BIOINFORMATICS

Vol. 00 no. 00 2014
Pages 1–8

A quantitative assessment of the Hadoop framework for analyzing massively parallel DNA sequencing data

Alexey Siretskiy [1]*, Luca Pireddu [2], Tore Sundqvist [1] and Ola Spjuth [3,4]

[1]Department of Information Technology, Uppsala University, Uppsala, Sweden
[2]CRS4, Polaris, Pula, Italy
[3]Department of Pharmaceutical Biosciences, Uppsala University, Uppsala, Sweden
[4]Science for Life Laboratory, Uppsala University, Uppsala, Sweden

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: New high-throughput technologies such as massively parallel sequencing have transformed the life sciences into a data-intensive field. With increasing data volumes comes the necessity to analyze data in parallel using high-performance computing resources, but doing this effectively can be laborious and challenging.

Results: Hadoop is a framework that automatically distributes data and computation and has been shown to scale to thousands of nodes. Herein we introduce metrics and report a quantitative comparison of Hadoop to regular high-performance computing resources for aligning short reads and calling variants for five datasets of different sizes up to 250 gigabases. In order to increase performance of existing software and obtain a better comparison we modified and wrote new analysis scripts. From the observed scaling relations we are able to draw conclusions about the perspectives of the approaches, leading to the conclusion that as data set sizes reach 100 gigabases, the Hadoop-based pipelines become performance-competitive with a canonical high-performance cluster solution. As data sets in biological sequencing are sure to increase with time, Hadoop and similar frameworks are very interesting technologies that we envision will play a key role in the future of biological data analysis.

Availability: All code and data in this study is freely available as open source in public repositories.

Contact: alexey.siretskiy@it.uu.se

1 INTRODUCTION

Since its inception, massively parallel DNA sequencing, also referred to as Next Generation Sequencing (NGS) technology, has been an extremely bountiful source of data providing insight into the workings of biological machinery (Metzker, 2010; Marx, 2013). Decreasing sequencing costs facilitates and promotes larger and larger studies with increasingly larger data sizes. Extracting useful information from these voluminous amounts of data is transforming biology into a data-intensive discipline requiring a high-performance computing (HPC) infrastructure to produce results. As an example of the scale of the demands, consider that a single Illumina high-throughput sequencing run produces approximately 1800 gigabases (Gbases) corresponding to 2 terabytes (TB) of raw data in 3 days (http://www.illumina.com/systems/sequencing.ilmn).

A common step of NGS data analysis consists of aligning short reads to a reference sequence and then finding the genetic variations specific to the sample. Many of the most widespread tools for this purpose – like BWA (Li and Durbin, 2009), Bowtie (Langmead *et al.*, 2009b) and Samtools (Li *et al.*, 2009) – are not designed with distributed computing in mind. Many others do not even have the native ability to use multiple cores.

The most common approach to accelerate NGS tools is to parallelize within a compute node (multi-core parallelization) using shared memory parallelism (OMP) (http://openmp.org/), but this approach is naturally limited by the number of cores per node, which does not usually exceed 16 (http://www.top500.org/statistics/list/). For the tools which do not support OMP natively – e.g. Samtools – variant calling can be parallelized by creating a separate process for each chromosome or using the GNU Parallel Linux utility (Tange, 2011). It is important to note that a multi-core approach does not improve the performance of operations that are limited by local disk or network throughputs, whereas splitting the dataset and using multi-node parallelization is not generally bound by those constraints. A common way to implement multi-node parallelization is by using Message Passing Interface (MPI) (http://www.mcs.anl.gov/research/projects/mpi/). However, writing efficient MPI-programs for hundreds of cores is a non-trivial task since thread synchronization (or load balancing) has to be woven into the software code, and there are only a few existing solutions available for processing sequencing data (Clement *et al.*, 2010) (http://erne.sourceforge.net, http://bmi.osu.edu/hpc/software/pmap/pmap.html).

Another common way to introduce parallelization into NGS analysis pipelines in Linux systems is to use custom scripting in a language such as Bash, Perl, or Python. This involves using existing utilities and cluster tools to split the data into chunks, process them on the separate nodes, and merge the results afterwards. This kind of solution benefits from both MPI-like and OMP parallelization and provides good performance, but the development requires substantial expertise in order to be efficient. Since the process is tightly coupled to the local computational cluster and network architectures, it might also be nearly impossible to re-use such developments in other settings.

*to whom correspondence should be addressed

Table 1. Datasets used in the comparison.

Dataset	Organism	Size in Gbases
I	*A.thaliana*	1.4
II	*A.thaliana*	7.0
III	*A.thaliana*	30.0
IV	*A.thaliana*, the artificial dataset created using Samtools package	100.0
V	*H.sapiens*, two individuals (GM12750 and GM12004), sample SRR499924	250.0

The Map-Reduce (MR) programming paradigm (Dean and Ghemawat, 2004a) offers a compelling alternative for running tasks in a *massively* parallel way. This paradigm, however, shifts the focus from the best performance to scalability, suited for managing huge datasets of sizes up to several terabytes (Lin and Dyer, 2010). The most prevalent open source implementation of Map-Reduce is Hadoop (Dean and Ghemawat, 2004b; White, 2009). The Hadoop MR framework provides automatic distribution of computations over many nodes as well as automatic failure recovery (failure of individual jobs or computing nodes by storing multiple copies of data on different nodes), and automated collection of results (White, 2009). Hadoop Distributed File System (HDFS) is a complementary component that stores data by automatically distributing it over the entire cluster, writing data blocks onto the local disk of each node and therefore effectively enabling moving the computation to the data and thus reducing network traffic. HDFS provides a storage system where the bandwidth and size scales with the number of nodes in the cluster (Sammer, 2012), which is very different from the properties of the usual HPC cluster network architecture.

