Application of Microarray Analysis on Computer Cluster and Cloud Platforms*

C. Bernau[1]; A.-L. Boulesteix[1]; J. Knaus[2]

[1]Department for Medical Informatics, Biometry and Epidemiology (IBE), Ludwig-Maximilians-University Munich, Munich, Germany;
[2]Institute of Medical Biometry and Medical Informatics, Albert-Ludwigs-University Freiburg, Freiburg, Germany

Keywords
Biostatistics, cloud computing, computing methodologies, microarray analysis, statistical computing

Summary
Background: Analysis of recent high-dimensional biological data tends to be computationally intensive as many common approaches such as resampling or permutation tests require the basic statistical analysis to be repeated many times. A crucial advantage of these methods is that they can be easily parallelized due to the computational independence of the resampling or permutation iterations, which has induced many statistics departments to establish their own computer clusters. An alternative is to rent computing resources in the cloud, e.g. at Amazon Web Services.
Objectives: In this article we analyze whether a selection of statistical projects, recently implemented at our department, can be efficiently realized on these cloud resources. Moreover, we illustrate an opportunity to combine computer cluster and cloud resources.
Methods: In order to compare the efficiency of computer cluster and cloud implementations and their respective parallelizations we use microarray analysis procedures and compare their runtimes on the different platforms.
Results: Amazon Web Services provide various instance types which meet the particular needs of the different statistical projects we analyzed in this paper. Moreover, the network capacity is sufficient and the parallelization is comparable in efficiency to standard computer cluster implementations.
Conclusion: Our results suggest that many statistical projects can be efficiently realized on cloud resources. It is important to mention, however, that workflows can change substantially as a result of a shift from computer cluster to cloud computing.

Correspondence to:
Christoph Bernau
Ludwig-Maximilians-University Munich
Department for Medical Informatics, Biometry and Epidemiology (IBE)
Marchioninistr. 15
81377 Munich
Germany
E-mail: bernau@ibe.med.uni-muenchen.de

Methods Inf Med 2013; 52: 65–71
doi: 10.3414/ME11-02-0043
received: November 7, 2011
accepted: March 5, 2012
prepublished: November 27, 2012

1. Introduction

With the arrival of new biological methods like gene expression microarrays and high throughput technology, statisticians are faced with huge amounts of data and computationally intensive methods for analyzing them. Moreover, the growing popularity of resampling approaches and permutation tests requires iterating a basic analysis many times which is demanding even more computational power. An important advantage of these statistical tools is that they can be easily parallelized which drastically reduces computation times provided that enough computational resources are available. Therefore, many statistics departments established their own computer clusters in order to apply these computationally intensive methods. These computer clusters are significant investments and especially for smaller departments it is questionable whether they are really necessary. A possible alternative to buying and maintaining one's own computer cluster could be renting computational resources 'in the cloud', e. g. at Amazon Web Services [1]. Meanwhile, several statistical open source projects like Bioconductor, which provides many routines for the statistical analysis of modern biological data, have developed Amazon Machine Images for their respective users [2]. In this paper we want to assess whether such statistical methods, which are implemented using the R language and environment [3], can sensibly be realized at Amazon Web Sercvices Elastic Compute Cloud (AWS EC2) and which problems might occur. Please note that AWS EC2 possibly does not match one's personal definition of the term cloud computing. The term "utility computing" [4] may be more adequate since one actually rents a computer cluster in a specific datacenter. We do not integrate machines from all over the world which might be a different interpretation of cloud computing.

In this article we will primarily focus on programming and parallelization aspects whereas we will mostly disregard financial, security, privacy and general technical aspects which will be subject of other articles

* Supplementary material published on our website www.methods-online.com

in this special topic. The closely related topic of costs for cloud computing is treated in more detail in [5]. In this perspective we arrange our paper in the following way: the first section will be devoted to basic benchmarks of the different EC2 instance types and their network capacity. In the following section, we will present two statistical projects whose realization in the cloud will be compared to the common computer cluster implementation. Subsequently, we will introduce an alternative approach which tries to combine machines from both sources. At the end, we will discuss the results and experiences we obtained while implementing our projects in the cloud.

