Bioinformatics, 34(9), 2018, 1457–1465
doi: 10.1093/bioinformatics/btx808
Advance Access Publication Date: 15 December 2017
Original Paper

OXFORD

Sequence analysis

Analyzing large scale genomic data on the cloud with Sparkhit

Liren Huang[1,2,3], Jan Krüger[1,2] and Alexander Sczyrba[1,2,3,*]

[1]Faculty of Technology, Bielefeld University, Bielefeld 33615, Germany, [2]Center for Biotechnology – CeBiTec, Bielefeld University, Bielefeld 33615, Germany and [3]Computational Methods for the Analysis of the Diversity and Dynamics of Genomes, Bielefeld University, Bielefeld 33615, Germany

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

Received on July 14, 2017; revised on November 9, 2017; editorial decision on December 11, 2017; accepted on December 14, 2017

Abstract

Motivation: The increasing amount of next-generation sequencing data poses a fundamental challenge on large scale genomic analytics. Existing tools use different distributed computational platforms to scale-out bioinformatics workloads. However, the scalability of these tools is not efficient. Moreover, they have heavy run time overheads when pre-processing large amounts of data. To address these limitations, we have developed Sparkhit: a distributed bioinformatics framework built on top of the Apache Spark platform.

Results: Sparkhit integrates a variety of analytical methods. It is implemented in the Spark extended MapReduce model. It runs 92–157 times faster than MetaSpark on metagenomic fragment recruitment and 18–32 times faster than Crossbow on data pre-processing. We analyzed 100 terabytes of data across four genomic projects in the cloud in 21 h, which includes the run times of cluster deployment and data downloading. Furthermore, our application on the entire Human Microbiome Project shotgun sequencing data was completed in 2 h, presenting an approach to easily associate large amounts of public datasets with reference data.

Availability and implementation: Sparkhit is freely available at: https://rhinempi.github.io/sparkhit/.

Contact: asczyrba@cebitec.uni-bielefeld.de

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Genome and transcriptome projects have provided unprecedented knowledge and insights for life science by interpreting large amounts of next-generation sequencing data (Consortium and Auton, 2015; Peterson *et al.*, 2009; R Genomes Project, 2014; Wyatt *et al.*, 2014). These genomic datasets, sequenced from thousands of samples, take days or weeks to analyze on private clusters (Bao *et al.*, 2014). For researchers with limited computational resources, processing terabytes (TB) of data remains a fundamental bottleneck. Although several existing tools have been developed and utilized on single computers (Bray *et al.*, 2016; Langmead and Salzberg, 2012; Li and Durbin, 2009; Li *et al.*, 2009, 2008; Niu *et al.*, 2011; Wood and Salzberg, 2014), most of the parallelizations on multi-computer networked clusters are done by manually splitting and distributing in smaller batches (Droop, 2016). To be compatible with a cluster, methods involving message passing and graph representation between computers must be re-implemented with higher level programming interfaces (Gropp *et al.*, 1996). To address this challenge, we need both easy-access to large computational infrastructure and bioinformatics tools that are compatible and scalable on such platforms.

Cloud infrastructure and platform services (IaaS and PaaS) have been well established in the informatics discipline (Dean and Ghemawat, 2008; Schadt *et al.*, 2010; Shvachko *et al.*, 2010; Zaharia *et al.*, 2012). To fully exploit distributed cloud computing systems, methods and programs should be scalable, fault tolerant and platform independent. In genomics applications, there are several tools [e.g. ABySS (Simpson *et al.*, 2009) and Ray (Boisvert *et al.*, 2010)] using the message passing interface (MPI) for distributed implementations. However, programming on top of MPI has to tangle with thread synchronization and load balance. Moreover,

Table 1. Features of different cloud based bioinformatics tools

Tools	Platform	Implementation	Application	Methods and tools
Cloudburst (Schatz, 2009)	Hadoop	Native Java re-implementation	Mapping	Seed and extend
Crossbow (Langmead *et al.*, 2009)	Hadoop	Invoking external utilities	Mapping and genotyping	Bowtie and SOAPsnp (Li *et al.*, 2008)
Halvade (Decap *et al.*, 2015)	Hadoop	Invoking external utilities	Mapping	BWA and GATK (McKenna *et al.*, 2010)
Myrna (Langmead *et al.*, 2010)	Hadoop	Invoking external utilities	Mapping and gene expression profiling	Bowtie (Langmead, 2010)
CS-BWAMEM (Chen *et al.*, 2015)	Spark	Native Scala re-implementation	Mapping	Burrows-Wheeler transform
SparkSW (Zhao *et al.*, 2015)	Spark	Native Scala re-implementation	Mapping	Smith-Waterman
SparkBWA (Abuin *et al.*, 2016)	Spark	Invoking external utilities	Mapping	BWA (Li and Durbin, 2009)
MetaSpark (Zhou *et al.*, 2017)	Spark	Native Scala re-implementation	Fragment recruitment	Seed and extend
Sparkhit	Spark	Both native and external	Fragment recruitment, mapping, genotyping, taxonomy profiling and gene expression profiling	Seed and extend and a collection of tools

there is no complete fault tolerance mechanism built inside of MPI. The conventional Apache Hadoop framework is designed to offer higher scalability and a supervised fault tolerance mechanism by providing a distributed data storage system called HDFS (Shvachko *et al.*, 2010) and a distributed computing engine, Hadoop MapReduce (Dean and Ghemawat, 2008). Cloudburst (Schatz, 2009) is a Hadoop based sequence mapping tool programmed in the MapReduce model. The common seed and extend alignment pipeline is split and implemented in 'map' and 'reduce' steps, where the 'map' step searches K-mer matches (as seeds) and the 'reduce' step extends the seeds to apply dynamic alignments. The limitation for such method is that the 'reduce' step introduces large data shuffling across cluster nodes that impacts its performance. Crossbow (Langmead *et al.*, 2009), Halvade (Decap *et al.*, 2015) and Myrna (Langmead *et al.*, 2010), on the other hand, directly use Hadoop to invoke existing sequence aligner [Bowtie (Langmead, 2010)] and SNP caller [SOAPsnp (Li *et al.*, 2008) or GATK (McKenna *et al.*, 2010)] for sequence mapping and genotyping on large datasets. The three tools have successfully reduced the run times for mapping, genotyping and gene expression quantification. Yet, their data preprocessing step introduces a heavy overhead and their options for handling distributed data are limited.

