluster for each SpaRC cluster, and completeness as the maximum percentage of reads from a known cluster captured by a SpaRC cluster. A, E show the purity distribution of SpaRC clusters for PacBio (A) and Illumina (E) reads, respectively. C, G show the completeness distribution of the SpaRC clusters for PacBio (C) and Illumina (G) reads, respectively. B, F show the purity (*Y-axis*) versus the cluster size (*X-axis*) for PacBio (B) and Illumina (F), respectively. Only the impure clusters are shown; D, H show the completeness (*Y-axis*) versus the cluster size (*X-axis*) for PacBio (D) and Illumina (H), respectively. Only the incomplete clusters are shown

modules on different datasets shows very different behavior. First of all, as the datasets have the same size, they contain similar number of raw k-mers, therefore the KMR step takes about similar time. Second, reads from the complex metagenome dataset typically have fewer edges than transcriptomes because many species do not have sufficient coverage. Longer reads tend to produce more edges because they have more overlapping k-mers (Table 5). Finally, even given comparable number of total edges (Table 5), LPA takes significantly longer for the long read transcriptome dataset than for the short read transcriptome dataset. This is because each long read vertex has more edges than short read, and GraphX's implementation uses vertex-cut for graph partition (Xin *et al.*, 2013), resulting more copies of vertices, which in turn translates into higher time cost on each reduction in each LPA iteration.

Among the steps in the workflow, AddSeq is the simplest step and takes very little time (no more than 1 min) for all datasets.

3.4 Degree of parallelism on SpaRC's computing performance

Resilient distributed datasets (Zaharia *et al.*, 2012) are Spark's low-level, fault-tolerant data structure. They are partitioned and distributed across different nodes due to its huge size. Spark runs one task per partition, and the total number of partitions defines the parallelism. It has been reported parallelism level has a major effect on the performance of the Spark applications (Abu-Doleh and Çatalyürek, 2015). Therefore, we evaluated the effect of parallelism level on the overall execution time of the current SpaRC software.

**Table 3.** Isoform detection performance comparison between SpaRC and Minimap clustering

Dataset	SpaRC		Minimap	
	TP[a]	FP[b]	TP	FP
PacBio2 (375 k reads)	57	13	54	17
PacBio3 (623 k reads)	61	11	61	14

[a]TP: true positive, transcripts derived from the loci.
[b]FP: false positive, transcripts that do not belong to the loci.

Table 4. Clustering accuracy comparison on a 100 GB simulated metagenome dataset

Tools	#clusters	#reads	% of pure cluster	Median purity	Median compl
LSA	619	213 447 604	0.16	33.42	85.49
bwtCluster	218 154	285 947 805	47.33	99.50	25.48
SpaRC	1 347 826	296 027 232	69.48	100.00	11.62
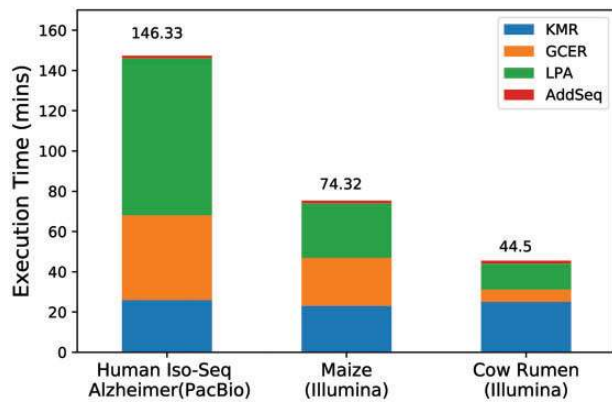
Fig. 3. Execution time difference between datasets: the execution time (*Y-axis*) of each step on three sets of data (*X-axis*), from bottom to top: KMR, GCER, LPA and KMR. (see text for details about the datasets). The number on each stacked column shows the total runtime of the whole dataset in minutes. Each experiment was repeated three times and the average runtime in minutes is shown

We ran multiple SpaRC experiments over 20 and 50 GB Cow Rumen dataset on OTC, each with a Spark default parallelism (spark.default.parallelism) value ranging from 50 to 20 000. Once set, Spark automatically sets the number of partitions of an input file according to its size for distributed shuffles.

