High performance framework to analyze microarray data

Fabrizio Marozzo[1], Loris Belcastro[1]

University of Calabria [fmarozzo,lbelcastro]@dimes.unical.it

Summary. Pharmacogenomics is an important research field that studies the impact of genetic variation of patients on drug responses, looking for correlations between Single Nucleotide Polymorphisms (SNPs) of patient genome and drug toxicity or efficacy. The large number of available samples and the high resolution of the instruments allow microarray platforms to produce huge amounts of SNP data. To analyze such data and find correlations in a reasonable time, high-performance computing solutions must be used. Cloud4SNP is a bioinformatics tool, based on Data Mining Cloud Framework (DMCF), for parallel preprocessing and statistical analysis of SNP pharmacogenomics microarray data.

This work describes how Cloud4SNP has been extended to execute applications on Apache Spark, which provides faster execution time for iterative and batch processing. The experimental evaluation shows that Cloud4SNP is able to exploit the high-performance features of Apache Spark, obtaining faster execution times and high level of scalability, with a global speedup that is very close to linear values.

Key words: Pharmacogenomics, Single Nucleotide Polymorphisms, Statistical Analysis, Cloud Computing

Running head: High performance framework to analyze microarray data

1 Introduction

In recent years there has been a growing interest about the biomedical *omics* disciplines, such as genomics, proteomics, and interactomics. In particular, genomics is the study of the activity of genes, proteomics focuses on the activity of proteins, and interactomics concerns the study of protein interactions inside a cell [10].

In a typical case-control study, microarray technology is able to measure the expression level of genes present in biological samples, which results in a matrix of real numbers where the value (i, j) represents the expression of the gene i on the sample j. More recently, genotyping microarrays make it possible to detect the genetic variant between samples, i.e., to detect variations in nucleotides with respect to a reference population.

A nucleotide variation or Single Nucleotide Polymorphism (SNP) is defined as a stable substitution of a single base of DNA[1] occurring with a frequency of more than 1% in at least one population. For instance, in the short sequences ATGT and ACGT a base change occurs at position 2. Each individual has a unique sequence of DNA that determines his/her characteristics and differences can be measured in terms of substitutions of bases in the same position. In a case-control genotyping study, microarray technology produces a matrix of SNPs where the (i, j) value represents the SNP found on the DNA sequence or gene i on sample j.

Pharmacogenomics is the branch of genomics that studies the impact of individual genetic variations on the response to drugs. In particular, pharmacogenomics correlates gene expression or SNPs with the toxicity or efficacy of a drug, with the aim of improving drug therapies according to the patients' genotype (e.g., choosing drugs that best match each patient's genetic profile) [20]. Pharmacogenomics experiments involve gene sequencing and SNP detection using microarray technology and computational analysis. DMET (Drug Metabolism Enzymes and Transporter) is an Affymetrix[2] microarray platform for gene profiling specifically designed to detect in human samples the presence/absence of SNPs on 225 genes that are related with drug absorption, distribution, metabolism and excretion (ADME) [9]. SNPs data generated by the DMET platform are preprocessed and analyzed for discovering a correlation between the presence/absence of SNPs and condition of samples (e.g., the type of drug treatment or response to a drug). Main issues in analyzing DMET SNPs data are: *i*) the efficient management of huge data due to the high number of samples and genes investigated in pharmacogenomics studies; *ii*) the fast analysis of SNP symbolic data, which need to undergo different preprocessing compared to gene expression numeric data; *iii*) obtaining a result that is biologically interpretable.

The huge amount of data generated represent a challenge to the current process and analysis capabilities. Then to extract results in a reasonable time from such kind of data, novel frameworks and data analysis approaches have been developed. Moreover, the usage of sequential algorithms for analyzing large volumes of data requires a very long time for extracting useful models and patterns. In this scenario, high performance computers, such as many and multi-core systems, Clouds and clusters, paired with parallel and distributed algorithms, are commonly used [7]. Nowadays several programming frameworks are available for developing computing intensive applications to be executed in distributed environments. Among open-source projects, Apache Hadoop[3] and Apache Spark[4] are considered the leading open-source data-processing frameworks, which was contributed by IT giants such as Facebook and Yahoo. For several classes of applications, Spark is considered a better alternative to Hadoop, since it stores data in RAM memory and queries it repeatedly so as to obtain better performance [6].

