Large-Scale DNA Sequence Analysis in the Cloud: A Stream-Based Approach

Romeo Kienzler[1], Rémy Bruggmann[2], Anand Ranganathan[3], and Nesime Tatbul[1]

[1] Department of Computer Science, ETH Zurich, Switzerland
romeok@student.ethz.ch, tatbul@inf.ethz.ch
[2] Bioinformatics, Department of Biology, University of Berne, Switzerland
remy.bruggmann@biology.unibe.ch
[3] IBM T.J. Watson Research Center, NY, USA
arangana@us.ibm.com

Abstract. Cloud computing technologies have made it possible to analyze big data sets in scalable and cost-effective ways. DNA sequence analysis, where very large data sets are now generated at reduced cost using the Next-Generation Sequencing (NGS) methods, is an area which can greatly benefit from cloud-based infrastructures. Although existing solutions show nearly linear scalability, they pose significant limitations in terms of data transfer latencies and cloud storage costs. In this paper, we propose to tackle the performance problems that arise from having to transfer large amounts of data between clients and the cloud based on a streaming data management architecture. Our approach provides an incremental data processing model which can hide data transfer latencies while maintaining linear scalability. We present an initial implementation and evaluation of this approach for SHRiMP, a well-known software package for NGS read alignment, based on the IBM InfoSphere Streams computing platform deployed on Amazon EC2.

Keywords: DNA sequence analysis, Next-Generation Sequencing (NGS), NGS read alignment, cloud computing, data stream processing, incremental data processing.

1 Introduction

Today, huge amounts of data is being generated at ever increasing rates by a wide range of sources from networks of sensing devices to social media and special scientific devices such as DNA sequencing machines and astronomical telescopes. It has become both an exciting opportunity to use these data sets in intelligent applications such as detecting and preventing diseases or spotting business trends, as well as a major challenge to manage their capture, transfer, storage, and analysis.

Recent advances in cloud computing technologies have made it possible to analyze very large data sets in scalable and cost-effective ways. Various platforms and frameworks have been proposed to be able to use the cloud infrastructures

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part II, LNCS 7156, pp. 467–476, 2012.

for solving this problem such as the MapReduce framework [2], [4]. Most of these solutions are primarily designed for batch processing of data stored in a distributed file system. While such a design supports scalable and fault-tolerant processing very well, it may pose some limitations when transferring data. More specifically, large amounts of data has to be uploaded into the cloud before the processing starts, which not only causes significant data transfer latencies, but also adds to the cloud storage costs [19], [26].

In this short paper, we mainly investigate the performance problems that arise from having to transfer large amounts of data in and out of the cloud based on a real data-intensive use case from bioinformatics, for which we propose a stream-based approach as a promising solution. Our key idea is that data transfer latencies can be hidden by providing an incremental data processing architecture, similar in spirit to pipelined query evaluation models in traditional database systems [15]. It is important though that this is done in a way to also support linear scalability through parallel processing, which is an indispensable requirement for handling data and compute-intensive workloads in the cloud. More specifically, we propose to use a stream-based data management architecture, which not only provides an incremental and parallel data processing model, but also facilitates in-memory processing, since data is processed on the fly and intermediate data need not be materialized on disk (unless it is explicitly needed by the application), which can further reduce end-to-end response time and cloud storage costs.

The rest of this paper is outlined as follows: In Section 2, we describe our use case for large-scale DNA sequence analysis which has been the main motivation for the work presented in this paper. We present our stream-based solution approach in Section 3, including an initial implementation and evaluation of our use case based on the IBM InfoSphere Streams computing platform [5] deployed on Amazon EC2 [1]. Finally, we conclude with a discussion of future work in Section 4.

2 Large-Scale DNA Sequence Analysis

Determining the order of the nucleotide bases in DNA molecules and analyzing the resulting sequences have become very essential in biological research and applications. Since 1970s, the Sanger method (also known as dideoxy or chain terminator method) had been the standard technique [22]. With this method, it is possible to read about 80 kilo *base pairs* (bp) per instrument-day at a total cost of $150. The Next-Generation Sequencing (NGS) methods, invented in 2004, dramatically increased this per-day bp throughput, and therefore, the amount of data generated that needed to be stored and processed [27]. To compare with the Sanger method above, with NGS, the cost for sequencing 80 *kbps* has fallen to less than $0.01 and is done in less than 10 seconds. Table 1 shows an overview of speed and cost of three different NGS technologies compared to the Sanger method. The higher throughput and lower cost of these technologies have led to the generation of very large datasets that need to be efficiently analyzed. As stated by Stein [25]:

