Economics of Cloud Computing: a Statistical Genetics Case Study

Jeremy M. R. MARTIN [a], Steven J. BARRETT [a], Simon J. THORNBER [a], Silviu-Alin BACANU [a], Dale DUNLAP [b], Steve WESTON [c]

[a] GlaxoSmithKline R&D Ltd, *New Frontiers Science Park, Third Avenue, Harlow, Essex, CM19 5AW, UK*
[b] Univa UD, *9737 Great Hills Trail, Suite 300, Austin, TX 78759, USA*
[c] REvolution Computing, *One Century Tower, 265 Church Street, Suite 1006, New Haven, CT 06510, USA*

Abstract. We describe an experiment which aims to reduce significantly the costs of running a particular large-scale grid-enabled application using commercial cloud computing resources. We incorporate three tactics into our experiment: improving the serial performance of each work unit, seeking the most cost-effective computation cycles, and making use of an optimized resource manager and scheduler. The application selected for this work is a genetics association analysis and is representative of a common class of embarrassingly parallel problems.

Keywords. Cloud Computing, Grid Computing, Statistical Genetics, R

Introduction

GlaxoSmithKline is a leading multinational pharmaceutical company which has a substantial need for high-performance computing resources to support its research into creating new medicines. The emergence of affordable, on demand, dynamically scalable hosted computing capabilities (commonly known as 'Cloud Computing' [1]) is of interest to the company: potentially it could allow adoption of an 'elastic' grid model whereby the company runs a reduced internal grid with the added capability to expand seamlessly on demand by harnessing external computing infrastructure. As these clouds continue to accumulate facilities comprising powerful parallel scientific software [2] and key public databases for biomedical research [3], they will become increasingly attractive places to support large, collaborative, *in silico* projects. This will improve as efforts [4,5] to evaluate and identify the underlying benefits continue within commercial and public science sectors.

In this article we explore the economic viability of transferring a particular Statistical Genetics application from the GSK internal PC grid to an external cloud. The application is written using the R programming language. One run of this application requires 60,000 hours of CPU time on the GSK desktop grid, split across 250,000 jobs which require approximately fifteen minutes of CPU time each. Each job uses approximately 50MB of memory and performs negligible I/O. The GSK desktop grid comprises the entire fleet of R&D desktop PCs within GSK: these are Pentium machines with an average speed of about 3 GHz. At any given time up to 1500 of these devices may be used concurrently, harvested for spare compute cycles using the Univa UD Grid MP system. Based on raw performance, each node in this grid is roughly 3 times faster than a 'small' instance on Amazon EC2 [6]. This suggests that the entire application would require approximately 180,000 hours of CPU time on the Amazon EC2 cloud which implies that the naïve cost for running the

application on the EC2, without any optimization, would be around $18,000 ($0.10 per CPU hour). This figure is too high to be attractive when compared with the running costs of the internal GSK grid infrastructure. However if we were able to reduce that price by an order of magnitude it would open up new possibilities for GSK to run an economical elastic grid.

In order to tackle this problem we have established a three-way collaboration between GSK, Univa UD which is a pioneering company in the area of grid and cloud computing resource management and scheduling, and REvolution Computing which specialises in high-performance and parallel implementation of the R programming language. Our cost reduction strategy is based on three tactics:

- Pursuit of low-cost computing resources
- Reduction of serial execution time
- Efficient scheduling and resource management

The rest of the paper is organized as follows. In section 1 we describe the Statistical Genetics grid application which we are using for this experiment. Section 2 gives a brief introduction to the R programming language. In section 3 we review the current status of cloud computing. Section 4 provides a detailed explanation of our methods, and the results are presented in section 5. Finally we review the results and make predictions for the future uptake of cloud computing in science and technology.