2. Methods and Results

Our test of the usability and efficiency of the individually assembled computer clusters at AWS EC2 consists of four stages. At first we analyze the basic computation and network capacity of the different instances and compare them to other computer clusters used by our statistical department, namely the computer cluster at the IBE (ibec) and the mpp1-cluster (mpp) at the Leibniz Computing Center in Munich. Subsequently, we will implement in the cloud some important parts of two statistical projects which have already been realized on common computer clusters. This is principally the crucial part of our article since it will answer our main question, namely whether the statistical analysis of high-dimensional biological data can be reasonably realized in the cloud. In our last experiment we will go one step further by combining machines from several computer clusters at different departments and EC2. This experiment will present an opportunity to let machines from very heterogeneous sources jointly compute different sub-tasks of a statistical analysis.

2.1 Simple Trend Benchmarks

Synthetically benchmarking computer systems is basically a science on its own, which we do not want to treat here. But for an overview and estimation of runtime behavior of our real applications on the different EC2 instances, two simplistic synthetic benchmarks were used. Both are not benchmarking the complete (virtual) machine, but speed and memory aspects of the destination platform R.

Overall system benchmarks like the Spec suite [6] are far more elaborated and precise, but lack the integration of the R interpreter in the results. Moreover, the Spec test is rather difficult to configure and time consuming to run (Spec results for some instance types can be found in [7]). Our small benchmarks are using the arithmetic system of R – as desired for the biostatistical software used in the bigger tests afterward.

The performance of single cores of the CPU is tested by calculation of a specific subset of the Mandelbrot set [8], which is done using a recursive loop calculating whether a complex polynomial is bound or not. The inner loop is very small, so depending on the specific R environment it can be held and executed inside the processor's cache.

Our memory test is even simpler. It copies memory blocks inside a matrix guaranteed bigger than the processors cache. Overall runtime of the benchmarks is short, between one and three minutes depending on the system. Therefore the benchmark can be repeated many times (at least 20 iterations), giving a measurement of the variance in execution time, which is only taken inside the R program and therefore no startup time is included.

These short running benchmarks are giving a basic trend of the systems processor's speed using R and they provide an opportunity to quickly and therefore cheaply test the basic system performance, which is important due to the sheer amount of different EC2 instance types.

▶Table 1 provides an overview over the instances and servers we used for our analysis. We focused on the instance types m1.small, m1.large and c1.medium, which are basically the cheapest instances. Nonetheless, they provide enough memory for the statistical projects which we will analyze in the subsequent sections. The first impression is that all three instance types are slower than our computer cluster machines which were bought about four years ago. Especially the m1.small instance, which is EC2's standard instance, needs about two to three times more time for our first benchmark. For compatibility issues, this instance type is representing and running at the speed of a 1 Ghz CPU from 2006 when EC2 was introduced. Via virtualization, it only gets time shares of about 40% of a single physical core [9]. The m1.large and c1.medium instances are remarkably faster whereby both instances provide two virtual cores. In our ▶Supplementary Online File, we also provide the results of these two benchmarks for the larger, faster instances.

For computations with low memory demands c1.medium is basically the instance of choice having almost the same price per core as the slower m1.small instance. The m1.large instance provides 3.75 GB per core. However, its costs are four-fold. This price structure clearly indicates that writing memory efficient code can save money. Thus, it is quite probable that a statistical analysis in the cloud will produce some avoidable costs. For example, we usually

Table 1 Technical information and simple trend benchmark results for computer cluster and cloud instances. * Prices from the Amazon US east data center in North Virginia (September 2011). Prices vary between the individual data centers. ** Although running on a 2.7 GHz CPU this instance only features the performance of a 1.7 GHz Xeon processor [9].

Identifier/ API-Name	Pro-vider	CPU [Ghz]	Cores/ machine	RAM per core	Costs/ h*	Benchmark 1 Mean (sd) [s]	Benchmark 2 Mean (sd) [s]
ibec	IBE	2.7	4	1.5 GB	–	79.1 (1.03)	71.8 (0.7)
mpp	LRZ	2.0	16	2 GB	–	98.6 (2.5)	91.9 (3.6)
gvs1/3	LRZ	2.3	16	8 GB	–	100.6 (1.8)	97.8 (0.23)
m1.small	AWS	2.7	1	1.7 GB	0.085$	309.1 (6.9)	317.6 (6.6)
c1.medium	AWS	2.3**	2	0.88 GB	0.17$	132.6 (4.9)	135.1 (5.3)

had to perform our computations using only one core per c1.medium instance. It is also worth mentioning that the performance of the AWS instances is subject to higher variability.