Table 1. Compared to the Sanger method, NGS methods have significantly higher throughput at a significant fraction of their costs

	Sanger	**Roche 454**	**Illumina 2k**	**SOLID 5**
read length	700-900	500	100	75
GB per day	0.00008	0.5	25	42
cost per GB	$2,000,000	$20,000	$75	$75

> *"Genome biologists will have to start acting like the high energy physicists, who filter the huge datasets coming out of their collectors for a tiny number of informative events and then discard the rest."*

NGS is used to sequence DNA in an automated and high-throughput process. DNA molecules are fragmented into pieces of 100 to 800 *bps*, and digital versions of DNA fragments are generated. These fragments, called *reads*, originate from random positions of DNA molecules. In re-sequencing experiments the reads are mapped back to a reference genome (e.g., human) [19] or - without a reference genome - they can be assembled *de novo* [23]. However, de novo assembly is more complex due to the short read length as well as to potential repetitive regions in the genome. In re-sequencing experiments, polymorphisms between analyzed DNA and the reference genome can be observed. A polymorphism of a single bp is called *Single Nucleotide Polymorphism (SNP)* and is recognized as the main cause of human genetic variability [9]. Figure 1 shows an example, with a reference genome at the top row and two SNPs identified on the analyzed DNA sequences depicted below. As stated by Fernald et al, once NGS technology becomes available on a clinical level, it will become part of the standard healthcare process to check patients' SNPs before medical treatment (a.k.a., "personalized medicine") [12]:

> *"We are on the verge of the genomic era: doctors and patients will have access to genetic data to customize medical treatment."*

Aligning NGS reads to genomes is computationally intensive. Li et al give an overview of algorithms and tools currently in use [19]. To align reads containing SNPs, probabilistic algorithms have to be used, since finding an exact match between reads and a given reference is not sufficient because of polymorphisms and sequencing errors. Most of these algorithms are based on a basic pattern called *seed and extend* [8], where small matching regions between reads and the reference genome are identified first (seeding), and then further extended. Additionally, to be able to identify seeds that contain SNPs, a special algorithm that allows for a certain difference during seeding needs to be used [16]. Unfortunately, this adaptation further increases the computational complexity. For example, on a small cluster used by FGCZ [3] (25 nodes with a total of 232 CPU compute cores and 800 GB main memory), a single genome alignment process can take up to 10 hours.

Read alignment algorithms have been shown to have a great potential for linear scalability [24]. However, sequencing throughput increases faster than

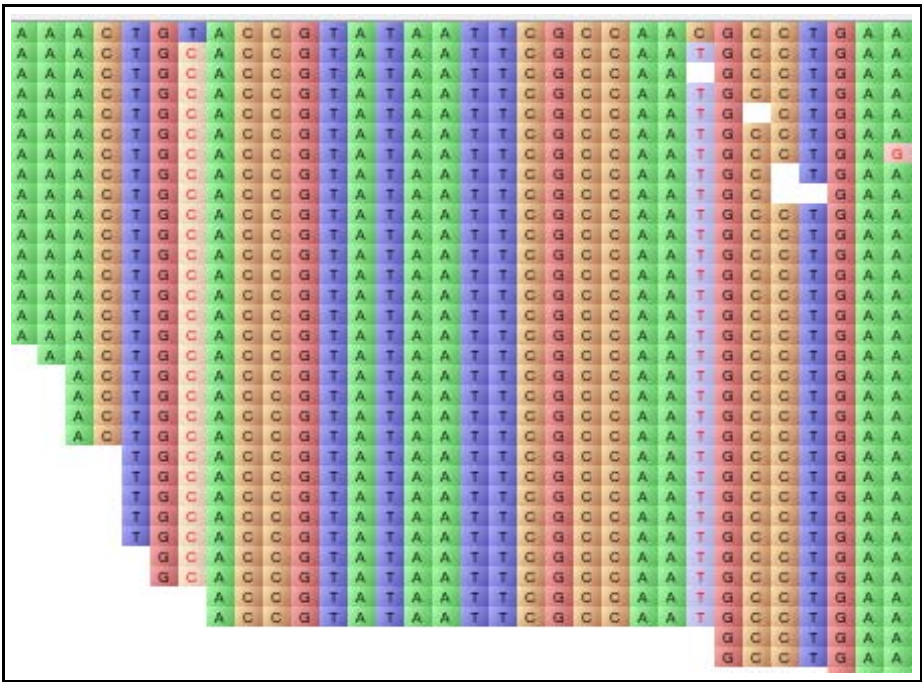

Fig. 1. SNP identification: The top row shows a subsequence of the reference genome. The following rows are aligned NGS reads. Two SNPs can be identified. T is replaced by C (7th column) and C is replaced by T (25th column). In one read (line 7), a sequencing error can be observed where A has been replaced by G (last column). Source: http://bioinf.scri.ac.uk/tablet/.

