journal homepage: www.intl.elsevierhealth.com/journals/cmpb

Optimizing R with SparkR on a commodity cluster for biomedical research

Martin Sedlmayr [a,*], Tobias Würfl [a], Christian Maier [a], Lothar Häberle [b], Peter Fasching [b], Hans-Ulrich Prokosch [a], Jan Christoph [a]

[a] Friedrich-Alexander University Erlangen-Nürnberg, Wetterkreuz 13, 91058 Erlangen, Germany
[b] Department of Gynecology and Obstetrics, Erlangen University Hospital, Universitätsstrasse 21-23, 91054 Erlangen, Germany

A R T I C L E I N F O

Article history:
Received 18 April 2016
Received in revised form
16 August 2016
Accepted 6 October 2016

Keywords:
Computing methodologies
Genome-wide association study
Big data
Cluster computing
SparkR

A B S T R A C T

Background and Objectives: Medical researchers are challenged today by the enormous amount of data collected in healthcare. Analysis methods such as genome-wide association studies (GWAS) are often computationally intensive and thus require enormous resources to be performed in a reasonable amount of time. While dedicated clusters and public clouds may deliver the desired performance, their use requires upfront financial efforts or anonymous data, which is often not possible for preliminary or occasional tasks. We explored the possibilities to build a private, flexible cluster for processing scripts in R based on commodity, non-dedicated hardware of our department.

Methods: For this, a GWAS-calculation in R on a single desktop computer, a Message Passing Interface (MPI)-cluster, and a SparkR-cluster were compared with regards to the performance, scalability, quality, and simplicity.

Results: The original script had a projected runtime of three years on a single desktop computer. Optimizing the script in R already yielded a significant reduction in computing time (2 weeks). By using R-MPI and SparkR, we were able to parallelize the computation and reduce the time to less than three hours (2.6 h) on already available, standard office computers. While MPI is a proven approach in high-performance clusters, it requires rather static, dedicated nodes. SparkR and its Hadoop siblings allow for a dynamic, elastic environment with automated failure handling. SparkR also scales better with the number of nodes in the cluster than MPI due to optimized data communication.

Conclusion: R is a popular environment for clinical data analysis. The new SparkR solution offers elastic resources and allows supporting big data analysis using R even on non-dedicated resources with minimal change to the original code. To unleash the full potential, additional efforts should be invested to customize and improve the algorithms, especially with regards to data distribution.

* *Corresponding author.* Friedrich-Alexander University Erlangen-Nürnberg, Wetterkreuz 13, 91058 Erlangen, Germany. Fax: +49 (9131) 85 26754.
E-mail address: martin.sedlmayr@fau.de (M. Sedlmayr).

http://dx.doi.org/10.1016/j.cmpb.2016.10.006

1. Introduction

Medical researchers are challenged today by the enormous amount of data collected in healthcare [1]. Especially research of high dimensional data in cancer genomics and translational medicine becomes increasingly common in medicine [2]. Data mining algorithms and genome-wide association studies (GWAS) are used to detect correlations between genomic and phenotypic features [3].

Such analysis methods are often computationally intensive and thus require enormous resources to be performed in a reasonable amount of time. Fortunately, many algorithms can be parallelized and therefore the tasks can be allotted to many computers [4].

On an infrastructure level, cluster computing is often used to distribute the workload on a (local) network of dedicated resources [4]. In recent years, cloud computing became a viable alternative, because of its elasticity and cost-effectiveness [5]. On a platform level, Hadoop-based services provide standard interfaces for distributed storage and computing [6,7]. Especially Spark, an in-memory computing service based on Hadoop, begins to surpass previous solutions such as the Message Passing Interface (MPI) used in high-performance computing (HPC) [8].

Unfortunately, using external resources (e.g. the cloud) is often limited or prohibited for sensitive patient data due to privacy regulations. Researchers also face the problem to justify an investment into dedicated resources (e.g. clusters and blades) in smaller projects or for occasional use.

Therefore, we explored the possibilities to build a private, flexible cluster based on commodity hardware using open source solutions. We focused on the R-package parallel based on MPI [9] and SparkR [10], not only because R is a popular software for analyzing biomedical data [11] but it is as well the tool of choice for our researchers. It also challenges two different paradigms: traditional HPC cluster computing and Hadoop-based Big Data technologies.

In the following, we compare a GWAS-calculation in R on a single desktop computer, an MPI-cluster and a SparkR-cluster with regards to the performance, scalability, quality, and ease of use.