1. Development of Methods for Genetic Analysis Using Simulation

Discovery of genetic variants responsible for certain diseases or adverse drug reactions is currently a high priority for both governmental and private organizations. As described in Bacanu *et al* [7], the tools for discovery consist of statistical methods for detecting association between phenotype of interest and biomarkers (genotypes in this case). While these methods are general in spirit, the type of data they are applied to is unfortunately very specific. For instance, the data for a very common "genome scan" consists of around one million or more variables (markers) that can be very correlated regionally and, at the same time, the correlation structure is highly irregular and its magnitude varies widely between even adjacent genomic regions. Due to the size and irregularity of genetic data, performance of association methods has to be assessed for very large simulation designs with very time-consuming individual simulations. The computational complexity of the problems is further compounded when the number of available subjects is small; under these circumstances asymptotic statistical theory cannot be used and the statistical significance has to assessed via computationally-intensive permutation tests. Consequently, to successfully develop and apply methods, statistical genetics researchers need extensive computational power such as very large clusters, grid or cloud computing.

Association analysis is the statistical technique for linking the presence of a particular genetic mutation in subjects to the susceptibility of suffering from a particular disease.. To assess the presence of associations, subjects in study cohorts are divided into *cases* (those with the disease) and *controls* (those without). The DNA of each subject is *genotyped* at particular markers where relatively common mutations are present. The combination of genotype and phenotype (disease symptom) data is then analyzed to arrive at a statistical significance (for association) at each studied marker.

A typical design for testing association methods involves running many instances (sometimes more than 250,000) of an identical program, each with different input data or, if

permutations are necessary, a randomly permuted copy of the input data. Each instance produces for each marker a single data point within a simulated probability distribution. Once all the instances have completed the resultant probability distribution is used to approximate the statistical significance of the actual data, and hence whether there is a significant link between a particular genetic marker and the disease under study.

2. The R Programming Language

The R programming language[8] is a popular public-domain tool for statistical computing and graphics. R is a very high level function-based language and is supported by a vast online library of hundreds of validated statistical packages[9]. Because R is both a highly productive environment and is freely available it appeals both to those academics who are teaching the next generation of statisticians and to others who are developing new 'state-of-the-art' statistical algorithms, which are becoming increasingly complex as methods become progressively more sophisticated.

The flip-side of this level usefulness and programming productivity is that R programs may take a long time to execute when compared with equivalent programs written in low-level languages like C. However the additional learning and effort required to develop complex numerical and statistical applications using C is not a realistic option for most statisticians, so there have been many initiatives to make R programs run faster.

These fall into three general categories:

1. Task farm parallelisation: running a single program many times in parallel with different data across a grid or cluster of computers. This is the approach that has been used for the genetics application of this project – using the GSK desktop grid a program which would take *seven years* to run sequentially can complete in a few days.

2. Explicit parallelisation in the R code using MPI or parallel loop constructs (e.g. R/Parallel [10]). The drawback of this approach is that it requires the R programmer to have some understanding of concurrency in order to use the parallel constructs or functions within the program.

3. Speeding up the performance of particular functions by improved memory handling, or by using multithreaded or parallelised algorithms 'beneath the hood' to accelerate particular R functions e.g. REvolution R, Parallel R[11] or SPRINT[12]. With this approach the programmer does not need to make any changes to code – the benefits will be automatic. However at present only certain R functions have been optimised in this way – so this approach will only work for codes which require significant use of those functions.

For the purpose of this experiment we will be using a combination of approach 1 with approach 3. The code has already been enabled to run as a collection of 250,000 independent work units (as described above). By also aiming to reduce the serial execution time of the code (by improved memory handling rather than by multithreading or parallelised algorithms) we shall hope to reduce our overall bill for use of cloud virtual machines.

3. Cloud Computing

Cloud computing is a style of computing in which dynamically scalable and virtualized resources are provided as a service over the Internet. Major cloud computing vendors, such as Amazon, HP and Google, can provide a highly reliable, pay-on-demand, virtual infrastructure to organizations. This is attractive to small commercial organizations which do not wish to invest heavily in internal IT infrastructure, and who need to adjust the power of their computing resources dynamically, according to demand. It may also be attractive to large global companies, such as pharmaceutical companies [13], who will potentially be able to choose to set their internal computing capacity to match the demands of day-to-day operations, and provide an elastic capability to expand on to clouds to deal with spikes of activity. In either case the economics of cloud computing will be crucial to the likelihood of uptake 14].