Depending on the structure of the underlying Xen virtualization [10], cores and memory are separated quite well on the instances. But network and storage I/O cannot be separated this way and therefore all instances on a host can affect each other. In order to check the network capacity we used Rmpi's broadcast command [11] for sending a 2 gigabytes AffyBatch object [12], which is the commonly used R-object for microarray batches, to a varying number of worker cores.

As shown in ▶Figure 1, we can observe that this broadcast takes slightly longer on the EC2 instances and that their runtimes are growing faster if the number of worker cores is increased. In the case of the mpp-cluster, which is equipped with an infiniband network connection, the broadcasting time basically does not increase after 10 cores. However, network performance seems to be sufficient for most statistical methods where larger amounts of data are only sent once. Further analysis of the network connection and description of technical details can be found in the ▶Supplementary Online File. The result of these first tests are that computations might take longer on the chosen EC2 instances but the network capacity seems to be sufficient for the planned projects

2.2 Project 1: Optimization Bias

The first project to be conducted in the cloud deals with nested resampling procedures in the context of supervised classification with high-dimensional microarray data as predictors. Supervised classification algorithms, like Support Vector Machines (SVM) [13], are commonly assessed through a resampling procedure such as cross-validation. Such a resampling procedure whose aim is to estimate the misclassification rate is here denoted as external. Many supervised classification algorithms for high-dimensional data involve a tuning parameter, for instance the cost parameter for SVM that has to be adequately chosen. Simply trying several values of the tuning parameter and reporting the best performance only yields an optimistic bias [14]. A possible approach to avoid this optimization bias consists in performing nested resampling which means that, in each external resampling iteration, the optimal value of the tuning parameter is chosen by performing an additional internal resampling loop within the corresponding external training set.

The goal of this project is to estimate the bias induced by simply trying several predefined cost parameter values and reporting the minimal resampling misclas-

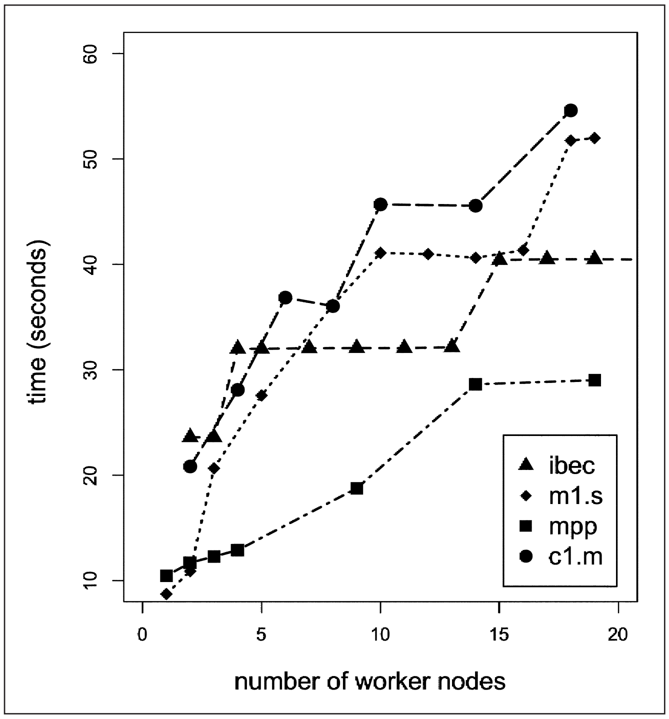

Figure 1 Time needed to broadcast an AffyBatch object of size 2GB to an increasing number of worker cores