2. Methods

The starting point of our work was an R script which was developed and validated by clinical statisticians together with medical researchers of the Department of Gynecology and Obstetrics of the University Hospital Erlangen. They also provided an excerpt of a dataset derived from breast-cancer patients who participated in previous clinical research. The calculation aims to identify significant correlations between mutations in the genome (single nucleotide polymorphisms, SNPs) and expression levels of any other genes (GENs), using basic clinical variables to minimize confounders (similar to Ref. [12]).

The script reads an anonymous dataset comprising 199,960 single nucleotide polymorphism (SNPs) and 18,609 gene expressions (GENs) as well as 17 clinical variables (such as age and body mass index) from 122 patients. Then, it performs a linear regression fit (*lmfit*) for each pair of SNP and GENE with the clinical variables adjusting confounders and calculates the standard error, the p-value and the beta-value. The result is written to an output file.

Initially, performance was not in the focus of the developers who rather concentrated on the correctness of the outcome. Based on small subsets of the data, the estimated runtime for the whole dataset on a standard desktop PC was about three years. Our goal was to reduce this time significantly by optimizing the original code and distributing the workload on a network of computing nodes. While working on the optimization of the code, we focused on four success indicators:

- Performance: The time required for computation of the full dataset (the shorter, the better).
- Scalability: The extent, to which adding more computing nodes yields a performance gain (ideally a linear gain should be achieved).
- Quality: There should be no (relevant) deviation between the output of the optimized and the original script.
- Simplicity: The resulting script should retain the flexibility of the original while requiring only minimal changes.

The last point was especially important because the medical experts should still be able to proof and modify the algorithm using their existing libraries (flexibility, trust). This is why the options to completely rewrite the algorithm in another programming language or to use alien tools were discarded.

First, the initial code in R was optimized. Then, the algorithm was parallelized, and the workload was distributed in a local MPI-cluster using the parallel-package of R. Finally, SparkR, which is a relatively new combination of Hadoop in-memory computing and R, was evaluated as an alternative to MPI. All changes to the functional part of the script have been checked for deviations in the results.

2.1. *Optimizing R*

The R script was analyzed using the R-profiler to identify time bottlenecks caused by inefficient data structures or control structures. Based on numerous tutorials, best practices and recommendations by the R-community [13], the R script was improved by a couple of means:

- The original script iterated over the list of SNPs and GENs using nested *for*-loops. Using *apply* instead yielded not only a performance gain but also facilitated the parallelization in the later steps.
- An R *DataFrame* is a flexible, but inefficient data structure. It was replaced by a *matrix* with a simple typecast.
- The linear regression performed by LM requires parsing of the given formula. Using *lm.fit* instead of LM avoids the parsing overhead, but requires to manually calculate the p-value and standard error.
- Replacing the built-in *lm.fit* function by a third-party library (qtl2scan) resulted in another, substantial performance gain [14].

Each step was benchmarked and the results were validated against the original script as gold standard. An intermediate test using two SNPs revealed a speedup of factor 122 (7.6s vs. 932.3s) or an estimated duration of nine days (vs. 3 years) for the full dataset.

2.2. *Parallelizing R using the message passing interface*

The nature of the algorithm allows for parallelization as each combination of SNP and GEN can be calculated independently.

While there are several options to implement parallel R (multicore, explicit cluster) [9] we concentrated on the Message Passing Interface (MPI) as it is a standard used by many HPC clusters and comes packaged for most Linux distributions. Each node of an MPI cluster should provide the same environment (software, folders) so that a locally started process can easily be copied to worker nodes. In principle, the local PC becomes the master, logs into each worker node, and starts the software which keeps connected to the master process. If the master quits, all dependent worker processes are terminated automatically.

Refactoring the script is straightforward:

- *cluster = createCluster(number of nodes, type = „MPI“)*: Commence the cluster by starting R on each node
- *clusterExport(cluster, vector of variables & methods)*: Copy all required data and code to each node of the cluster.
- *parApply(cluster, vector, function)*: Apply the function on each element of the vector. It equals the standard R function *apply* but distributes the load across the remote nodes instead of computing it sequentially on the local node.
- *stopCluster(cluster)*: Stop R on all the nodes in the cluster.

The *clusterExport* causes the data to move to each instance of R on the remote nodes which might require substantial time depending on the size of data and network speed. Using *clusterCall* instead to read the data in parallel from the file system had no beneficial effect on execution time in our experiments.

The *parApply* method not only distributes the workload, but also collects the partial results into an overall result. The joined result has to fit into the memory of the node of the R master.

2.3. *Parallelizing R using SparkR*

SparkR is an R-package that provides functions to compute and read/write resilient data frames (RDD) in a Spark cluster. The current distribution of Spark already includes a preconfigured environment so that a user can immediately start using R with SparkR.