Cloud4SNP [2] is a parallel version of the DMET-Analyzer [12] that exploits data parallelism to perform statistical tests on multiple computing nodes on the Cloud. This paper presents an extension of Cloud4SNP for enabling the execution of

[1] DNA is made up of four sub-units, or bases, called adenine (A), cytosine (C), guanine (G) and thymine (T).
[2] www.affymetrix.com
[3] http://hadoop.apache.org/
[4] http://spark.apache.org/

data analysis applications based on Apache Spark, which provides faster, in-memory executions for iterative and batch processing. The experimental evaluation shows that Cloud4SNP is able to exploit the high-performance features of Apache Spark, obtaining faster execution times and high level of scalability, with a global speedup that is very close to linear values.

The rest of the paper is structured as follows. Section 2 describes the main tools for analyzing SNP genotyping data for pharmacogenomics, including the most popular frameworks for designing and executing parallel and distributed data analysis applications. Section 3 discusses the main functionalities and implementation details of Cloud4SNP. Section 4 presents the performance evaluation of Cloud4SNP. Finally, Section 5 concludes the paper.

2 Related work

Microarray technology is used in biology and medicine to study the behavior of genes in biological processes, with the aim of understanding how their behavior can be related to disease progression. Microarray technology includes two main categories of microarray chips: *i*) *expression microarrays that aim to investigate the activity of genes* in different conditions, and *ii*) *genomic microarrays* (e.g., DMET arrays) that aim to study the variations on sequence of genomes.

In recent years, the average size of a dataset from microarray platforms has been growing steadily. This trend is mainly due to two main factors: *i*) the increase of size of files that can be encoded on a single chip; and *ii*) the increasing number of samples (and therefore of arrays) that are usually produced in a single experiment. Therefore, the increase in the size of the datasets to be analyzed, and in the complexity of the analyzes to be carried on, give rise to the need to use high-performance tools and technologies for processing such data efficiently. The usage of parallel and distributed algorithms on multicore architectures could lead to an efficient preprocessing of microarray data. In such a way, the computation is distributed across several CPU cores, which perform a parallel computation on smaller data chunks. As first step in this direction, the MapReduce programming models can be exploited to perform the preprocessing algorithm on different data chunks that are distributed to the different computing nodes. However, partitioning and distributing data to nodes can often be inefficient due to network overhead.

One of the main research works is affyPara [1] that is a Bioconductor[5] package for parallel preprocessing of Affymetrix microarray data. It is freely available from the Bioconductor project. Similarly, the micro-CS project [13] presents a framework for the analysis of microarray data based on a distributed architecture made of different web-services for annotating and preprocessing data.

EMAAS (Extensible MicroArray Analysis System) [3] is a web-application for the management and analysis of Affymetrix arrays. It is based on a high performance computing architecture that uses Grid technology for the analysis of large datasets. Among its main characteristics, EMAAS *i*) is able to process only a subset of Affymetrix Expression arrays; *ii*) supports collaboration among users; and *iii*) requires the upload of data onto the web server, so it may require large upload time

[5] http://www.bioconductor.org/

when Internet speed is not sufficient or it may cause legal problems for SNP data in some countries.

Since most biology labs or clinical centers that use microarrays do not have parallel or HPC computers, parallelizing the microarray data analysis pipeline is not enough. The use of Cloud computing systems can also offer small research groups the possibility of exploiting parallel bioinformatics tools. Cloud computing is a valid and cost-effective solution for executing complex data analysis applications [22]. In fact, thanks to elastic resource allocation and high computing power, Cloud computing represents a compelling solution for faster data analysis, that means more timely results and then greater data value. Actually, despite the Cloud is an affordable solution for many users, the number of data processing framework available is very limited. Most available solutions today are based on open source frameworks, such as Apache Hadoop and Apache Spark.

Hadoop is the most used open source MapReduce implementation for developing parallel applications that analyze big amounts of data. It can be adopted for developing distributed and parallel applications using many programming languages (e.g., Java, Ruby, Python, C++). Hadoop relieves developers from having to deal with classical distributed computing issues, such as load balancing, fault tolerance, data locality, and network bandwidth saving.