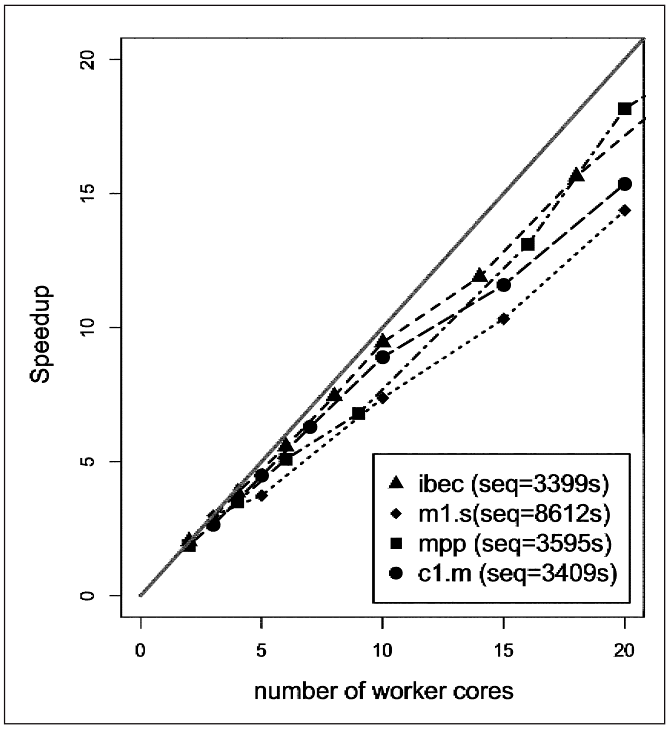

Figure 2 Speedup and sequential runtimes in seconds for the 100 resampling iterations in Project 1 on the various platforms

sification rate only. In order to analyze this bias, one has to reiterate many times the whole resampling approach with SVM as a classification method. Additionally, the whole procedure has to be repeated for several microarray data sets because the optimization bias, just as the cost parameter itself, strongly depends on the classification task at hand [15]. For financial reasons, we do not implement the whole project on the EC2 instances but demonstrate the efficiency of the parallelization for the most important part of this analysis. This central routine consists in the resampling approach for a specific cost parameter value. If this part scales sufficiently we can expect the whole project to be efficiently scaling as well. ▶Figure 2 shows the speedup for computer cluster and cloud instances, which consists of the sequential time divided by the parallel time for an increasing amount of workers. The gray line represents a perfect linear speedup on adding additional workers. Technically, this is based on a parallelization in R using MPI. We provide the code used for our analyses on a publically available Amazon Machine Image (▶Supplementary Online File). The consumed time is only measured inside the R program, so the overhead of instance startup is not included. Starting up 20 EC2 instances can take several minutes whereby this start up time is quite variable. Including this overhead into our analysis would severely distort our results since we analyze a small part of our project. Moreover, AWS charges fees per started hour and instance which also has to be considered appropriately. Both of these issues are negligible if the whole project is calculated. We can see that scaling is almost identical for cloud and computer cluster instances. Of course, the absolute computation times are remarkably higher for the m1.small instance. However, since the implementation scales well, the higher sequential runtime can be easily reduced by renting more instances. The main result of this experiment is that the main computation part can also be efficiently parallelized on EC2 instances.

2.3 Project 2: Normalization of Microarray Data

The characteristics of the second project are quite different since it combines a resampling approach with a computationally intensive task which can be parallelized itself. Gene expression microarray data have to be normalized before they can be used for high-level statistical analysis. Since in resampling procedures training and test data sets have to be strictly separated and classifiers should be trained without "seeing" the test data, microarrays should in principle be normalized either individually or, for multi-array approaches, in each resampling iteration anew. In the latter case, which is considered here, two variants are conceivable: the data can be normalized either separately for training and test set or an add-on normalization procedure can be used to normalize the test set subsequently using normalization parameters estimated from the training set [16]. In practice, however, researchers often normalize the whole data set only once before splitting it into training and test sets for computational reasons. Doing so, they indirectly violate the separation between training and test sets. The goal of project 2 is to analyze the effect of this imperfect separation on the classification results by comparing this approach to the more computationally intensive variants mentioned above performing normalization in each iteration anew and requiring a strong computational effort.