As well as economic considerations, cloud computing comes with a number of additional complications and there exists a certain amount of skepticism in the Computing world about its true potential. Armbrust *et al* [15] list ten obstacles to the adoption of cloud computing as follows:

1. *Availability of Service.* Risks of system downtime might be unacceptable for business-critical applications.
2. *Data Lock-In.* Each cloud vendor currently has its own proprietary API and there is little interoperability which prevents customers from easily migrating applications between clouds.
3. *Data Confidentiality and Auditability*. Cloud computing offerings are essentially public rather than private networks and so are more exposed to security attacks than internal company resources. There are also requirements for system auditing in certain areas of business (such as Pharmaceuticals) which need to be made feasible on cloud virtual machines.
4. *Data Transfer Bottlenecks.* Migrating data to clouds is currently costly (around $150 per terabyte) and may also be very slow.
5. *Performance Unpredictability.* Significant variations in performance have been observed between cloud virtual machine instances of the same type from the same vendor. This is a common problem with virtualisation and can be a particular issue for high-performance computing applications which are designed to be load-balanced across homogeneous compute clusters.
6. *Scalable Storage*. Providing high-performance, shared storage across multiple cloud virtual machines is still considered to be an unsolved problem.
7. *Bugs in Large Distributed Systems*. Debugging distributed applications is notoriously difficult. Adding the cloud dimension increases the complexity and will require the development of specialised tools.
8. *Scaling Quickly.* Cloud vendors generally charge according to the quantity of resources that are reserved: virtual CPUs, memory and storage, even if they are not currently in use. Therefore a system which can scale the reserved resources up and down quickly according to actual usage, hence minimizing costs, would be very attractive to customers.

9. *Reputation Fate Sharing*. One customer's bad behaviour could potentially tarnish the reputation of all customers of a cloud, possibly resulting in blacklisting of IP addresses.
10. *Software Licensing*. Current licensing agreements tend be node-locked, assigned to named users, or floating concurrent licenses. What is needed for cloud computing is for software vendors to charge by the CPU hour.

Each of these matters needs to be carefully reviewed when a company makes a decision whether to adopt cloud computing. They are all subject to ongoing technical research within the cloud computing community. Scientific software standards also need to be consistently adhered to and specific issues such as random number generation addressed.

For the Genetics application described here, GSK is able to avoid data confidentiality issues by taking steps to make the data anonymous. Also data transfer bottlenecks and scalable storage are not major issues here as this problem is highly computationally bound. The concern of data lock-in is to be handled by Univa UD's Unicloud product, which provides a standard interface whichever cloud vendor is used. The R programming environment is open source and so there are no associated license costs.

4. Tactics for Reducing Cost

The main objective of our experiment is to reduce the cost of running the GSK Statistical Genetics application by an order of magnitude from our starting estimate of $18K. We are using three separate, largely independent mechanisms to achieve this.

4.1 Pursuit of Low-Cost Computing Resources

Cloud vendors usually offer several different varieties of virtual computer, with differing levels of resources and correspondingly different costs. We will be trialing a number of these from two vendors, Amazon and Rackspace, as listed in Tables 1 and 2. (Note that R is platform neutral: it runs on both Windows and Linux).

Table 1. Amazon EC2 instance types.

Instance	Memory	EC2 Compute Units	Storage	Architecture	Price per CPU hour (Linux)
Standard Small	1.7GB	1	160GB	32-bit	$0.10
Standard Large	7.5GB	4	850GB	64-bit	$0.40
Standard XL	15GB	8	1690GB	64-bit	$0.80
High CPU Medium	1.7GB	5	350GB	32-bit	$0.20
High CPU XL	7GB	20	1690GB	64-bit	$0.80

Table 2. Rackspace cloud instance types.