The more recent Apache Spark framework addresses these weaknesses with its unique data sharing primitive, called resilient distributed dataset (RDD) (Zaharia *et al.*, 2012). RDD offers a 'cache' function to store distributed data in the memory across computers on a cluster, thus, avoiding run time overhead of iterative data I/O (input and output). Moreover, Spark has more build-in functions for RDD to facilitate methods implementation and data handling via its application programming interface (API). These advantages make Spark suitable for large scale genomic data analysis. Nevertheless, existing Spark based bioinformatics tools have their own limitations. For instance, SparkBWA (Abuin *et al.*, 2016) adopts the same idea of Crossbow by using Spark to invoke the BWA (Li and Durbin, 2009) aligner. However, it does not provide data preprocessing functions for large amounts of compressed sequencing data. Thus, manually decompressing the sequencing data introduces a significant run time overhead. Instead of directly invoking existing aligners, MetaSpark (Zhou *et al.*, 2017) re-implemented a fragment recruitment algorithm (Rusch *et al.*, 2007). It has the same 'seed and extend' pipeline as Cloudburst, but implemented its algorithm on top of Spark. Thus, it also introduces large data shuffling in the reduce step that impacts its run time performance. Moreover, it requires a self-defined input format rather than the standard Fastq and Fasta format, which introduces an overhead to manually covert large Fastq files (Table 1).

We present Sparkhit, an open source computational framework that is easy to use on a local cluster or on the cloud (Fig. 1). Sparkhit is built on top of the Apache Spark platform, integrates a series of analytical tools and methods for various genomic applications: (i) We have natively implemented a metagenomic fragment recruitment tool and a short-read mapping tool (Sparkhit-recruiter and Sparkhit-mapper). The short-read mapper implements the pigeonhole principle to report the best hit of a sequencing read. Whereas the fragment recruitment tool implements the q-Gram algorithm to allow more mismatches during the alignment, so that extra information is provided for the metagenomic analysis; (ii) For using external software on Sparkhit, we built a general tool wrapper (Sparkhit-piper) to invoke and parallelize existing executables, biocontainers [e.g. Docker containers (Merkel, 2014)] and scripts; (iii) For downstream data mining, we integrated and extended Spark's machine learning library. All methods and tools are programmed and implemented in a new MapReduce model extended by Spark, where parallelization is optimized (load balanced) and supervised (fault tolerance).

Our benchmarks on Sparkhit demonstrated its high scalability. In comparison, Sparkhit ran 18–32 times faster than Crossbow on data preprocessing and Sparkhit-recruiter ran 92–157 time faster than MetaSpark on fragment recruitment (Figs 2 and 3). Utilizing a powerful compute cloud, Sparkhit can quickly analyze large amounts of genomic data (Fig. 4). Our use case presents a 21 h' pay-as-you-go' cloud application that analyzed 100 TB genomic data from 3 genome projects and a transcriptomics study (Wyatt *et al.*, 2014). The analysis on the Human Microbiome Project (HMP), associated public 'big data' with private datasets, demonstrates how Sparkhit can be widely applied in different genomic studies (Supplementary file S2, Figs S1, S2 and S3). Thus, our framework enables the broader community to engage genomic investigations by leveraging cloud computing resources.