The core part to be analyzed consists in 6 repetitions of 5-fold cross validation procedure for a relatively small microarray data set of size n = 47 [17]. In this case we have two possible options for parallelization. The first option consists in parallelizing the normalization step itself based on a distributed data approach. This approach will also decrease the amount of memory necessary to perform the task which therefore also might decrease EC2 costs. The corresponding implementation can be found in the package affyPara [18]. This package was actually designed with larger data sets in mind (▶Supplementary Online File). Nonetheless we can compare the efficiency of the parallelization in the cloud to its efficiency on computer cluster machines. As can be seen from ▶Figure 3, the parallelization is scalable up to about five cores and saturates afterwards, whereby the speedup curves are comparable for computer cluster and cloud instances.

The actual implementation of the project is based on the parallelization of the resampling iterations. On our local computer cluster, the serial implementation requires about two hours even for this

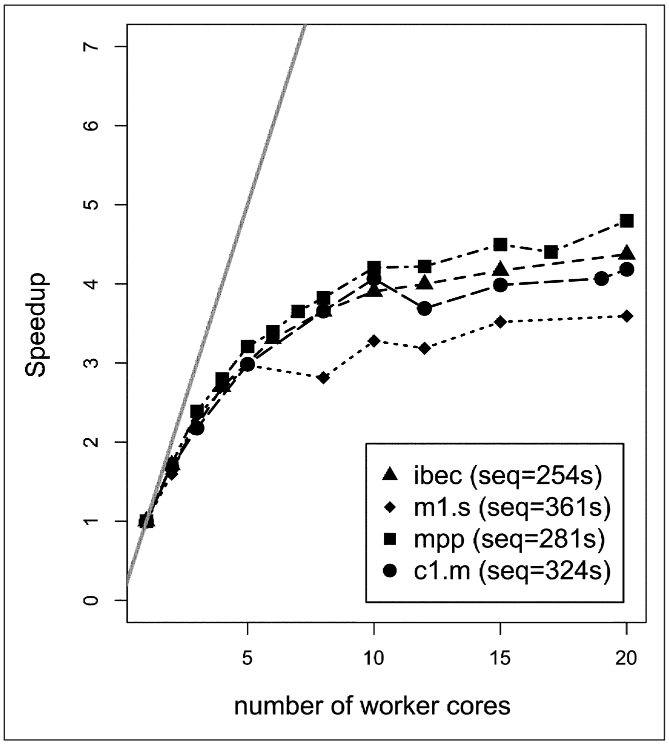

Figure 3
Speedup and sequential runtime in seconds for an AffyPara-Preprocessing routine on the different platforms

small core part which clearly demonstrates the need of parallelization. The second version scales well up to at least 20 cores which is the upper limit for EC2 instances in our analysis. We know from an analysis on different computer clusters that this implementation can scale up to approximately 125 cores depending on the size of the data set at hand. Again there is not much difference between the efficiency of the parallelization on computer cluster and cloud instances. Speedups are almost perfectly linear on all architectures. For the m1.small instances computation times are more than doubled in comparison to our computer cluster machines and to the other EC2 instances. However, this can be alleviated by simply taking more instances or the more expensive, faster instances. Of course, one advantage of slower instances is that parallelization yields even more profit as far as the reduction of the absolute computation time is concerned. Please note that the required memory of our computation forces the second c1.medium core to remain idle in this implementation.

