Informatics in Medicine Unlocked 25 (2021) 100684

Contents lists available at ScienceDirect

Informatics in Medicine Unlocked

journal homepage: www.elsevier.com/locate/imu

Implementation of human whole genome sequencing data analysis: A containerized framework for sustained and enhanced throughput

Abhishek Panda [a,1], Krithika Subramanian [a,1], Bratati Kahali [a,*]

[a] Centre for Brain Research, Indian Institute of Science, Bangalore, India

ARTICLE INFO

Keywords:
Containers
Resource optimization
Whole genome sequencing
Variant calling pipeline
Human

ABSTRACT

Whole Genome Sequencing (WGS) provides information for each base of the entire 3.2 billion base pairs of the diploid human genome. Therefore, WGS plays an important role in identifying genetic variations for populations and understanding disease signatures in cohort studies or cases with rare genetic disorders. Nonetheless, discoveries from high throughput WGS are dependent on efficient processing, analyzing, and storing this enormous amount of genomic sequencing data, often in the scale of petabytes. Although there has been a significant reduction in genome sequencing costs in recent years, high-performance computation costs have not decreased in a directly proportional fashion.

The objective of the present work is to develop a Docker-based container method for human whole genome sequencing data processing and analysis for detecting genetic variations from paired end WGS short reads. Our method provides an approach to simultaneously process multiple genomes within a single compute system while guaranteeing sustained and stable handling of the memory requirements for the genomic data processing and ensuring no unwanted termination of the currently running parallel jobs. This method also achieves a 40 % reduction in execution time. To encourage widespread adoption and ease of WGS analysis, our containerized pipeline will be made publicly available. We have tested this approach for human genome data from Illumina WGS platforms and report the benchmark metrics in two different workstation environments in this communication. Compared to truth sets, our approach calls variants with 99 % precision and recall.

1. Introduction

Technological advances in the field of genomics over the past couple of decades have led to substantial reductions in the cost of whole genome sequencing. This is extremely beneficial for carrying out large-scale WGS, which subsequently provides a deeper understanding of genetic makeup in several world populations, and helps in identification of disease susceptible genetic variations [1,2]. Moreover, large-scale genomic data from population groups could also shed light on some of the unanswered questions of their evolution and migration patterns [3].

New paradigms for WGS data analyses and storage are thus becoming increasingly crucial due to the lowering of sequencing costs as the tens of billions of sequence reads generated from high throughput WGS demands sophisticated computational resources for data analyses and storage. Even though the WGS of one human sample costs approximately $1000, the computational analysis and long-term storage costs can substantially run up to thousands of dollars for on-premise infrastructural requirements or costs for cloud instances.

For identifying genetic variants from WGS raw data, we call Single Nucleotide Polymorphisms (SNPs), short insertions and deletions (InDels), using standard Genome Analysis Toolkit (GATK) best practices, as also followed by other worldwide consortia [4]. Each stage in the WGS data processing and analysis pipeline is crucial because of diverse requirements for data handling. We depict the entire pipeline in Fig. 1a.

The computational infrastructure requirement of each stage of the pipeline is largely heterogeneous because they vary from each other in terms of memory requirement, multi-thread execution compatibility, and overall utilization of available computing power (often termed as load average). This heterogeneous nature makes it challenging to tune and stabilize it in different computational platforms, such as standalone workstations as well as large-scale distributed systems or high-performance computing clusters while expecting guaranteed sustainable throughput. Compared to high maintenance and highly expensive cluster infrastructure, multiple standalone servers will be a more

* Corresponding author.
E-mail address: bratati@iisc.ac.in (B. Kahali).
[1] Equal contribution.

https://doi.org/10.1016/j.imu.2021.100684
Received 27 April 2021; Received in revised form 10 July 2021; Accepted 28 July 2021
Available online 30 July 2021

feasible solution in terms of cost, maintenance, and manpower for processing a few thousand genomes. But it is futile trying the WGS data analysis of multiple genomes simultaneously upon a workstation because at some point in time, the memory becomes a core bottleneck, and the time of execution is prohibitively high. Memory bottleneck leads to the automatic termination of one or more simultaneously running processes by the kernel to free up memory. Serial processing of whole genomes one after the other will take an unreasonably massive amount of time even to process a few hundred genomes.

In this paper, we have conceived Docker-based containerization to process human WGS data to circumvent the memory bottleneck problem, so that a server will be able to process simultaneous genomes. A key motive for adopting containerization in this paper, is to have control upon computational resource allocation within a compute server for human WGS data analytics processing. Containerization is application-level virtualization that is lightweight in nature, binds application software and its necessary dependencies into one executable bundle. By using a container deployment engine named Docker, we segregate a computational server into multiple smaller logical units called containers with a fixed number of central processing unit (CPU) cores and allocated memory, and hence we have better control over the WGS analysis process in terms of efficient resource utilization. This kind of mechanism helps in setting an upper limit constraint to the resources allocated to a container. The performance overhead in containers is very less in comparison to virtual machines [5,6], which makes it ideal to containerize WGS workloads.

1.1. Related work

In the past decade, there has been extensive research for boosting performance in WGS data analytics. The existing literature for performance in WGS data analysis consists of parallel implementation algorithms (like alignment, sorting) upon CPU as well as GPU infrastructure [7], co-design of hardware and software for genomic applications, and Field Programmable Gateway Arrays (FPGA) [8] based implementations. In IBM's whitepaper on GATK, they have shown optimizations for GATK upon expensive POWER9 processors (https://www.ibm.com/downloads/cas/ZJQD0QAL). IBM has employed splitting sequence intervals and source code modifications for one publicly available genome NA12878 upon a single POWER9 compute node. They have launched POWER9 optimized GATK through the IBM cloud using Kubernetes and Docker. Additional publications in recent years have reported Docker-based next generation sequencing processing pipelines [9,10]. However, they cannot be replicated on a lab server and do not provide detailed information about each stage of computing resource utilization, which is important from the implementation standpoint, because even in a containerized environment, some steps in the WGS pipeline will perform differently depending on the compute resources. Docker utilization by Broad GATK (https://gatk.broadinstitute.org/hc/en-us/articles/360035889991–How-to-Run-GATK-in-a-Docker-container) is chiefly targeted at making the software user-friendly by automating the deployment of the original GATK within a container with all the dependencies in a single bundle.

However, the high-end enterprise-level IBM Power9 Server is designed for heavy workloads and is thus prohibitively expensive for WGS -based research in academia. The automated pipelines reported does not guarantee resource optimizations and enhanced throughput which is one of the primary objectives of our work.

In this manuscript, we detail this procedure with appropriate performance benchmarks, so that it is achievable by different researchers, and requires minimal intervention. By deploying our method that uses resource control and allocation features of Docker, 1000 human WGS can be processed in a regular standalone server in less than 6 months. Our study here shows a comparative analysis between the containerized parallel processing and non-containerized serial processing of multiple WGS data on a standalone server environment, highlighting the following pointers: execution time analysis of the WGS data analytics pipeline, efficient resource utilization benchmarks, safe handling of increased memory demands while parallel processing of multiple WGS datasets and quantification of the minimum configurational threshold for a server to handle WGS datasets.

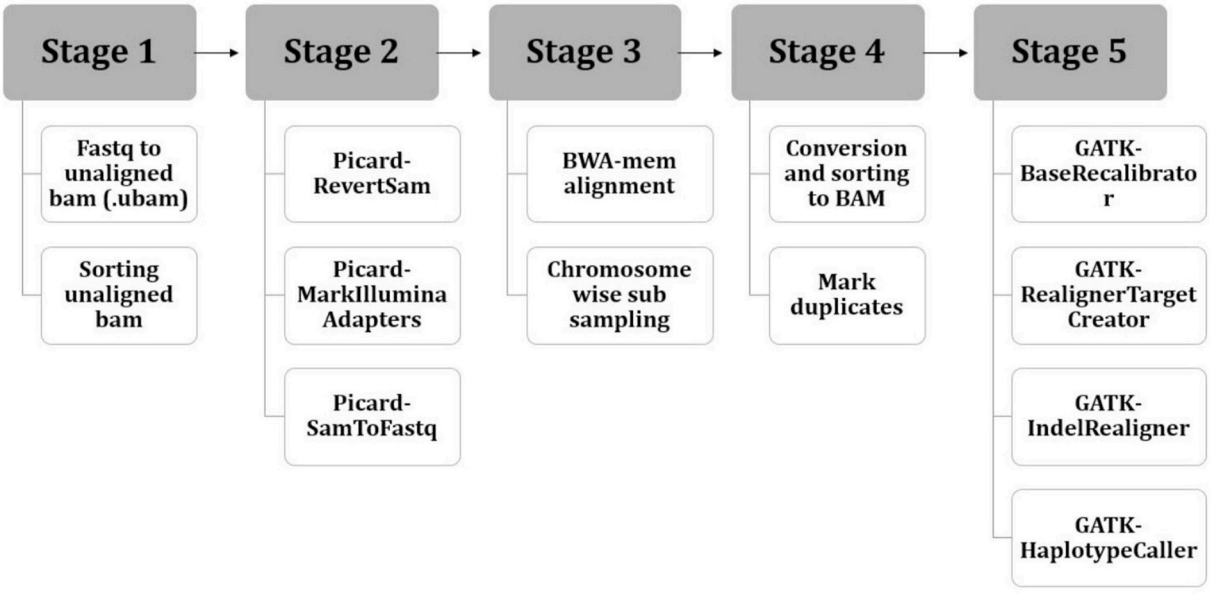

Fig. 1. Overview of the key elements of human whole genome sequencing: (a) Workflow depicting the human whole genome sequencing (WGS) data analysis pipeline starting from fastq files to obtain GVCF files for identifying germline short variants (SNPs and Indels) in one or more individuals. (b) Schematic representation of the computational models in container and non-container modes for germline variant calling from human WGS Illumina HiSeq and NovaSeq experimental data.