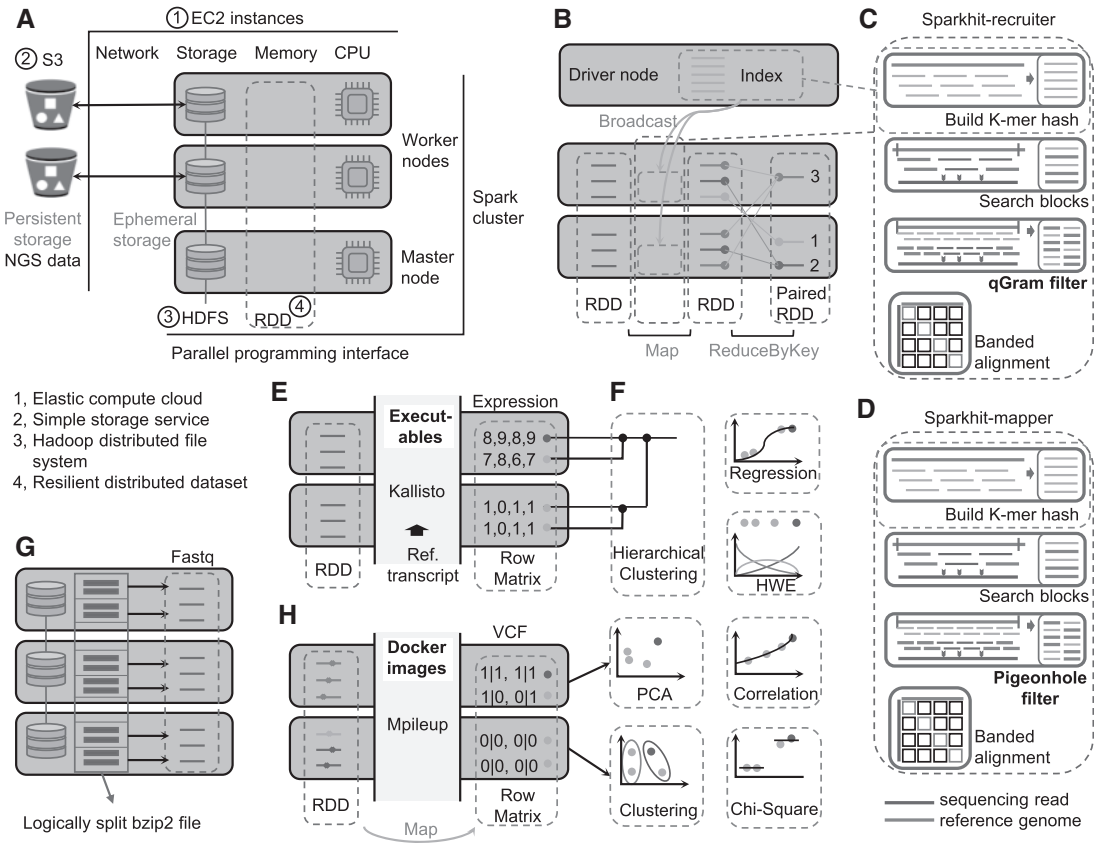

Fig. 1. A distributed computational framework for large scale genomic analysis. (**A**) Architecture of a Spark cluster deployed on the Amazon cloud. The yellow boxes represent Amazon EC2 instances that are virtualized into Spark master/worker nodes. (**B**) Distributed implementation of Sparkhit-recruiter. The reference index, illustrated in blue dashed box, is built on a driver node and broadcasted to each worker node. Sequencing reads, illustrated in Red dashes, are loaded into an RDD and queried to the broadcasted reference index in parallel as a 'Map' step. A 'Reduce' step is followed to summarize the mapping result. (**C**) Pipeline of Sparkhit-recruiter. The reference genome, illustrated in bold blue dash, is extracted and built into a K-mer hash table. The sequencing read, illustrated in bold red dash, will be searched against the reference hash table for exact matches. A smaller Kmer is used to apply the q-Gram filter. (**D**) Pipeline of Sparkhit-mapper. It is similar to (C), but uses the pigeonhole principle. (**E, H**) Using external tools and Docker containers for different analyses. Genomic data is loaded into an RDD and distributed across worker nodes. Each partition of the RDD is sent to external tools to be processed independently. (**F**) Different modules of the machine learning library. Colored dots denote vectors of either genotypes or gene expressions. (**G**) Parallel decompression. A Bzip2 file is split into blocks and stored in three worker nodes. Each block is decompressed independently (Color version of this figure is available at *Bioinformatics* online.)

2 Materials and methods

2.1 Spark cluster's architecture and cloud deployment

The architecture of Sparkhit is organized by a Spark cluster, which virtualizes each computer and forms a network consists of one or more master node(s) and worker nodes. To organize all virtual compute nodes, a master node is selected to supervise worker nodes, allocate resources and balance workloads, whereas worker nodes carry out assigned tasks (Fig. 1A). On such architecture, Spark extends and integrates the MapReduce model into RDD's functions, where the 'map' step loads, processes and saves data in parallel on each worker node and the 'reduce' step sorts, shuffles and aggregates intermediate data. We used Spark-ec2 and BiBiGrid (https://github.com/BiBiServ/bibigrid), a Java-based tool developed in our research group, to easily setup Spark clusters on the Amazon cloud with one command line (see Supplementary file 1 Method-1.2).

2.2 Parallel data downloading

Downloading and decompressing large genomic files are significant bottlenecks before the actual data analysis begins. Spark's architecture can greatly benefit from high-performance networks during large data transfers. In particular, when commencing a download task, files can be split into chunks and transferred from distributed storage system, such as Amazon simple storage service (S3), to each worker node. This parallel transfer method fully utilizes high network bandwidth of a distributed cluster (Fig. 1A).

When downloading data from Amazon S3 to HDFS, we have used Hadoop Distcp (distributed copy), a tool designed for large inter-cluster copying. For downloading data from Amazon S3 to a shared network file system (NFS), we used a Java-based tool developed in our research group called BiBiS3 (https://wiki.cebitec.uni-bielefeld.de/bibiserv/index.php/BiBiS3). BiBiS3 not only parallelizes downloading jobs to multiple computer nodes, but also applies multi-thread downloading on each computer node to fully exploit the capacities of every computer's network connection (see Supplementary File 1 Method-1.3).

2.3 Parallel data decompression

Most genomic datasets are compressed and stored on public repositories. To directly access and analyze the compressed data, we also improved the performance on data preprocessing by introducing parallel decompression after downloading the data. We developed a parallel decompression tool, Sparkhit-spadoop, that is built on top