Apache Spark is another framework for Big Data processing. Differently from Hadoop, in which intermediate data are always stored in distributed file systems, Spark stores data in RAM memory and queries it repeatedly so as to obtain better performance for some class of applications (e.g., iterative machine learning algorithms). For many years, Hadoop has been considered the leading open source Big Data framework, but recently Spark has become the more popular so that it is supported by every major Hadoop vendors. In fact, for particular tasks, Spark is up to 100 times faster than Hadoop in memory and 10 times faster on disk.

The use of Cloud and of high-level programming are at the basis of the Data Mining Cloud Framework (DMCF) [19], i.e., the data analysis framework used to implement Cloud4SNP applications. DMCF enables the design and the execution of distributed data analysis workflows on Clouds by using a high-level visual language (VL4Cloud) or a script-based language (JS4Cloud). Recently, DMCF has been extended to include the execution of MapReduce tasks [5].

There are other representative workflow-based systems that can be used to implement applications for analyzing large amount of data on parallel and distributed computing systems. Swift [23] is a system for creating and running distributed workflows, which uses an implicit data-driven task parallelism. In fact, it looks like a sequential language, but all variables are futures, thus the execution is based on data availability (i.e., when the input data is ready, functions are executed in parallel).

COMPSs [14] is another workflow system that aims at easing the development and execution of workflows in distributed environments, including Grids and Clouds. With COMPSs, users create a Java sequential application and select which methods will be executed remotely by providing an annotated interfaces. The runtime intercepts any call to a selected method creating a representative task and finding the data dependencies with all the previous ones that must be considered along the application run [18].

Pegasus [11] is a workflow management system developed at the University of Southern California for supporting the implementation of scientific applications

also in the area of data analysis. Pegasus includes a set of software modules to execute workflow-based applications in a number of different environments, including desktops, Clouds, Grids, and clusters. It has been used in several scientific areas including bioinformatics, astronomy, earthquake science,gravitational wave physics, and ocean science.

3 Cloud4SNP

Cloud4SNP allows to statistically test the significance of the presence of SNPs in two classes of samples using the well known Fisher test. The Fisher test allows to test if two distributions are significantly different, i.e. the eventual difference is not due by chance. In particular, Cloud4SNP performs the following main steps: *i*) loading of the input dataset and sample class assignment; *ii*) execution of statistical tests (e.g., Fisher test); and *iii*) statistical correction of p-values.

3.1 Loading of the input dataset and sample class assignment

The input dataset is a table (*SNPs Input Table*) containing for each sample and for each probe the detected SNPs, as produced by the DMET Console. The SNPs Input Table is $np \times ns$ matrix of alleles, where np is the number of probes ($np = 1936$ for current DMET chips) and ns is the number of samples. Table 1 shows a portion of this table showing the SNPs detected in 8 samples (S_1, ..., S_8) on 2 probes. Samples are assigned to class A or B by mouse selection or by providing a list of samples for class B. In the example, some samples (S_1, ..., S_4) belong to class A, while the remaining ones (S_5, ..., S_8) belong to class B.

Table 1. Example dataset containing alleles detected in 8 subjects through 2 probes ($np = 2$ and $ns = 8$).

SNPs Table								
Probes	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
$Probe_1$	a/a	a/a	a/a	c/t	t/t	t/t	t/t	t/t
$Probe_2$	a/a	a/c	a/a	t/t	a/c	a/c	c/t	t/t

3.2 Execution of statistical tests and correction of p-values

During this step, the *Cloud4SNP Statistical Tests* module performs the following operations:

- Preprocess the input data, and eventually filtering of probes having the same or similar distributions in class A and B by using the Fisher Significance threshold (Fs).
- Compute Fisher tests (Ft) and discards tests that are not statistically significant according to Ft (i.e., the tests whose p-value is greater than Ft). More details about this step are provided in as dedicated paragraph below.

- Annotate results with URLs to dbSNP[6] and to PharmaGKB[7].

After computing Fisher tests, the *Cloud4SNP Statistical Corrections* module performs one of the available multiple tests corrections (none, Bonferroni or False Discovery Rate) that adjust p-values derived from multiple statistical tests to correct for occurrence of false positives. Annotated and eventually corrected results are finally displayed to the user through the Graphical User Interface. In summary, Cloud4SNP Statistical Tests and Cloud4SNP Statistical Corrections are the core modules of Cloud4SNP and provide the analysis of SNP data.