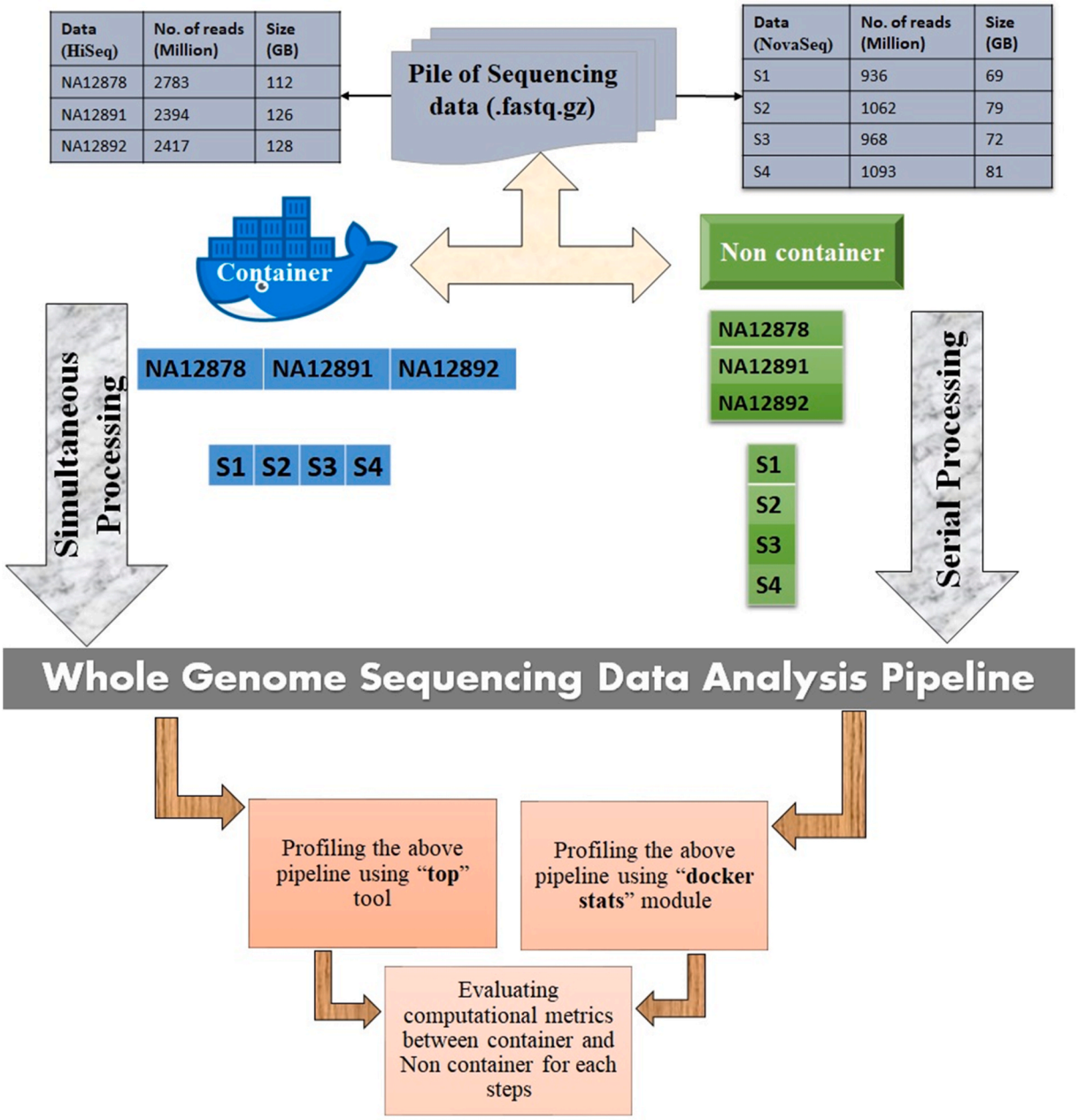

Fig. 1. (*continued*).

2. Methods

2.1. Experimental design

WGS data analysis pipeline consists up of broadly 5 stages (Fig. 1a). We started with converting raw image base calls (.bcl) to analyzable sequence format (.fastq) for NovaSeq 6000 data. Then, we converted the. fastq of HiSeq 2000 and NovaSeq 6000 to unaligned bam files for preprocessing for alignment of read pairs against current hg38 human genome build using Picard tool (http://www.broadinstitute.github.io/picard/). We used BWA-mem (Burrows-Wheeler aligner) (https://arxiv.org/pdf/1303.3997.pdf) for alignment and Genome Analysis Toolkit (GATK, https://software.broadinstitute.org/gatk/) for variant calling. After alignment, we sorted the output BAM files and marked the duplicates as the same DNA sequence can be sequenced several times and downstream GATK tools will then ignore these duplicate reads by default, through the internal application of a read filter. Then we generated the individual level genomic variants calls file format (.g.vcf). We launched the containers using the Docker engine upon the workstation environment to do all these operations. We have set up this experiment as a one-to-one mapping of a container to a WGS sample for processing the four sets of NovaSeq and three sets of HiSeq WGS data upon a single machine/node and hence achieve an optimum resource utilization when we deploy them in parallel execution mode. Using Dockers, we configured containers with limits on memory usage and used swap efficiently to ensure a controlled environment for our pipeline for processing. Fig. 1b shows the schematic representation of the workflow of containers and the non-container-based approach for all these executions for the whole genome sequencing data analysis pipeline.

WGS pipeline consists of more than ten broad steps (Fig. 1a), which as mentioned earlier differs greatly in computational requirements. The five stages highlighted in this work depict the compute-intensive stages so that the utility of our Docker-based method is discernible.

2.2. Datasets

The datasets used for the deployment of the WGS data analytics pipeline are mentioned in Table 1. We downloaded NovaSeq 6000 NA12878 data from Illumina BaseSpace platform in. bcl format (https://basespace.illumina.com/run/50719672/S1_DVT_N350_DFL). It consists of four replicates from the same individual. The total size of the. cbcl files of the 4 replicates is 218 GB. Each of these replicates consists of ~1000 million reads. We retrieved the raw WGS data from HiSeq experiment from the 1000 genomes (http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/) (through FTP) in. fastq format for individual id NA12878 (.fastq files ~119 GB), NA12891 individual (.fastq files ~ 134 GB), NA12892 individual (.fastq files ~136 GB). Each of these genome data consists of ~2400 million reads.

2.3. Configuration and command line parameters

2.3.1. Dockerfile configuration

A Dockerfile is a configuration file where step-by-step instructions of building a Docker image is embedded. In our case, all the software modules involved in the pipeline and other development and performance tools like **'git'** and **'numactl'** are marked in this file along with necessary instruction tags. We have provided this Dockerfile link in this google drive (https://drive.google.com/open?id=1xMWKfuIzYQMukz6EKp95yeLo61t0WiH2) and GitHub link (https://github.com/BratatiKahaliLab/Containerized_WGS_Data_Pipeline) that we have used in this paper.

2.4. Creating docker image

The Docker image named "pipeline" was built from the Dockerfile (https://drive.google.com/open?id=1xMWKfuIzYQMukz6EKp95yeLo61t0WiH2) using the following command: "docker build -t pipeline/Docker/file/path/.", where build parameter builds the image from the Dockerfile and a context, -t parameter specifies the name of the image that is the pipeline in this case, "." in the above command represents the context. The google drive link (https://drive.google.com/open?id=1k4IHXFWPp-uh3Q1-Ie8QrX5BThewVgK5) shows our built image of the pipeline present in the local Docker repository.

2.5. Deploy containers

We launch the containers from the image using the following command:

- "docker run -i –security-opt seccomp = unconfined \
- m "X" G –memory-reservation "M" G \
- memory-swap −1 –cpuset-cpus "k-p" \
- v/host/path/Output/:/Container/path/Output/ \
- v/host/path/Output/:/Container/path/Output/ \
- t "image-name" bash"

Where **run** parameter deploys a container, **-i** parameter launches container in interactive mode**, –security-opt seccomp=unconfined** parameter helps to use restricted system calls like **'numactl'** commands to be used inside the docker container**, -m "X" GB** is the maximum upper limit of memory allocated to the container**, –memory-reservation "M" G** is the soft limit of memory allocated to the container**, –memory-swap -1** is the parameter which allows the container to use the swap space available on the host machine**, –cpuset-cpus "k-p"** parameter allocates the number of cores to the container**, -v** parameter helps to mount host directories inside the containers where even after we stop the containers from running, the files and directories mounted to the container will still be available on the host side, **-t** is the parameter that takes image name, **bash** parameter provides the bash prompt within the container environment. Based on the above command line parameters, we launched the containers to the configurations listed in Table 2.