In this manuscript we focus on the question of *when* is Hadoop an appealing alternative to the programs for DNA-seq analysis generally found in HPC centers. Since the Hadoop framework poses a significant performance overhead, we seek to estimate the average data size at which point it starts to pay off from a performance perspective. We also study the analysis execution times for datasets of different sizes and evaluate Hadoop and the HPC approaches from a scaling perspective.

2 METHODS

Analysis pipelines

For our work to determine when Hadoop is an appealing option for genomic sequence analysis, we chose as a representative use case for our experiments the task of identifying single-nucleotide polymorphisms (SNPs) from short-read data. We constructed two pipelines that perform the same task: one that runs in an HPC environment and the other on the Hadoop platform. Although there are many different approaches and software tools to perform this task in an HPC environment (Li *et al.*, 2013) there is only a limited number of options for Hadoop. Given this limitation, we decided to keep the analysis pipeline as simple as possible to ensure a fair comparison between the two platforms.

1. HPC approach

   Short-read alignment: Bowtie ver. 0.12.8

   SNP calling: Samtools ver. 0.1.19

2. Hadoop approach: Crossbow

   Short-read alignment: Bowtie ver. 0.12.8

   SNP calling: SOAPsnp 1.02

In the HPC pipeline, reads were aligned with Bowtie (Langmead *et al.*, 2009b), followed by sorting the aligned reads and SNP calling with Samtools (Li *et al.*, 2009). The Bowtie aligner natively implements OMP – therefore, using 8 cores on the same computer will theoretically produce a result 8 times faster than using a single core. Likewise, Samtools (as of version 0.1.19) also offers shared memory parallelism for several of its functions. Where available these features were used to improve the analysis speed. The exact workflow used is available from Github (https://github.com/raalesir/HPC_bash_align).

The equivalent Hadoop-based pipeline was implemented with Crossbow (Langmead *et al.*, 2009a). The input data and the indexed genome reference were copied to Hadoop's storage system (HDFS) before starting the experiments. Specifically, the storage volume containing the input data was mounted with SSHFS to one of the Hadoop nodes, from where the `bzip2`-compressed data were transferred to the HDFS. Crossbow implements a short pipeline that pre-processes the input data, transforming it into a format suitable for the alignment stage, and then continues to use Bowtie for alignment and SoapSNP (http://soap.genomics.org.cn/soapsnp.html) to call SNPs. Unfortunately, unlike Crossbow's main program, its preprocessor does not scale well to large datasets (it is not implemented as a MR program). Due to this limitation, this basic step threatened to be the most time-consuming procedure in our test pipeline and bias our experiments. To overcome this bottleneck we substituted Crossbow's preprocessor with our own MR implementation (available from Github https://github.com/raalesir/mr_python). Our implementation is described in Section *MapReduce-based Data Preprocessor*.

Datasets

For our experiments we used three publicly available DNA-seq datasets (I–III), a synthetic dataset (IV) of *A.thaliana* – the well-known model plant – and a dataset (V) of two *H.sapiens* individuals (Table 1). Datasets I-III and V are composed of Illumina HiSeq sequencing data. Additional information about the source of the datasets is provided in the Supplementary Data.

All data was standardized to FASTQ format and compressed. The `bzip2` compression codec was selected as it provides very good compression and is splittable by Hadoop, meaning that the archive can be expanded in a parallel manner allowing Hadoop to process multiple segments simultaneously. Also, due to the lack of a standard FASTQ record id, which is required by our test pipelines to extract the read number, we also standardized the read ids in the datasets by generating new ids based on the SHA-1 hash function and labelled read mates with ".1" and ".2" suffixes.

MapReduce-based Data Preprocessor

The Crossbow package includes a native preprocessing stage that provides great flexibility in delivering the data to the Hadoop cluster; it can autonomously download data from Amazon S3, FTP and HTTP servers over the Internet (Langmead *et al.*, 2009a). However, this preprocessor does

Table 2. Time (in minutes) employed to preprocess Datasets II, III, and IV by Crossbow's native read preprocessor and our MR preprocessor on a 56-core Hadoop cluster, and by the custom multi-core Bash script on a single 16-core HPC node.

Dataset	II (7 Gbases)	III (30 Gbases)	IV (100 Gbases)
Crossbow native	60.6 ± 0.7	299.0 ± 2.3	673.0 ± 1.0
Hadoop, this work	7.0 ± 0.0	20.7 ± 0.4	52.4 ± 0.1
Bash, this work	7.4 ± 0.0	31.1 ± 0.1	114.5 ± 0.3

not parallelize its operations. In theory one can manually introduce some degree of parallelization by manually splitting the original input read data into smaller files and to explicitly configuring Crossbow's preprocessor to download each one (this is done by listing all of them in a special *manifest file*).