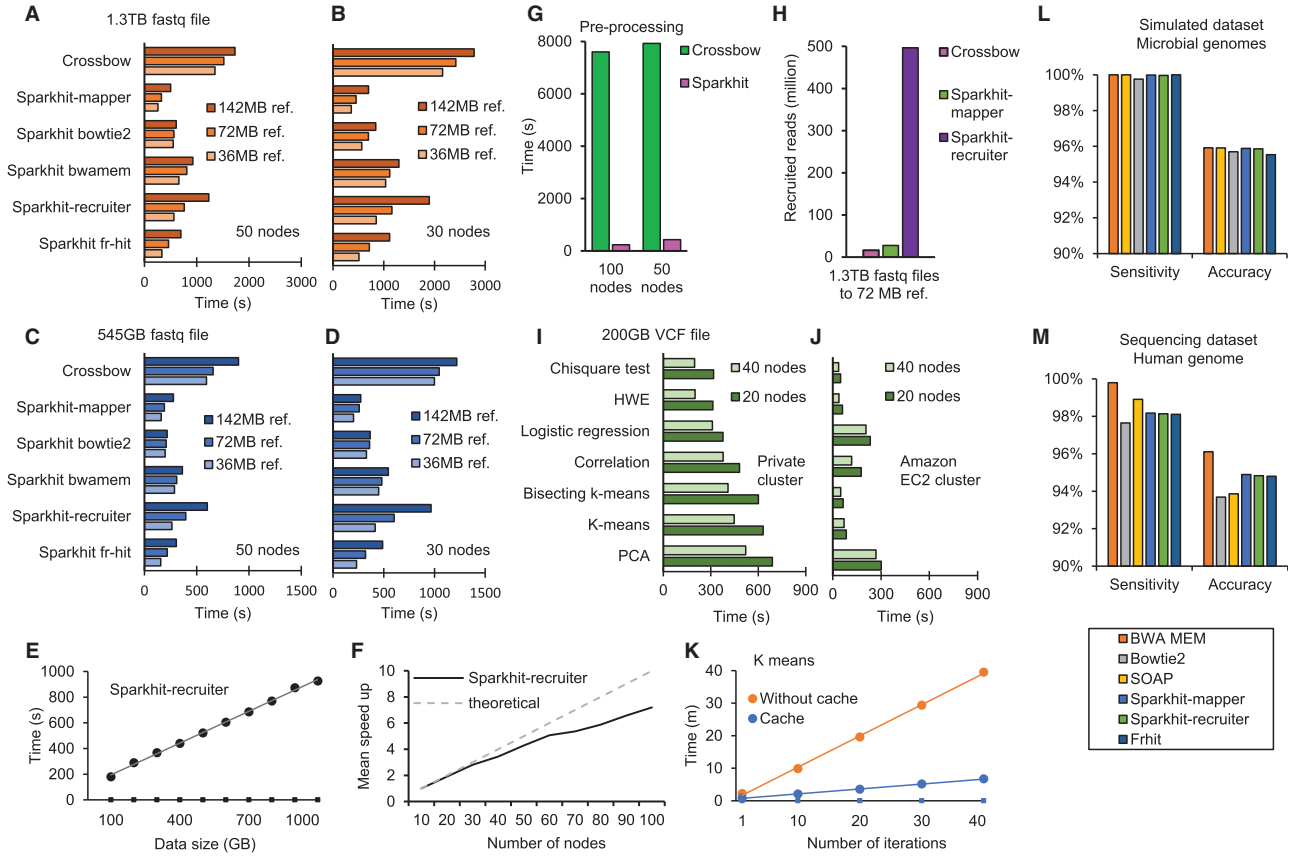

Fig. 2. Performance benchmarks for Sparkhit. (**A–D**) Run time comparisons between different aligners. The comparisons were carried out across different sizes of input fastq files, different sizes of reference genomes and different numbers of worker nodes. (**E**) Run time performance of Sparkhit-recruiter for recruiting 100–1000 GB sequencing data to a 72 MB reference genome on a 30 nodes Spark cluster deployed on the Amazon EC2 cloud. Each node has 32 vCPUs. (**F**) Scaling performance of Sparkhit-recruiter. When increasing the number of worker nodes, the mean speed ups are measured by comparing their run times to the run time on 10 worker nodes. We recruited 1.3 TB fastq files (Data-1) to a 72 MB reference genome (Ref-2) on the same cluster of (E). (**G**) Run time comparisons between Crossbow and Sparkhit for preprocessing 338 TB compressed fastq files on 50 and 100 worker nodes. (**H**) Comparing the recruited number of reads between Crossbow and Sparkhit-recruiter when mapping 1.3 TB fastq files to a 72 MB reference genome. (**I–J**) Run times of the machine learning library on a private cluster and the Amazon EC2 cloud. All computations were performed on a 200 GB VCF file cached in the memory. (**K**) Run times for different iterations of the K means clustering. We ran iterations on the same VCF file from I, J, with data caching and non data caching. (**L–M**) Sensitivity and accuracy comparisons between mapping tools

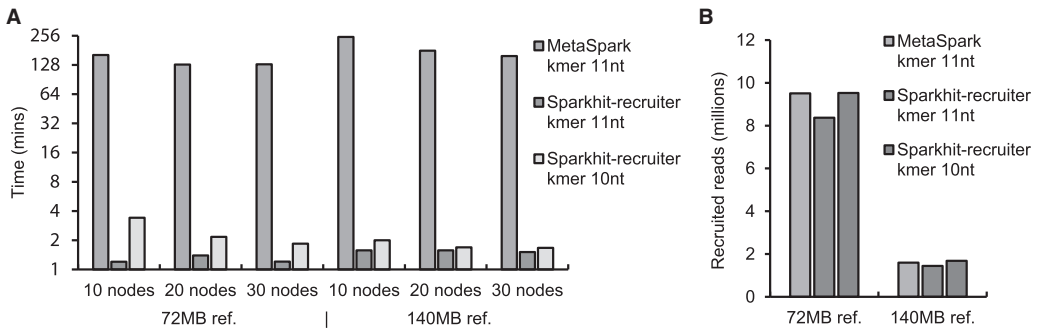

Fig. 3. Comparisons between Sparkhit-recruiter and MetaSpark on metagenomic fragment recruitment. (**A**) Run times on recruiting simulated sequencing reads to 72 MB and 142 MB reference genomes. All tests were carried out on 10, 20 and 30 worker nodes Spark clusters. Each worker node has 16 vCPUs. Run times are presented in logarithmic scale base 2. (**B**) Numbers of recruited reads on recruiting 6 million simulated reads to 72 MB reference genome and 1 million simulated reads to 142 MB reference genome

of the Apache Hadoop (Spark's parallel decompression has a thread safety issue at 2.0.0 version) and included it in our toolkit.

Since Bzip2 files are compressed in blocks (900 KB per block by default), a large Bzip2 compressed file with sequencing data can be decompressed in parallel. In particular, each block is an independent component that is delimited by a 48-bit pattern, which makes it possible to find the block boundaries. When block boundaries are found, parallel decompression can be applied to each block and processed by multiple CPUs, so that more compute cores are utilized. Then, we distributed all processes in a Hadoop MapReduce

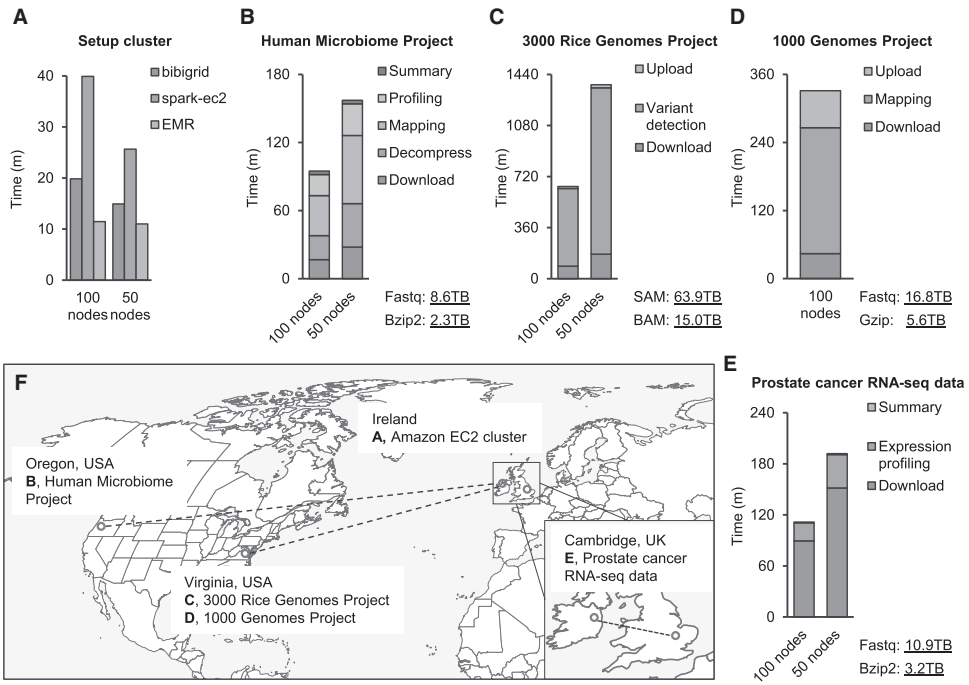

Fig. 4. Large scale genomic data analyses on the cloud with Sparkhit. (**A**) Run time comparison between three auto-scaling tools for deploying a Spark cluster on the Amazon EC2 cloud. Durations include pending for approval of EC2 spot request and waiting for SSH connection to each EC2 instance. EMR, Amazon Elastic MapReduce service. (**B**) Run times for processing all WGS data from the Human Microbiome Project. Mapping was carried out using Sparkhit-recruiter while profiling was carried out using Sparkhit invoked Kraken. (**C**) Run times for processing 15 TB BAM files of the 3000 Rice Genome Project. We uploaded the variant calling result to Amazon S3. (**D**) Run times for processing 5.6 TB compressed sequencing data. Mapping was carried out using Sparkhit invoked BWA aligner. We uploaded the SAM files to Amazon S3. (**E**) Run times for processing 3.2 TB RNA-seq data. Gene expression profiling is carried out using Sparkhit invoked Kallisto. (**F**) Fast access to genomic data on public repositories. Datasets of the Human Microbiome Project, the 3000 Rice Genome Project and the 1000 Genomes Project are hosted in different regions on Amazon S3. Whereas the RNA-seq data of a prostate cancer transcriptomic study is stored on the ENA ftp server