2.6. Hardware used

We have executed and benchmarked the container as well as non-container-based WGS data analytics pipeline on two workstations with different processor microarchitectures and different RAM configurations, for which we describe the hardware infrastructure in Table 3.

2.7. Configuration used for container v/s non-container mode

For non-container mode, we have executed the processing of genomic data serially in the hardware configuration mentioned in Table 3. In container mode, we have mapped one genome to one container. Each container is configured according to the host machine, which is stated in Table 2.

2.8. Benchmark

During the experiment, we have captured the resource utilization statistics of the processes running in the container and non-container mode using Linux-based top profiler version 3.3.10 (http://www.man7.org/linux/man-pages/man1/top.1.html). In container mode, we also captured the statistics relevant to the launched containers using the docker stats tool from the Docker engine.

The following metrics have been used to measure and compare the runs in container and non-container environments:

Load Average: It is the measure of the total number of processes available in the kernel run queue. Here the normalized load in terms of percentage is calculated by dividing the load value reported by the top profiler with the number of processors present in the machine and then multiplying it with 100.

CPU Utilization: CPU utilization is a measure of a task's share of the elapsed CPU time in percentage. Here the normalized CPU % is calculated by dividing the CPU% value (reported by the top profiler for the particular task) by the total number of processors present in the machine.

Resident memory: This represents the current physical memory

Table 1
Details of publicly available datasets used in this study: sequencing data from two different sequencing platforms have been chosen. One set is from HiSeq 2000 which consists of 3 samples and the other is from NovaSeq 6000 which consists of 4 samples.

Platform	Sample	Total number of reads	Read length	%GC
HiSeq	NA12878	2,783,700,988	101	40
	NA12891	2,394,130,484	101	40
	NA12892	2,417,332,230	101	39
NovaSeq	S1 Replicate	936,633,060	35–151	41
	S2 Replicate	1,062,504,340	35–151	41
	S3 Replicate	968,921,366	35–151	41
	S4 Replicate	1,093,524,074	35–151	41

Table 2
Configuration of the containers: Containers configured for HiSeq data and NovaSeq data varies in terms of memory and CPU core limits allocated. In two different workstation environments, for HiSeq data, 3 containers have been configured whereas for NovaSeq data, 4 containers have been configured.

Dataset	Platform	Number of containers deployed	Configuration per container	
			Cores allocated	Memory allocated
NovaSeq	Workstation 1	4	12	23 GB
HiSeq	(Centos-7)	3	16	30 GB
NovaSeq	Workstation 2	4	6	16 GB
HiSeq	(Centos-7)	3	8	20 GB

Table 3
Hardware infrastructure: In this study, two different workstations with different micro-architecture-based processors have been used to benchmark the container versus non-container mode of execution of the human whole genome sequencing variant calling pipeline.

Machine	Microarchitecture of processor	Clock speed (GHz)	Number of cores	Memory (GB)	Storage Hard disk (TB)	Operating System
Workstation 1	Intel Skylake	2.6	48	96	30	CentOS-7
Workstation 2	Intel Sandy Bridge	2.4	24	64	6	

(Random Access Memory/RAM) of a task used at that point in time. Henceforth, we will be using the terms RAM and resident memory interchangeably in this communication as it has same connotations.

We have used the above-mentioned metrics are considered for benchmarking the container and non-container mode of the run because they are captured seamlessly through the native top profiler in Linux operating system. The top profiler provides capabilities of recording real-time information of a running system in batch mode. It does not require to instrument the source code, which in turn saves from having performance overheads and inaccuracies with the final output of the source code. We captured the benchmark metrics through the top profiler by providing process-id for a specific stage of the WGS process. These metrics, Load average and CPU utilization resemble information for CPU, while resident memory represents RAM for a specific stage of the WGS pipeline.

2.9. Software details

We have used BWA software version 0.7.17-r1194-dirty (http://bio-bwa.sourceforge.net/bwa.shtml) to perform the alignment of sequence reads with the reference genome. For genome variant calling, we have used GATK 3.7–0 [11]. We have used the following software for other intermediate steps of the pipeline: Picard-2.17.4 (http://www.broadinstitute.github.io/picard/), Sambamba-0.6.5 [12], Samblaster-0.1.24 [13], Samtools-1.3.1 [14], bamUtil-NonPrimaryDedup-1.0.14 (https://github.com/statgen/bamUtil). We used Docker Engine version 18.06.1-ce (https://www.docs.docker.com/engine/install/centos/) to build and launch containers.

3. Results

WGS data analytics pipeline is a combination of multithreaded and serially implemented modules. Processing different stages of the WGS pipeline for multiple genomes upon a single node at a time will lead to a reduced performance by creating unwanted bottlenecks for computing resources. That's where the method of containerization provides a "Goldilocks environment" which allows defining explicit limits for memory and CPU cores, which are otherwise constraining the WGS data analytics process.

In this work, we implement the WGS analytics in a parallel fashion for all the datasets in container mode, whereas in non-container mode, each dataset is processed serially. We compare the container and non-container modes by benchmark metrics, namely, resident memory (RAM), load average, and CPU utilization for different stages of the analytics. In Figs. 2 and 3 and supplementary figures S1A through S1E, we have depicted the trends of benchmark metrics plotted against execution time for Novaseq and Hiseq datasets respectively. In Tables 4 and 5, we have reported maximum, average and total values of the benchmark metrics for NovaSeq and HiSeq datasets respectively.

3.1. Sorting uBAM stage

For the NovaSeq dataset, while sorting the bam files, we observe that each sample has a restrained memory utilization pattern in container mode whereas the non-container mode has higher memory utilization per sample. We depict this pattern for each of the 4 samples in container mode where the memory has reached a maximum of 5 GB (Fig. 2A i.a) and the overall memory utilization has efficiently summed up to 17.78 GB (Table 4), while in non-container mode each sample has a memory utilization higher than 7.5 GB (Fig. 2A i.b) and the average memory utilized is only 8.66 GB (Table 4). We were able to specify the memory utilization limit in the container, hence it shows efficient usage of memory, whereas such adaptation towards memory utilization is not observed in the unrestrained non-container execution. A similar memory trend is observed in the HiSeq dataset, the total memory utilization in container mode (Figure S1A i.a) is 23.52 GB (Table 5) and 14 GB (Table 5) in non-container mode (Figure S1A i.b).

For a majority period of time, the load average is consistently more than 100 % (Fig. 2A ii.a), reaching a maximum of 108.48 % (Table 4) in the container run. The cause for overload is the high number of threads spawned in each container. However, the non-container counterpart shows a trend of load average between 75 % and 100 % with a high degree of variability (Fig. 2A ii.b), because each sample acquires the complete workstation serially one after the other. A similar load pattern is found in the HiSeq dataset for container mode (Figure S1A ii.a) and non-container mode (Figure S1A ii.b). The total load for HiSeq data in container mode is 106.98 % (Table 5) which is slightly overloaded whereas in non-container mode the load is 100 % (Table 5), akin to behavior in Novaseq data.

For NovaSeq data, the CPU utilization fluctuates between the period of 4000 s–12000 s for each of the 4 samples in container mode (Fig. 2A iii.a). This is because the number of cores allocated per container (Table 2) is 12 which is way less than the number of cores utilized by the non-container mode i.e., 48 (Table 3). However, the non-container mode is stable at 100 % for most of the time for each of the 3 samples (Fig. 2A iii.b), and all 48 cores are available for processing each sample serially. By reducing the number of cores and CPU utilization tuned accordingly, container mode is achieving a higher sample processing throughput as compared to the non-container mode. We observe a similar CPU utilization trend for the HiSeq dataset (Table 5) in the container (supplementary Figure S1A iii.a) versus non-container mode (supplementary Figure S1A iii.b).