An example of Fisher test applied to SNPs

The Fisher Test is applied to couples of SNPs, e.g. SNP_h vs SNP_k, occurring on a probe $Probe_i$ on classes A and B. The algorithm uses the occurrences of the two alleles SNP_h and SNP_k on each class A and B that are reported in a 2×2 contingency table used to compute the Fisher test. Thus, to perform the Fisher Test, Cloud4SNP has to count the occurrences of the SNPs for each probe and class. Tables 2 and 3 contain the occurrences of SNPs in class A and B respectively for the dataset reported in Table 1. For the sake of simplicity, such tables do not report the occurrences of alleles that are not present in any of the two classes, since such occurrences are of course zero.

As an example, Table 4 is the input data to perform the F-test on SNPs A/A vs T/T on probe $Probe_1$, while Table 5 is the input data to perform the F-test on SNPs A/A vs C/T on probe $Probe_2$. Applying F-test on such data, the distributions of SNPs A/A vs T/T on probe $Probe_1$ in the two classes are statistically different ($p-value = 0.0286$), while the distributions of SNPs A/A vs C/T on probe $Probe_2$ not ($p-value = 0.3333$). The p-value threshold to accept or refuse the test (usually set to 0.05) is a parameter of Cloud4SNP, called *Fisher Filter threshold* (Ft). The Ft parameter is used to accept or not the performed Fisher tests, i.e. Fisher tests results with $p - value > Ft$ are discarded and not visualized to the user.

Table 2. Class A SNPs Table.

Class A				
Probes	S_1	S_2	S_3	S_4
$Probe_1$	a/a	a/a	a/a	c/t
$Probe_2$	a/a	a/c	a/a	t/t

3.3 Data Mining Cloud Framework

The DMCF's architecture has been designed to be implemented on different Cloud systems, so as to take advantage of main cloud computing features, such as elasticity of resources provisioning. In DMCF, at least one Virtual Web Server runs continuously in the Cloud, as it serves as user front-end. In addition, users specify the

[6] http://www.ncbi.nlm.nih.gov/projects/SNP
[7] http://www.pharmgkb.org

Table 3. Class B SNPs Table.

Class B				
Probes	S_5	S_6	S_7	S_8
$Probe_1$	t/t	t/t	t/t	t/t
$Probe_2$	a/c	a/c	c/t	t/t

Table 4. $Probe_1$ A/A and T/T SNPs occurrences for Fisher Test

	Class A	Class B
a/a	3	0
t/t	0	4

Table 5. $Probe_2$ A/A and C/T SNPs occurrences for Fisher Test

	Class A	Class B
a/a	2	0
c/t	0	1

minimum and maximum number of Virtual Compute Servers, which are in charge of executing the data mining tasks. The DMCF can exploit the auto-scaling features that allows dynamic spinning up or shutting down Virtual Compute Servers, based on the number of tasks ready for execution in the DMCF's Task Queue. Since storage is managed by the Cloud platform, the number of storage servers is transparent to the user.

The DMCF allows creating data mining and knowledge discovery applications using workflow formalisms. Workflows may encompass all the steps of discovery based on the execution of complex algorithms and the access and analysis of scientific data. In data-driven discovery processes, knowledge discovery workflows can produce results that can confirm real experiments or provide insights that cannot be achieved in laboratories. In particular, DMCF allows to program workflow applications using two languages:

1. *VL4Cloud* (Visual Language for Cloud), a visual programming language that lets users develop applications by programming the workflow components graphically [17].
2. *JS4Cloud* (JavaScript for Cloud), a scripting language for programming data analysis workflows based on JavaScript [16].

Both languages use two key programming abstractions:

1. *Data* elements denote input files or storage elements (e.g., a dataset to be analyzed) or output files or stored elements (e.g., a data mining model).
2. *Tool* elements denote algorithms, software tools or service performing any kind of operation that can be applied to a data element (data mining, filtering, partitioning, etc.).

In particular, three different types of Tools can be used in a DCMF workflow:

1. A *Batch Tool* is used to execute an algorithm or a software tool on a Virtual Compute Server without user interaction. All input parameters are passed as command-line arguments.

2. A *Web Service Tool* is used to insert into a workflow a Web service invocation.
3. A *MapReduce Tool* is used to insert into a workflow the execution of a MapReduce algorithm or application running on a cluster of virtual servers [5].