The parallel approach used before can easily be replaced by SparkR functions. Creating and destroying the cluster is not necessary because the Spark cluster runs independently; the client just connects to the Spark master which distributes the workload. The distribution of data and function definitions is handled automatically, so there is no need to export the data to the nodes explicitly.

The main differences are:

- *dframe = createDataFrame*: Convert the matrix of SNPs into a Spark DataFrame, which is inherently parallel.
- *rdd = SparkR:::map(dframe, function)*: Prepares to apply the given function on each element of dframe.
- *collect(rdd)*: Actually executes the *map* and performs the calculation. Partial results will be collected on the master.

The described modifications only concentrate on the computational aspect. SparkR also provides options to read and write data in a distributed way using the Hadoop Distributed File System (HDFS) which was not used, because the R script already saves partial results from the worker nodes and using HDFS would have required a full Hadoop cluster next to the Spark cluster.

The java-based method *glm* of the machine learning package of Spark has been evaluated, but discarded because it is significantly slower than the native R function (*glm* is optimized for distributing the linear regression of huge matrices among a cluster).

3. Results

The core of our cluster consists of a pool of 13 desktop PCs (Intel core i5, 8 GB RAM) in a lecture hall, which are mainly used for student education. A virtual machine was set up on each PC (Ubuntu Linux 64bit, 4 cores, 4 GB RAM). All measurements were taken using these nodes.

The cluster is complemented by a 6-year old desktop server (2 Intel Xeon, 8 GB RAM), one desktop and one spare PC (Intel core i5, 8 GB RAM) and two virtual machines (each 6 cores, 12 GB RAM) on two local VmWare ESX hosts. The full dataset was computed on the full cluster (76 cores) in less than three hours.

Each run was performed three times, and the minimum, maximum and mean runtimes were recorded. A significant difference between the three runs (30–70%) was only observable if using the smallest dataset. The cause is the variation in startup time of the client and the connection to the cluster which contributes a large portion to the overall runtime. There was no notable difference between the three runs (i.e. less than 5%) with larger datasets. As cluster computations are only reasonable with larger datasets, only the mean time is shown here.

Table 1 shows the times measured for each approach. For each number of worker nodes (cores) in the cluster (1/5/10/20/50) a combination of SNPs (1000/10,000/100,000), and GENs (100/1000) have been calculated using two adjustment variables (age and body mass index). Based on the number of SNP $\times$ GEN combinations and the time measured, a performance index (thousand calculations per second, Tps) has been determined. For MPI and SparkR, a factor relative to the single case is calculated based on the Tps—ideally, it should be the same as the number of cores.

For the single case, the performance is between 13 and 14 thousand calculations per second, independent of the size of the dataset.

A 5-node MPI cluster is about 2.8 times faster than the single case, and the 50-node cluster is only about 3–15 times faster. Although the number of combinations of 10,000 SNPs/1000 GENs equals 100,000 SNPs/100 GENs, the relative performance is better if GENs is larger because the transfer time only depends on the number of SNPs.

The 5-node SparkR cluster is about 4.5 times faster than the single case and scales up to factor 22 on the 50-node cluster. The number of calculations per second from which the factor

Table 1 – Mean time in seconds for calculating all correlations of SNPs and GENs using core-number of worker nodes. The number of thousand calculations per second (Tps) is the basis for the performance factor relative to the single R measurement.

Single

Cores	SNPs	GENs	Mean	Tps
1	1,000	100	8	13
1	1,000	1,000	72	14
1	10,000	100	75	13
1	10,000	1,000	717	14
1	100,000	100	788	13
1	100,000	1,000	7690	13

MPI

Cores	SNPs	GENs	Mean	Tps	Factor
5	1,000	100	4	27	2.0
5	1,000	1,000	26	39	2.8
5	10,000	100	30	33	2.5
5	10,000	1,000	245	41	2.9
5	100,000	100	326	31	2.4
5	100,000	1,000	2744	36	2.8
10	1,000	100	2	55	4.1
10	1,000	1,000	13	75	5.3
10	10,000	100	17	60	4.5
10	10,000	1,000	124	81	5.8
10	100,000	100	168	60	4.7
10	100,000	1,000	1364	73	5.6
20	1,000	100	2	51	3.8
20	1,000	1,000	13	78	5.5
20	10,000	100	12	83	6.2
20	10,000	1,000	71	141	10.1
20	100,000	100	111	90	7.1
20	100,000	1,000	714	140	10.8
50	1,000	100	5	19	1.4
50	1,000	1,000	17	58	4.2
50	10,000	100	26	38	2.8
50	10,000	1,000	51	197	14.1
50	100,000	100	251	40	3.1
50	100,000	1,000	491	204	15.7