job that creates a 'mapper' for each chunk of the input HDFS data. Each 'mapper' loads a data chunk and commences a decompression process on the Bzip2 blocks in the chunk. (see Supplementary file 1 Method-1.4)

2.4 Data loading and saving

The Spark RDD stores data chunks in line-based text format, where identifying entries requires finding line boundaries denoted by new-line characters. After decompression, most NGS data is stored in line-based text files, e.g. fastq, SAM and VCF files. For SAM and VCF files, each line is an independent unit that contains its corresponding information (mapping records or genotypes). Thus, when loading these files, each line is read and stored as an element of an RDD. However, a fastq file stores its basic information in a four-line unit, where each line is an essential part of a sequencing read. When loading fastq files into Spark RDD, a filter step is applied to check each four-line unit before sending it out to be processed.

Both loading and saving run in parallel on each partition of the RDD (the sequencing data) with a default size of 256 MB per partition. Sparkhit can also directly load and save data from and to both HDFS and Amazon S3 by using the HDFS and the S3 URL scheme.

2.5 Fragment recruitment and short-read mapping implementations of Sparkhit

In metagenomic studies, fragment recruitment is a key step to understand the genome structure, evolution, phylogenetic diversity and gene functions of biological samples (Rusch *et al.*, 2007). As a special case of read mapping, fragment recruitment supports more mismatches during the alignment. Thus, the computational complexity for fragment recruitment can be higher than for standard short read mapping.

We implemented a fast and sensitive fragment recruitment tool, called Sparkhit-recruiter. Sparkhit-recruiter extends the Fr-hit (Niu *et al.*, 2011) pipeline and is implemented natively on top of the Apache Spark. The pipeline consists of four steps: (i) building reference index, (ii) searching candidate blocks, (iii) block filtering and (iv) banded alignment (Fig. 1C). When building the reference index, we use a K-mer hash table to store each K-mer's location on the reference genome (Fig. 1C in blue color). Once the index is built, we extract the K-mers from sequencing reads (Fig. 1C in red color) and search against the reference hash table for exact matches. The matched K-mers will be placed on the genome as seeds to extend candidate blocks. The block filtering step incorporates a q-Gram threshold to remove badly matched blocks, therefore, improves run time performance. After filtering, banded alignment is applied to give a final mapping result.

We also implemented a short-read aligner, called Sparkhit-mapper (Fig. 1B and D). It adopts the same pipeline of Sparkhit-recruiter, but uses a more strict pigeonhole principle for the filtering step. Compared to the q-gram filter implementation in Sparkhit-recruiter, the pigeonhole principle allows fewer mismatches on the sequence so that less identical blocks are filtered and high similarity candidate blocks are preserved (see Supplementary file 1 Method-1.5.5). In this case, less candidate blocks are sent to the next step for banded alignment, resulting in much faster runtime than Sparkhit-recruiter.

To implement the pipelines in a distributed fashion, we split the mapping processes into two parts: building the reference index and

querying sequencing reads (Fig. 1C, D, blue and red dashed boxes). When starting a Sparkhit-recruiter job, the reference index is, firstly, built on the driver node (usually the master node), where the main Spark program runs. Once the reference index is built, the driver program executes a 'broadcast' command to ship one copy of the index to each worker node shared by all local querying tasks. On the worker nodes, sequencing data chunks are loaded from HDFS to a Spark RDD. Each partition of the RDD is, then, independently queried to the broadcasted reference index as a 'map' step of the MapReduce pipeline. In the end, a 'reduce' step summarizes the mapping result (Fig. 1B, see also Supplementary file S1 Method-1.5).

2.6 Parallelizing external tools using Sparkhit

To be flexible for different kinds of analyses, we built a general tool wrapper called Sparkhit-piper. It extends Spark's 'pipe' function to invoke existing tools for analyses like sequence mapping, taxonomic profiling, gene expression quantification and genotyping (Fig. 1E). We use Spark RDD to split and distribute NGS data across cluster nodes. Then, distributed datasets are sent to the invoked tool via a standard input (stdin) stream. The tool processes input data in a batch and sends back the result to another RDD via a standard output (stdout) stream. In this case, the Spark RDD splits and distributes NGS data, while external tools carry out correspond computations. Sparkhit-piper is intuitive and flexible for users to parallelize their own scripts or tools directly without modifying their codes.

The implementation is based on Spark RDD's 'pipe' function, where the RDD is able to send its data out of the JVM for processing in the operating system, like a Linux pipe operator ('|'). The Java code can be expressed as:

```
JavaRDD<String> MapRDD = FastqRDD.pipe
(param.tool + param.toolOptions);
```

where '`FastqRDD`' is the input RDD that stores sequence data and '`MapRDD`' is the output RDD that stores mapping result. The '`param.tool`' represents the full path of the tool executable while '`param.toolOptions`' represents the corresponding tool parameters. Together, they assemble an external command that runs as an independent process on each partition of the input RDD.

2.7 Machine learning library

Downstream genomic data mining usually applies statistical tests and machine learning methods to analyze upstream results (gene abundance, taxonomic profile of different samples and genotypes of a certain cohort). We extended Spark's machine learning library (Mllib) and integrated a variety of algorithms: (i) clustering, (ii) regression, (iii) chi-square test, (iv) correlation test and (v) dimensional reduction (Fig. 1F). These algorithms are implemented with more RDD functions and iterative processes (see Supplementary file 1 Method-1.6).

3 Results

3.1 General performance benchmarking for Sparkhit

In the following section we present a series of benchmarks for our framework (Fig. 2).