As shown in Figure 4, we found the performance of SpaRC does not vary much over several orders of magnitude in parallelism, for both of the two datasets tested. As long as the parallelism is not extreme (<100 or over 1 million), SpaRC performance is quite consistent. When there are too few data partitions, performance suffers because of cluster resource under utilization. In contrast, when there are too many data partitions, there might be excessive overhead in managing small tasks. It is not necessary, at least in this case, to adjust the default parallelisms.

It is worth noting that Spark relies on Hadoop file system (HDFS) which has a default partition size 64 MB. Our previous work showed that bioinformatics applications can benefit from setting it to 32 MB (Shi *et al.*, 2017), therefore, in SpaRC we recommend setting HDFS default partition size to 32 MB.

Table 5. Metrics of SpaRC intermediate data on different datasets

Dataset	# of k-mers	# of edges	# of nodes	Avg degrees/node
PacBio1	179 039 835	263 116 527	1 027 204	512
Maize	96 643 966	298 631 852	11 465 314	52
Cow Rumen	46 027 775	41 155 061	4 001 389	20
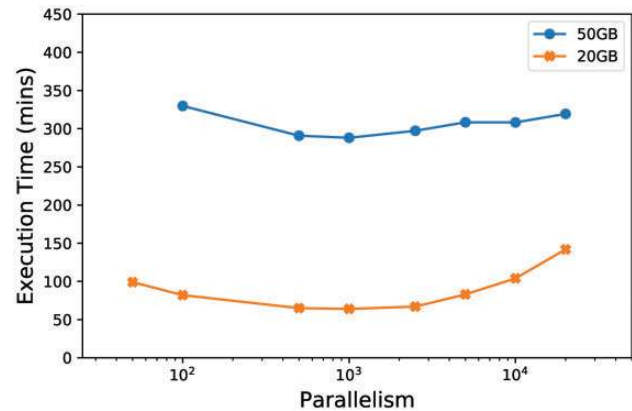
Fig. 4. Effect of parallelism level on the total execution time. The execution time (*Y-axis*) of two datasets derived from the Cow Rumen metagenome dataset (20 GB and 50 GB) with different settings of Spark parallelism in log10 scale (*X-axis*). The execution time reflects the average of three independent runs

3.5 SpaRC scales near linearly with input data and compute nodes

We designed two different experiments to measure the scalability of SpaRC. The first tests data scalability as more input are added on a fixed-sized cluster; the second measures horizontal scalability as more nodes are added to the cluster to compute the same input. For the data scalability test we use 20, 40, 60, 80 and 100 GB fastq datasets from Cow Rumen metagenome. The sequence retrieval step (AddSeq) is not shown due to its negligible processing time (as mentioned in the above).

We report in Figure 5 the result of the first experiment varying input data size and maintaining the number of nodes in the OTC cluster to a fixed value (20). The KMR and GCER step scale up linearly as expected, while LPA step scales up near linearly, consistent with its design (Raghavan *et al.*, 2007).

We next tested SpaRC performances by keeping the input size fixed (10, 50 GB) but using different number of nodes. As shown in Figure 6, the compute time required for each stage and the total time decreases as the number of nodes increases. However, there is a 'sweet spot' for each specific input size (10 nodes for 10 GB, 50 for 50 GB, respectively). For node counts less than the optimum, every doubling of nodes translates into approximately halving the compute time. However, the rate of compute time improvements decreases when the node number increases beyond the optimum. This phenomenon can be explained by the Amdahl's law (https://en.wikipedia.org/wiki/Amdahl's_law) in parallel computing. Overall, we achieve the near-linear scalability as other spark-based tools (de Castro *et al.*, 2017; Rasheed and Rangwala, 2013), suggesting SpaRC scales well with the number of nodes.