Memory	Storage	Price per CPU hour (Linux)
256MB	10GB	$0.015
512MB	20GB	$0.03
1024MB	40GB	$0.06
2048MB	80GB	$0.12

Note that the CPU power of each Amazon EC2 instance is given as a multiple of their so-called 'compute unit' which is claimed to be equivalent to a 1.0-1.2 GHz 2007 Opteron processor[6]. The CPU performance of Rackspace cloud instances, however, is not provided on their website[16] and so for the purpose of our project has been determined by running benchmarks.

4.2 Reduction of Serial Execution Time

Given our starting estimate of $18K to run our application on a cloud, it would be well worth investing effort in optimizing the serial performance of each of the 250,000 work units. This can be achieved in two ways: either by performance tuning the R code of the application directly, or by optimizing the R run-time environment.

REvolution Computing's version of the R runtime environment contains optimized versions of certain underlying mathematical functions with improved memory and cache utilization. Programs which make heavy use of the optimized functions may show a significant speed up in serial execution time. In section 4 we shall describe the results of using REvolution's tools in this case study as well as transforming the R code directly using application profiling.

Another potential approach to improving the serial performance would be to rewrite the code in an efficient low-level language such as C or Fortran. However, this would be a very costly undertaking because of the complexity of the R packages used in the program. Also the R program used for our Genetics Analysis is subject to frequent revisions and improvements and the flexibility to make frequent changes would disappear using this approach.

4.3 Efficient Scheduling and Resource Management

In order to achieve the highest throughput in a cloud environment, we need to be able to quickly provision a large environment, and also efficiently schedule a large number of jobs within that environment.

An HPC cloud is essentially a cluster of a large number of virtual machines. To be useful, we need the ability to quickly create a large clustered environment, consisting of dozens, or possibly hundreds or thousands of virtual cores. Ideally, this process should be highly scalable: requiring little (if any) additional effort to build a 1,000 node cluster, as opposed to a 2-node cluster. In our experiment, we used Univa UD's UniCloud product to provision the cluster.

UniCloud works as follows: one virtual machine ("installer node") is created manually within the cloud environment (for example, Amazon EC2). This machine is installed with CentOS Linux 5.2. The customers' account information is configured in a special file (which varies by cloud vendor), and this allows UniCloud to use the cloud vendor's API to create additional virtual machines programmatically. A UniCloud command ("ngedit") is used to define the configuration for additional nodes, and the "addhost" command is then

used to actually provision the nodes. This includes building the operating system, setting up `ssh` keys for password-less login, and installation of optional software packages.

One of the optional packages installed by UniCloud is Sun Grid Engine [17], which is used as the batch job scheduler. This allows us to submit all 250,000 jobs at once. The jobs are queued by SGE and then farmed out to different machines as they become available. Without a scheduler, it would be impossible to fully utilize all the compute resources.

5. Experimental Results

We ran a representative subset of jobs on three flavours of cloud VM, two from Amazon and one from Rackspace, using the UniCloud scheduler and resource manager in order to achieve an efficient throughput of jobs. Because of the large number of individual jobs that are run in our experiment, the scheduler was able to achieve near perfect efficiency in utilization of processor cores. We used these results to make the cost and execution time predictions in Table 3.

Table 3. Initial performance results.

	EC2 Standard XL	EC2 High CPU XL	Rackspace 256MB
Average Runtime Per Job	29.1 min	17.9 min	19.6 min
Jobs Per Hour Per Instance	8.25	26.82	12.23
Total Number of Jobs	250,000	250,000	250,000
Wall Clock Time (200 instances)	151.6 hr	46.6 hr	102.2 hr
Cost per hr	$0.80	$0.80	$0.015
Total Cost	$24,250.00	$7,458.33	$306.72

Using the Rackspace clouds works out over twenty times less expensive that the lowest cost seen with Amazon. But, there are fundamental differences between the two environments that make direct comparison difficult. An instance at Amazon EC2 provides a guaranteed minimum performance, as well as a certain amount of memory and disk space. Each Rackspace VM provides access to four processing cores, and a specific amount of memory (in our case, 256MB). Each physical machine at Rackspace actually has two dual-core processors, and 16GB of memory. There could be up to 64 separate 256MB VMs sharing this hardware. Each VM can use up to the full performance of the machine, if the other VMs are idle. But, it is possible (although unlikely) that the performance will be significantly less. Rackspace's provisioning algorithm attempts to avoid this problem by distributing new VMs instead of stacking them and most applications already deployed to run on the Rackspace cloud are not at all CPU-hungry.

