Metadata of the chapter that will be visualized in SpringerLink

Book Title	Bioinformatics and Biomedical Engineering	
Series Title		
Chapter Title	Exploiting In-memory Systems for Genomic Data Analysis	
Copyright Year	2018	
Copyright HolderName	Springer International Publishing AG, part of Springer Nature	
Author	Family Name	**Shah**
	Particle	
	Given Name	**Zeeshan Ali**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	King Faisal Specialist Hospital and Research Center (KFSHRC)
	Address	Riyadh, Saudi Arabia
	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)
	Address	Riyadh, Saudi Arabia
	Email	
Author	Family Name	**El-Kalioby**
	Particle	
	Given Name	**Mohamed**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	King Faisal Specialist Hospital and Research Center (KFSHRC)
	Address	Riyadh, Saudi Arabia
	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)
	Address	Riyadh, Saudi Arabia
	Email	
Author	Family Name	**Faquih**
	Particle	
	Given Name	**Tariq**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	King Faisal Specialist Hospital and Research Center (KFSHRC)
	Address	Riyadh, Saudi Arabia

	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)
	Address	Riyadh, Saudi Arabia
	Email	
Author	Family Name	**Shokrof**
	Particle	
	Given Name	**Moustafa**
	Prefix	
	Suffix	
	Role	
	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)
	Address	Riyadh, Saudi Arabia
	Email	
Author	Family Name	**Subhani**
	Particle	
	Given Name	**Shazia**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	King Faisal Specialist Hospital and Research Center (KFSHRC)
	Address	Riyadh, Saudi Arabia
	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)
	Address	Riyadh, Saudi Arabia
	Email	
Author	Family Name	**Alnakhli**
	Particle	
	Given Name	**Yasser**
	Prefix	
	Suffix	
	Role	
	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)
	Address	Riyadh, Saudi Arabia
	Email	
Author	Family Name	**Aljafar**
	Particle	
	Given Name	**Hussain**
	Prefix	
	Suffix	
	Role	
	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)

	Address	Riyadh, Saudi Arabia
	Email	
Author	Family Name	**Anjum**
	Particle	
	Given Name	**Ashiq**
	Prefix	
	Suffix	
	Role	
	Division	Department of Computing and Mathematics
	Organization	University of Derby
	Address	Derby, UK
	Email	
Corresponding Author	Family Name	**Abouelhoda**
	Particle	
	Given Name	**Mohamed**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	King Faisal Specialist Hospital and Research Center (KFSHRC)
	Address	Riyadh, Saudi Arabia
	Division	Saudi Human Genome Program
	Organization	King Abdulaziz City for Science and Technology (KACST)
	Address	Riyadh, Saudi Arabia
	Email	mabouelhoda@yahoo.com

Abstract	With the increasing adoption of next generation sequencing technology in the medical practice, there is an increasing demand for faster data processing to gain immediate insights from the patient's genome. Due to the extensive amount of genomic information and its big data nature, data processing takes long time and delays are often experienced. In this paper, we show how to exploit in-memory platforms for big genomic data analysis, with focus on the variant analysis workflow. We will determine where different in-memory techniques are used in the workflow and explore different memory-based strategies to speed up the analysis. Our experiments show promising results and encourage further research in this area, especially with the rapid advancement in memory and SSD technologies.

Keywords (separated by '-')	Bioinformatics - Big data - In memory processing - Next generation sequencing

Exploiting In-memory Systems for Genomic Data Analysis

Zeeshan Ali Shah[1,2], Mohamed El-Kalioby[1,2], Tariq Faquih[1,2], Moustafa Shokrof[2], Shazia Subhani[1,2], Yasser Alnakhli[2], Hussain Aljafar[2], Ashiq Anjum[3], and Mohamed Abouelhoda[1,2(✉)]

[1] King Faisal Specialist Hospital and Research Center (KFSHRC), Riyadh, Saudi Arabia
mabouelhoda@yahoo.com
[2] Saudi Human Genome Program, King Abdulaziz City for Science and Technology (KACST), Riyadh, Saudi Arabia
[3] Department of Computing and Mathematics, University of Derby, Derby, UK