We reimplemented the main functionality in the Crossbow preprocessor as an MR program to benefit from its massively parallel nature. The new MR preprocessor was implemented in Python using the Hadoop streaming library. The resulting program efficiently processes short reads produced by different versions of the Illumina sequencing platform as well as FASTQ files generated from SRA files (NCBI Sequence Read Archive, http://www.ncbi.nlm.nih.gov/Traces/sra). However, the Hadoop-based approach imposes the requirement that the input data is stored where the Hadoop nodes can access it, which is in our opinion does not bias the comparison since sequencing platforms in the general case have direct access to storage and computing facilities.

Computational resources

To run the HPC analysis pipeline we used a computational cluster at UPPMAX, which is heavily used for analysis of NGS-data (Lampa *et al.*, 2013). Data and reference genomes were stored on a parallel, shared storage system. On the other hand, the Hadoop test platform was deployed on a private cloud at UPPMAX using the OpenNebula (http://opennebula.org) cloud manager. The cluster was configured with Cloudera Hadoop distribution version 2.0.0-mr1-cdh4.5.0 (http://www.cloudera.com/content/cloudera/en/why-cloudera/hadoop-and-big-data.html). For the fully detailed hardware configuration of the two computational resources please see the Supplementary Data.

3 RESULTS

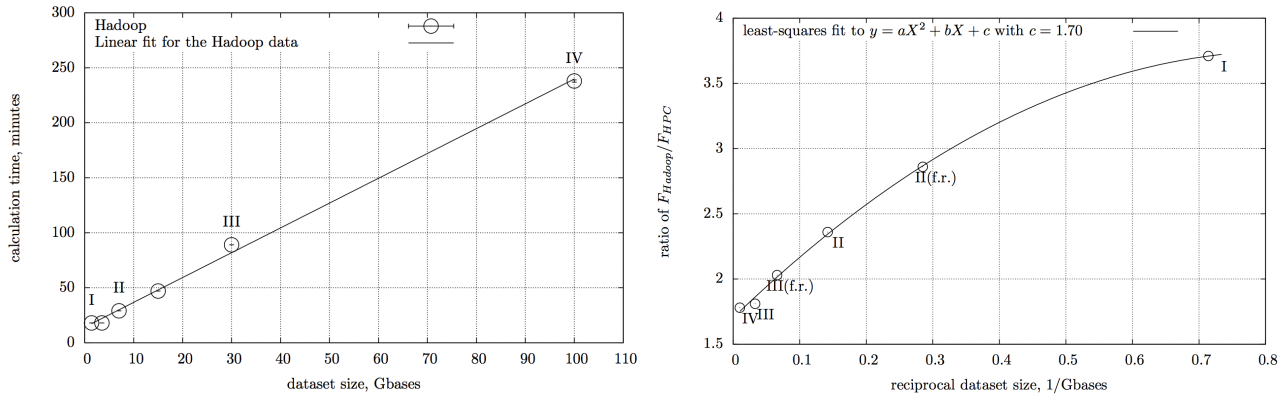
We executed the two pipelines for each of the five datasets on the HPC and Hadoop platforms, and measured the wall-clock run time for each pipeline stage. All experiments were repeated at least three times, and the results averaged to obtain a data point for the particular combination of data size and computational platform.

Accuracy of pipelines

Since our HPC and Hadoop pipelines use different SNP callers (Samtools and SOAPsnp) we could not expect them to deliver perfectly matching SNP lists. Instead, we verified the accuracy of the test pipelines by verifying their ability to correctly identify the mutation $C \rightarrow T$ on chromosome 4 at position 16702262 (Schneeberger *et al.*, 2009) in Dataset I.

Efficiency of preprocessing stage

To quantify the processing time saved by adopting the MR approach for preprocessing the read data, we ran an experiment comparing the new preprocessor described in Section *MapReduce-based Data Preprocessor* to the original preprocessor furnished with Crossbow.



(a) (b)

Fig. 1: **(a)** Runtime for the Hadoop-based pipeline on the 56-core Hadoop cluster with increasing input dataset size. Each test dataset (labelled with Roman numerals) was used. The image clearly highlights the linear scaling characteristics of the Hadoop-based approach.
(b) The ratio of the F_{Hadoop}/F_{HPC} as a function of a reciprocal dataset size in Gbases. The pipelines were run on a 56-core Hadoop cluster and a 16-core HPC node, respectively. The points are fit to a quadratic least-squares curve, which makes it possible to predict *infinite* dataset size. Datasets are labeled with Roman numerals; f.r. stands for "forward reads".

Table 3. Time (in minutes) for processing different dataset sizes on the HPC and Hadoop deployments with their respective pipelines. Datasets are labelled by Roman numerals, ("f.r." stands for "forward reads"). The large variance for the HPC deployment is due to Samtools BAM sorting, which is I/O and memory intensive; the data spillage to the file system makes it very susceptible to the load of the entire shared HPC cluster.