SPARK

Cores	SNPs	GENs	Mean	Tps	Factor
5	1,000	100	2	41	3.1
5	1,000	1,000	15	65	4.6
5	10,000	100	16	62	4.6
5	10,000	1,000	159	63	4.5
5	100,000	100	179	56	4.4
5	100,000	1,000	1830	55	4.2
10	1,000	100	2	50	3.8
10	1,000	1,000	9	112	8.0
10	10,000	100	9	109	8.1
10	10,000	1,000	88	114	8.1
10	100,000	100	101	99	7.8
10	100,000	1,000	1003	100	7.7
20	1,000	100	3	29	2.2
20	1,000	1,000	5	189	13.5
20	10,000	100	6	179	13.3
20	10,000	1,000	46	217	15.6
20	100,000	100	55	181	14.2
20	100,000	1,000	542	184	14.2
50	1,000	100	4	27	2.1
50	1,000	1,000	4	241	17.2
50	10,000	100	5	217	16.2
50	10,000	1,000	30	331	23.7
50	100,000	100	57	176	13.9
50	100,000	1,000	347	288	22.1
72	196,000	18,600	9360	389	

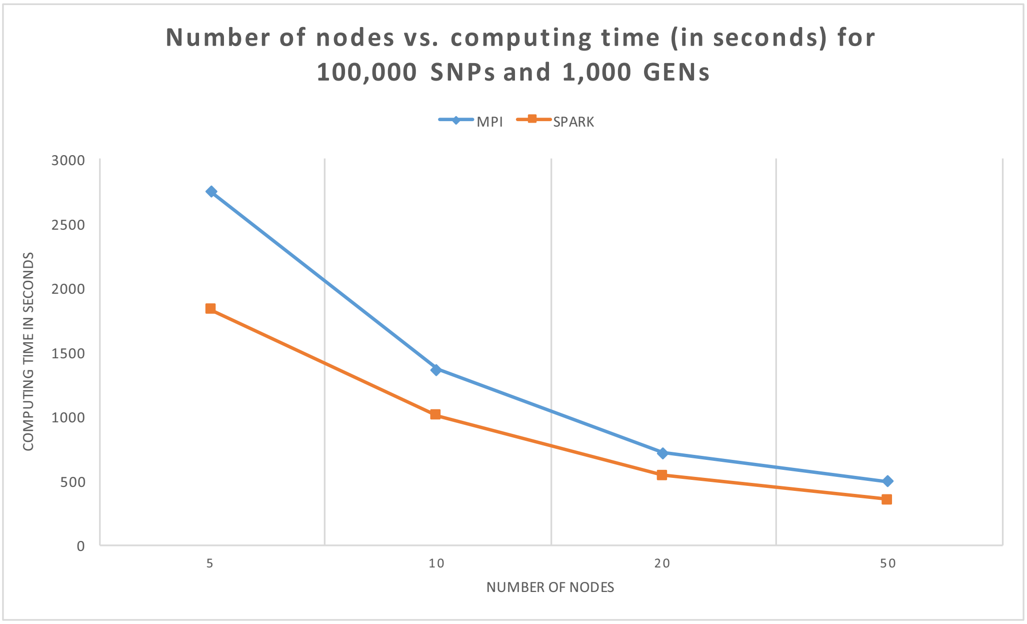

Fig. 1 – Time in seconds for performing a 100,000 SPNs, 1000 GENs analysis dependent on the number of cores used in the cluster.

is determined is less prone to variations in the data size than with MPI.

In both cases MPI and SparkR, the scalability is not linear to the number of nodes. While SparkR generally scales better than MPI, it worsens with the cluster size (Fig. 1 shows the performance gain in relation to the hypothetical linear scaling).

For MPI as well as for SparkR, the relative performance drops the larger the cluster gets (Fig. 2). The main contributing factor is the distribution of the data, which requires more time as the cluster grows. SparkR generally scales better than MPI.

An additional test was performed to compare the native Hadoop/SparkR installation on Windows 7 with the virtual Linux environment on the same physical hardware using 4 nodes (16 cores), 1000 SNPs and 1000 GENs. While splitting the tasks into 16 parts (1 part per core) showed no significant difference in runtime (both 11s), a split into 160 parts (10 tasks per core) revealed a massive drop in the native Windows implementation (32 s vs. 17 s). Apparently, the set-up of R as an external process and the communication required is less efficient in Windows than in Linux.