3.1.1 Run time comparison between different mappers

We first measured the run times of all Sparkhit based mappers and compared them to Crossbow (Fig. 2A–D). Our toolkit ran faster than Crossbow across different numbers of worker nodes (30 and 50), different sizes of input data (1.3 TB and 545 GB) and different sizes of reference genomes (36 MB, 72 MB and 142 MB, see Supplementary file 1 Materials and Methods). Although Sparkhit-recruiter was slower than other Sparkhit based mappers, it recruited many more reads than standard short-read mappers such as BWA (Fig. 2H).

3.1.2 Scaling performance of Sparkhit-recruiter

Sparkhit-recruiter scaled linearly with the increasing amount of input data on a 30 worker nodes Spark cluster (Fig. 2E). When scaled-out to more compute nodes, Sparkhit experienced slight slowdown after we increased the number of worker nodes to 60 (Fig. 2F). The slowdown is introduced by the overhead of building the reference index. However, since metagenomic fragment recruitment applications actively change reference genomes between different studies, the index building overhead is quite small and it runs much faster compared to other Burrows-Wheeler transform (BWT) based methods (Supplementary file S2, Fig. S5 and Discussion section).

3.1.3 Preprocessing comparison with Crossbow

Data preprocessing is a critical step for interpreting cloud stored public datasets. Manually decompressed and distributed large amounts of genomic data on a cluster introduce significant overheads before data analysis. Although, several existing cloud tools have provided preprocessing functions (Decap *et al.*, 2015; Langmead *et al.*, 2009; Schatz, 2009), their proprocessing speeds are limited by their non-parallel implementations. We have compared the run time performances on data preprocessing between Sparkhit and Crossbow. For 338 TB Bzip2 compressed data (Data-1), Sparkhit ran 18 to 32 times faster than Crossbow on 50 and 100 worker nodes (Fig. 2G). Since Sparkhit utilizes all vCPUs for parallel decompression, its run time performance almost doubled from 50 nodes to 100 nodes, whereas Crossbow had similar run times on both clusters (Supplementary file S2, Table S12).

3.1.4 Machine learning library benchmarking and run time performances on different clusters

For modules of the machine learning library, we have compared their run time performances on both a private cluster and the Amazon EC2 cloud (Fig. 2I and J). For the private cluster, data was stored on magnetic disks of a shared network file system (NFS). Whereas on Amazon EC2, data was stored on an HDFS deployed on solid state disks (SSD). We ran every module on both 20 and 40 nodes Spark clusters using a 200 GB VCF file from the 1000 Genomes Project and compared their run times between a private cluster and Amazon EC2. Since each module opened 640 and 1280 I/O tasks (20 nodes and 40 nodes, each node has 32 cores) to read input data and write output result, the run time performance on the private cluster was significantly slower than on the Amazon EC2 cloud (Supplementary file S2, Table S14). We also observed a significant improvement on run time for cached iterative computations (K-means clustering), compared to non-cached ones (Fig. 2K).

3.1.5 Fragment recruitment comparison with MetaSpark

We have compared Sparkhit-recruiter to another Spark based distributed fragment recruitment tool (Fig. 3), MetaSpark (Zhou *et al.*, 2017). Sparkhit-recruiter ran 92–157 times faster than MetaSpark across different numbers of worker nodes (10, 20 and 30), different numbers of input reads (1 million and 6 millions) and different sizes of reference genomes (72 and 142 MB). Although our tool recruited

10–12% reads less than MetaSpark using the same K-mer size, we have adjusted to a smaller K-mer size that recruits more reads than MetaSpark, while still ran 47–124 times faster (see Discussion section).

3.1.6 Accuracy and sensitivity of natively implemented tools

To assess the accuracy and sensitivity of Sparkhit-recruiter and Sparkhit-mapper, we used public datasets from the Genome in a Bottle Consortium (GIAB) (Zook *et al.*, 2014) and simulated 6 datasets (doi: 10.4119/unibi/2914921) from three microbial reference genome templates (Ref-1, Ref-2 and Ref-3; Supplementary file S2, Table S2). In general, Sparkhit-recruiter has slightly higher accuracy than Fr-hit, Bowtie2 and SOAP, while having slightly lower accuracy than BWA (Fig. 2M–L and Supplementary file S2, Table S1). For sensitivity, Sparkhit-recruiter, Fr-hit, Bowtie2 and BWA are higher than SOAP on 100 nt simulated reads. For 150 nt simulated reads, all mappers have similar sensitivities. Sparkhit-mapper has slightly higher accuracy and sensitivity than Sparkhit-recruiter on the GIAB data.

3.2 Large scale genomic data analyses on the cloud

To demonstrate that large scale genomic analyses on the cloud can be easily accessible with Sparkhit, we tested our framework on 100 TB of genomic data from 3 genome projects and a transcriptomics study on the Amazon EC2 cloud (Fig. 4). These data were compressed and stored in different regions around the world on Amazon S3 and the European Nucleotide Archive (ENA) ftp server (Fig. 4F). We rented 100 c3.8xlarge Amazon EC2 instances (3200 cores, 6TB memory and 60TB disk space in total; see Supplementary file 1 Materials) with a total spot price around 38 dollars per hour. Sparkhit completed the entire process, including cluster deployment, data downloading, decompression and various data mining, in 21 h (the entire duration that the cloud provider charges) with a total cost less than 800 dollars (Fig. 4A–E).

We started deploying a Spark cluster of 100 worker nodes in Ireland region using Spark-ec2 script, which is the slowest one among three cluster deployment tools (the worst scenario for users' cloud budget). This step took 39 min and 54 s on the entire run time clock (Fig. 4A and Supplementary file S2, Table S3).

3.2.1 Processing all WGS data of the Human Microbiome Project

The Human Microbiome Project hosts 2.3 TB of compressed metagenomic whole genome shotgun (WGS) data on Amazon S3 located in the Oregon region. The data enables comprehensive characterization of the human microbiome and serves as a metagenomic database for microbiome studies. To associate these public datasets with microbial reference genomes, we downloaded all WGS data to the Spark cluster located in the Ireland region. After decompression, we recruited all sequencing reads to a collection of 21 microbial reference genomes (72 MB total, defined as Ref-2) and summarized the fragment recruitment results. We also profiled the taxonomy abundance using Sparkhit invoked Kraken (Wood and Salzberg, 2014). All processes were completed in 1:34:52 h (Fig. 4B and Supplementary file S2, Table S4).