For each Tool in a workflow, a *Tool descriptor* includes a reference to its executable, the required libraries, and the list of input and output parameters. Each parameter is characterized by *name, description, type*, and can be *mandatory* or *optional*. Another common element is the task concept, which represents the unit of parallelism in our model. A *task* is a Tool, invoked in the workflow, which is intended to run in parallel with other tasks on a set of Cloud resources. According to this approach, VL4Cloud and JS4Cloud implement a *data-driven task parallelism*. This means that, as soon as a task does not depend on any other task in the same workflow, the runtime asynchronously spawns it to the first available virtual machine. A task T_j does not depend on a task T_i belonging to the same workflow (with $i \neq j$), if T_j during its execution does not read any data element created by T_i.

In VL4Cloud, workflows are directed acyclic graphs whose nodes represent data and tools elements. The nodes can be connected with each other through direct edges, establishing specific dependency relationships among them. When an edge is being created between two nodes, a label is automatically attached to it representing the type of relationship between the two nodes. Data and Tool nodes can be added to the workflow singularly or in array form. A data array is an ordered collection of input/output data elements, while a tool array represents multiple instances of the same tool.

In early versions, DMCF has exploited the default storage provided by public cloud infrastructures for any I/O operations. This implies that DMCF's I/O performance was limited by the performance of the storage provided by cloud providers. In work [21] it was proposed to use the Hercules system within DMCF as a storage system for temporary data generated by workflow-based applications. Hercules is a highly scalable, in-memory, distributed storage system. In a later work [15], a data-aware scheduling runtime that exploits data locality has been used. An experimental evaluation was carried out to evaluate the advantages of these strategies and to demonstrate the effectiveness of the solution. Using the proposed data-aware strategy and Hercules as a temporary storage service, I/O overhead was reduced by 55% compared to standard Azure storage-based execution, leading to a 20% reduction in total execution of the workflow.

3.4 Workflow implementation

Starting from the sequential DMET-Analyzer software several modules have been exported and ported as individual tools to the Data Mining Cloud Framework. In particular, the Cloud4SNP Statistical Tests and Cloud4SNP Statistical Corrections modules have been implemented in DMCF as tools, called DMETAnalyzer and Corrector respectively.

In addition, three new tools have been added in the Data Mining Cloud Framework to support parallel processing of SNP input data: *i*) a *Partitioner* tool, which creates a set of partitions from a single SNP dataset; (ii) a *ModelMerger* tool, which merges into a single model the partial models generated by the *DMETAnalyzer*, either corrected or not by a *Corrector*; *iii*) a *ModelsMerger* tool, which takes in input three single models (with FDR, Bonferroni or none correction) and produces

a single HTML file. Using the web GUI of Data Mining Cloud Framework, the tools described above have been composed into the Cloud4SNP workflow shown in Figure 1.

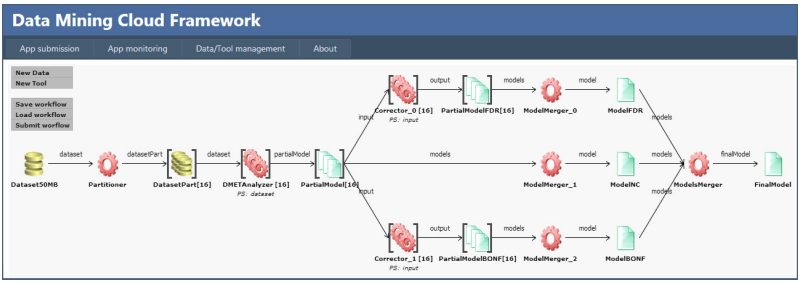

Fig. 1. Cloud4SNP workflow in the Data Mining Cloud Framework.

The workflow performs the following steps. As first step, the initial dataset is partitioned into n chunks by the *Partitioner*, where n is the number of workers available for parallel data processing (16 in this case). By increasing the number of workers, it is possible to improve the level of parallelism and, therefore, system performance. Each data chunk $DatasetPart[i]$, $i = 1, ..., n$, is analyzed by an instance of the *DMETAnalyzer* tool (i.e., *DMETAnalyzer[i]*), which produces a partial model (i.e., *PartialModel[i]*) containing the p-values of each probe. Such partial models are corrected using two different instances of the *Corrector* tool: *Corrector_0* that uses an *FDR* correction, and *Corrector_1* that uses a *Bonferroni* correction. Then, three instances of the *ModelMerger* tool are used to create three models, *ModelNC*, *ModelFDR* and *ModelBONF*, which are respectively the model with no corrections (composition of *PartialModel[n]*), the model with FDR correction (composition of *PartialModelFDR[n]*), and the model with Bonferroni correction (composition of *PartialModelBONF[n]*). Finally, the *ModelsMerger* tool combines *ModelNC*, *ModelFDR* and *ModelBONF* to produce a single HTML file with all the output results (see Figure 2).