Abstract. With the increasing adoption of next generation sequencing technology in the medical practice, there is an increasing demand for faster data processing to gain immediate insights from the patient's genome. Due to the extensive amount of genomic information and its big data nature, data processing takes long time and delays are often experienced. In this paper, we show how to exploit in-memory platforms for big genomic data analysis, with focus on the variant analysis workflow. We will determine where different in-memory techniques are used in the workflow and explore different memory-based strategies to speed up the analysis. Our experiments show promising results and encourage further research in this area, especially with the rapid advancement in memory and SSD technologies.

AQ1

Keywords: Bioinformatics · Big data · In memory processing
Next generation sequencing

1 Introduction

Genomics based medicine, referred to as personalized or precision medicine, became an important component in the healthcare system. This is basically due to the recent advancements in next generation sequencing (NGS) technology, which reduced the cost and time of reading the genome. NGS is currently used in the clinic to find variants (mutations) related to the disease to improve the diagnosis, prognosis, or to find optimized treatment plans.

For computational scientists, the wide use of NGS in the clinic has introduced new challenges. The clinical grade data analysis requires more optimized algorithms to reach reliable results, which accordingly increases the running time. Moreover, to reach a list of variants with the necessary information for the clinic, a sophisticated computational workflow of many software tools should be used.

I. Rojas and F. Ortuño (Eds.): IWBBIO 2018, LNBI 10813, pp. 1–10, 2018.
https://doi.org/10.1007/978-3-319-78723-7_35

The input to this workflow is the list of NGS reads and the output is the list of significant annotated variants related to the disease.

The output of an NGS machine is a large set of short reads (DNA fragments). The number of these reads depends on the technology and the model of the NGS instrument. For Ion technology, one expects around 80 million reads per run for the Ion Proton model. For Illumina technology, one expects up to 20 billion reads per run for the recent NovaSeq model. Processing such huge number of reads entails huge I/O operations, especially when a workflow of multiple independent programs is used. This causes two problems: First, a considerable fraction of the analysis time is spent in reading/writing of the data. Second, such intensive I/O mode of operation reduces the lifetime of the hard disks, which interrupts the operation and increases the operational costs. To solve these problems, it is important to avoid reading and writing to the mechanical hard disk and to keep the processing in the RAM as much as possible.

Fortunately, the recent advancements in hardware and computer architecture coupled with modern operating systems make this possible. Currently, one can find commercially available servers at an affordable price with a RAM size reaching tens of terabytes. Parallel to this, we observe a continuous advancement in the software side as well. One can find many options for RAM resident data structures and in-memory database systems, where issues like fault-tolerance, efficient synchronized read-write, and data integrity are addressed.

In this paper, we discuss how the in-memory techniques can be used in the clinical bioinformatics workflows. We will focus on the variant analysis workflow, which is the mostly used workflow in the clinic. We will explain the mode of use of in-memory systems at each step of the workflow, either within the program itself or to pass the data from one tool to the next. We will also show by experiment that the use of in memory systems at different steps indeed leads to improved running times.

This paper is organized as follows: In the following section, we shortly review different technologies like high size RAM computers and new storage technology. In the same section, we will briefly review in-memory data systems and some of their use in bioinformatics. In Sect. 3, we will also discuss the variant analysis workflow and its basic steps. In Sect. 4, we will show how in-memory techniques can used in the variant analysis workflow. Finally, Sects. 5 and 6 include experimental results and conclusions.

2 Background

2.1 Advanced Hardware Architecture

It is already well known that accessing data in memory is much faster than doing so on hard disks. A memory access by one CPU can take up to 200 ns, while one disk access can take up to 10 million nanoseconds (10 ms). Although different protocols to speed up the transfer of hard disk data have been developed (like SAS with 12 Gbps and SATA with 6 Gbps), reading or writing to disks is still many orders of magnitude slower than memory access.