In our testing, the Rackspace 256MB instance was the most cost efficient, because each of our jobs requires about 50MB of memory. So, four concurrent jobs can fully utilize one of these virtual machines without swapping. The scheduler was able to keep each of these cores constantly busy until all the jobs had been processed.

Note also that there is currently a limit of 200 concurrent VMs that a Rackspace user may hire in a single session, so Rackspace would appear to be significantly less scalable for HPC than Amazon. The best execution time on the Rackspace cloud is 102 hours. Amazon can do the analysis in 46 hours using 200 VM instances, and could potentially go much faster than this by creating 1000 or more concurrent VMs.

5.1 R Code Optimisation

Our performance optimization for the test R program was largely focused in two areas: language-level and processor optimizations.

The R language is a rich, easy to use, high-level programming language. The flexibility of the language makes it easy to express computational ideas in many ways, often simplifying their implementation. However, that flexibility also admits many less than optimal implementations. It is possible for several mathematically-equivalent R programs to have widely different performance characteristics.

Fortunately, R code is easy to profile for time and memory utilization, allowing us to identify bottlenecks in programs and compare alternate implementations. The Rprof function produces a detailed log of the time spent in every function call over the course of a running program, as well as optional detailed memory consumption statistics. There exist several excellent open-source profile analysis and visualization packages that work with Rprof, incuding 'proftools' and 'profr.' These are essential tools for R performance tuning.

Processor optimizations represent the other main approach to R performance optimization. Unlike the language-level optimization, processor optimizations are usually implemented through low-level numeric libraries. The R language relies on low-level libraries for basic computational kernels like linear algebra operations.

Running the Statistical Genetics R code using REvolution R showed a marginal performance improvement (6%) over running with the public domain version of R. In order to explore further we used the RProf profiling package to analyse the code. This shows a breakdown of how much time the program spends in each function: for our program the RProf output begins as follows:

```
Each sample represents 0.02 seconds.
Total run time: 1774.31999999868 seconds.

Total seconds: time spent in function and callees.
Self seconds: time spent in function alone.

   %       total       %       self
 total    seconds     self    seconds    name
100.00   1774.32      0.00      0.00     "source"
100.00   1774.32      0.00      0.02     "eval.with.vis"
 99.99   1774.18      0.00      0.00     "t"
 99.99   1774.18      0.34      6.12     "sapply"
 99.99   1774.18      0.00      0.00     "replicate"
 99.99   1774.14      1.09     19.34     "lapply"
 99.99   1774.10      9.79    173.74     "FUN"
 99.98   1774.02      0.09      1.68     "getpval"
 64.55   1145.30      1.51     26.88     "lm"
 32.21    571.48      0.03      0.60     "anova"
 32.17    570.88      0.02      0.40     "anova.lm"
 30.21    536.02      1.22     21.72     "eval"
 29.92    530.80      0.18      3.20     "anova.lmlist"
```

This shows that most of the execution time was within the ‘lm’ function (and related functions) for fitting linear models. The test program runs a large number of linear models, which are ultimately based on the QR matrix factorization. The particular QR factorization

used by R has not yet been adapted to take advantage of tuned numeric libraries. This explains why the speedup obtained was small. (However, optimization of the QR factorization code is work in progress and we expect it will provide about 5 to 10% further speed.)