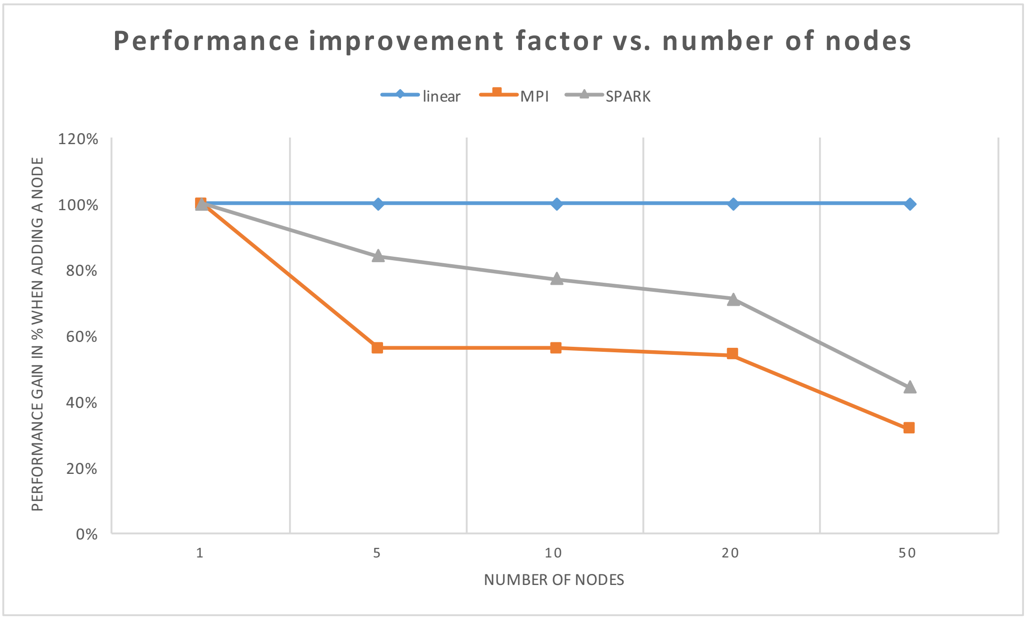

Fig. 2 – Factor to which extend adding nodes increase the performance of the cluster (indicator for scalability).

	R (original)	R (optimized)	parallel R (MPI)	SparkR
Pseudocode outer loop	`for each snp`	`R <- apply(snps,`	`cl <- createCluster()`	`Parallelize(...)`
	`for each gen`	`apply (gens,`	`Export(cl, ...)`	`R <- Map(snps,`
	gwas	*gwas*	`R <- parApply(cl, snps,`	`apply(gens,`
	`end for`	`)`	`apply(gens,`	*gwas*
	`end for`	`)`	*gwas*	`)`
		`save(R)`	`)`	`)`
			`)`	`save(R)`
			`save(R)`	

Fig. 3 – Modifications to the code required to utilize the cluster (in pseudo-code).

Fig. 3 illustrates the main differences between implementations. It can be seen that bringing the script to the cluster requires no major changes. While the original script was based on *for*-loops, the optimized version used *apply*. For parallel computing, a *createCluster* starts R on all nodes as defined in the MPI-configuration. Required data have to be exported to each node of the cluster before the parallel version of *apply* performs the calculation. When SparkR is invoked, it automatically connects to a Spark cluster, which has to be started in advance. Data should be parallelized so that the distribution can also be handled automatically. Therefore, the modifications required in the MPI-case and SparkR are very similar.

4. Discussion

The goal of reducing the runtime of the data analysis has been achieved: from an initially targeted three years to less than three hours. Refactoring the initial script by rewriting the loops and using native data structures also paved the way towards parallelization using MPI and SparkR. No new or external resources were needed, only the existing office infrastructure was used. With regards to our success criteria:

– Performance: Improving the original script already reduced the computing time from an estimated 3 years to about 9 days. As the algorithm can be parallelized, the workload can easily be distributed among worker nodes.
– Scalability: All three tested approaches use R for the actual computation, so the parallelization scales with the number of R instances in the network. However, the overhead for communication (of data and results) among the worker nodes increases with the number of nodes. The efficiency of communication limits the performance gain. SparkR scales generally better than MPI.
– Quality: As all calculations are performed by the very same functions in R, there is no deviation in the results. Our experiments with the equivalent function *glm* (from the machine learning package of Spark [10]) shows only a minor difference in the rounding of the eighth to seventh decimal place.
– Simplicity: Using the R-package parallel required only four additional lines and one changed line of code, SparkR required two lines. Both approaches match closely the optimized R script and hinder in no way the readability of the code.