Modern commercial servers can be equipped with huge memory; for example Dell PowerEdge R940 can have up to 6 TB RAM. Furthermore, distributed shared memory architecture combine memories from different physical machines into one logical address space. The machines in this case are linked with high speed low latency interconnection. The operating system can still see this system as a single computing server with collective number of processors and RAM. This architecture can lead to a server with much higher RAM size in the range of tens of terabytes. Interestingly, these architectures also provide battery powered non-volatile RAM (NVDIMM), but with limited size, which could help keep important information in case of power failure.

To narrow the gap between RAM and hard disk, solid state drives (SSD) (and the new 3D XPoint of Intel) have recently become available with faster interface based on the NVMe (Non-volatile memory express) protocol. NVMe is based on PCIe and it assumes the SSDs are mounted to physical slot architecture; i.e., direct connection to the CPU. While the maximum throughput of SATA is 6 Gbps and that of SAS is 12 Gbps, the throughput of NVMe based on PCIe Gen3 can reach 24 Gbps. It is expected by many researchers that the performance of SSD will converge to that of the RAM. That is, in the near future we will have huge non-volatile high speed memory.

2.2 In-memory Data Processing

Linux Pipes: Linux systems offer two options for using the RAM instead of the disk. The first is through the folder /dev/shm (tmpfs), where data in this folder resides actually in the RAM and not in the disk. The second option is through the use of pipes where the output of one program is fed to the next through intermediate buffering in the RAM. Another interesting feature about the piping is that the two involved processes run actually in parallel, where the data items are processed whenever they are ready. For example, assume a task B is piped to task A (i.e., $A|B$). The data item $A(D_i)$ output by A can be processed by B while A still processes the data item D_j, $i < j$. This characteristic is of great advantage when the data set is a list of items that can be processed independent of one another, like our NGS reads and variants.

In-memory database systems: The emergence of in-memory database systems dates back to 1980s [1–4]. Recent in-memory systems can come in different flavors: There are relational databases, column-oriented, key-value, document, and graph-oriented. There are also systems that can offer many of these models, like Apache Ignite, Couchbase, Arange DB, SAP-HANA among others. This is in addition to systems supporting memory resident data structures, like the Redis system. For a survey on these systems, we refer the reader to [5,6]. For genome analysis, Schapranow et al. [7] demonstrated how in-memory systems can be used to speed up the alignment of NGS reads.

3 The Variant Analysis Workflows

The variant analysis workflow is the mostly used workflow in the clinical practice. The workflow is used to identify the variants in the patient's genome and to annotate them with as much information as possible to enable interpretation and clinical decisions. In Fig. 1, we show the basic steps of the workflow. The input to the workflow is a set of NGS reads coming from the sequencing machine. The first step is to check the quality of the reads and to exclude those with low quality. The second step includes the alignment (mapping) of the reads to a reference genome. This alignment arranges the reads with respect to the reference genome to assemble the target genome, a step sometimes referred to as reference-based assembly. The program BWA [8] is the most commonly used tool for this task. Once the reads are mapped, variant calling is performed to spot the variants and to distinguish them from sequencing errors using different