computational power and storage size [25]. As a result, although NGS machines are becoming cheaper, using dedicated compute clusters for read alignment is still a significant investment. Fortunately, even small labs can do the alignment by using cloud resources [11]. Li et al state that cloud computing might be a possible solution for small labs, but also raises concerns about data transfer bottlenecks and storage costs [19]. Thus, existing cloud-based solutions such as CloudBurst [24] and Crossbow [17] as well as the cloud-enabled version of Galaxy [14] have a common disadvantage: before processing starts, large amounts of data has to be uploaded into the cloud, potentially causing significant data transfer latency and storage costs [26].

In this work, our main focus is to develop solutions for the performance problems that stem from having to transfer large amounts of data in and out of the cloud for data-intensive use cases such as the one described above. If we roughly capture the overall processing time with a function $f(n, s) \propto cs + \frac{s}{n}$, where n is the number of CPU cores, s is the problem size[1], and c is a constant for data

[1] Problem size for NGS read alignment depends on a number of factors including the number of reads to be aligned, the size of the reference genome, and the "fuzziness" of the alignment algorithm.

transfer rate between a client and the cloud, our main goal is to bring down the first component (cs) in this formula. Furthermore, we would like to do it in a way that supports linear scalability. In the next section, we will present the solution that we propose, together with an initial evaluation study which indicates that our approach is a promising one.

3 A Stream-Based Approach

In the following, we first present our stream-based approach in general terms, and then describe how we applied it to a specific NGS read alignment use case together with results of a preliminary evaluation study.

3.1 Incremental Data Processing with an SPE

We propose to use a stream-based data management platform in order to reduce the total processing time of data-intensive applications deployed in the cloud by eliminating their data transfer latencies. Our main motivation to do so is to exploit the incremental and in-memory data processing model of Stream Processing Engines (SPEs) (such as the Borealis engine [7] or the IBM InfoSphere Streams (or Streams for short) engine [13]).

SPEs have been primarily designed to provide low-latency processing over continuous streams of time-sensitive data from push-based data sources. Applications are built by defining directed acyclic graphs, where nodes represent operators and edges represent the dataflows between them. Operators transform data between their inputs and outputs, working on finite chunks of data sequences (a.k.a., sliding windows). SPEs provide query algebras with a well-defined set of commonly used operators, which can be easily extended with custom, user-defined operators. There are also special operators/adapters for supporting access to a variety of data sources including files, sockets, and databases. Once an application is defined, SPEs take care of all system-level requirements to execute it in a correct and efficient way such as interprocess communication, data partitioning, operator distribution, fault tolerance, and dynamic scaling.

In our approach, we do not provide any new algorithms, but we provide an SPE-based platform to bring existing algorithms/software into the cloud in a way that they can work with their input data in an incremental fashion. One generic way of doing this is to use command line tools provided by most of these software. For example, in the NGS software packages that we looked at, we have so far seen two types of command line tools: those that are able to read and write to standard Unix pipes, and those that can not. We build custom streaming operators by wrapping the Unix processes. If standard Unix pipe communication is supported, using one thread, the Unix process is provided with incoming data streams and results are read by a second thread. Otherwise, data is written in chunks to files residing on an in-memory file system. For each chunk, the Unix process is run once and the produced output data is read and passed on to the next operator as a data stream.