3.5 Using Apache Spark for faster in-memory processing

Apache Spark[8] is considered one of the leading open source Big Data systems and thus it is supported by every major Cloud providers. Differently from other systems (e.g., Apache Hadoop), in which intermediate data are always stored in distributed file systems, Spark stores data in RAM memory and queries it repeatedly so as to obtain better performance for some classes of applications. Apache Spark allows also to cache intermediate data, providing a significant performance improvement while running multiple queries on the same data. Spark is commonly used to develop in-memory iterative batch applications using many programming languages (e.g., Java, Scala, Python, R). Many powerful libraries are built on top of Spark:

[8] https://spark.apache.org

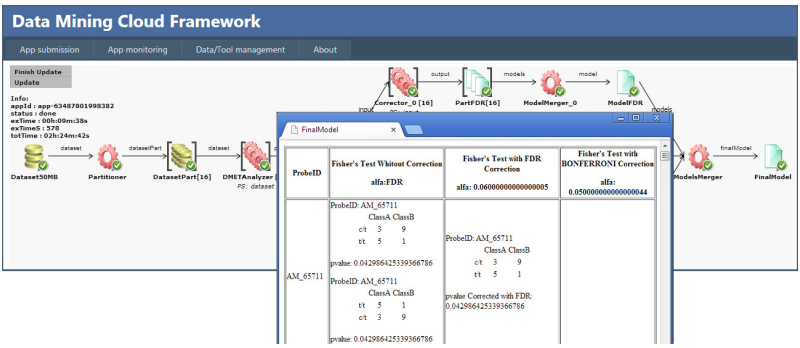

Fig. 2. Cloud4SNP workflow at the end of the execution, with visualization of the final result.

MLlib for machine learning, *GraphX* for graph-parallel computation, *ParSoDa*[8] for processing social media data, *Spark Streaming* for stream processing.

Apache Spark provides a *low-level of abstraction*, because a programmer can define an application using APIs which are powerful but require high programming skills. Developing an application based on Spark requires a quite small number of lines of code, especially when *Scala* is used as programming language [4]. Input data is divided in chunks and processed in parallel by different computing nodes. It supports also *task parallelism*, when independent stages of the same application are executed in parallel. Spark is designed to process very large amounts of data in *large-scale infrastructures* with up to tens of thousands of nodes.

On the other hand, the DMCF provides a *high-level of abstraction*, because a programmer can define workflows by using a high-level visual language (VL4Cloud) or a script-based language (JS4Cloud). VL4Cloud is a convenient design approach for high-level users, for example, domain-expert analysts having a limited understanding of programming. Conversely, JS4Cloud allows skilled users to program complex applications more rapidly, in a more concise way, and with greater flexibility.

Cloud4SNP has been redesigned and extended to allow the configuration of Spark-based applications in two ways:

1. Using a *Web Tool* in DMCF, specially configured to invoke an external web service that invokes the execution of a Spark application.
2. Using a *Batch Tool* in DMCF to launch a Spark application on a VM on which Spark is installed in single-node mode

Since the preprocessing, analysis and statistical correction algorithms have been entirely rewritten in Spark, it is possible to use them natively for creating and executing the entire workflow on a Spark cluster, without passing through DMCF. In such a way, the execution of the Spark application is driven by a central coordinator (i.e., the main process of the application), which can connect with different cluster managers (e.g., *Apache Mesos*[9] or *Hadoop YARN*[10]) for better managing

[9] https://mesos.apache.org/
[10] https://hadoop.apache.org/

distributed resources. including the usage of data locality techniques for reducing the network overhead due to data moviment among nodes.

4 Performance Evaluation

To evaluate the execution times and scalability with increasing workloads, several experiments have been carries out on three SNP datasets ($D1$, $D2$, $D3$) with a constant number of samples (28) and with an increasing number of probes: around 10,000 probes for the dataset $D1$, 20,000 for the $D2$, and 40,000 for $D3$.