Fortunately it turns out that our program's performance may be significantly improved by a code transformation. The original code contains the following 'for loop' construct:

```
for(i in 1:ncov){
      if(i==1){
            model0<-paste(model0,"covs[,",i,"]")
            model1<-paste(model1,"+","covs[,",i,"]")
      } else{
            model0<-paste(model0,"+","covs[,",i,"]")
            model1<-paste(model1,"+","covs[,",i,"]")
      }
      mod0<-lm(as.formula(model0))
      mod1<-lm(as.formula(model1))
      pvalcov[[i]]<-anova(mod0,mod1)[2,"Pr(>F)"]
}
```

Loops are notoriously inefficient in R. In this case we noticed that each iteration of the loop is independent so we may safely transform this by moving the loop body into a function and then replacing the 'for' statement with a more efficient 'apply' operation (which is equivalent to a functional 'map' statement). We produced an optimized version that runs significantly faster (even using the public-domain version of R).

```
xxmod <- function (i) {
      if(i==1){
            model0<-paste(model0,"covs[,",i,"]")
            model1<-paste(model1,"+","covs[,",i,"]")
      } else{
            model0<-paste(model0,"+","covs[,",i,"]")
            model1<-paste(model1,"+","covs[,",i,"]")
      }
      mod0<-lm(as.formula(model0))
      mod1<-lm(as.formula(model1))
      anova(mod0,mod1)[2,"Pr(>F)"]
}
pvalcov <- sapply (1:ncov, xxmod)
```

Table 4 shows the results of re-running the experiment using the optimized version of the R program (but still using the public domain version of R). We see a further 20% cost reduction.

Table 4. Performance results following code optimisation.

	EC2 Standard XL	EC2 High CPU XL	Rackspace 256MB
Average Runtime Per Job	19.2 min	13.5 min	15.8 min
Jobs Per Hour Per Instance	12.50	35.56	15.24
Total Number of Jobs	250,000	250,000	250,000
Wall Clock Time (200 instances)	100.0 hr	35.2 hr	82.0 hr
Cost per hr	$0.80	$0.80	$0.015
Total Cost	$16,000.00	$5,625.00	$246.09

So we have brought down the total cost of running our application from $18,000 to around $250 using the methods described in this paper. The bottom line is that the reason Rackspace is such a good deal is that it *perfectly* matches the jobs to be run. That is, a 256MB instance is very cheap, and it just so happens that 4 of these jobs fit perfectly in 256MB. Also this instance only has 10GB of disk but our job mix does not need much disk. (The two EC2 instances that we tested provided 15GB and 7GB of RAM respectively and 1690GB of storage space, most of which was unused.) The UniCloud tool made it possible to use the Rackspace cloud instance efficiently and the R profiling tools also provided a reduction in serial performance time, applicable across every work unit.

An important caveat to the Rackspace lowest cost is that it is a best case scenario for that cloud infrastructure, when it is relatively lightly loaded with high-performance jobs. Since the performance of cloud instances is potentially highly variable, the cloud brokerage layer would ideally be set up to constantly monitor the performance of the resources under its control and incorporate the ability to feedback information to the customer and to automatically shut down badly performing VMs.

Note that the cost figures presented above are based purely on the cloud computing resource usage – they do not include any software license fees for UniCloud or REvolution R. Actually REvolution R is an open source product and UniCloud is based on the open source Sun Grid Engine product so it would be possible for a programmer to implement the solution described in this article without incurring any software license costs. However the provision of a commercial cloud brokerage system offering simple standardized access to a variety of cloud offerings, allowing its user to shop around for the most suitable virtual infrastructure will be attractive to organizations such as GSK.

6. Conclusions

Use of commercial clouds for Grid computing may seem surprisingly expensive at first sight and may appear off-putting to companies that require HPC services. However, by taking a significant example from a major pharmaceutical company, we have shown how to reduce these costs using a combination of cloud brokerage, efficient scheduling, and application tuning.

The application described here was computationally bound, having negligible memory and I/O requirements. If we were to look at different classes of applications: such as those which use large-scale databases, then we would have to consider a wider spectrum of economic variables: such as costs for cloud storage, and cloud data transfer. Applications requiring large amounts of memory would not show the same cost benefits in using the Rackspace cloud as we have achieved here. Another important variable is level of data security: customers such as GSK will require watertight guarantees of security from vendors before sending any commercially sensitive data to a cloud environment.