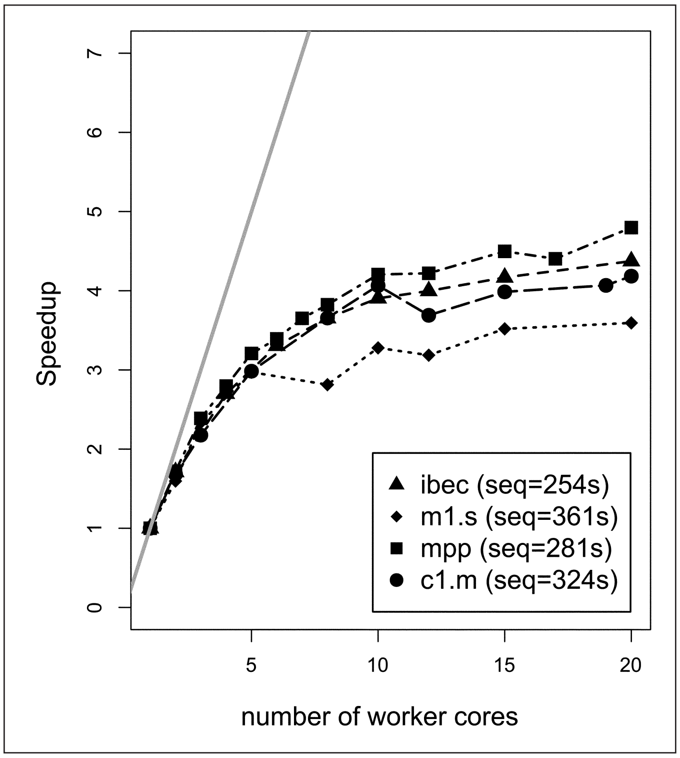

Figure 4
Speedup and sequential runtime in seconds for Project 2 on computer cluster and cloud instances

2.4 A Hybrid Cloud Solution: Combining Computer Cluster and Cloud Resources

An advantage of the cloud is its scalability, i.e. the possibility to get more resources if more resources are needed. If a given infrastructure is fully utilized in peak times, the possibility to add machines from the cloud to existing computer clusters would be helpful. With some efforts, EC2 instances can be directly integrated into local computer clusters manually. Far more convenient is the usage of a system which can handle this automatically. In the following, we show how to use the "data structure server" Redis [19] together with its R connector doRedis [20] to combine local computer clusters and cloud instances. As networking is partly routed over the public Internet, it is clearly a bottleneck. But for embarrassingly parallel [21] programs like the examples shown above, networking speed is not essential.

The concept of the Redis approach is that the server manages all the subtasks (e. g. a single resampling iteration) of a job and sends them to whatever worker node we connect to it. To illustrate this approach we use an enlarged version of the core part of Project 1. We compute 3000 subsampling steps instead of one hundred. Once we have a function for a single subsampling iteration (here: clrun) the whole approach can be implemented in less than ten lines (see appendix). An important optional feature is the opportunity to specify the number of tasks a certain connected worker machine shall perform. If we do not want some worker machines to participate until the very end of the whole computation we can set this parameter to a small number and the corresponding worker machine will stop as soon as it has performed the respective number of tasks. A really big advantage of this separation of job administration and computation is that we do not encounter any problems if we temporarily do not have any workers at our disposal, i.e. our job can be paused. Results and the description of the individual subtasks of the job are stored independently from the workers. For example, if one wants to use the staff's computers for larger computation tasks at night, one can use a small script connecting the respective computer to the Redis server as soon as the staff leaves the office. In our example we combined resources from the LRZ (gvs1, gvs3), IBE and EC2.

Since this example was intended to demonstrate the sheer possibility of this approach our pool of worker machines primarily consisted of free computing resources. ►Figure 5 shows how many tasks have been performed by the different machines. Of course, the main part of the tasks has been processed by the 30 IBEC cores. However, we can see that the c1.medium instances processed more tasks per core than the IBEC machines although their connection to the Redis server is slower. In practice, the actual number of integrated cloud instances can be chosen depending on the availability of free computing resources and the urgency of the job.

In this context, one has to consider some of the problematic aspects of cloud computing. For example, some of the machines in the previous illustration used R 2.12 whereas other ones used R 2.13. The individual worker cores actually do not know about each other. These consistency problems might be overcome by the solutions already available at AWS, namely virtual machines and images.