Figure 3 shows the turnaround times of the different implementation of Cloud4SNP using the dataset $D3$. As shown, Apache Spark ensures a reduction in the processing times by an average of 35% compared to the other systems. In particular, using Spark natively ensures the best performance, followed by DMCF with Spark as Tool. In general, the differences in processing between the two Spark-based implementations are minimal, mainly due to the overhead required by DMCF for managing and executing the workflow in a distributed environment. For the other two smaller datasets, $D1$ and $D2$, the results obtained are quite similar and, therefore, are not reported for reasons of brevity.

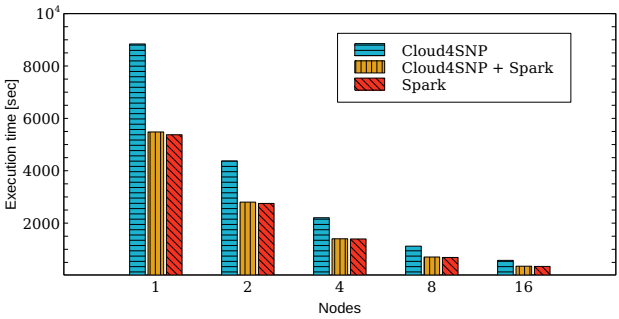

Fig. 3. Execution times of the different Cloud4SNP implementations using the dataset $D3$.

Although using Spark natively offers the best performance, DMCF continues to be the recommended choice for low skilled users that have limited programming capabilities. In addition, due to the complexity of managing a Spark cluster, the use Spark only makes sense for large datasets, so as to make the most of its in-memory processing capabilities.

Figure 4 reports the speedup obtained by the different systems using the dataset $D3$. As reported, the results obtained is very close to linear values, demonstrating the high scalability of the implemented solutions.

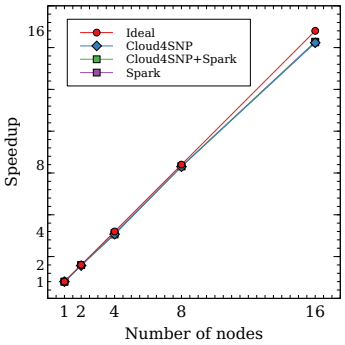

Fig. 4. Speedup values of the different Cloud4SNP implementations using the dataset *D3*.

5 Conclusion

This paper discussed an extension of Cloud4SNP, a bioinformatics tool for parallel preprocessing and statistical analysis of SNP pharmacogenomics microarray data, for enabling the execution of data analysis applications based on Apache Spark. Apache Spark is one of the leading open source Big Data systems, which is able to provide faster, in-memory executions for iterative and batch processing. The experimental evaluation shows that Cloud4SNP is able to exploit the high-performance features of Apache Spark, obtaining faster execution times and high level of scalability, with a global speedup that is very close to linear values.

References

1. affypara-a bioconductor package for parallelized preprocessing algorithms of affymetrix microarray data. Bioinform Biol Insights 30(22), 83–7 (2009)
2. Agapito, G., Cannataro, M., Guzzi, P.H., Marozzo, F., Talia, D., Trunfio, P.: Cloud4snp: Distributed analysis of snp microarray data on the cloud. In: Proc. of the ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics 2013 (ACM BCB 2013). p. 468. ACM Press, Washington, DC, USA (September 2013), iSBN 978-1-4503-2434-2
3. Barton, G., Abbott, J., Chiba, N., Huang, D., Huang, Y., Krznaric, M., Mack-Smith, J., Saleem, A., Sherman, B., Tiwari, B., Tomlinson, C., Aitman, T., Darlington, J., Game, L., Sternberg, M., Butcher, S.: Emaas: An extensible grid-based rich internet application for microarray data analysis and management. BMC Bioinformatics 9(1), 493 (2008), `http://www.biomedcentral.com/1471-2105/9/493`
4. Belcastro, L., Marozzo, F., Talia, D.: Programming models and systems for big data analysis. International Journal of Parallel, Emergent and Distributed Systems 34(6), 632–652 (2019)
5. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Programming visual and script-based big data analytics workflows on clouds. In: Grandinetti, L., Joubert, G., Kunze, M., Pascucci, V. (eds.) Post-Proc. of the High Performance