As the function of the linear fit remained the same in all three implementations and R is used for performing the calculations, the measured runtime difference between MPI and SparkR should result from cluster management overhead. This overhead consists of data distribution and interprocess communication with R. In the case of MPI, the time required for data distribution can be measured explicitly (*clusterExport*) and takes up to 90% of the overall time. Although our approach to copy all data to all nodes could be improved, it is fair to say that a special focus should be put on data management.

This is also true for collecting the results. Because we do no filter on the p-value, the number of raw results is huge and the result file grows larger than 100 GB. This does not fit in the memory of the R master and hence the computation fails. Third-party packages for huge matrices and Big Data exist ([15]), but they add another layer of complexity and are a mere workaround.

Both, MPI and SparkR, require additional effort in setting up and maintaining a cluster and also show other differences during installation and operation.

4.1. *MPI*

The *parallel* package based on MPI is easy to install and use because it is available preconfigured for major Linux distributions and requires only minimal setup (a textfile containing the IP-addresses of worker nodes). Further, MPI is a robust technology proven in large clusters and super computing centers [16]. Implementations are available for all major operating systems.

However, MPI is optimized for static clusters preferably using dedicated hardware and network resources [16]. The set of nodes is fixed and cannot dynamically grow and shrink which requires a stable environment, often not given when piggybacking office computers or using dynamic cloud resources.

A minor issue is the lack of a progress indicator when using the parallel package; therefore, the user gets no feedback on the state of long-running tasks.

4.2. *SparkR*

SparkR is a relatively new addition to R and suffers until now from a lack of documentation as well as from some contradictory tutorials due to a change in the interface. This causes an additional overhead for the programmer to learn and for the administrator to maintain a cluster. The pace of development of Spark is rather high, so major version changes often occur (every 4–8 weeks) which potentially affect or break current programs.

An advantage of SparkR is that it natively interfaces with Hadoop and can, therefore, utilize the distributed filesystem HDFS for reading and writing large datasets.

Spark splits a request into tasks and distributes tasks to workers and executors. The size of a single task should not exceed 100 kb which was the case for the larger data runs. Also, the cluster experienced timeouts if the required time in R exceeded the heartbeat time of the executor. Either the timeout has to be extended or—more preferably—the number of tasks has to be increased (e.g. from two to ten times the number of nodes). Having more tasks than nodes is also important for load-balancing heterogenous clusters so that faster nodes can complete more tasks than slower ones.

SparkR provides a function to apply a function to the Cartesian product of two vectors. So *SparkR:::cartesian(SNPs, GENs, function)* would have been an alternative to just parallelizing the SNPs. However, the ratio of number of tasks to runtime of each task became unfortunate (four billion small tasks in our case), and the runtime performance dropped due to the additional overhead.

An experiment implementing the functions of our R script using Spark/Scala had no apparent performance benefit in our case. However, as datasets become larger, the possibility to employ not only large compute clusters but also to use RDDs in HDFS is beneficial [17].

We were able to add successfully computing nodes running Microsoft Windows to the Spark-cluster using HDFS as the common file system. A sample benchmark with 4 nodes (16 cores) revealed that while the computing time of R does not differ between R on Windows and R in a VM in Linux, the overhead of the process communication between Spark and R is enormous. If the workload is split 1:1 to workers, there is no significant difference; if a split of 1:10 is used, the Windows-based solution is 4 times slower.

Although the performance can be below the dedicated Linux virtual machine, it could be an opportunity for employees to start the node at the end of a working day to provide additional computing resources during the night. The elasticity of Spark allows them to kill the node in the morning without compromising the overall stability.

4.3. *Related work*

Liang et al. [11] have evaluated and discussed the differences in runtime performance of other Hadoop packages in R (rHadoop and H2O). They also found the exchange of data between R and the Java-based cluster as the bottleneck while the changes to the R program are less than 10 lines.

Wiewiorka et al. [18] have used Spark for genomic analysis and found an almost linear speedup with the number of worker nodes. They also use HDFS to read the data directly to the client thus avoiding the data transfer overhead. Additionally, as they do not use R, they have no inter-process-communication overhead.

Zou et al. [19] provide an extensive survey of MapReduce frameworks for bioinformatics. They also argue that MPI has many disadvantages at it does not offer load balancing, no fault tolerance, and no distributed file system. O'Driscoll et al. [17] confirm that the Spark family clusters are more robust to node failures and automatically restart chunks upon failure [17].