3.2.2 Genotyping on 3000 samples of the 3000 Rice Genomes Project

The 3000 Rice Genomes Project hosts 200 TB of public data on Amazon S3 (Virginia region). This data enables large scale discovery of novel alleles for important rice phenotypes. Variant detection is a particularly expensive step in analyzing whole genome or exome sequencing data. To test the run time performance of Sparkhit on variant detection, we downloaded 15 TB BAM files to the Spark cluster. Using Sparkhit invoking Samtools-Mpileup (Li *et al.*, 2009), we genotyped 3000 rice samples and uploaded detected variants to Amazon S3. This analysis took 10:49:54 h (Fig. 4C and Supplementary file S2, Table S5).

3.2.3 Mapping 106 samples of the 1000 Genomes Project

The 1000 Genomes Project hosts all WGS data on Amazon S3 (Virginia region). It provides a detailed catalogue of human genetic variants in the studied populations. Before variant detection, a mapping step is required to present all matches and mismatches on the reference genome. To test the run time performance of Sparkhit on whole genome sequence mapping, we downloaded 5.6 TB compressed sequencing data to the Spark cluster. Using Sparkhit invoking BWA (Li and Durbin, 2009), we mapped 106 samples to a human reference genome. After sequence mapping, all results (SAM format) were uploaded to Amazon S3 for persistent storage. The entire process was completed in 5:31:17 h (Fig. 4D and Supplementary file S2, Table S6).

3.2.4 Gene expression profiling on prostate cancer RNA-seq data

Profiling tumor specific gene expression is a critical step for cancer research. To enable fast transcriptome quantification on the cloud, we tested the run time performance of Sparkhit on profiling 3.2 TB compressed RNA-seq data of a prostate cancer transcriptomics study (Wyatt *et al.*, 2014). Since all datasets are archived on the ENA ftp server (see Supplementary file 1 Materials), the downloading process consumed 1:29:28 h, whereas gene expression profiling with Sparkhit invoked Kallisto (Bray *et al.*, 2016) was completed in 21 min and 9 s (Fig. 4E and Supplementary file S2, Table S7). After uploading the final result to Amazon S3, we shut down all EC2 instances.

3.3 Recruiting HMP metagenomic data to microbial reference genomes

Metagenomics can capture and obtain genome fragments of uncultivated microbes (the microbial dark matter) by using shotgun sequencing to aggregated microorganisms sampled directly from the environment (Rinke *et al.*, 2013). The Human Microbiome Project applied metagenomic shotgun sequencing to characterize microbial communities at different human body sites. As all HMP data is hosted on Amazon S3, we can directly access and analyze these metagenomic datasets on the cloud.

To demonstrate that associating HMP metagenomic data with private datasets (e. g. cultured microbial genomes or uncultured single cell genomes) can provide more biological insights, we downloaded and recruited all HMP whole genome shotgun data (2.3 TB compressed) to 12 selected microbial genomes (Ref-1, see Supplementary file S2, Table S2). Based on the metadata of the HMP samples, we present the fragment recruitment profile of 7 different microbial genomes across 6 different body sites (15 sub-body sites) in Supplementary file S2, Figure S1. The abundance of each microbial genome was normalized and illustrated across different mapping identities (from 75 percent to 100 percent). In general, sub body sites of the same main body site have similar profiles, but with few exceptions. For instance, the abundances of *Neisseria meningitides* are different between saliva and gingival plaque from the oral body site. *Streptococcus aureus* has higher abundance in buccal mucosa, attached gingivae and saliva compared to other sub body sites in the oral body site. By changing the input reference genomes,

users can easily produce the abundance profile of other microbes (see discussion).

3.4 Functional study based on HMP fragment recruitment result

Mapping metagenomic sequences to a new strain could reveal potential structural variances. A common method for functional studies is based on the hypothesis that such structural variances can have functional impact on the gene level. When recruiting HMP data to the reference genomes, the recruitment result was sensitive enough to pick up structural differences between genome sequences of two *E.coli* strains (Supplementary file S2, Fig. S1A and B). We, then, followed up a functional analysis, gene prediction and pathway enrichment, based on the sequences (see Supplementary file 1 Method-1.10). We extracted all gaps on the *E.coli O104: H4* strain and predicted 158 new genes. These genes are enriched in two pathogenic pathways: Shigellosis and Pathogenic *E.coli* infection (Supplementary file S2, Fig. S2E–G). We annotated all genes and identified two toxic genes (Shiga toxin 2 subunit A and B) present in the *E.coli* strain that caused the 2011 German *E.coli* outbreak (Rasko *et al.*, 2011).

4 Discussion

In this study, we presented a Spark based distributed computational framework for large scale genomic analytics, called Sparkhit. Sparkhit incorporates a variety of tools and methods that are programmed in the Spark extended MapReduce model. We have described (i) the implementation of a fragment recruitment tool and a short-read mapping tool using Spark's RDD API, (ii) the construction of a general tool wrapper to invoke and parallelize external tools and (iii) the integration of Spark's machine learning library for downstream data mining. We also presented the architecture of Sparkhit and the utilities that we used for deploying Spark clusters and downloading public datasets. Sparkhit outperforms most Hadoop and Spark based bioinformatics tools in computational run time. Using our framework, we analyzed large amount of public genomic data on the cloud within a short time.

The performance benchmarks demonstrated the scalability of Sparkhit. Sparkhit-recruiter scaled linearly (Fig. 2E) with the increasing amount of input data, as we utilized Spark RDD to balance data distribution and optimized the computational parallelization. In addition, the distributed data I/O via HDFS further reduces latency. On HDFS, data is distributed and loaded locally or from the closest node (depending on the redundancy setting of HDFS), avoiding massive data transfer across the network. We also observed the advantage of using HDFS when comparing the run times of the machine learning library between the Amazon EC2 cloud and the private cluster, which stored input data on an NFS shared by all worker nodes. On NFS, all data was read and written through the network connection to a mounted volume that can saturate the bandwidth. When scaling out to more worker nodes, a slight slowdown was observed (Fig. 2F). The slowdown was caused by the overhead of building the reference index, which runs solely on the driver node. This can be improved by pre-building the reference index using our locally implemented recruiter (a Java-based tool included in our framework). Moreover, the overhead for constructing reference index is small compared to the run time of the fragment recruitment process.

Our tool had excellent run time performance on data preprocessing compared to Crossbow (18–32 times faster) and significant run time improvement on fragment recruitment compared to MetaSpark (92–157 times faster). Although we recruited 10–12% less reads than MetaSpark, we can adjust to a smaller K-mer size that recruits slightly more reads than MetaSpark, while still ran 47–124 times faster (Fig. 3). In addition, our tool has a reasonable accuracy and sensitivity on sequence mapping (Supplementary file S2, Table S1). Sparkhit-recruiter also offers more options for fragment recruitment, such as an option for reporting the best match for each read and an option to choose between global or local alignment, whereas MetaSpark can only apply local alignment.