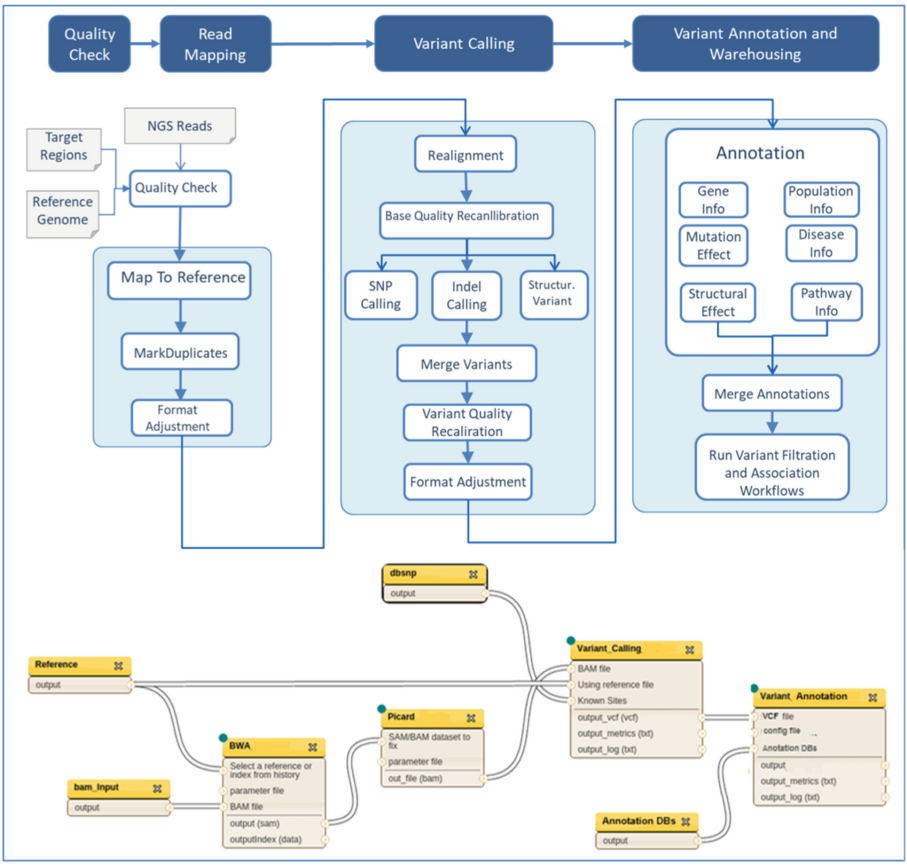

Fig. 1. The variant analysis workflow. The upper part is schematic diagram. The lower part as it is designed in the canvas of Tavaxy.

statistical models, taking technology specific parameters into consideration. The most commonly used tool for variant calling is the GATK pipeline [9]. In fact, the variant calling process is a sub-workflow that involves many steps, as shown in the figure. The final step includes the annotation of the variants using different information and knowledge databases.

One way to implement the variant analysis workflow is to write a (shell) script to run one phase after another. Another means is to use workflow management systems, where the workflow is visualized and the non-IT experts can modify the analysis parameters without any scripting or programming skills. Example workflow management systems with a graphical user interface include Galaxy [10], Taverna [11], and Tavaxy [12,13], among others. In the workflow systems, the programs run according to certain dependency plan and the results of one program is fed to the next one. These workflow systems have also an advantage of running on a high performance computing infrastructure, where many independent tasks can run in parallel. In the lower part of Fig. 1, we show the implementation of the variant analysis workflow in Tavaxy. In Tavaxy, each major step is represented by a node (which is actually a sub-workflow).

A drawback of these systems (including the current version of Tavaxy) is that the output of one step in the workflow is passed as input to the next step via intermediate files written to certain folders. Another drawback is that one task in the workflow cannot start before the completion of all the previous tasks it depends on. That is, these workflow systems do not directly support the use of Linux piping and the use of the main memory. In the following section, we will discuss how to overcome these limitations within these workflow systems using linux piping and in-memory systems.

4 In-memory Systems in Action

4.1 Linux Piping

The first steps in the variant analysis workflow involving the quality check and alignment can readily use Linux piping. The output of the program fastx for quality check can be piped to the alignment program BWA. With piping, the quality check and the alignment step can run in parallel, where any good quality read reported by fastx is directly processed by BWA. (Note that there might be some delay until BWA loads the index in the RAM). For distributed computation on a computer cluster, the set of NGS reads can be decomposed into subsets that can be processed on different compute nodes, and within each node the quality check and alignment can be still piped.

After the alignment, samtools can be used to format the output and decompose the results if required into subsets to be processed in parallel. Fortunately, samtools supports Linux piping.

Piping from the alignment to GATK cannot be achieved on the read level, because GATK computes some background statistics on the whole read set. One idea to overcome this limitation is to decompose the set of reads into different subsets. Each sub-set includes the reads covering large segment of the genomes

(the reads are sorted by samtools). In [14], a segment size of about 50 Mbp was suggested. Each of these subsets can be processed in parallel and fed one after another whenever ready. In other words, the piping will work on the subset level rather than the reads level.