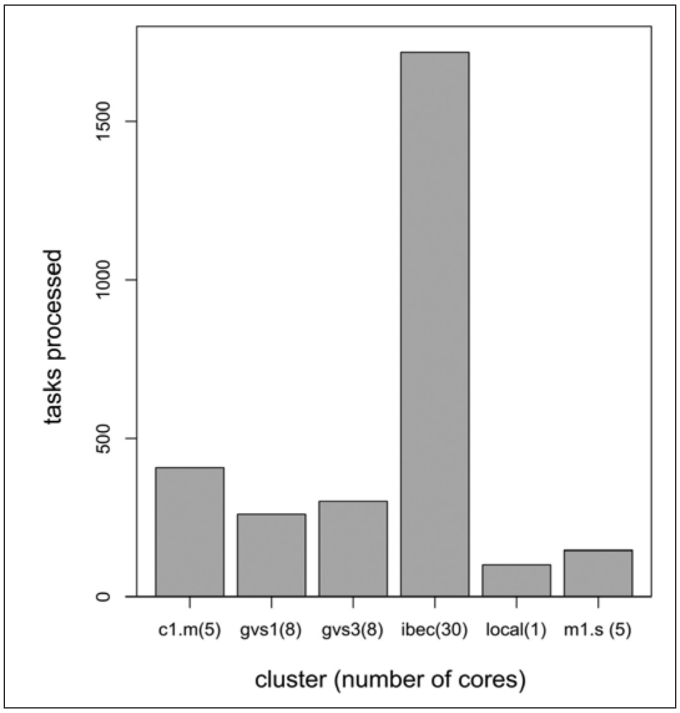

Figure 5
Distribution of the 3000 tasks among the different resources for the redis server based combination of computer cluster and cloud instances

3. Discussion

As illustrated in this paper, computationally intensive statistical projects can be realized in the cloud. Basically, EC2 provides different instance types for almost all computational requirements a statistical analysis of high-dimensional biological data may need. The default instance, m1.small, is quite slow. However, the c1.medium instance, which costs only slightly more per CPU, can keep up with usual computer cluster machines as far as speed is concerned. Since our projects scaled sufficiently well on the EC2 instances it is also possible to reduce computation time by simply increasing the number of instances. Especially if one needs more than 850MB RAM which means that the second c1.medium core remains idle, one can rent a larger number of m1.small instances instead. If projects are even more memory-intensive they can be realized on the m1.large instance with 7.5GB RAM and sufficiently fast processors. For financial reasons we run our projects on smaller data sets. As shown in Table 1 in the ►Supplementary Online File, EC2 provides several instance types which offer enough memory for performing the same analysis on remarkably larger data sets for remarkably higher prices. From our experience we do not expect the implementations of our projects to scale worse if applied to larger data sets.

We would also like to emphasize here that we performed our analysis on the small EC2 instances in order to reduce costs. Of course, it is possible to rent faster, more expensive instances at EC2 which will further reduce the computation time in the cloud such that runtimes might be even smaller than on the computer clusters we used.

Overall, we can state that the efficiency of the parallelization in the cloud is comparable to computer cluster implementations for the presented applications and therefore most likely also for a wide range of other biostatistical applications.

Finally, we would like to discuss some experiences concerning the workflow in the cloud and on computer clusters which might also have an impact on one's individual choice. A very convenient feature of utility computing is its high flexibility. You can build your computer cluster at EC2 almost whenever you want and you can customize it accordingly to your individual needs. You also have the opportunity to rent single instances for sequential code, e.g. if the memory requirements of your computations might exceed the capacity of your local machines. Another advantage is that you have full control over your resources and you do not have to resort to job scheduling systems coordinating the needs of the different computer cluster users. On the contrary, the workflow on cloud resources is remarkably changed by the fact that one has to pay for each computation. One always reflects twice whether a specific computation is really necessary which limits the opportunity to test and experiment. Of course, this will induce researches to plan their computations more elaborately and avoid unnecessary tests which might also be considered a positive side-effect. Cloud computing, however, also has some hurdles for inexperienced users. Simple mistakes like forgetting to shut down instances or moving results to the own computer can easily produce unnecessary costs. Moreover, one must always estimate time and memory needs in order to decide which instances should be preferred for the task at hand. Consequently, currently AWS and similar services are probably not the best place for the first steps in parallel computing. Another problem concerns experienced users as well. After finding an error or recognizing that a certain part of one's algorithm has to be changed, all the computations have to be repeated. Also, discovered problems in used packages can force a recalculation. In an advanced stage of a project this means that the computation costs will effectively multiply.

After all, utility computing seems to be a tool which has to be applied in a considered way. Instead of switching entirely to cloud computing, a possible way to get started consists of supporting one's own computer cluster resources by renting cloud instances in times of workload peaks. An option to flexibly combine computer cluster and cloud resources even within the scope of a single computation task consists in the redis-server based implementation we applied in this article. After using both resources for a while, one will probably be able to assess which solution better matches one's individual needs.