3.6 Performance comparison among cloud and HPC clusters

We have shown SpaRC can be run without modification on two cloud environments, OTC and EMR. To explore whether SpaRC

are applicable on HPC systems where spark jobs is incorporated into the scheduler along with other traditional HPC jobs, we ran SpaRC against PacBio1 dataset on an eight-node cluster on PSC's Bridge system (Section 2). Currently one can maximally provision eight nodes per interactive session without reservation. As on OTC and AWS EMR clusters, we used one of the nodes as our spark master and the remaining as spark workers. Again, we are able to run SpaRC without modification in the HPC environment. Moreover, the performance on HPC system is comparable to cloud environments, as the total execution time on the three systems are comparable (Table 6). These results demonstrate the flexibility of the Spark framework, enabling SpaRC to adapt to very different environments.

4 Conclusions and discussions

Metagenome and transcriptome assembly is challenging due to both its scale and complexity. We developed an efficient distributed algorithm, SpaRC, for large-scale metagenome and transcriptome read clustering to enable downstream assembly optimization. SpaRC takes advantage of Apache Spark for scalability, efficiency, rapid development and flexible running environments. SpaRC can handle large-scale datasets produced by current NGS technologies, including both long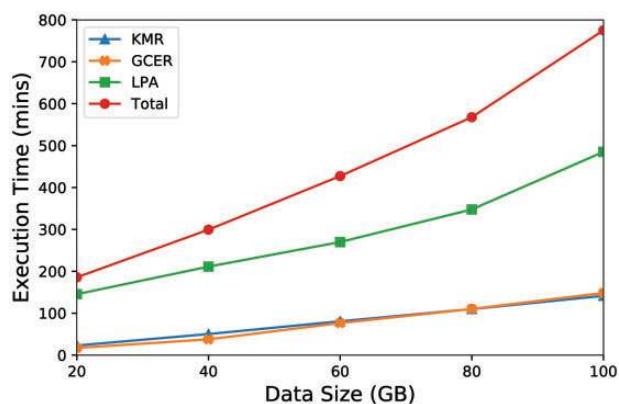 and short reads. We evaluated SpaRC on both transcriptome and metagenome datasets and demonstrated that SpaRC produces accurate results.

SpaRC is a generic read clustering tool. Although it has a function to produce k-mer counts for filtering purposes, it needs to create a map between k-mers and reads containing them for downstream clustering. We do not recommend using SpaRC for calculating simple k-mer statistics. There exist several k-mer counters such as KMC2 (Deorowicz *et al.*, 2015), DSK (Rizk *et al.*, 2013) and Jellyfish (Marçais and Kingsford, 2011) that are very fast as they exploit specific data structures. For larger datasets it should be straightforward to create a Spark-based k-mer counting program. We are actively exploring alternative data structures (bloom filters, minimizers, etc) for further improving the computing performance of SpaRC.

Since Apache Spark is a still very young project undergoing heavy development, some of its components have not been stabilized and/or optimized. For example, the current LPA is implemented in GraphX using the pregel interface (Malewicz *et al.*, 2010) instead of in GraphFrame (Dave *et al.*, 2016), which does not take the full advantage of the scalability and efficiency of the DataFrame API (Armbrust *et al.*, 2015). Current LPA function in GraphFrame is a simple wrapper of the method in GraphX. LPA performance in space and time efficiency could be improved further. Since it cumulatively caches the results of each iteration for job recovery, disk usage often explodes as the number of iterations increases. Furthermore, if one executor dies, all of its cached data is lost and the whole process has to start from scratch. Creating a checkpoint for each iteration like the GraphFrame version of connect component should alleviate this problem. Although it is known that Spark programs are much faster than their Hadoop equivalent, they may not be as efficient as MPI programs on a cluster, or non-scalable C/C++/JAVA programs on a single machine. Some overhead in Spark is necessary to ensure its robustness, which is critical for large jobs. Besides robustness, programing Scala/Python/R relatively is much easier than programing C/C++/JAVA and MPI.

We observed the clusters produced tend to be too small when the read is short (e.g. single-end metagenomic dataset on Illumina platform). For pair-end sequencing datasets one can merge (if they overlap) or concatenate the two ends to increase the cluster size. Decreasing k-mer size, or requiring less shared k-mers should also help increase cluster size. However, this may lead to decrease of purity. One potential solution is to run an additional binning or scaffolding step (using pair-end or long reads if available) after assembling each cluster of reads into contigs, a common practice in metagenome assemblies.