The output of GATK is a set of variants. These variants can then be piped to the subsequent annotation step. Piping and online processing of the annotation step is addressed in more detail later in the paper.

To sum-up, piping can be used on the read level from the beginning of the workflow until the alignment. From the alignment to variant calling using GATK, piping is done on the level of blocks of reads. From GATK to variant annotation, piping can be used again on the read level.

Piping in workflow management systems: The workflow management systems based on a data flow model (like Galaxy and Tavaxy) do not support such mode of operation. Usually, for the two tasks $A \rightarrow B$, where B depends on A, the data flow model assumes that task B cannot start before the completion of task A and the whole output of A become available. Usually intermediate files are used to store the output of A which will be in turn the input of B. In the piping or streaming model, we can allow that B starts processing once some result data from A becomes available. That is A and B can run in parallel.

Instead of changing the workflow engine itself, one can overcome this problem by defining a new workflow node representing a sub-workflow where tasks A and B run using Linux pipes. That is the command to be executed in association with this node is the command including piping in the form $run(A)|run(B)$. Combining this with parallel processing of $A \rightarrow B$ on subsets/blocks of the reads will lead to considerable speedup.

4.2 In Memory Systems for the Variant Annotation

The Annovar system [15] is used to annotate the variants with all possible information. The pieces of annotation information include the respective gene and the coding region, the frequencies in population databases, the structural effect of the mutation, and the relation to the disease. The Annovar system compiles public databases including this information and uses them for the annotation process. For hg19, Annovar has 32 text files including these databases of total size ≈350 GB. To complete the annotation, each variant should be searched for in these files to extract the respective record, if exists. Optimizing Annovar can be achieved in two ways:

- Decompose the list of variants into different sub-lists and process each list independently. The Annovar system uses Perl threads to achieve this. This can indeed speedup the annotation, because the queries can run quickly in parallel on the loaded databases.
- Query each variant against the different databases in parallel. This solution is not implemented in Annovar yet as it has the following challenge: To query the

databases in parallel, all the 350 GB files should be cached in the RAM and that caching time could outweigh the disk-based search. For large number of files to annotate, this strategy might pay off. If there is enough RAM, another possible solution is to keep these databases in the RAM and use them once an annotation is needed. There are two means to do this:

1. *Use of tmpfs:* One uses the /dev/shm folder to host the annotation databases.
2. *Use of in-memory database:* In this case, the Annovar databases should be hosted in an in-memory database. This option necessitates the re-implementation of the Annovar system, because some of the computation required to report the variants according to the HGVS nomenclature (www.hgvs.org). (mostly for handling ambiguities associated with indels.) In the following, we introduced an in-memory based strategy without changing the code of Annovar, and this strategy works for large scale genome projects requiring the annotation of large number of variant files.

Our optimization strategy is based on the observation that there is a large number of variants that are common in the population. Our strategy is that once a variant is annotated, we keep a copy of it in an in-memory database. We use the Redis in-memory system to handle the annotated variants. Redis uses key-value pairs to define and retrieve the objects. The key of the variant is its physical location in the genome and the base change, defined by the tuple (chromosome, start position, end position, reference base(s), alternative base(s)). The value is the annotation information related to the variant. When annotating a variant, we first check whether the variant is in the Redis database or not. If it exists, we report it. Otherwise, it is added to a list L. After collecting all new variants in L, we run Annovar on them. Once the new variants are annotated, they are reported and added to the Redis database. The size of the database in the main memory can be kept constant by deleting less frequent variants on a regular basis. Frequency of the variant can be kept in a separate Redis table, where the key is defined as above and the value is the number of times the variant was observed in that annotation process. As we will demonstrate by experiment, this strategy has proven very effective in reducing the annotation time.