Computing Workshop 2014. Advances in Parallel Computing, vol. 26, pp. 18–31. IOS Press, Cetraro, Italy (2015), iSBN: 978-1-61499-582-1
6. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Appraising spark on large-scale social media analysis. In: Euro-Par Workshops. pp. 483–495. Lecture Notes in Computer Science, Santiago de Compostela, Spain (28-29 August 2017), iSBN: 978-3-319-75178-8
7. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Big data analysis on clouds. In: Zomaya, A., Sakr, S. (eds.) Handbook of Big Data Technologies, pp. 101–142. Springer (December 2017), iSBN: 978-3-319-49339-8
8. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Parsoda: high-level parallel programming for social data mining. Social Network Analysis and Mining 9(1), 1–19 (2019)
9. Burmester, J.K., Sedova, M., Shapero, M.H., Mansfield, E.: Dmet microarray technology for pharmacogenomics-based personalized medicine. Microarray Methods for Drug Discovery, Methods in Molecular Biology 632, 99–124 (2010)
10. Cannataro, M., Guzzi, P.H., Veltri, P.: Protein-to-protein interactions: Technologies, databases, and algorithms. ACM Comput. Surv. 43(1), 1:1–1:36 (Dec 2010), http://doi.acm.org/10.1145/1824795.1824796
11. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., Da Silva, R.F., Livny, M., et al.: Pegasus, a workflow management system for science automation. Future Generation Computer Systems 46, 17–35 (2015)
12. Guzzi, P.H., Agapito, G., Di Martino, M.T., Arbitrio, M., Tagliaferrri, P., Tassone, P., Cannataro, M.: DMET-analyzer: automatic analysis of affymetrix DMET data. BMC Bioinformatics 13:258, 258+ (Oct 2012)
13. Guzzi, P.H., Cannataro, M.: mu-cs: An extension of the tm4 platform to manage affymetrix binary data. BMC Bioinformatics 11, 315 (2010)
14. Lordan, F., Tejedor, E., Ejarque, J., Rafanell, R., Alvarez, J., Marozzo, F., Lezzi, D., Sirvent, R., Talia, D., Badia, R.M.: Servicess: An interoperable programming framework for the cloud. Journal of grid computing 12(1), 67–91 (2014)
15. Marozzo, F., Rodrigo Duro, F., Garcia Blas, J., Carretero, J., Talia, D., Trunfio, P.: A data-aware scheduling strategy for workflow execution in clouds. Concurrency Computation 29(24) (2017)
16. Marozzo, F., Talia, D., Trunfio, P.: Js4cloud: Script-based workflow programming for scalable data analysis on cloud platforms. Concurrency Computation 27(17), 5214–5237 (2015)
17. Marozzo, F., Talia, D., Trunfio, P.: A workflow management system for scalable data mining on clouds. IEEE Transactions on Services Computing 11(3), 480–492 (2018)
18. Marozzo, F., Lordan, F., Rafanell, R., Lezzi, D., Talia, D., Badia, R.: Enabling cloud interoperability with compss. In: Proc. of the 18th International European Conference on Parallel and Distributed (Europar 2012). vol. 7484, pp. 16–27. Lecture Notes in Computer Science, Rhodes Island, Greece (27-31 August 2012)
19. Marozzo, F., Talia, D., Trunfio, P.: Scalable script-based data analysis workflows on clouds. In: Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science. pp. 124–133 (2013)
20. Phillips, C.: SNP Databases. In: Komar, A.A. (ed.) Single Nucleotide Polymorphisms, vol. 578, chap. 3, pp. 43–71. Humana Press, Totowa, NJ (2009)

21. Rodrigo Duro, F., Marozzo, F., Garcia Blas, J., Talia, D., Trunfio, P.: Exploiting in-memory storage for improving workflow executions in cloud platforms. Journal of Supercomputing 72(11), 4069–4088 (2016)
22. Talia, D., Trunfio, P., Marozzo, F.: Data Analysis in the Cloud. Elsevier (October 2015), iSBN 978-0-12-802881-0
23. Wilde, M., Hategan, M., Wozniak, J.M., Clifford, B., Katz, D.S., Foster, I.: Swift: A language for distributed parallel scripting. Parallel Computing 37(9), 633–652 (2011)