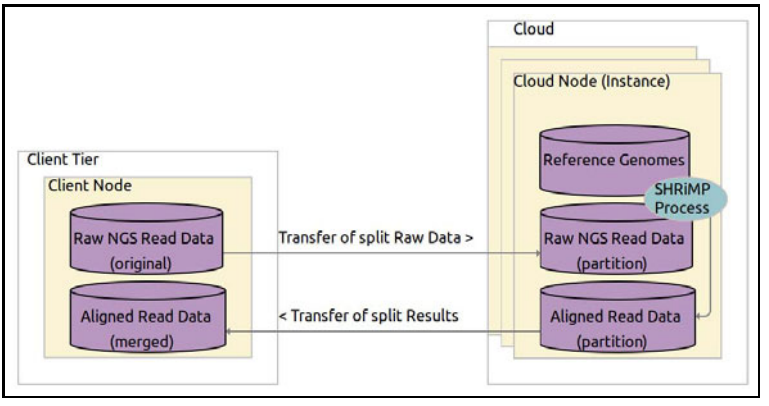

Fig. 2. Using SHRiMP on multiple nodes as standalone application requires to split the raw NGS read data into equal-sized chunks, transfer them to multiple cloud nodes, run SHRiMP in parallel, copy back the results to the client, and finally, merge them into a single file

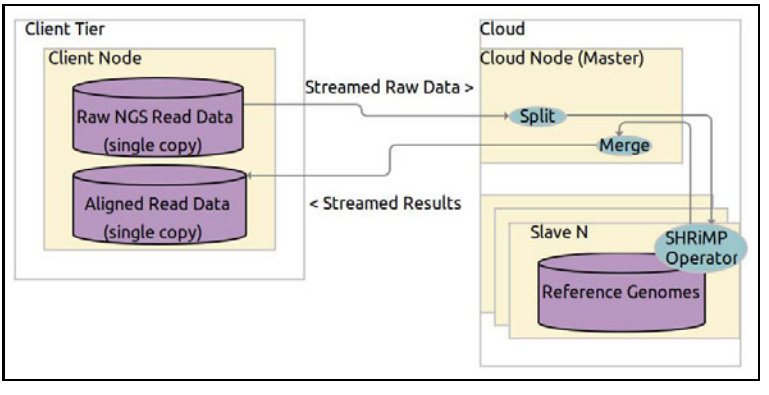

Fig. 3. With our stream-based approach, the client streams the reads into the cloud, where they instantly get mapped to a reference genome and results are immediately streamed back to the client

Figure 2 and Figure 3 contrast how data-intensive applications are typically being deployed in the cloud today vs. how they could be deployed using our approach, respectively. Although the figures illustrate our NGS read alignment use case specifically, the architectural and conceptual differences apply in general.

3.2 Use Case Implementation

We now describe how we implemented our approach for a well-known NGS read alignment software package called SHRiMP [21] using IBM InfoSphere Streams [5] as our SPE and Amazon EC2 [1] as our cloud computing platform.

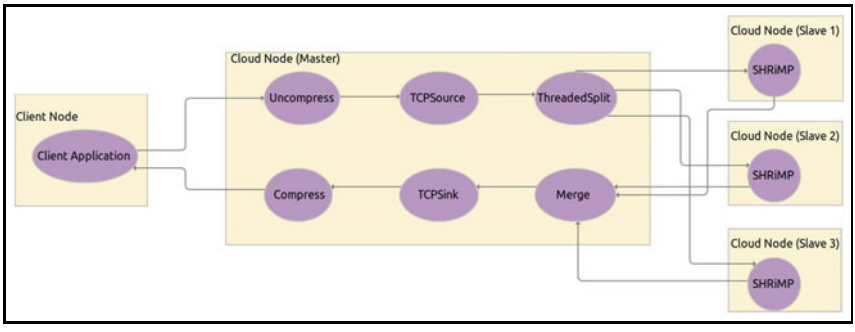

Fig. 4. Operator and dataflow graph for our stream-based incremental processing implementation of SHRiMP

Figure 4 shows a detailed data flow graph of our implementation. A client application implemented in Java compresses and streams raw NGS read data into the cloud, where a master Streams node first receives it. At the master node, the read stream gets uncompressed by an Uncompress operator and is then fed into a TCPSource operator. In order to be able to run parallel instances of SHRiMP for increased scalability, TCPSource operator feeds the stream into a ThreadedSplit operator. ThreadedSplit is aware of the data input rates that can be handled by its downstream operators, and therefore, it can provide an optimal load distribution. The number of substreams that ThreadedSplit generates determines the the number of processing (i.e., slave) nodes in the compute cluster, each of which will run a SHRiMP instance. SHRiMP instances are created by instantiating a custom Streams operator using standard Unix pipes. The resulting aligned read data (in the form of SAM output [6]) on different SHRiMP nodes are merged by the master node using a Merge operator. Then a TCPSink operator passes the output stream to a Compress operator, which ensures that results are sent back to the client application in compact form, where they should be uncompressed again before being presented to the user. The whole chain, including the compression stages, is fully incremental.