Fig. 5. Scalability of SpaRC on different input sizes. The execution time (*Y-axis*) of each step of SpaRC on the Cow Rumen dataset with different input sizes (*X-axis*) on 20 OTC nodes. Total runtime is shown at the top. Each experiment was executed three times and the figure shows the average runtime, standard deviations are too small to be shown
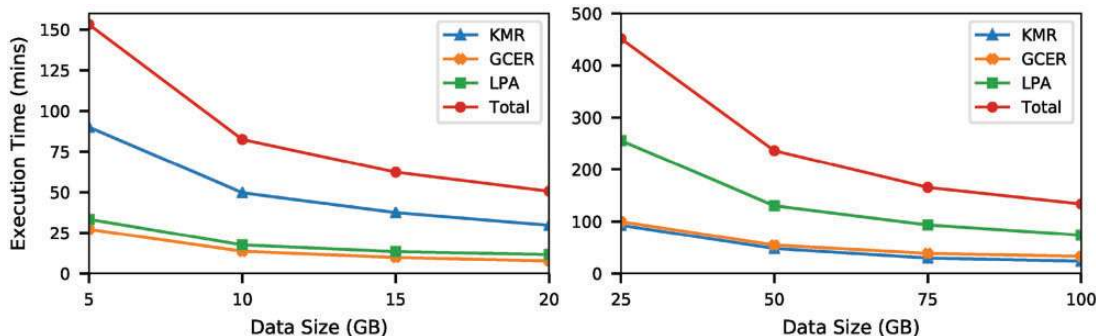

Fig. 6. Scalability of SpaRC on different number of nodes. (Left) The execution time (*Y-axis*) of each step plus the total time on different number of Amazon EMR nodes (*X-axis*) using a 10 GB Cow Rumen dataset; (right) same experiments were repeated using a 50 GB Cow Rumen dataset. Each experiment was executed three times and the figure shows the average runtime, standard deviations are too small to be shown

Table 6. Performance of SpaRC against PacBio1 on Cloud (OTC, AWS EMR) and HPC Clusters (Bridges)

	OTC	AWS EMR	Bridges
# of workers	19	19	7
Cores	8 (152)	8 (152)	28 (196)
Memory (GB)	64 (1216)	61(1159)	128 (896)
Time (min)	106	105	126

Running SpaRC on HPC systems sometimes is necessary because it avoids the need to move data into the cloud, as well as makes it easy to incorporate SpaRC as part of the assembly pipeline. However, Spark configuration on HPC systems is a complicated task. Once Spark is deployed, in our case on the Bridges system, it is flexible and can be run on a single node, or can be scaled up to a very large cluster.

References

Abu-Doleh,A. and Çatalyürek,Ü.V. (2015) Spaler: spark and graphx based de novo genome assembler. In: *2015 IEEE International Conference on Big Data (Big Data)*. pp. 1013–1018. IEEE.

Ajay,S.S. *et al.* (2011) Accurate and comprehensive sequencing of personal genomes. *Genome Res.*, **21**, 1498–1505.

Alanko,J. *et al.* (2017) A framework for space-efficient read clustering in metagenomic samples. *BMC Bioinformatics*, **18**, 59.

Armbrust,M. *et al.* (2015). Spark sql: relational data processing in spark. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. pp. 1383–1394. ACM.

Bahmani,A. *et al.* (2016) Sparkscore: leveraging apache spark for distributed genomic inference. In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops*. pp. 435–442. IEEE.

Cleary,B. *et al.* (2015) Detection of low-abundance bacterial strains in metagenomic datasets by eigengenome partitioning. *Nature Biotechnol.*, **33**, 1053.

Dave,A. *et al.* (2016). Graphframes: an integrated api for mixing graph and relational queries. In: *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*. p. 2. ACM.

de Castro,M.R. *et al.* (2017) Sparkblast: scalable blast processing using in-memory operations. *BMC Bioinformatics*, **18**, 318.

Deorowicz,S. *et al.* (2015) Kmc 2: fast and resource-frugal k-mer counting. *Bioinformatics*, **31**, 1569–1576.