3.2. BWA alignment stage

Another crucial stage in the genomic variant calling pipeline is the alignment of raw reads to the human reference genome. The total and/or peak utilization of all the three metrics for this stage for NovaSeq and HiSeq data are depicted in Tables 4 and 5 respectively. For NovaSeq data in container mode, we observe that each sample has maintained a persistent memory pattern of above 6 GB (Fig. 2B i.a) which is similar to that in non-container mode (Fig. 2B i.b). But the overall memory utilization of workstation-1 in container mode for NovaSeq data is around 28.82 GB (Table 4) whereas the non -container counterpart has an average peak memory utilization of only 7.46 GB (Table 4), as each sample is executed in a serial fashion. The pattern of memory in HiSeq data is different than the NovaSeq data for this alignment stage of the pipeline as HiSeq data is more than double in terms of the number of reading sequences that are processed (Table- 1). So, the amount of data to be stored in RAM in the HiSeq data is higher than the NovaSeq dataset. We could process all the three samples of the HiSeq dataset in a parallel manner in container mode without exceeding the upper limit of 30 GB memory (supplementary Figure S1.B i.a), whereas just to process a single sample of the HiSeq dataset in non-container mode, RAM of 41 GB is required (Figure S1.B i.b) in the BWA alignment module of the pipeline. According to the memory footprint of a single BWA alignment

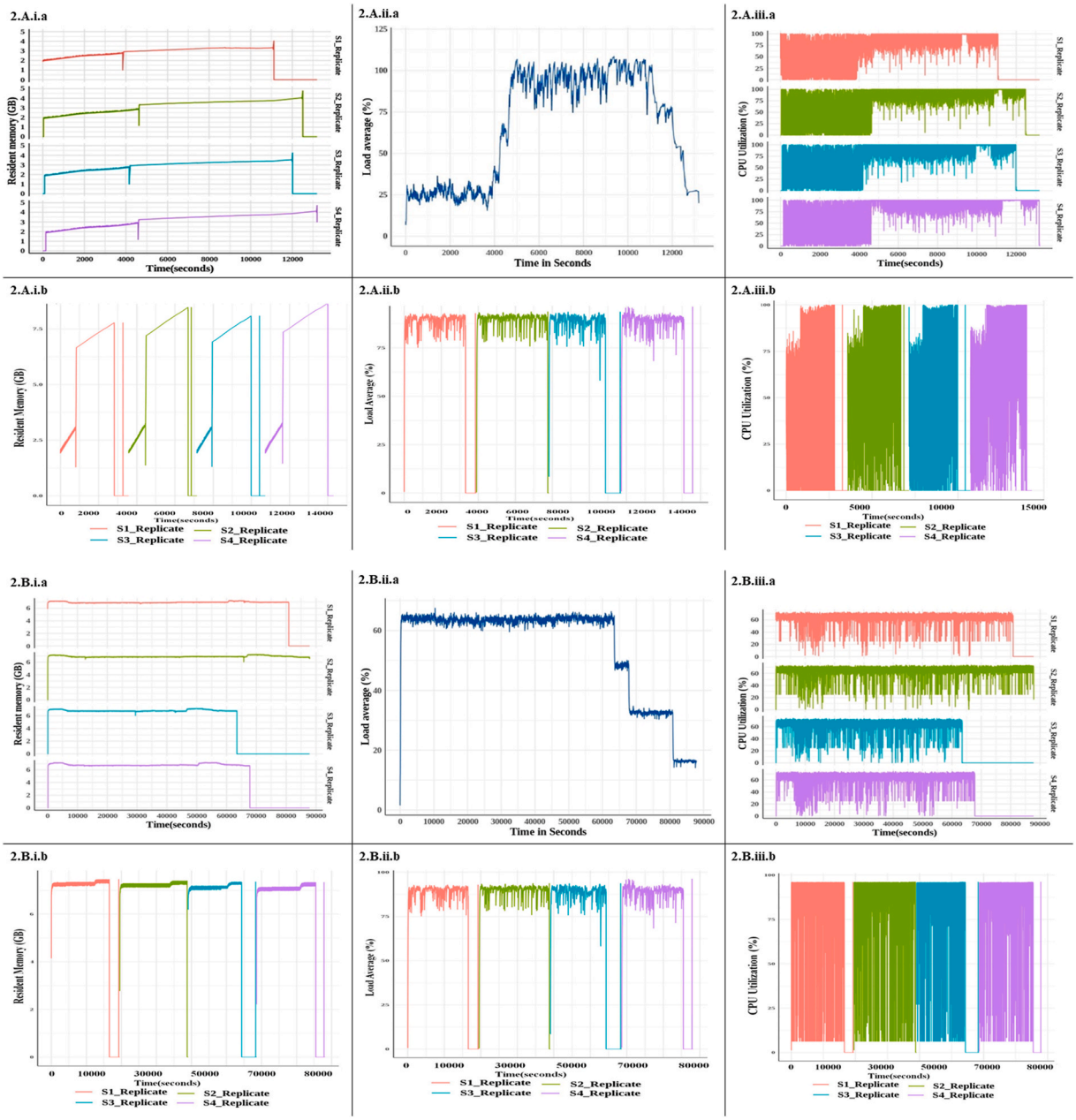

Fig. 2. A.i Memory patterns for sorting UBAM files in NovaSeq data: (a) Illustration of parallel memory utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode). . (b) Illustration of serial memory utilization with respect to execution time for 4 NovaSeq samples, one after the other (non-container mode). Fig. 2A.ii Load patterns for sorting UBAM files in NovaSeq data: (a) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. . (b) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, one sample is processed after the other serially (non-container mode). Fig. 2A iii CPU utilization patterns for sorting UBAM files in NovaSeq data: (a) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. . (b) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, when one sample is processed serially after the other (non-container mode). Fig. 2B i NovaSeq data memory patterns for BWA alignment stage: (a) Illustration of parallel memory utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode). (b) Illustration of serial memory utilization with respect to execution time for 4 NovaSeq samples, one after the other (non-container mode). Fig. 2B ii NovaSeq data load patterns for BWA alignment stage: (a) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. (b) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, one sample is processed after the other serially (non-container mode). Fig. 2B iii NovaSeq data CPU utilization patterns for BWA alignment stage: (a) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. (b) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, when one sample is processed serially after the other (non-container mode). Fig. 2C i NovaSeq data memory patterns for SAM to BAM conversion stage: (a) Illustration of parallel memory utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode). (b) Illustration of serial memory utilization with respect to execution time for 4 NovaSeq samples, one after the other (non-container mode). Fig. 2C ii NovaSeq data load patterns for SAM to BAM conversion stage: (a) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. . (b) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, one sample is processed after the other serially (non-container mode). Fig. 2C iii NovaSeq data CPU utilization patterns for SAM to BAM conversion stage: (a) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. (b) Illustration of CPU

utilization with respect to execution time for 4 NovaSeq samples, when one sample is processed serially after the other (non-container mode). Fig. 2D i NovaSeq data memory patterns for GATK BaseRecalibrator stage: (a) Illustration of parallel memory utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode). (b) Illustration of serial memory utilization with respect to execution time for 4 NovaSeq samples, one after the other (non-container mode). Fig. 2D ii NovaSeq data load patterns for GATK BaseRecalibrator stage: (a) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. (b) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, one sample is processed after the other serially (non-container mode). (a) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. (b) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, when one sample is processed serially after the other (non-container mode). Fig. 2E i NovaSeq data memory patterns for GATK HaplotypeCaller stage: (a) Illustration of parallel memory utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode). (b) Illustration of serial memory utilization with respect to execution time for 4 NovaSeq samples, one after the other (non-container mode). Fig. 2E ii NovaSeq data load patterns for GATK HaplotypeCaller stage: (a) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. (b) Illustration of load average on the processors with respect to execution time for 4 NovaSeq samples, one sample is processed after the other serially (non-container mode). Fig. 2E iii NovaSeq data CPU utilization patterns for GATK HaplotypeCaller stage: (a) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, each processed in separate containers (container mode) in parallel. (b) Illustration of CPU utilization with respect to execution time for 4 NovaSeq samples, when one sample is processed serially after the other (non-container mode).