3.3 Initial Evaluation

In this section, we present an initial evaluation of our approach on the implemented use case in terms of scalability, costs, and ease of use. For scalability, we have done an experiment that compares the two architectures shown in Figure 2 and Figure 3. In the experiment, we have aligned 30000 reads of Streptococcus suis, an important pathogen of pigs, against its reference genome. Doing this on a single Amazon EC2 m1.large instance takes around 28 minutes. In order to be able to project this to analyzing more complicated organisms (like humans), we have scaled all our results up by a factor of 60 (e.g., 28 hours instead of 28 minutes). In all cases, data is compressed before being transferred into the cloud. To serve as a reference point, assuming a broadband Internet connection, transferring the compressed input data set into the cloud takes about 90 minutes.

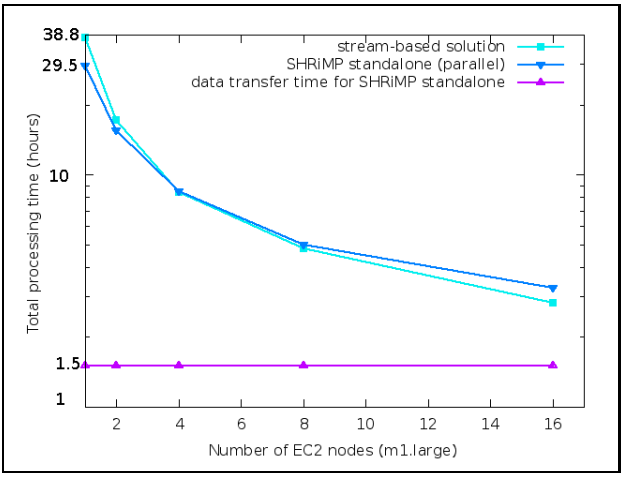

Fig. 5. At a cluster with size of 4 nodes and above, the stream-based solution incurs less total processing time than the standalone application. This is because data transfer time always adds up to the curve of the standalone application.

Scalability. Figure 5 shows the result of our scalability experiment. The bottom flat line corresponds to the data transfer time of 90 minutes for our specific input dataset. This time is included in the SHRiMP standalone curve, where input data has to be uploaded into the cloud in advance. On the other hand, the stream-based approach does not transfer any data in advance, thus does not incur this additional latency. Both approaches show linear scalability in total processing time as the number of Amazon EC2 nodes are increased. Upto 4 nodes, the standalone approach takes less processing time. However, we see that as the cluster size increases beyond this value, the relative effect of the initial data transfer latency for the standalone approach starts to show itself, reaching to almost a 30-minute difference in processing time over the stream-based approach for the 16-node setup. We expect this difference to become even more significant as the input dataset size and the cluster size further increase.

Costs. As our solution allows data processing to start as soon as the data arrives in the cloud, we can show that the constant c in the formula $f(n,s) \propto cs + \frac{s}{n}$ introduced in the previous section can be brought to nearly zero, leading to $f'(n,s) \propto \frac{s}{n}$ for the overall data processing time. Since we have shown linear scale out, we can calculate the CPU cost using $p(n,s) \propto nf'(n,s) \propto n\frac{s}{n} \propto s$. Since the cost ends up being dependent only on the problem size, one can minimize the processing time $f'(n,s)$ by maximizing n without any significant effect on the cost. Data transfer and storage costs are relatively small in comparison to the CPU cost, therefore, we have ignored them in this initial cost analysis. Nevertheless, it is not difficult to see that these costs will also decrease with our stream-based approach.

Ease of Use. Our client, a command line tool, behaves exactly the same way as a command line tool for any read alignment software package. Therefore, existing data processing chains can be sped up by simply replacing the existing aligner with our client without changing anything else. Even flexible and more complex bioinformatics data processing engines (e.g., Galaxy [14] or Pegasus [10]) can be transparently enhanced by simply replacing the original data processing stages with our solution.

4 Conclusions and Future Work

In this paper, we proposed a stream-based approach to bringing data- and CPU-intensive applications into the cloud without transferring data in advance. We applied this idea to a large-scale DNA sequence analysis use case and showed that overall processing time can be significantly reduced, while providing linear scalability, reduced monetary costs, and ease of use.