4.3 Fault tolerance

One of the main concerns about in-memory processing is the loss of data, due to power loss, hardware malfunctioning or software errors. Fault tolerance mechanisms are critical to assure the transactions' consistency during the occurrence of failures. For introducing an additional layer of fault tolerance, a copy of result data in each step in the workflow can be written in parallel to another stable storage, preferrably of type SSD. Logging information about each step enables resumption of workflow in case of any failure without re-computation of already finished tasks. "Write-ahead-logging" techniques can be traditionally used to keep copy of the variants in Redis in permanent storage and/or in battery-powered non-volatile RAM.

5 Experimental Results

5.1 Experiment 1: Linux Piping

The performance gain for using the Linux pipes has been addressed before in [16]. We could also achieve similar results on similar test data like the standard human NGS exome dataset (NA12878) [9], whose size is about 9 Gbp. Using the pipes, a speed up of about 30% could be achieved for the steps of quality check followed by read mapping; and for the formatting the final BAM file after alignment to be ready for removing duplicates and variant calling. Combining this to the usual parallelizaion for processing the data achieves more speedup as shown in Table 1.

Table 1. Running times in minutes using different number of cores. For these computations, we used a machine of 64 Cores, 512 GB RAM, and 8 TB disk.

Mode	Nodes		
	16	32	64
Without piping	452 min	237 min	170 min
With piping	385 min	194 min	139 min

5.2 Experiment 2: In-memory Databases

In this experiment, we tested our strategy for speeding up Annovar. Table 2 shows the results of our approach using different scenarios. We measured the execution of Annovar on hard disk and measured the performance when the whole databases are hosted in the RAM (tmpfs). We also measured our optimization strategy where we stored 200 previously annotated variant files (with about 8 million variants) in-memory using the Redis and MySQL (memory-based). For about 1000 exomes and 3000 gene panel files that have been annotated, we observed an average hit rate of 93%. That is, only around 10% new variants has to go for Annovar for annotation for each file.

We also measured the time assuming all variants are in the database; i.e., we have hit rate of 100%. This is the best case, and it unlikely happens in practice even with large number of variants are in the database due to individual variations.

From the results in the table, we can confirm the advantage of hosting the Annovar databases in the RAM compared to storing it in the hard drive. We can confirm that our strategy based on using memory-based database (Redis or MySQL) significantly speeds up the annotation process. Overall, the annotation time could be reduced by 50% compared to the use of tmpfs only and by 80% compared to the hard-disk based version. We also observe that Redis performs slightly better than the memory-based version of MySQL.

It is important to note that we did not observe much time reduction when processing small gene panel files. This can be attributed to the overhead of

Table 2. Running times in seconds using different variant files from Illumina and Ion Technology (Exome and Gene Panels). The column titled "HDD" includes time when using hard drives, the column titled "tmpfs" is when the Annovar system including its databases is hosted in the RAM. The columns titled "Redis100" and MySQL100 includes the time when we store previously annotated variants in Redis and MySQL assuming hit rate of 100%, respectively. The column titled "Redis90" includes the time when we have an average hit rate of ≈90%. For these computations, we used a machine of 64 Cores, 512 GB RAM, and 8 TB disk.

Input	#Variants (FileSize)	HDD	tmpfs	Redis100%	MySQL100%	Redis (90%)
VF1_Illumina_Exome	84287 (18M)	1351	474	26	34	241
VF2_Illumina_Exome	88387 (19M)	1275	480	28	34	251
VF3_Illumina_Exome	88410 (19M)	1351	490	28	31	259
VF4_Illumina_Exome	85307 (19M)	1275	481	27	33	246
VF5_Ion_Exome	54249 (27M)	870	305	17	23	185
VF6_Ion_Exome	55265 (27M)	844	307	18	21	188
VF7_Ion_GP	1344 (622K)	619	213	2	3	150
VF8_Ion_GP	1498 (642K)	623	215	2	3	150

reading the annotation databases. To overcome this problem for gene panels, we recommend to merge multiple gene panel files and processing them as one big variant file to eliminate that overhead.