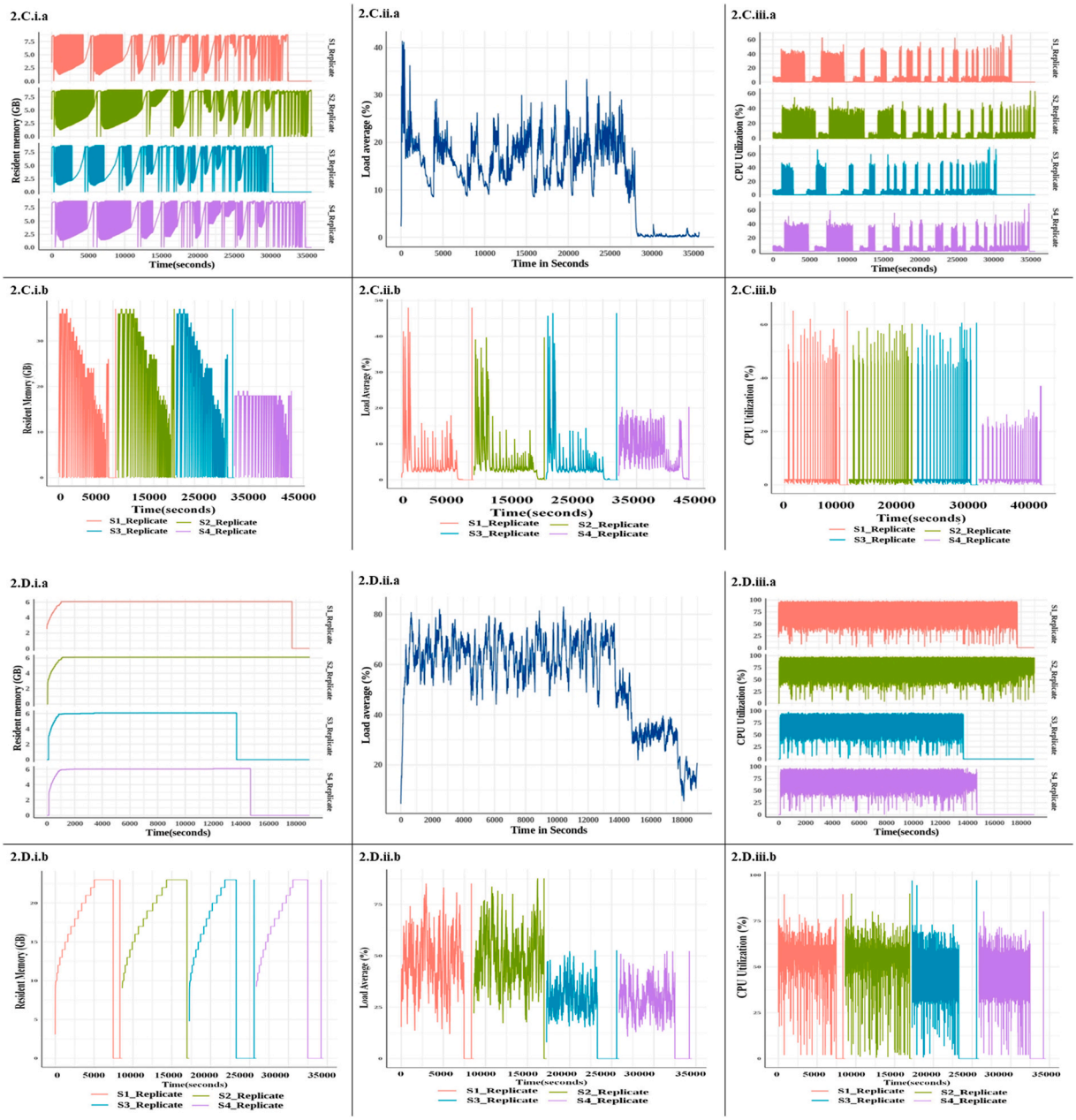

Fig. 2. (*continued*).

process, it will be tough to handle the memory requirement if we want to multi-process BWA alignment for all the three HiSeq platform samples (Table 1) or even more than one sample in a regular local server, because the physical memory available on Workstation-1 is 96 GB, and more than 120 GB RAM will be required. To tackle this excess memory required, we mapped each HiSeq sample to a container (configurational

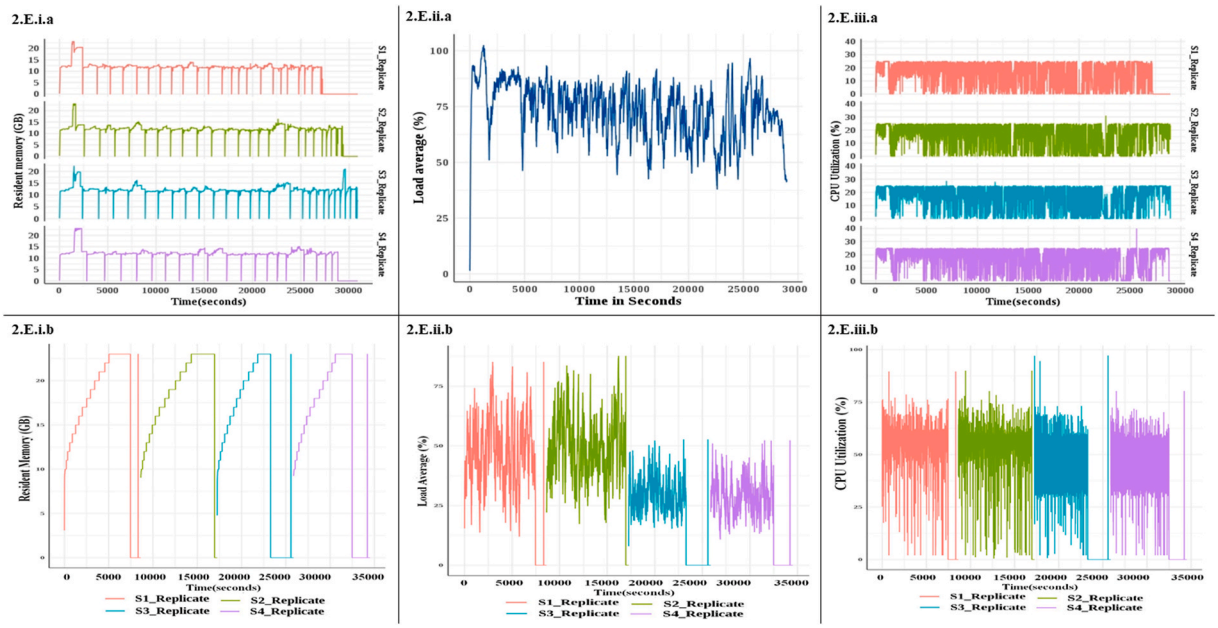

Fig. 2. (*continued*).

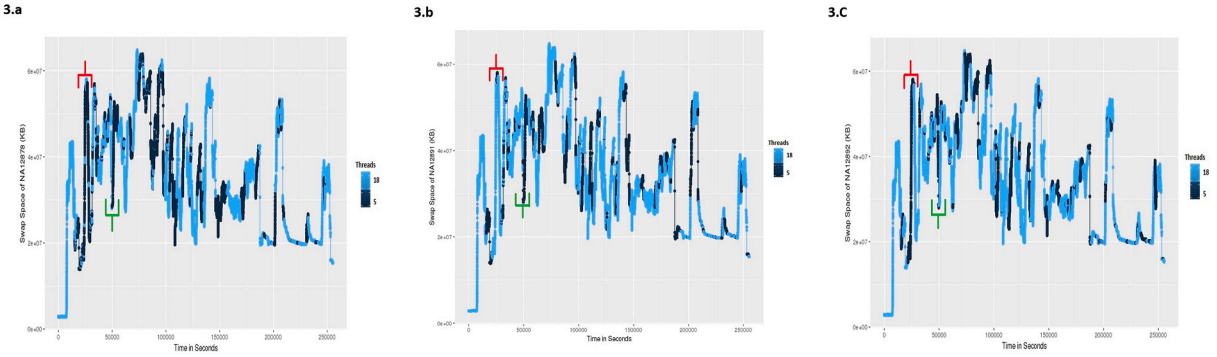

Fig. 3. BWA alignment threads and swap memory behavior during parallel execution of HiSeq data in container mode. The red highlighted portion represents the BWA thread pattern 5,18,5 modulated differently at the same point of time across all the three subfigures. Similarly, the green highlighted portion is showing 5,5,18 BWA thread patterns across all three subfigures. (a) Plot depicting swap memory usage with respect to execution time for NA12878 sample. The light blue trace on the memory curve represents 18 threads whereas the dark blue trace represents 5 threads being utilized. (b) Plot depicting swap memory usage with respect to execution time for NA12891 sample. The light blue trace on the memory curve represents 18 threads whereas the dark blue trace represents 5 threads being utilized. (c) Plot depicting swap memory usage with respect to execution time for NA12892 sample. The light blue trace on the memory curve represents 18 threads whereas the dark blue trace represents 5 threads being utilized. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

details specified in Table 2) with memory (RAM) configured to an upper limit of 30 GB. The overall memory utilization is 87 GB in container mode (Table 5) whereas that in non-container mode is 41 GB (Table 5). This shows that even with such a bulky dataset, container mode can efficiently manage the memory utilization keeping the parallel processing of all the 3 HiSeq datasets intact.

The load average for NovaSeq data is maintained at 65 % (Fig. 2B ii. a) in container mode, as compared to the non-container mode (Fig. 2B ii. b) where load average is consistent at 96 %. In HiSeq data, the load average in container mode (supplementary Figure S1.B ii.a) is varying highly from 25 % to 80 % whereas non-container mode (supplementary Figure S1.B ii.b) is persistent at 80 %. The reason for this variability in container mode is the suspension of BWA threads and accommodating the respective pages in the swap memory (explained in Fig. 3). This reiterates that existing resources in a single compute node cannot support processing of some particular stages of the WGS analytical pipeline, like the BWA alignment, for even a few human whole genomes. CPU utilization in container mode for NovaSeq data is around 74 % (Fig. 2B iii.a) while the non-container mode depict a CPU utilization of 96.16 % (Fig. 2B iii.b). The reason behind the reduced CPU utilization in container mode in NovaSeq data is that fewer number of CPU cores are allocated.