Acknowledgments

We would like to thank Ferdinand Jamitzky for helping us to implement our analysis methods in parallel and run them at the Leibniz Computing Center and Monika Jelizarow for providing details about the computation hours of her project on global tests for ordinal data. Moreover, we want to thank the anonymous referees for their helpful remarks and suggestions for improvements.

References

1. Amazon Web Services (2011): Amazon Elastic Compute Cloud (EC2). Available: http://aws.amazon.com/ec2 (accessed: Jan 20, 2012).
2. Bioconductor (2011): Bioconductor – Cloud AMI. Available: http://www.bioconductor.org/help/bioconductor-cloud-ami/ (accessed: Oct 28, 2011).
3. R development Core Team (2011): R: A Language and Environment for Statistical Computing. Available: http://www.R-project.org/ (accessed: 2012 Jan 14)
4. Wikimedia Foundation. Wikipedia (2012): Utility Computing. Available: http://en.wikipedia.org/wiki/Utility_computing (accessed: Jan 29, 2012)
5. Knaus J, Hieke S, Binder H, Schwarzer G. Costs of Cloud Computing for a Biometry Department – A Case Study. Methods Inf Med 2013; 52: 72–79.
6. Standard Performance Evaluation Corporation (2011): Spec CPU [online]. Available: http://www.spec.org/benchmarks.html (accessed: 2011 Oct 28).
7. Coffey P, Beliveau J, Mogre N, Harner A (2011): Benchmarking the Amazon Elastic Compute Cloud (EC2) [online]. Available: http://www.wpi.edu.Pubs/E-project/Available/E-project-030811-115350/unrestricted/AmaznEC2_MQP_Final.pdf (accessed: Oct 23, 2011).
8. Mandelbrot BB. The fractal geometry of nature. New York: W.H. Freeman and Company; 2003.
9. Evangelinos C, Hill CN. Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Proceedings of CCA-08.2008.
10. Citrix Systems (2012): Xen. Available: http://xen.org/ (accessed Jan 25, 2012).
11. Yu H. (2010): Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface). R package version 0.5–9 (online). Available: http://CRAN.R-project.org/package=Rmpi (accessed: Oct 28, 2011).
12. Gautier L, Cope L, Bolstad B M, Irizarry RA. affy-analysis of Affymetrix GeneChip data at the probe level. Bioinformatics 2004; 20 (3): 307–315.
13. Hastie T, Tibshirani R, Friedman .H. The Elements of Statistical Learning. New York: Springer-Verlag; 2001.
14. Varma S, Simon, R. Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics 2006; 7: 91.
15. Bernau C, Augustin T, Boulesteix A-L. Correcting the optimally selected resampling-based error rate: A smooth analytical alternative to nested cross-validation. Department of Statistics: Technical Reports, Nr. 105, 2011. Available: http://epub.ub.uni-muenchen.de/12231/ (accessed: Jan 29, 2012).
16. Kostka D, Spang R. Microarray Based Diagnosis Profits from Better Documentation of Gene Expression Signatures. PLoS Computational Biology 2008, 4, e22.
17. Ancona N, Maglietta R, Piepoli A, D'Addabbo A, Cotugno ., Savino M, Liuni S, Carella M, Pesole G, Perri F. On the statistical assessment of classifiers using DNA microarray data. BMC Bioinformatics 2006; 19 (7): 387.
18. Schmidberger M, Vicedo E, Mansmann U. affyPara: Parallelized preprocessing methods for Affymetrix Oligonucleotide Arrays. Rpackage version 1.13.0, 2011 (online). Available: http://bioconductor.org/packages/2.9/bioc/html/affyPara.html (accessed: Oct 28, 2011).
19. Sanfilippo S, Noordhuis P. Redis. Version 2.4.2, 2011 (online). Available: http://redis.io/download (accessed: Oct 28, 2011).
20. Lewis BW. (): doRedis: Foreach parallel adapter for the redis package. R package version 1.0.4, 2011 (online). Available: http://CRAN.R-project.org/package=doRedis (accessed: Oct 28, 2011).
21. Wikimedia Foundation.Wikipedia. Embarrassingly parallel, 2012. Available: http://en.wikipedia.org/wiki/Embarrassingly parallel (accessed: Jan 28, 2012).