Spark and Hadoop based bioinformatics tools are not widely used in genomic studies as they require users to comprehend certain amount of knowledge on cloud computing. Therefore, we particularly focus on providing a simple and comprehensive cloud application to directly access and analyze public genomic datasets. We described a simple way to setup a Spark cluster on the Amazon cloud with one line of command. We also facilitated downloading and preprocessing large amounts of data by leveraging distributed Amazon S3 storage and optimizing parallel data decompression. In addition, Sparkhit enables users to parallelize their own tools or public bio-containers. Our large scale data analyses on 100 TB data presented how we completed the entire cloud utilization cycle in only 21 h. Moreover, the fragment recruitment application on all WGS data of HMP was completed in less than 2 h on Amazon EC2.

The fragment recruitment application presented a study model that users can easily query the entire HMP data to a personalized reference dataset on the cloud. In microbial studies, combining metagenomic data with microbial reference genomes has been commonly used (Eloe-Fadrosh *et al.*, 2016). In our use case, we recruited all WGS data of the HMP to two different strains of *E.coli*: a toxic strain and a non-toxic strain. The intention is to find genome sequence segments that are not presented in the metagenomic samples. These sequence segments, which are unique for the toxic strain, might have functional impact that differs from the non-toxic one. We applied a functional analysis using the sequence segments and reproduced two toxic genes that caused the 2011 German *E.coli* outbreak. The same method can be applied to other isolates or single cell genomes.

Acknowledgements

We thank Georges Hattab for proof reading the preliminary manuscript. Gratitude to Raunak Shrestha and Dr. Faraz Hach for bringing insights to the project.

Funding

This project has been supported by the German-Canadian DFG international research training group 'Computational Methods for the Analysis of the Diversity and Dynamics of Genomes' (DiDy) GRK 1906/1. All Amazon cloud benchmarks and applications are funded by an Amazon research grant.

Conflict of Interest: none declared.

References

Abuin,J.M. *et al.* (2016) Sparkbwa: speeding up the alignment of high-throughput dna sequencing data. *PLoS One*, **11**, e0155461.

Auton,A. *et al.* (2015) A global reference for human genetic variation. *Nature*, **526**, 68–74.

Bao,R. *et al.* (2014) Review of current methods, applications, and data management for the bioinformatics analysis of whole exome sequencing. *Cancer Inf.*, **13**, 67–82.

Boisvert,S. *et al*. (2010) Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *J. Comput. Biol*., **17**, 1519–1533.

Bray,N.L. *et al*. (2016) Near-optimal probabilistic RNA-seq quantification. *Nat. Biotechnol*., **34**, 888–5257.

Chen,Y.-T. *et al*. (2015) Cs-bwamem: A fast and scalable read aligner at the cloud scale for whole genome sequencing. In: *High Throughput Sequencing Algorithms and Applications* (*HITSEQ*).

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

Decap,D. *et al*. (2015) Halvade: scalable sequence analysis with mapreduce. *Bioinformatics*, **31**, 2482–2488.

Droop,A.P. (2016) qsubsec: a lightweight template system for defining sun grid engine workflows. *Bioinformatics*, **32**, 1267–1268.

Eloe-Fadrosh,E.A. *et al*. (2016) Global metagenomic survey reveals a new bacterial candidate phylum in geothermal springs. *Nat. Commun*., **7**, 10476.

Gropp,W. *et al*. (1996) A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Comput*., **22**, 789–828.

Langmead,B. (2010) Aligning short sequencing reads with bowtie. *Curr. Protoc. Bioinf*., doi: 10.1002/0471250953.bi1107s32.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with bowtie 2. *Nat. Methods*, **9**, 357–359.

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

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

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, **25**, 1754–1760.

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

Li,R. *et al*. (2008) Soap: short oligonucleotide alignment program. *Bioinformatics*, **24**, 713–714.

McKenna,A. *et al*. (2010) The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome Res*., **20**, 1297–1303.

Merkel,D. (2014) Docker: lightweight linux containers for consistent development and deployment. *Linux J*., **2014**, 2.

Niu,B. *et al*. (2011) Fr-hit, a very fast program to recruit metagenomic reads to homologous reference genomes. *Bioinformatics*, **27**, 1704–1705.

Peterson,J. *et al*. (2009) The NIH human microbiome project. *Genome Res*., **19**, 2317–2323.

R Genomes Project. (2014) The 3,000 rice genomes project. *Gigascience*, **3**, 7.

Rasko,D.A. *et al*. (2011) Origins of the *E. coli* strain causing an outbreak of hemolytic-uremic syndrome in germany. *N. Engl. J. Med*., **365**, 709–717.

Rinke,C. *et al*. (2013) Insights into the phylogeny and coding potential of microbial dark matter. *Nature*, **499**, 431–437.

Rusch,D.B. *et al*. (2007) The sorcerer ii global ocean sampling expedition: northwest atlantic through eastern tropical pacific. *PLoS Biol*., **5**, e77.

Schadt,E.E. *et al*. (2010) Computational solutions to large-scale data management and analysis. *Nat. Rev. Genet*., **11**, 647–657.

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

Shvachko,K. *et al*. (2010) The hadoop distributed file system. In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST*), pp. 1–10.

Simpson,J.T. *et al*. (2009) Abyss: a parallel assembler for short read sequence data. *Genome Res*., **19**, 1117–1123.

Wood,D.E. and Salzberg,S.L. (2014) Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol*., **15**, R46.

Wyatt,A.W. *et al*. (2014) Heterogeneity in the inter-tumor transcriptome of high risk prostate cancer. *Genome Biol*., **15**, 426.

Zaharia,M. *et al*. (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, p. 15–28.

Zhao,G. *et al*. (2015) Sparksw: scalable distributed computing system for large-scale biological sequence alignment. In: *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (*CCGrid). IEEE, pages 845–852.

Zhou,W. *et al*. (2017) Metaspark: a spark-based distributed processing tool to recruit metagenomic reads to reference genomes. *Bioinformatics*, **33**, 1090–1092.

Zook,J.M. *et al*. (2014) Integrating human sequence data sets provides a resource of benchmark snp and indel genotype calls. *Nat. Biotechnol*., **32**, 246.