Data, Gbases	1.4 (I)	3.5 (II, f.r.)	7.0 (II)	15.0 (III, f.r.)	30.0 (III)	100.0 (IV)	250 (V)
Hadoop, 56 cores	18 ± 0	18 ± 0	29 ± 0	47 ± 0	89 ± 0	238 ± 1	1164 ± 14
HPC, 16 cores	17 ± 1	22 ± 6	43 ± 3	81 ± 9	172 ± 15	467 ± 60	> 48 hours

Table 4. Timings for Dataset I processed on different number of cores of a HPC node.

N cores	timing, minutes			speed-ups		
	alignment	SNP calling	total	total	alignment	SNP call
1	22.75 ±1.24	26.1±0.7	48.9	1.00	1.00	1.00
2	11.00±0.05	16.9±1.3	27.9	0.88	1.03	0.77
4	5.91±0.04	11.7±0.5	17.6	0.69	0.96	0.56
6	4.24±0.04	8.6±0.8	12.8	0.64	0.89	0.51
8	3.44±0.03	8.6±0.9	12.1	0.51	0.83	0.38
10	2.91±0.01	7.7±0.5	10.6	0.46	0.78	0.34
12	2.64±0.04	7.5±0.5	10.1	0.40	0.72	0.29
14	2.57±0.03	7.5±0.6	10.1	0.35	0.63	0.25
16	2.88±0.06	7.4±0.4	10.2	0.30	0.49	0.22

To further validate our strategy, we also compared the performance of our MapReduce approach to a different parallelization strategy based on a custom Bash script that uses all processing cores available on a single computing node to accelerate the operation (see Supplementary Data for details of the implementation). This program read its input directly from conventional HPC storage, rather than HDFS. The experiment was run on Datasets II, III, and IV. The Hadoop cluster used was configured with 56 cores (7 slave nodes) while the Bash approach ran on a single node equipped with 16 CPU cores.

Table 2 shows the results of the experiment, and we see that all of of the parallel approaches are much faster than the original tool, for all dataset sizes. Also, we noticed that the difference between the Bash and Hadoop approaches is minimal when running on Dataset II. Indeed, with the relatively small dataset size the overhead incurred by the Hadoop framework likely overwhelms the useful compute time used for each available CPU core. However, with increasing input size the superior scalability of MR becomes apparent, with the largest dataset requiring less than half the time of the Bash approach to be processed. At the same time, it is important to also notice that the Hadoop approach employed approx. 1.6 times more CPU time compared to the Bash approach.

Scalability

Given the wide range in genome dataset sizes processed in practice – e.g. exomes and whole-genome sequencing at various depths – it is important to understand the scaling characteristics of the Hadoop and HPC processing platforms with respect to input size. For each platform, we measured the time required to execute their respective variant calling pipeline for each of the test datasets (I-V). The results are shown in Table 3.

The measurements highlight that one of the advantages of using the Hadoop framework is its almost linear scalability with respect to input size – i.e. the calculation time linearly depends on the size of the dataset, regardless the scaling nature of the underlying program (Langmead *et al.*, 2009a; Pireddu *et al.*, 2011). Figure 1a shows the wall clock running time as a function of the dataset size for the 56-core Hadoop cluster and datasets I-IV.

To investigate the scalability characteristics of the HPC approach we ran the HPC pipeline using Dataset I as input and collected the running times as a function of the number of cores used (Table 4). The results show that the alignment process with Bowtie scales fairly well, but overall the pipeline suffers because of the SNP calling step with Samtools. Indeed, this operation is the bottleneck of the workflow as it scales worse than Bowtie and consumes a progressively larger portion of the total calculation time as the input grows – even after parallelizing analysis by chromosome as previously suggested (http://www.biostars.org/p/48781).

Efficiency of calculations

Since Hadoop was designed to digest huge datasets (Dean and Ghemawat, 2004b; Lin and Dyer, 2010) we hypothesize that the larger the biological sequence dataset is, the more suitable Hadoop becomes compared to the HPC approach. In order to compare the 'suitability' for the Hadoop and HPC platforms run on different number of cores, we construct the function showing the amount of used core-hours: $F = T_p \times p$, where T_p is the calculation time on p cores.

Using the data from Table 3, we plot the ratio F_{Hadoop}/F_{HPC} as a function of the reciprocal dataset size (Figure 1b). Extrapolation to zero on the X-axis estimates the ratio for the hypothetical infinite dataset. We note that the Hadoop approach becomes more and more effective compared to the HPC scenario as the dataset size

Table 5. Time (in minutes) required by Samtools to sort Dataset IV (BAM file containing 58 Gbases). The measurements are done on a single HPC node with $p = 16$ cores. Averaging is done over 5 independent simulations. The amount of available RAM reported is for all 16 cores.

RAM, GB	8	16	32	64
timing, minutes	208 ± 50	172 ± 13	136 ± 12	87 ± 11

increases. The parabolic extrapolation for our settings give the ratio as 1.70 ± 0.01, meaning that Hadoop running even in the virtualized environment of a private cloud assembled on moderate hardware, is competitive with the HPC approach run on "bare metal" of modern hardware for datasets larger than 100 Gbases (dataset IV), which is typical for human genome sequencing with a sequencing depth of about $30x$.