This behavior is also reflected in lesser CPU utilization during container mode in comparison to the non-container mode. We show that the CPU utilization in container mode for processing HiSeq data is 87 % (Figure S1.B iii.a), whereas in non-container mode the peak CPU utilization is 93 % (Supplementary Figure S1.B iii.b). This is due to the bulkier nature of HiSeq data leading to suspension of threads and offloading of memory pages to swap. In non-container, each HiSeq sample is processed serially using the full capacity of workstation-1, making the CPU utilization reach a higher value. We also want to emphasize that the constrained CPU utilization for parallel processing of multiple samples in container mode is achieved without changes to the BWA source code. Therefore, we could make efficient utilization of resources in the containerized mode for the alignment of 3 HiSeq samples in a single compute node.

Table 4
Workstation 1 – Benchmark metrics: This table represents the peak and/or total values[a] of all the benchmark metrics (Memory, Load, CPU) for both container and non-container modes of execution for NovaSeq data in the workstation-1 environment. Total time is also given in this table.

Mode of execution	Container			Non-container					
Total time (Hours)	100			165					
	Parameters used as benchmark metrics								
Stages of the variant calling pipeline	**RAM (GB)** [Total for 4 samples]	**LA (%)** [Total for 4 samples]	**CPU (%)** [Total for 4 samples]	**RAM (GB)** Peak for one sample	Total for 4 samples	**LA (%)** Peak for one sample	Total for 4 samples	**CPU (%)** Peak for one sample	Total for 4 samples
uBAM generation	9.63	15.90	50.5	8.67	34.17	12.08	35.85	69.13	249.33
Sorting uBAM	17.78	108.48	100	8.66	32.99	96	388.46	100	400
Picard-RevertSam	0.87	10.85	29.58	0.42	1.55	8.48	25.44	17.08	26.16
Picard-MarkIlluminaAdapters	0.94	10.60	27.23	0.38	1.22	5.81	16.67	17.08	34.67
Picard – Samtofastq	0.98	11.23	45.3	0.42	1.35	5.65	19.71	20.42	29.52
BWA-mem alignment	28.82	67.5	74.42	7.46	29.54	96.23	376.38	96.17	384.52
Chromosome wise sub sampling	0.60	94.94	22.69	0.413	1.612	58.35	212.98	68.79	264.83
Conversion and sorting to BAM	35.37	41.38	70.03	37	130	47.94	154.35	65.19	223.17
Mark duplicates	13.37	99.34	98.22	6.37	24.55	104.3	412.04	100	400
GATK-BaseRecalibrator	24.39	82.98	98.58	23	92	87.60	277.52	97.08	356.63
GATK-RealignerTargetCreator	17.76	100.15	99.5	14	56	104.5	401.48	98.60	372.75
GATK-IndelRealigner	24.5	13.29	83.83	13	52	18.83	59.81	100	338.23
GATK-HaplotypeCaller	91.2	102.33	39.69	33	132	92.10	357.94	100	400

[a] The numbers are rounded off to the nearest two decimal places.

Table 5
Workstation 1 – Benchmark metrics: This table represents the peak and/or total values[a] of all the benchmark metrics (Memory, Load, CPU) for both container and non-container modes of execution for HiSeq data in the workstation-1 environment. Total time is also given in this table.

Mode of execution	Container			Non-container					
Total time (Hours)	167			215					
	Parameters used as benchmark metrics								
Stages of the variant calling pipeline	**RAM (GB)** [Total for 3 samples]	**LA (%)** [Total for 3 samples]	**CPU (%)** [Total for 3 samples]	**RAM (GB)** Peak for one sample	Total for 3 samples	**LA (%)** Peak for one sample	Total for 3 samples	**CPU (%)** Peak for one sample	Total for 3 samples
uBAM generation	10.63	14.35	81.56	8.95	26.75	15.90	22.96	50.5	207.23
Sorting uBAM	23.52	106.98	99.56	14	38	100	267.79	100	300
Picard-RevertSam	0.88	9.54	18.31	0.44	1.2	10.85	16.63	29.58	13.52
Picard-MarkIlluminaAdapters	0.89	9.67	13.38	0.38	1.06	10.60	13.85	27.23	10.69
Picard – Samtofastq	0.77	10.42	74.94	0.36	0.92	11.23	18.31	45.3	10.80
BWA-mem alignment	87	86.52	87.81	41	120	67.5	280.44	93	288.75
Chromosome wise sub sampling	1.27	61.27	57.63	0.83	2.45	94.94	85.06	22.69	77.60
Conversion and sorting to BAM	47.3	48.08	100	40	120	41.38	142.56	70.03	211.98
Mark duplicates	9.75	107.79	100	7.75	22.23	102.73	302.15	100	300
GATK-BaseRecalibrator	23.76	73.81	75.88	24	72	82.98	173.23	98.58	265.10
GATK-RealignerTargetCreator	20.13	99.98	90.13	14	42	100.15	304.77	99.5	257.13
GATK-IndelRealigner	23.97	17.21	81.88	13	36	13.29	56.958	83.83	237.08
GATK-HaplotypeCaller	70.8	113.29	100	34	94	102.33	303.56	39.69	300

[a] The numbers are rounded off to the nearest two decimal places.

3.3. SAM to BAM conversion stage

During the conversion of aligned reads to the binary alignment map file, in NovaSeq data, the total memory utilization in container mode is 35.37 GB (Table 4; BAM conversion), whereas the non-container counterpart has an average memory utilization of 37 GB (Table 4; BAM conversion). To note here is that we have specifically controlled the memory for each sample in container mode to utilize only 8 GB (Fig. 2C i.a). In non-container mode, higher utilization of memory is observed for each sample as there is no resource constraint (Fig. 2C i.b). Containerization forces all the three HiSeq samples to process together within 47 GB RAM, whereas the non-container mode requires 40 GB RAM per sample (Figure S1.C i.a & Figure S1.C i.b). The peak and total RAM reflect the same pattern as depicted in Table 5.

In both execution modes, the load average reaches a peak value just at the beginning of the run (41.38 % for container and 47.94 % for non-container, see Table 4) and then maintains two distinct consistent patterns between 15 and 25 % in container mode (Fig. 2C ii.a) and for NovaSeq data, for a major duration of the execution, whereas in non-container mode (Fig. 2C ii.b) lesser load average between 7 % and 15 % is being maintained. For HiSeq data, the load average in containers (Figure S1.C ii.a) is persisting between 10 % and 30 % whereas in non-container (Figure S1.C ii.b) the load average is not showing a consistent pattern throughout the execution time.

For NovaSeq data, the CPU utilization in the container and non-container mode has shown almost similar trends (Fig. 2C iii.a and b) which is maintained across all 4 samples. The only exception in non-container mode is the fourth sample which depicts a lower CPU

utilization of around 20 % (Fig. 2C iii.b). The reduction in CPU utilization in the non-container mode for the fourth sample (S4_replicate) reflects the fact that the CPU is idle for a substantial time period as the I/O is slow due to the storage being filled to a major extent with the processed data from other 3 samples before S4 replicate processing. For the HiSeq dataset, the CPU utilization pattern in container mode (Figure S1.C iii.a) for all the samples reaches a peak of 100 % but this ranges between 45 and 65 % for all the samples in the non-container mode (supplementary Figure S1.C iii.b). The peak CPU utilization for this stage for both NovaSeq and HiSeq data is provided in Tables 4 and 5 respectively.

3.4. GATK BaseRecalibrator stage

In the stage of recalibration of the quality scores of bases, for NovaSeq data the memory utilization is 24.39 GB in containers (Fig. 2D i.a), compared to total 92 GB RAM ((Fig. 2D i.b) (Table 4) in non-containers. Therefore, 6 GB RAM is utilized per sample in container (Fig. 2D i.a) compared to an average of 24 GB in noncontainers (Fig. 2D i.b).

The load average is maintained across the 4 containers (Fig. 2D ii.a) consistently within 60 %–80 %. But in the non-container mode (Fig. 2D ii.b), for the first two samples, the load is maintained at 50 %–75 % whereas the other two samples occupy a lesser load of 25 %–50 %. The lesser load average for S3_replicate and S4_replicate samples means that there is an overall decrease in the number of processes available in the run queue of the kernel as compared to for processing of S1_replicate and S2_replicate samples. This behavior could be due to the unstable nature of java garbage collection threads employed by the operating module of GATK. The peak load usages for this stage are depicted in Table 4 for Novaseq and Table 5 for Hiseq datasets.

We observe that the CPU utilization is enhanced in containers (Fig. 2D iii.a) to 90 % whereas, in non-container (Fig. 2D iii.b), the CPU utilization has been stuck at 75 % for the majority of the execution time. The reason for the enhancement of CPU utilization in container mode because threads are spawned within the container for a lesser number of cores (Table 2) than the entire number of cores available in workstation-1 (Table 3).

For the same recalibration operation for HiSeq data, the total memory utilization in the containers (Figure S1D i.a) for all three samples is observed to be ~24 GB. However, the total RAM required in a non-container run (supplementary Figure S1D i.b) for all three samples is 70 GB as each sample has a memory utilization of 23 GB. The load average in containers (Figure S1D ii.a) is around 40 %–60 % whereas in the non-containers (supplementary Figure S1D ii.b), the load average is around 10 %–40 %. The CPU utilization in container run (supplementary Figure S1D iii.a) is around 50 % for all the three samples whereas the CPU utilization in non-container run (supplementary Figure S1D iii. b) is around 60 %. The prime reason for reduced CPU utilization in container mode is because of the CPU cores being idle waiting for the data to be loaded onto the RAM as the size of HiSeq data is much bulkier, being 3 times the size of the NovaSeq dataset per sample (refer Table 1).