We would like to extend this work along several directions. At the moment, only SHRiMP [21] and Bowtie [18] have been enabled to run on our system. We would like to explore other algorithms (e.g., SNP callers [20]) that can benefit from our solution. As some of these presuppose sorted input, this will be an additional challenge that we need to handle. Furthermore, we would like to take a closer look at recent work on turning MapReduce into an incremental framework and compare those approaches with our stream-based approach. The last but not the least, we will explore how fault-tolerance techniques in stream processing can be utilized to make our solution more robust and reliable.

Acknowledgements. This work has been supported in part by an IBM faculty award.

References

1. Amazon Elastic Compute Cloud, `http://aws.amazon.com/ec2/`
2. Apache Hadoop, `http://hadoop.apache.org/`
3. Functional Genomics Center Zurich, `http://www.fgcz.ch/`
4. Google MapReduce, `http://labs.google.com/papers/mapreduce.html`
5. IBM InfoSphere Streams, `http://www.ibm.com/software/data/infosphere/streams`
6. The SAM Format Specification, `samtools.sourceforge.net/SAM1.pdf`
7. Abadi, D., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.: The Design of the Borealis Stream Processing Engine. In: Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA (January 2005)
8. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. Journal of Molecular Biology 215(3) (October 1990)
9. Collins, F.S., Guyer, M., Chakravarti, A.: Variations on a Theme: Cataloging Human DNA Sequence Variation. Science 278(5343) (November 1997)
10. Deelman, E., Mehta, G., Singh, G., Su, M., Vahi, K.: Pegasus: mapping large-scale workflows to distributed resources. In: Workflows for e-Science, pp. 376–394 (2007)

11. Dudley, J.T., Butte, A.J.: In Silico Research in the Era of Cloud Computing. Nature Biotechnology 28(11) (2010)
12. Fernald, G.H., Capriotti, E., Daneshjou, R., Karczewski, K.J., Altman, R.B.: Bioinformatics Challenges for Personalized Medicine. Bioinformatics 27(13) (July 2011)
13. Gedik, B., Andrade, H., Wu, K.L., Yu, P.S., Doo, M.: SPADE: The System S Declarative Stream Processing Engine. In: ACM SIGMOD Conference, Vancouver, BC, Canada (June 2008)
14. Goecks, J., Nekrutenko, A., Taylor, J., Team, G.: Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences. Genome Biology 11(8) (2010)
15. Graefe, G.: Query Evaluation Techniques for Large Databases. ACM Computing Surveys 25(2) (June 1993)
16. Keich, U., Ming, L., Ma, B., Tromp, J.: On Spaced Seeds for Similarity Search. Discrete Applied Mathematics 138(3) (April 2004)
17. Langmead, B., Schatz, M.C., Lin, J., Pop, M., Salzberg, S.L.: Searching for SNPs with Cloud Computing. Genome Biology 10(11) (2009)
18. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.L.: Ultrafast and Memory-efficient Alignment of Short DNA Sequences to the Human Genome. Genome Biology 10(3) (2009)
19. Li, H., Homer, N.: A Survey of Sequence Alignment Algorithms for Next-Generation Sequencing. Briefings in Bioinformatics 11(5) (September 2010)
20. Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K., Wang, J.: SNP Detection for Massively Parallel Whole-Genome Resequencing. Genome Research 19(6) (June 2009)
21. Rumble, S.M., Lacroute, P., Dalca, A.V., Fiume, M., Sidow, A., Brudno, M.: SHRiMP: Accurate Mapping of Short Color-space Reads. PLOS Computational Biology 5(5) (May 2009)
22. Sanger, F., Coulson, A.R.: A Rapid Method for Determining Sequences in DNA by Primed Synthesis with DNA Polymerase. Journal of Mol. Biol. 94(3) (May 1975)
23. Schatz, M., Delcher, A., Salzberg, S.: Assembly of large genomes using second-generation sequencing. Genome Research 20(9), 1165 (2010)
24. Schatz, M.C.: CloudBurst: Highly Sensitive Read Mapping with MapReduce. Bioinformatics 25(11) (June 2009)
25. Stein, L.D.: The Case for Cloud Computing in Genome Informatics. Genome Biology 11(5) (2010)
26. Viedma, G., Olias, A., Parsons, P.: Genomics Processing in the Cloud. International Science Grid This Week (February 2011), `http://www.isgtw.org/feature/genomics-processing-cloud`
27. Voelkerding, K.V., Dames, S.A., Durtschi, J.D.: Next-Generation Sequencing: From Basic Research to Diagnostics. Clinical Chemistry 55(4) (February 2009)