Going forward we foresee a model of cloud brokerage emerging whereby a layer of middleware is provided to help satisfy customers constraints in utilising software services based on factors such as cost or execution time. In our example we made cost the primary factor which led us to choosing the Rackspace cloud offering. Had execution time been our driver we would have opted for Amazon.

The issue of ‘data lock in’ is particularly relevant to the experiment described here. At present there is no recognised standard for connecting to cloud resources and each vendor provides its own unique programming interface. By using a middleware layer, such as UniCloud, a cloud customer is insulated from these differences and is easily able to exploit

potential economic or performance advantages that we have shown to be possible by careful choice of cloud supplier. Without that additional layer, the additional programming costs required for moving applications around between different clouds could be restrictive.

Once the brokerage infrastructure is in place to allow customers dynamic access to powerful and economically attractive computing resources, and major security issues have been ironed out, we foresee major uptake of cloud computing within the scientific community both for collaborative activities and also for metered use of commercial software and database services.

At present there is still a great deal of work to do for this vision to become reality. In the meantime if you can find a vendor that has an instance that perfectly matches your workload (or, if it's possible to vary your workload to perfectly match an inexpensive instance), cloud computing can be *very* inexpensive.

References

[1] Rajkumar Buyya, Chee S. Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer System.* Vol. 25, No. 6. (2009), pp. 599-616. Elsevier.

[2] Michael Schatz. CloudBurst: Highly Sensitive Short Read Mapping with MapReduce http://sourceforge.net/apps/mediawiki/cloudburst-bio/index.php?title=CloudBurst

[3] Amazon Web Services for Biology. http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=246

[4] Bateman A, Wood M.(2009) Cloud computing. *Bioinformatics.* Oxford Journals (Oxford University Press, Oxford, UK) Jun 15;25(12):1475

[5] Brian D. Halligan, Joey F. Geiger, Andrew K. Vallejos, Andrew S. Greene and Simon N. Twigger. Low Cost, Scalable Proteomics Data Analysis Using Amazon's Cloud Computing Services and Open Source Search Algorithms. *J. Proteome Res*., 2009, 8 (6), pp 3148–3153. ACS publications.

[6] Amazon Elastic Cloud. http://aws.amazon.com/ec2/

[7] Bacanu SA, Nelson MR and Ehm MG. Comparison of association methods for dense marker data (2008). *Genet Epidemiol*. 32(8):791-9. Wiley-Liss, Inc.

[8] Sun Grid Engine. http://www.sun.com/software/sge/

[9] R Development Core Team (2008) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

[10] Gonzalo Vera, Ritsert C Jansen and Remo L Suppi. R/parallel – speeding up bioinformatics analysis with R, *BMC Bioinformatics* 2008, 9:390

[11] REvolution Computing. http://www.revolution-computing.com/

[12] Jon Hill , Matthew Hambley , Thorsten Forster , Muriel Mewissen , Terence M Sloan , Florian Scharinger , Arthur Trew and Peter Ghazal. SPRINT: A new parallel framework for R. *BMC Bioinformatics* 2008, 9:558

[13] Rick Mullin. The New Computing Pioneers, *Chemical and Engineering News* Volume 87, Number 21 pp. 10-14. ACS publications.

[14] Derrick Kondo, Bahman Javadi, Paul Malecot, Franck Cappello, David P. Anderson. Cost-benefit analysis of Cloud Computing versus desktop grids. *Proceedings of IEEE International Symposium on Parallel & Distributed Processing 2009* pp 1-12

[15] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia, Above the Clouds: A Berkeley View of Cloud Computing, *Technical Report No. UCB/EECS-2009-28*, Electrical Engineering and Computer Sciences, University of California at Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[16] The Rackspace Cloud, http://www.rackspacecloud.com/cloud_hosting_products/servers

[17] Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, Ulrich Mansmann. State of the Art in Parallel Computing with R. *Journal of Statistical Software,* Vol. 31, No. 1. (June 2009), pp. 1-27. American Statistical Association.