6 Conclusions

In this paper, we explored how in-memory systems (both hardware and software) can be exploited for clinical genomics data, with focus on the variant analysis workflow. We have shown the points in the workflow where the in-memory techniques in the form of Linux piping and memory-resident databases can be used.

We demonstrated that piping techniques leads to speeding up the variant analysis workflow. We also explained how the piping techniques can be wrapped in the workflow management systems, even those based on the data flow computational model.

For the annotation step, we introduced a new strategy based on storing previously annotated variants in in-memory databases. Interestingly, with a reasonable number of stored variants, we can reduce the running time by about 80% compared to the disk based systems.

The use of SSDs based on the NVMe protocol is very effective and it should be exploited on a large scale in genome analysis. In fact, SSD can be soon extend the RAM, and this implies that the sequence analysis tools should be re-engineered to make use of this feature.

Acknowledgments. This publication was supported by the Saudi Human Genome Project, King Abdulaziz City for Science and Technology (KACST).

References

1. DeWitt, D.J., Katz, R.H., Olken, F., Shapiro, L.D., et al.: Implementation techniques for main memory database systems, vol. 14, no. 2. ACM (1984)
2. Eich, M.H.: Mars: the design of a main memory database machine. Database Mach. Knowl. Base Mach. **43**, 325–338 (1988)
3. Garcia-Molina, H., Salem, K.: Main memory database systems: an overview. IEEE Trans. Knowl. Data Eng. **4**(6), 509–516 (1992)
4. Sikka, V., Färber, F., Lehner, W., et al.: Efficient transaction processing in SAP HANA database: the end of a column store myth. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 731–742. ACM (2012)
5. Han, J., Haihong, E., Guan, L., Jian, D.: Survey on NoSQL database. In: 6th International Conference on Pervasive Computing and Applications (ICPCA), pp. 363–366 (2011)
6. Ganesh Chandra, D.: BASE analysis of NoSQL database. Future Gener. Comput. Syst. **52**, 13–21 (2015)
7. Schapranow, M.P., Plattner, H.: An in-memory database platform enabling real-time analyses of genome data. In: 2013 IEEE International Conference on Big Data, pp. 691–696, October 2013
8. Li, H., Durbin, R.: Fast and accurate short read alignment with burrows and wheeler transform. Bioinformatics **25**(14), 1754–1760 (2009)
9. DePristo, M., Banks, E., et al.: A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nat. Genet. **43**(5), 491–498 (2011)
10. Goecks, J., Nekrutenko, A., Taylor, J., Team, T.G.: Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. **11**(8), R86+ (2010)
11. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., et al.: Taverna: a tool for building and running workflows of services. Nucleic Acids Res. **34**, W729–W732 (2006)
12. Abouelhoda, M., Issa, S., Ghanem, M.: Tavaxy: integrating Taverna and galaxy workflows with cloud computing support. BMC Bioinform. **13**(1), 77+ (2012)
13. Ali, A.A., El-Kalioby, M., Abouelhoda, M.: Supporting bioinformatics applications with hybrid multi-cloud services. In: Ortuño, F., Rojas, I. (eds.) IWBBIO 2015. LNCS, vol. 9043, pp. 415–425. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16483-0_41
14. Elshazly, H., Souilmi, Y., Tonellato, P., Wall, D., Abouelhoda, M.: MC-GenomeKey: a multicloud system for the detection and annotation of genomic variants. BMC Bioinform. **18**, 49 (2017)
15. Wang, K., Li, M., Hakonarson, H.: ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Res. **38**(16), e164 (2010)
16. GATK: How to Map and clean up short read sequence data efficiently. https://gatkforums.broadinstitute.org/gatk/discussion/6483/how-to-map-and-clean-up-short-read-sequence-data-efficiently. Accessed December 2017

Author Queries

Chapter 35

Query Refs.	Details Required	Author's response
AQ1	Per Springer style, both city and country names must be present in the affiliations. Accordingly, we have inserted the city name in affiliation '3'. Please check and confirm if the inserted city name is correct. If not, please provide us with the correct city name.	