3.5. GATK HaplotypeCaller stage

In the stage of calling haplotypes from sequencing data, the total memory utilization in container mode (Fig. 2E i.a) is around 90 GB, mostly due to a single peak of 20 GB RAM for each of the samples. While, in non-container mode (Fig. 2E i.b) the memory usage peaks at 33 GB multiple times throughout the entire execution of this step in the pipeline, and the rest of the time it hovers around 12 GB.

In the container run (Fig. 2E ii.a), the load average is maintained at ~80 % throughout the entire processing time, except for a point of time (around 1250 s timepoint) for a few seconds where the load has gone slightly beyond 100 %, which symbolizes that the workstation is overloaded, thus fully and efficiently utilizing our implementation of Dockers, validating one of the primary reasons we created it. In the non-container (Fig. 2E ii.b) mode of execution for each of the samples, the load average reaches a peak of almost 80 % towards the start and end of the process but is maintained at around 25 % for the rest of the execution time. We also observe that the load average is minimal around 14000–26000 s, 42000 to 51000, 66000–77000 s, 94000–120000 s, which could be attributed to processes being minimally active due to I/O during these points.

The CPU utilization in container mode (Fig. 2E iii.a) is controlled to 25 % throughout the execution time for each of the samples, so that all samples are processed simultaneously without processes being terminated or running 'out of memory'; while the non-container mode (Fig. 2E iii.b) reaches maximum CPU utilization of 100 %. This behavior will be reflected when researchers run hundreds of samples using our Docker method, and effectively for each sample processed in a regular unrestrained compute environment, our Docker method will be able to process and analyze four samples. The peak utilization of all the three metrics for this stage is depicted in Table 4 for NovaSeq data.

For HiSeq data, the total memory utilization in container mode (supplementary Figure S1E i.a) is around 45 GB for all three samples whereas, in the non-container (supplementary Figure S1E i.b) mode, the memory utilization for each sample is around 15 GB.

The load average in the container mode (supplementary Figure S1E ii.a) has persisted at 60 % whereas, in the non-container counterpart, the load average is around 25 % (supplementary Figure S1E ii.b) for each sample.

The CPU utilization in container mode (supplementary Figure S1E iii.a) is around ~55 % for all three samples whereas in the non-container mode (supplementary Figure S1E iii.b) this is above 60 % for each of the three samples. The peak utilization of all the three metrics for this stage is depicted in Table 5 for HiSeq data.

Therefore, we note from the above results that for various stages of the variant calling pipeline, when we predefine a controlled and optimized usage of memory and number of cores using Docker containerization, we obtain a sustained performance of all the stages of the pipeline as is evident from RAM, load-average and CPU utilization as reported above. Therefore, containerization, by virtue of logical allocation of compute resources, and utilizing the same leads to better throughput in sample processing and efficient system utilization, as compared to the unrestrained or nonformulaic non-container mode.

3.6. Swap memory behavior

Fig. 3a,3b, 3c depicts the BWA alignment threads and swap memory behavior within the container mode for three HiSeq samples. We also observed variability in load average pattern in container mode correlated to the suspension and activation of threads, which happens because of swapping memory pages. The increasing and decreasing pattern of threads is maintained within the Docker container that we have configured (Fig. 3a, 3b, 3c). Initially, 18 threads are maintained by each BWA aligner (depicted in light blue in Fig. 3 a 3. b, 3. c) for the same duration of time from 0 to 12000 s for each of the samples. However, after that the aligner of sample NA12878 maintains 5 threads (depicted in dark blue in Fig. 3a) at around 25000 s, aligner of sample NA12891 sample maintains 18 threads (depicted in light blue in figure-3. b) at around 25000 s and aligner of NA12892 maintains 5 threads (depicted in dark blue in Fig. 3c) at around 25000 s. The region highlighted in red in all the three plots depicts the thread pattern 5,18,5 respectively. In the region highlighted in green, a thread pattern of 5,5,18 is visible in all three plots respectively. The thread pattern 5,18,5 depicts that to maintain 18 threads for the NA12891 sample, the threads of samples NA12878 and NA12892 are reduced from 18 threads to 5. The reason for this reduction is that respective threads are suspended as they are performing swap operations because of container memory limits. The same reason is valid for the 5,5,18 thread pattern also. Here first two samples (NA12878 and NA12891) have reduced their number

of threads and the third sample (NA12892) has maintained maximum threads at 18. These ups and downs in the mechanism of threads enforced by containers are also evident at 100000,150000 and 180000 s. As the aligners are running in container limits, these thread patterns are occurring implicitly, and hence all the 3 alignment processes going forward a highly sustainable parallel processes. This pattern shows that the alignment process gets executed in a multiprocessing fashion for the three samples since the threads get suspended and their respective memory pages get offloaded into the swap space for accommodating newer memory pages within the allocated container memory of 30 GB for each HiSeq data (Table 2). This approach helps in circumventing the over-demand of memory and successfully performs alignment of multi-samples in a parallel fashion within a compute node.

3.7. Time utilization

All the 4 NovaSeq samples, namely, S1, S2, S3, S4 replicates processed in workstation-1 (architecture summary in Table- 2) serially in non-container mode depicts a significantly greater amount of time required as compared to the container-based parallel approach for most of the modules of the pipeline (Fig. 4). Only Picard SamtoFastq conversion and BWA alignment are the slight exceptions that require less time in the non-container approach compared to that in container mode of execution (Fig. 4). Nonetheless, this is compensated by the lesser amount of total time required for the overall execution of the entire pipeline in container mode as compared to non-container mode. We observe the same pattern of time required for the execution of HiSeq samples as well (Fig. S2). Fig. 5 provides the overall time required for executing the variant calling analysis pipeline in the container and non-container modes for four samples of NovaSeq and three samples of the HiSeq dataset (dataset descriptions in Table 1) in both the workstation architectures (Table 3). The overall time required for HiSeq data in Workstation-1 shown for the container mode is around 167 h (Fig. 5) whereas the non-container mode requires 215 h. In container mode, we can save around 48 h (precisely 2days) while processing 3 HiSeq samples on Workstation-1 kind of infrastructure. We report in this study that, in the same infrastructure, the container mode takes 100 h, whereas the non-container mode takes 165 h (Fig. 5) for the execution of 4 NovaSeq samples, thus saving 2 days and 17 h. We are describing the processing of 4 human genome samples on a single workstation (architecture in Table 3). Therefore, our Docker-based execution of analyzing WGS data enforces optimal utilization of computing resources and a significantly lesser amount of processing time required.

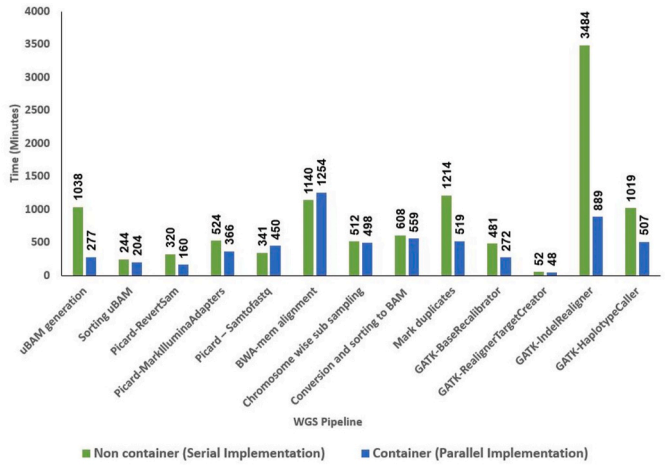

Fig. 4. Depiction of time required for executing each stage of the human WGS data analytics pipeline for germline variant calling for container and non-container modes for NovaSeq Data.

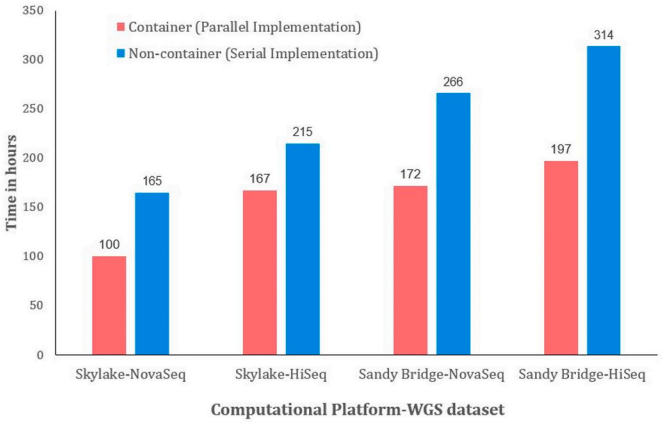

Fig. 5. Comparison of time required for executing the human WGS germline variant calling pipeline for container and non-container modes of execution in two workstation architecture for HiSeq as well as NovaSeq datasets.