Agarwal and Owzar [4] mention CUDA as a possible alternative for massively parallel execution using a graphics card (GPU). Davis et al. [20] and Lee et al. [21] give examples of genetic algorithms using GPUs. A package exists for R which brings the *lm.fit* to the GPU [22]. Unfortunately, similar to the function *glm* in Spark it helps to accelerate computation on large matrices, but requires to move the matrix from RAM to GPU-memory. The additional overhead to move the data of each cell to the graphics card back and forth is larger than the benefit. In order to unleash the card, one should move the whole matrix to graphics memory and implement a kernel to compute on CUDA only.

Jha et al. [8] compare different architectures, Spark and MPI among them. They also find an almost linear efficiency with a small benefit for MPI over Spark due to data requirements of the implemented K-means clustering algorithm which differs from our linear regression approach.

Large cloud providers such as Google and Amazon provide huge resources [5] at high service levels [23]. And also in Europe, scalable infrastructures for life science research are under development [24]. However, legal restrictions or mental reservations (e.g. regarding data ownership or trust) often prohibit the use of external services for sensitive data; especially as long as genetic data cannot be strictly anonymized [25,26].

4.4. *Limitations*

Computing the correlation of each SNP and GEN is a naive brute-force approach. In the final setting, one would use additional knowledge from databases like HapMap [27] to limit the combinations that have to be investigated. This reduces the computational effort and especially minimizes the risk that the necessary p-value adjustment (like Bonferroni correction or further advanced methods of FDR [28]) will hide genuine results. But our aim was to explore the possibilities to support a high number of calculations using big data/cluster technologies. Even if the number could be reduced in our case, it explodes if e.g. SNP–SNP–GEN interactions will be tested and makes it inevitable to use technologies as outlined above.

The authors are no professional experts in MPI and Spark. Both technologies are used according to their manuals and as close to the default installation as possible. Customizations and tweaks may exist that improve the overall performance. For economy, the amount of time required to optimize the runtime is probably larger than the achieved performance gain.

Further improvement may be achieved if SparkR is configured together with a Hadoop file system or a scalable database instead of files which would also relieve the master from sending data (input and results) back and forth the worker nodes. But this requires an extra Hadoop cluster.

We have not explored the possibility to use MPI in Windows. We do not expect any relevant runtime difference between using MPI/R on Linux and Windows: the actual calculation should not differ given our experiments with SparkR on Linux and Windows. The distribution of the data (cluster management) also remains the same with Windows and Linux.

While our tests focused on a single dataset and a particular algorithm, we are confident that our findings can be transferred to similar algorithms as literature supports our general findings as shown above. For example, it benefits the analysis of GEN–GEN as well as GEN–GEN–SNP interactions that would be too computationally demanding otherwise.

5. Conclusions

We have compared the potential to speed up an R-script for genetic analysis by refactoring the script, and by parallelizing it with MPI and SparkR.

Under the condition that the dataset and the algorithm can be parallelized and the time required to transfer the data to worker nodes is less than the actual computation time, a substantial increase can be observed when contributing additional nodes/ cores to such a cluster. It enables us to enlarge the dataset from 122 patients to more than 500 for which the researchers have data, to enrich the amount of confounder variables and to analyze GEN–GEN–SNP interactions that have not been considered before.

SparkR generally scales better than MPI in such a case. And although SparkR can be used in a standalone-cluster mode (as presented here) with comparably low efforts to set-up and maintain, its full potential should lie in the integration with Hadoop (HDFS) improving data management. The elasticity of such a cluster and the possibility to utilize free commodity resources is a real benefit.

The modifications required to an R script to utilize a cluster are minimal, but to unleash the full potential, additional efforts should be invested to customize and improve the code, especially with regards to data distribution.

Acknowledgments

The research has been supported by the Smart Data Program of the German Federal Ministry for Economic Affairs and Energy (1MT14001B).