The simplest curve to fit the points nicely is a parabolic function, importantly not a constant. Since the Hadoop approach for the datasets reveals the linear scaling (Figure 1a), a deviation of the ratio from the constant shows the lack of scalability of the HPC approach. We suspect that the main reason for such a behavior is that for large datasets (Dataset III-V, in our case) Samtools' SNP calling routine becomes more memory- and time-demanding than the alignment with Bowtie. Since Samtools memory requirements scale with input size, the application swaps data to disk for large input data creating multiple (up to hundreds) temporary files while it sorts the BAM file. These files can end up on shared network-accessed file systems and thus generate large amounts of network traffic. Table 5 shows the time required by Samtools to sort dataset IV with different amount of RAM available on a single HPC node. Though there is a large deviation when using small amounts of RAM – we believe due to other users' load on the shared cluster network – the trend clearly indicates that a relatively large amount of RAM per node is needed in order to use Samtools effectively. In contrast, our tests Hadoop performed the SNP calling on much more economic hardware in linear time with just 12GB of RAM per virtual node.

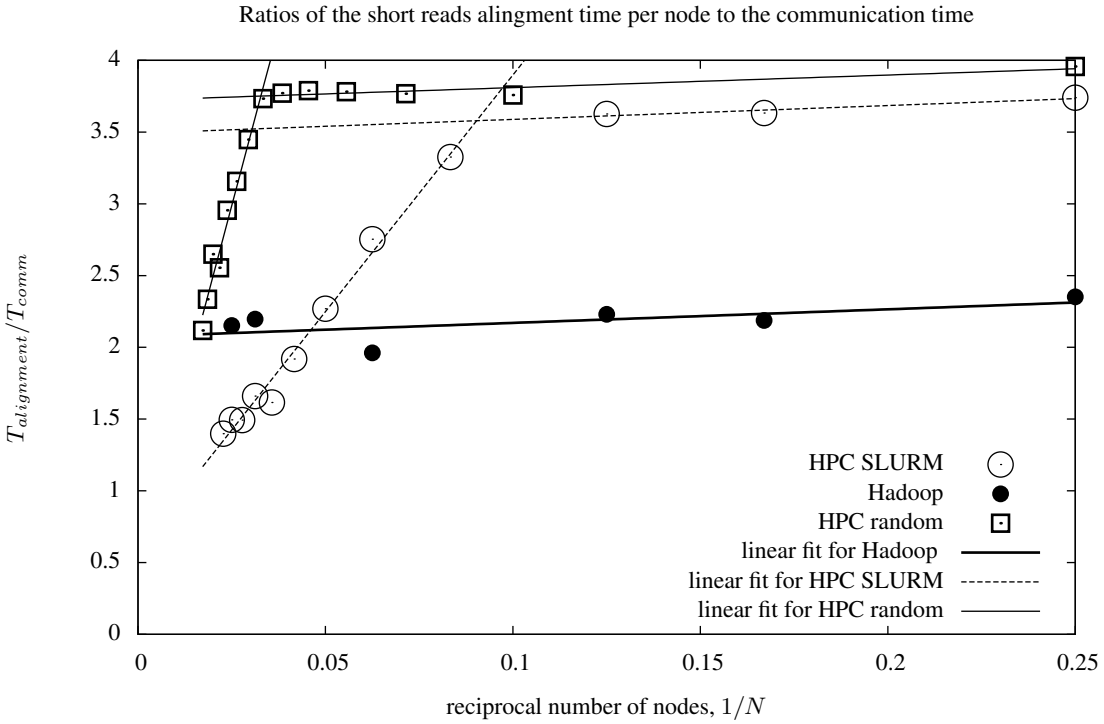


Fig. 2: Ratios of alignment time $T_{alignment}$ to communication costs T_{comm} for the HPC and Hadoop clusters as a function of reciprocal number of nodes $1/N$. All runs used Dataset IV as input. Two HPC scenarios are shown as "HPC SLURM" and "HPC random", which correspond to standard SLURM behavior and a modified one that allocates nodes from random racks. Linear fit to $ax + b$ was done with the least-squares method. The estimated coefficients for the Hadoop-based approach are $(a, b) \approx (0.95, 2.07)$. One can see two defined linear regions for the HPC approach with very different tangents for linear and non-linear scalings correspondingly. For the "HPC SLURM" $(a, b) \approx (0.96, 3.49)$ and $(a, b) \approx (32.99, 0.60)$, and for the "HPC random" $(a, b) \approx (0.88, 3.72)$ and $(a, b) \approx (98.24, 0.53)$. The constant term for HPC goes below unity – i.e. for a large number of nodes more time will be spent on communication than the actual alignment.

Table 6. Timings for alignment and the ratio $T_{alignment}/T_{comm}$ for HPC and Hadoop clusters for Dataset IV. For the "HPC random" approach, data chunks have to be copied to the local scratch disks first and the alignments (SAM files) copied back while Hadoop keeps all the data inside HDFS and hence does not need data staging. Hadoop however needs to preprocess reads before the actual alignment stage in order to be able to operate in MR manner resulting in what we term "communication costs". Note that each HPC node has 16 cores, while each Hadoop node has 7 (one core is dedicated to run the virtual machine).