3.8. Pipeline validation

In this study, we validated our WGS pipeline using NA12878 sample against Genome in a Bottle (GIAB) high confidence dataset using haplotype comparisons tool [https://github.com/Illumina/hap.py]. Benchmarking the WGS analytical pipeline is important because it will help us to assess the performance of our analysis pipeline and also identify true positive genetic variants. Table S1 depicts our benchmarking results of NA12878 samples for each chromosome. In our pipeline, the recall calls of 0.94–0.99, and precision is 0.99 for each chromosome. NA12878 is the worldwide standard sample used for validating variant calls generated by the WGS pipeline [15,16]. That is why we have reported our benchmarking against the gold standard NA12878 SNPs and InDels results so that the accuracy of our method is established.

The IBM whitepaper which implemented expensive POWER9 processors for WGS variant calling reported an execution time of about 20 h maximum for the NA12878 sample. We report in this paper that 4 samples can be run in 100 h by our Docker-based method. However, this is not a direct comparison, since the processing environments are different.

4. Discussion

Our work shows how to restrain the variant calling pipeline system usage in terms of CPU utilization and memory utilization that is achieved through the CPU and memory limiting parameters of Docker containers. This inexpensive implementation upon local server's infrastructure increases the throughput of WGS data analysis by more than 2.5 times. To the best of our knowledge, this study is one of the first of its kind, where we have successfully implemented containerization of the entire human WGS data analytics process where we have firm control over the computational resource utilization in each of the WGS pipeline steps without explicitly making any changes in the source code of any of the software modules. In all the instances described in results section above, it is evident that the memory footprint per sample has reduced to accommodate to the upper limit that we have allocated to the respective container as compared to the non-container runs. This has helped us in executing data analysis of multiple samples with increased throughput, while conspicuously reducing the overall time of execution. By this kind of arrangement, WGS data analytics can be deployed more efficiently with increased sample processing throughput on small-scale infrastructures of servers, workstations, as well as extendable to high-performance compute clusters.

Our method is important because, to increase the computation throughput and obtain timely results, one tends to process multiple WGS

samples within a compute server. Yet, while running multiple samples on a compute server, due to the hybrid nature of the pipeline, although some steps run fine, there arises an increased demand for memory for the memory-intensive stages, for example, BWA alignment, and GATK HaplotypeCaller as obvious from results above. Packing of multiple WGS samples without a control limit on resource utilization will lead to memory bottleneck issues prompting out of memory errors, and perilous overload on the CPU, leading to abrupt termination of the WGS data analytics pipeline, at which point our method comes to rescue.

Commercial packages are available for WGS data processing, for example, Isaac/Strelka2 (https://github.com/sequencing/isaac_variant_caller), Parabricks (GPU-accelerated) (https://www.e2enetworks.com/nvidia-parabricks-gpu-accelerated-gatk), Sentieon (CPU-accelerated) (https://www.sentieon.com/products/), IBM POWER9 based processing (https://www.ibm.com/downloads/cas/ZJQD0QAL), or GATK container bundle with allied dependencies (https://gatk.broadinstitute.org/hc/en-us/articles/360035889991–How-to-Run-GATK-in-a-Docker-container). However, these approaches, more often than not, requires expensive computer processor infrastructure or cloud services that might not be feasible in terms of funding availability or cloud compliance policy of institutions. Furthermore, these methods usually have done modifications to source code, internal working mechanism remain opaque to users, or original codes have been bundled in containers to make it user friendly. Therefore, they tend to use as much resource, required inherently by the WGS pipeline as it is not configured with resource limiting parameters, unlike our method which is capable of handling simultaneous sample processing within a single machine or limited nodes as per their availability in laboratory servers. Our method is seamlessly applicable to any future versions of the GATK modules for WGS. As a future implementation, this can also be packed onto Nextflow workflows with appropriate plugins. Our method also shows high precision and recall for variant calling against established world standard truth sets.

In high-performance cluster setups where there is always a chance of node failure, for such compute-intensive operations like WGS data analysis, if a node fails or is powered down in an arbitrary fashion then the genome(s) which is getting processed within the node will also get crashed resulting in considerable data loss, and wastage of time and resources. In our method, thus, by leveraging the lightweight nature and loose coupling with the host ensured by containers, we actually have a substantial advantage of efficient fault tolerance as live migration of the genomes being processed from failed node to a safe compute node is assured in this computing environment.

The benchmark parameters and results in this study can be considered as a reference for deciding how many samples and what type of samples to be processed on what kind of available hardware. Research groups working in this area with moderate infrastructure capacity and budding bioinformatics startups who have limited hardware resources will get the true benefits of increasing their sample processing capacity by adopting containerized deployment of genomic analytics. Additionally, this communication will encourage research groups working in domains of big data besides genomics but dealing with programs of heterogeneous nature to exploit the benefits of containerization and harness resource utilization for their desired throughput as depicted herein.

5. Conclusion

We have demonstrated a novel Docker-based computational resource allocation through optimization of memory, load-average, and CPU utilization that leads to a better throughput in sample processing and efficient system utilization in container runs as compared to the non-container runs for various stages of the human whole genome sequencing variant calling pipeline with no changes to source codes. This effectively reduces the overall execution time of the pipeline by about 40 % in regular workstation environments. Our publicly available method will achieve similar sustained parallel processing and throughput when replicated across multiple nodes of the cluster. Adopting such a methodology for human whole genome sequencing variant calling will greatly facilitate data processing and analysis for several research groups across the world without investing in expensive hardware infrastructure or being bound by cloud-based contracts.

Funding

This work was supported by DBT Ramalingaswami Fellowship Grant BT/RLF/Re-entry/29/2016 and SERB Early Career Research Award ECR/2018/001429 grant to BK. The funders had no role in study design, data collection, and analysis, decision to publish, or preparation of the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.imu.2021.100684.

Conflicts of interest

The authors have declared that no competing interests exist.

References

[1] Gonzaga-Jauregui C, Lupski JR, Gibbs RA. Human genome sequencing in health and disease. Annu Rev Med 2012 Feb;63:35–61. 18.
[2] Bycroft C, Freeman C, Petkova D, Band G, Elliott LT, Sharp K, Motyer A, Vukcevic D, Delaneau O, O'Connell J, Cortes A. The UK Biobank resource with deep phenotyping and genomic data. Nature 2018 Oct;562(7726):203–9.
[3] Albers PK, McVean G. Dating genomic variants and shared ancestry in population-scale sequencing data. PLoS Biol 2020 Jan;18(1):e3000586. 17.
[4] Leung YY, Valladares O, Chou YF, Lin HJ, Kuzma AB, Cantwell L, Qu L, Gangadharan P, Salerno WJ, Schellenberg GD, Wang LS. VCPA: genomic variant calling pipeline and data management tool for Alzheimer's Disease Sequencing Project. Bioinformatics 2018 Oct;35(10):1768–70. 23.
[5] Merkel D. Docker: lightweight linux containers for consistent development and deployment. Linux J 2014 Mar;2014(239):2. 1.
[6] Di Tommaso P, Palumbo E, Chatzou M, Prieto P, Heuer ML, Notredame C. The impact of Docker containers on the performance of genomic pipelines. PeerJ 2015 Sep;3:e1273. 24.
[7] Langdon WB, Lam BY. Genetically improved barraCUDA. BioData Min 2017 Dec;10(1):28.
[8] Houtgast EJ, Sima VM, Bertels K, Al-Ars Z. An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm. In2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS) 2015 Jul:221–7. 19.
[9] Causey JL, Ashby C, Walker K, Wang ZP, Yang M, Guan Y, Moore JH, Huang X. DNAp: a pipeline for DNA-seq data analysis. Sci Rep 2018 May;8(1):6793. 1.
[10] Kim B, Ali T, Lijeron C, Afgan E, Krampis K. Bio-Docklets: virtualization containers for single-step execution of NGS pipelines. GigaScience 2017 Jun;6(8):gix048. 27.
[11] McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, Garimella K, Altshuler D, Gabriel S, Daly M, DePristo MA. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res 2010 Sep;20(9):1297–303. 1.
[12] Tarasov A, Vilella AJ, Cuppen E, Nijman IJ, Prins P. Sambamba: fast processing of NGS alignment formats. Bioinformatics 2015 Feb;31(12):2032–4. 19.
[13] Faust GG, Hall IM. SAMBLASTER: fast duplicate marking and structural variant read extraction. Bioinformatics 2014 May;30(17):2503–5. 7.
[14] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. The sequence alignment/map format and SAMtools. Bioinformatics 2009 Aug 15;25(16):2078–9.
[15] Zook Justin M, et al. An open resource for accurately benchmarking small variant and reference calls. Nat Biotechnol 2019;37(5):561–6.
[16] Krusche Peter, et al. Best practices for benchmarking germline small-variant calls in human genomes. Nat Biotechnol 2019;37(5):555–60.