Georganas,E. *et al.* (2015) Hipmer: an extreme-scale de novo genome assembler. In: *2015 SC-International Conference for High Performance Computing, Networking, Storage and Analysis*. pp. 1–11. IEEE.

Gordon,S.P. *et al.* (2015) Widespread polycistronic transcripts in fungi revealed by single-molecule mrna sequencing. *PLoS One*, **10**, e0132628.

Guo,X. *et al.* (2015) Dime: a novel framework for de novo metagenomic sequence assembly. *J. Comput. Biol.*, **22**, 159–177.

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

Howe,A.C. *et al.* (2014) Tackling soil diversity with the assembly of large, complex metagenomes. *Proc. Natl. Acad. Sci.USA*, **111**, 4904–4909.

Hughes,J.B. *et al.* (2001) Counting the uncountable: statistical approaches to estimating microbial diversity. *Appl. Environ. Microbiol.*, **67**, 4399–4406.

Klein,M. *et al.* (2017) Biospark: scalable analysis of large numerical datasets from biological simulations and experiments using hadoop and spark. *Bioinformatics*, **33**, 303–305.

Li,D. *et al.* (2015) Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, **31**, 1674–1676.

Li,H. (2016) Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, **32**, 2103–2110.

Malewicz,G. *et al.* (2010) Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. pp. 135–146. ACM.

Marçais,G. and Kingsford,C. (2011) A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, **27**, 764–770.

Martin,J.A. and Wang,Z. (2011) Next-generation transcriptome assembly. *Nat. Rev. Genet.*, **12**, 671–682.

Martin,J.A. *et al.* (2014) A near complete snapshot of the zea mays seedling transcriptome revealed from ultra-deep sequencing. *Sci. Rep.*, **4**, 4519.

Massie,M. *et al.* (2013) Adam: genomics formats and processing patterns for cloud scale computing. Technical report UCB/EECS-2013-207. EECS Department, University of California, Berkeley.

Miller,J.R. *et al.* (2010) Assembly algorithms for next-generation sequencing data. *Genomics*, **95**, 315–327.

Nurk,S. *et al.* (2017) metaSPAdes: a new versatile metagenomic assembler. *Genome Res.*, **27**, 824–834.

Nystrom,N.A. *et al.* (2015). Bridges: a uniquely flexible hpc resource for new communities and data analytics. In: *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. p. 30. ACM.

Raghavan,U.N. *et al.* (2007) Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, **76**, 036106.

Rasheed,Z. and Rangwala,H. (2013). A map-reduce framework for clustering metagenomes. In: *2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. pp. 549–558. IEEE.

Rizk,G. *et al.* (2013) Dsk: k-mer counting with very low memory usage. *Bioinformatics*, **29**, 652–653.

Sczyrba,A. *et al.* (2017) Critical assessment of metagenome interpretation—a benchmark of metagenomics software. *Nat. Methods*, **14**, 1063.

Shi,L. *et al.* (2017) A case study of tuning mapreduce for efficient bioinformatics in the cloud. *Parallel Comput.*, **61**, 83–95.

Shi,W. *et al.* (2014) Methane yield phenotypes linked to differential gene expression in the sheep rumen microbiome. *Genome Res.*, **24**, 1517–1525.

Singer,E. *et al.* (2016) Next generation sequencing data of a defined microbial mock community. *Sci. Data*, **3**, 160081.

Sunagawa,S. *et al.* (2015) Structure and function of the global ocean microbiome. *Science*, **348**, 1261359.

Tringe,S.G. and Rubin,E.M. (2005) Metagenomics: dna sequencing of environmental samples. *Nat. Rev. Genet.*, **6**, 805–814.

Wang,Y. *et al.* (2012) Metacluster 5.0: a two-round binning approach for metagenomic data for low-abundance species in a noisy sample. *Bioinformatics*, **28**, i356–i362.

Xin,R.S. *et al.* (2013) Graphx: a resilient distributed graph system on spark. In: *First International Workshop on Graph Data Management Experiences and Systems*. p. 2. ACM.

Xu,X. *et al.* (2016) Cloudphylo: a fast and scalable tool for phylogeny reconstruction. *Bioinformatics*, **33**, 438–440.

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*. pp. 2–2. USENIX Association.