Hadoop			HPC random		
Number of nodes (cores)	Alignment time, minutes	$\frac{T_{alignment}}{T_{comm}}$	Number of nodes (cores)	Alignment time, minutes	$\frac{T_{alignment}}{T_{comm}}$
4(28)	293.5	2.33	4(64)	74.4	3.89
6(42)	189.8	2.19	10(160)	32.4	3.76
8(56)	136.0	2.23	14(224)	22.7	3.77
16(112)	70.3	1.96	18(288)	17.9	3.78
32(224)	39.3	2.20	22(352)	14.5	3.79
40(280)	32.5	2.15	26(416)	12.3	3.77
			30(480)	10.7	3.73
			34(544)	9.5	3.45
			38(608)	8.5	3.16
			42(672)	7.6	2.96
			46(736)	7.0	2.55
			50(800)	6.4	2.65
			54(864)	5.9	2.34
			58(928)	5.5	2.12

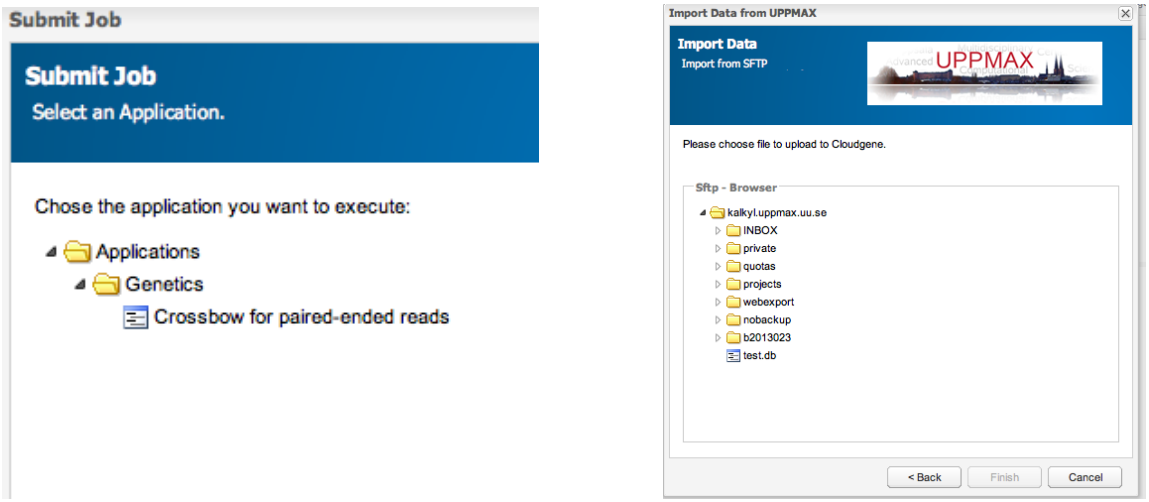
Efficiency of network communication

The network communication model for Hadoop has a major difference from the usual HPC cluster network architecture (n-tier tree) with NAS or NFS attached storages. The effective bandwidth of the Hadoop network increases with the cluster size (Sammer, 2012), opposite to that of the HPC network where cluster growth results in network saturation and performance degradation. To study this phenomenon, we compared HPC and Hadoop network communication costs as a function of the number of nodes involved in a computation on a fixed dataset size (58 Gbases, dataset IV). Since the alignment process is embarrassingly parallel – the read-pairs are independent of each other and thus can be aligned independently – it is possible to distribute the work over more than one computational resource, e.g. by splitting the initial data into chunks to processing them in parallel. In this approach, reducing the size of each data chunk reduces the size of the aligner job, $T_{alignment}$, but at the same time more chunks have to be sent over the network increasing the communication time, T_{comm}.

There are several program packages for short-read alignment with MPI support (Clement *et al.*, 2010) (http://bmi.osu.edu/hpc/software/pmap/pmap.html) and almost linear scaling up to 32 nodes for pair-ended reads has been reported (Bozdag *et al.*, 2010). We noted however that the Pmap package functions poorly for datasets larger than 20 Gbases, and instead implemented a highly optimized Bash script for alignment based on standard Unix utilities that makes very efficient use of the HPC cluster network (source code available on Github https://github.com/raalesir/HPC_bash_align). We then compared the network performance with the standard Hadoop HDFS approach. We separated alignment time, $T_{alignment}$, and communication time, T_{comm}, and plotted the ratio $T_{alignment}/T_{comm}$ as a function of the reciprocal number of nodes $1/N$ (Figure 2). This measure is applicable to both HPC and Hadoop, however T_{comm} has a different explanation: For Hadoop, the short reads in FASTQ format have to be preprocessed to be able to run in MR-fashion, while the data locality will be automatically achieved during the data ingestion into HDFS. For the HPC approach, T_{comm} involves the chunks that are sent from the sequence delivery location to the local node scratch disks where the actual alignment happens, and the transfer of the aligned SAM files back to the delivery location over the network.