REFERENCES

[1] W. Raghupathi, V. Raghupathi, Big data analytics in healthcare: promise and potential, Health Inf. Sci. Syst. 2 (1) (2014) 3.
[2] A.M. Noor, L. Holmberg, C. Gillett, A. Grigoriadis, Big data: the challenge for small research groups in the era of cancer genomics, Br. J. Cancer 113 (10) (2015) 1405–1412.
[3] R. Bellazzi, M. Diomidous, I.N. Sarkar, K. Takabayashi, A. Ziegler, A.T. McCray, Data Analysis and data mining: current issues in biomedical informatics, Methods Inf. Med. 50 (6) (2011) 536–544.
[4] P. Agarwal, K. Owzar, Next generation distributed computing for cancer research, Cancer Inform. 13 (Suppl. 7) (2014) 97–109.
[5] L. Griebel, H.-U. Prokosch, F. Köpcke, D. Toddenroth, J. Christoph, I. Leb, et al., A scoping review of cloud computing in healthcare, BMC Med. Inform. Decis. Mak. 15 (1) (2015) 17.
[6] R.C. Taylor, An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics, BMC Bioinformatics 11 (12) (2010) 1.
[7] P. Hodor, A. Chawla, A. Clark, L. Neal, cl-dash: rapid configuration and deployment of Hadoop clusters for bioinformatics research in the cloud, Bioinformatics 32 (2) (2016) 301–303.
[8] S. Jha, J. Qiu, A. Luckow, P. Mantha, G.C. Fox A tale of two data-intensive paradigms: applications, abstractions, and architectures. IEEE; 2014. pp. 645–652.
[9] D. Eddelbuettel CRAN task view: high-performance and parallel computing with R. https://cran.r-project.org/web/views/HighPerformanceComputing.html, 2016.
[10] SparkR. (R on Spark) [Internet]. https://spark.apache.org/docs/1.6.0/sparkr.html, 2016 (accessed 20.03.16).
[11] M. Liang, C. Trejo, L. Muthu, L.B. Ngo, A. Luckow, A.W. Apon Evaluating R-based big data analytic frameworks. IEEE; 2015. pp. 508–509.
[12] A.L. Dixon, L. Liang, M.F. Moffatt, W. Chen, S. Heath, K.C.C. Wong, et al., A genome-wide association study of global gene expression, Nat. Genet. 39 (10) (2007) 1202–1207.
[13] H. Wickham Advanced R—Performance [Internet]. http://adv-r.had.co.nz/Performance.html, 2016 (accessed 09.05.16).
[14] K. Broman Benchmark of linear regression routines [Internet]. http://kbroman.org/qtl2/assets/vignettes/linreg_benchmarks.html, 2016 (accessed 16.03.16).
[15] G. Ostruchov, W.-C. Chen, P. Patel, D. Schmidt Programming with big data in R [Internet]. http://r-pbd.org/, 2016 (accessed 11.04.16).
[16] M. Rak, M. Turtur, U. Villano, L. Pino A portable tool for running MPI applications in the cloud. IEEE; 2014. pp. 10–17.
[17] A. O'Driscoll, V. Belogrudov, J. Carroll, K. Kropp, P. Walsh, P. Ghazal, et al., HBLAST: parallelised sequence similarity–A Hadoop MapReducable basic local alignment search tool, J. Biomed. Inform. 54 (2015) 58–64.
[18] M.S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, M.J. Okoniewski, SparkSeq: fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision, Bioinformatics 30 (18) (2014) 2652–2653.
[19] Q. Zou, X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, K. Chen, Survey of MapReduce frame operation in bioinformatics, Brief Bioinform. 15 (4) (2014) 637–647.
[20] N.A. Davis, A. Pandey, B.A. McKinney, Real-world comparison of CPU and GPU implementations of SNPrank: a network analysis tool for GWAS, Bioinformatics 27 (2) (2011) 284–285.
[21] S. Lee, M.-S. Kwon, T. Park, CARAT-GxG: CUDA-accelerated regression analysis toolkit for large-scale gene-gene interaction with GPU computing system, Cancer Inform. 13 (Suppl. 7) (2014) 27–33.
[22] J. Buckner, Package gputools [Internet]. https://cran.r-project.org/web/packages/gputools/gputools.pdf, 2015.
[23] S. Yazar, G.E.C. Gooden, D.A. Mackey, A.W. Hewitt, Benchmarking undedicated cloud computing providers for analysis of genomic datasets, PLoS ONE 9 (9) (2014) e108490.
[24] A.M.S. Duarte, F.E. Psomopoulos, C. Blanchet, A.M.J.J. Bonvin, M. Corpas, A. Franc, et al., Future opportunities and trends for e-infrastructures and life sciences: going beyond the grid to enable life science data analysis, Front. Genet. 6 (2015) 197.
[25] J.L. Raisaro, E. Ayday, J.-P. Hubaux, Patient privacy in the genomic era, Praxis (Bern 1994) 103 (10) (2014) 579–586, http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=24800770&retmode=ref&cmd=prlinks.
[26] Z. Lin, A.B. Owen, R.B. Altman, Genetics. Genomic research and human subject privacy, Science 305 (5681) (2004) 183.
[27] T.A. Manolio, F.S. Collins, The HapMap and genome-wide association studies in diagnosis and therapy, Annu. Rev. Med. 60 (1) (2009) 443–456.
[28] S.R. Narum, Beyond Bonferroni: less conservative analyses for conservation genetics, Conserv. Genet. 7 (5) (2006) 783–787.