Hadoop results (filled circles in Figure 2): The ratio $T_{alignment}/T_{comm}$ indicates weak dependency in a wide range of number of nodes N: from 4 up to 40. It is known that Bowtie provides almost linear scaling between alignment time and dataset chunk size D: $T_{alignment} \propto D \propto 1/N$, see (Langmead *et al.*, 2009b), and our experiments summarized in Figure 1a. Since the ratio $T_{alignment}/T_{comm}$ is approximately constant, one can conclude that $T_{comm} \propto 1/N$, meaning that the more nodes involved, the faster the communication is in the preprocessing stage.

HPC results (open circles in Figure 2): The data from the delivery location is split into chunks in parallel, which are simultaneously pushed to the local scratch disks of the nodes allocated by SLURM. One can see two distinct linear stretches. One stretch is for the range from 4 to about 12 nodes, and the another is from 12 up to 60. The former (horizontal stretch) is explained as for Hadoop – the more nodes involved, the faster the chunks are being distributed. The latter stretch with the positive slope could be explained as follows: In the region of about 12 nodes the network becomes saturated and unable to pass more data in a time unit, while the alignment time is still proportional to the chunk size: $T_{comm} \approx \text{const}, T_{alignment} \propto D \propto 1/N \rightarrow T_{alignment}/T_{comm} \propto 1/N$, i.e., a linear dependency in the selected coordinates, which can be observed in the plot. The transition area between the two modes is based on the fact that each hard disk drive on the local node can write the data at a speed of about 1Gbit/sec. Ten nodes will consume the data with the rate of 10Gbit/sec, which is the limiting speed for the standard 10Gbit Ethernet cable connecting the cluster's rack with the switch. The



(a) (b)

Fig. 3: An example of a job setup view with the graphical Hadoop front-end Cloudgene, providing a smooth user experience even for novice users. **a)** Pipeline selection – in our case containing the Crossbow pipeline. **b)** The UPPMAX-adapted functionality to browsing and import data from the user's home folder in the shared file system.

nodes are being allocated on the same rack, which is the default SLURM behavior.

Scalability can be improved by overriding the default behavior of SLURM and allocating the nodes not from the same rack, but randomly from all available racks ("HPC random", open squares in Figure 2). Allocating the nodes on random racks allows for engaging more nodes without network saturation. For our cluster we could go up to 30-35 nodes with perfect linear scaling. For the most resources used (58 nodes) the deviation from a linear speedup is $\approx 7\%$ i.e. 5.50 minutes against the ideal 5.14, see Table 6. The threshold number of nodes in this strategy (≈ 35) is because of the uplink cable with the throughput of 50Gbit/sec being saturated. The proposed HPC strategies aimed at getting the maximum performance from the storage resources show that while even properly adjusted and tuned, the HPC approaches suffer from the network saturation at higher number of nodes. HDFS on the other hand maintains data locality, reducing communication and data transfer and hence lead to better scalability. Our Hadoop cluster has no more free nodes to continue to investigate the scaling as in plot at Figure 2, but we do not expect any significant deviations, since the observed behavior is a generic for Hadoop with HDFS(Lin and Dyer, 2010; White, 2009).

Usability aspects

Hadoop presents a computing paradigm that is different from what most users are accustomed to using. Therefore, adopting it requires an investment in learning how it works, how to use it, and how to understand what the programs running on the platform are doing. A popular way to define and run bioinformatics pipelines on HPC and cloud resources is via the Galaxy(Giardine *et al.*, 2005; Afgan *et al.*, 2010) platform, which provides a Web-based graphical user interface (GUI) to bioinformatic programs, simplifying the experience for the end user. A similar project is Cloudgene (Schönherr *et al.*, 2012), which provides a GUI to help lower the learning curve for Hadoop adopters and is specifically targeted at users in the life sciences. It consists of a light-weight and flexible Web-based solution for both public and private clouds. We implemented our Hadoop-pipeline in Cloudgene and extended the platform with functions to import data from the central file system at UPPMAX into HDFS on the private cloud (Figure 3). For our particular task related to DNA sequencing, Cloudgene makes it easy to select data and parameters for its execution. Most of the data management work is done automatically and the results can be downloaded to the client machine. The modular structure allows modification of the source code to adapt to the existing computing center architecture. For example, UPPMAX users can import their data from the sequencing platform directly to the Hadoop cluster by pressing a button and entering the credentials; at the same time they can be sure that their sensitive data is kept private.

4 DISCUSSION

In this report we have described two approaches for high-performance analysis of DNA sequencing data; one based on regular HPC using a batch system and one based on Hadoop. In order to compare the two platforms we developed highly optimized pipelines for both scenarios and found that Hadoop becomes comparable with HPC when processing datasets larger than 100 Gbases. As this corresponds to typical human genome data with a sequencing depth of about $30x$, we consider Hadoop not only to be a technology for the future but an attractive option for bioinformatics data analysis already today. While implementations on HPC are more efficient on the use of CPU time than the Hadoop-based approach, the price paid for this is increased overall runtime. The

main benefit of Hadoop is however its almost linear scaling with the problem size, and we show that this also holds for analysis of NGS data. Our results further reveal that the calculations on Hadoop with the file system HDFS scales better than the network attached parallel storage commonly used in the HPC centers. Finally, we demonstrated that an existing and extensible publicly available web-based GUI (Cloudgene) provides an easy way to execute bioinformatics analysis on Hadoop for those who are less experienced with Linux scripting.

Hadoop has so far seen relatively low adoption in bioinformatics. This is likely due to that existing high-performance infrastructure in most cases are built as traditional HPC systems with batch processing queues. Also, the bioinformatics software flora is still not as big for Hadoop and most analysis tools are implemented as regular linux applications. This may very well change soon with the demonstrated scalability and wide uptake of Hadoop in other settings. The results in this paper show that Hadoop enables highly scalable massively parallel analysis of DNA sequence data, and as the size of data sets are expected to increase over time we envision that Hadoop will become increasingly popular also in bioinformatics to deliver results faster.

ACKNOWLEDGEMENT

We thank system experts Pontus Freyhult, and Peter Ankerstål at UPPMAX for valuable discussions on effective storage and network usage. We also thank Jonas Hagberg (BILS, Stockholm, Sweden) for implementing the Cloudgene extensions to import data from UPPMAX filesystem, as well as Wesley Schaal for proofreading.

Funding: This work was supported by SNIC through Uppsala Multidisciplinary Center for Advanced Computational Science (SNIC-UPPMAX) [p2013023]; the Swedish strategic research programme eSSENCE; and COST Action BM1006 Next Generation Sequencing Data Analysis Network SeqAhead. Part of L.P.'s activity was performed within the context of the Ph.D. program in Biomedical Engineering at the University of Cagliari, Italy.

REFERENCES

Afgan, E., Baker, D., Coraor, N., Chapman, B., Nekrutenko, A., and Taylor, J. (2010). Galaxy cloudman: delivering cloud compute clusters. *BMC Bioinformatics*, **11 Suppl 12**, S4.

Bozdag, D., Hatem, A., and Catalyurek, U. V. (2010). Exploring parallelism in short sequence mapping using Burrows-Wheeler Transform. *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8.

Clement, N. L., Snell, Q., Clement, M. J., Hollenhorst, P. C., Purwar, J., Graves, B. J., Cairns, B. R., and Johnson, W. E. (2010). The GNUMAP algorithm: unbiased probabilistic mapping of oligonucleotides from next-generation sequencing. *Bioinformatics*, **26**(1), 38–45.

Dean, J. and Ghemawat, S. (2004a). MapReduce: simplified data processing on large clusters. In *OSDI '04: 6th Symposium on Operating Systems Design and Implementation*.

Dean, J. and Ghemawat, S. (2004b). MapReduce: Simplified data processing on large clusters. *Sixth Symposium on Operating System Design and Implementation: 2004; San Francisco, CA*.

Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., Miller, W., Kent, W. J., and Nekrutenko, A. (2005). Galaxy: a platform for interactive large-scale genome analysis. *Genome Res*, **15**(10), 1451–5.

Lampa, S., Dahlö, M., Olason, P., Hagberg, J., and Spjuth, O. (2013). Lessons learned from implementing a national infrastructure in Sweden for storage and analysis of next-generation sequencing data. *GigaScience*, **2**(1), 9.

Langmead, B., Schatz, M. C., Lin, J., Pop, M., and Salzberg, S. L. (2009a). Searching for SNPs with cloud computing. *Genome Biol*, **10**(11), R134.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009b). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, **10**(3), R25.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**(16), 2078–2079.

Li, Y., Chen, W., Liu, E. Y., and Zhou, Y.-H. (2013). Single Nucleotide Polymorphism (SNP) Detection and Genotype Calling from Massively Parallel Sequencing (MPS) Data. *Stat Biosci*, **5**(1), 3–25.

Lin, J. and Dyer, C. (2010). *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool Publishers.

Marx, V. (2013). Biology: The big challenges of big data. *Nature*, **498**(7453), 255–60.

Metzker, M. L. (2010). Sequencing technologies – the next generation. *Nat Rev Genet*, **11**(1), 31–46.

Pireddu, L., Leo, S., and Zanetti, G. (2011). SEAL: a distributed short read mapping and duplicate removal tool. *Bioinformatics*, **27**(15), 2159–60.

Sammer, E. (2012). *Hadoop Operations*. O'Reilly Media, Inc., 1st edition.

Schneeberger, K., Ossowski, S., Lanz, C., Juul, T., Petersen, A. H., Nielsen, K. L., Jorgensen, J.-E., Weigel, D., and Andersen, S. U. (2009). SHOREmap: simultaneous mapping and mutation identification by deep sequencing. *Nat Meth*, **6**(8), 550–551.

Schönherr, S., Forer, L., Weissensteiner, H., Kronenberg, F., Specht, G., and Kloss-Brandstatter, A. (2012). Cloudgene: a graphical execution platform for MapReduce programs on private and public clouds. *BMC Bioinformatics*, **13**, 200.

Tange, O. (2011). Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, **36**(1), 42–47.

White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly, first edition edition.