Distributed stream processing for genomics pipelines

Francesco Versaci, Luca Pireddu* and Gianluigi Zanetti

CRS4: Center for Advanced Studies, Research and Development in Sardinia, Pula, Italy

*Corresponding author email: luca.pireddu@crs4.it

Introduction

Personalized medicine is in great part enabled by the progress in data acquisition technologies for modern biology. These same technologies however are transforming biology into a data-intensive science, as the acquisition techniques that permit the observation of finer details of biological machinery generate ever larger dataset sizes. Genomics is the champion example of this phenomenon, with modern sequencing machines having a production capacity in the order of a terabyte of uncompressed data per day, which need to be computationally processed to allow the extraction of useful information. Conventional processing workflows are composed by independent, shared-memory tools which communicate by means of intermediate files. With increasing data sizes this approach is showing its limited scalability and robustness characteristics – problems that make it unsuitable for large-scale, population-wide personalized medicine applications.

In this work we propose the adoption of the stream computing architecture to make the genomics pipeline more scalable, and fault-tolerant. We decompose the first processing phases for Illumina sequencing data into two distinct and specialized modules (preprocessing and alignment) and we loosely link them through Kafka streams. This approach allows for easy composability of the modules and their integration with already existing pipelines. The proposed solution is experimentally validated on real data and shown to scale almost linearly.

Methods

The standard primary processing procedure when resequencing with Illumina sequencing machines consists of: 1) reconstructing reads from BCL files; 2) filtering reads based on machine quality checks; 3) sorting reads by sample (for multiplexed runs); 4) read alignment. In the state-of-the-art, sequencing operations are equipped with a conventional HPC cluster with a shared parallel storage system [4]. Within this context, the standard solution is to perform steps 1-3 using Illumina's own proprietary, open-source tool: bcl2fastq2. On the other hand, there is variety of read aligners in widespread use [3], though BWA-MEM [2] is quite popular among these and has been found to produce some of the best alignments [1]. These tools only provide parallelism within a single computing node (shared-memory parallelism). Though shared-memory parallelism is certainly beneficial to accelerating analysis, it is insufficient for large-scale operations. Processing the data produced by a single run of a modern sequencer can easily takes several hours if working on a single computing node.

Our method builds scalable, robust and easily composable tools by distributing the work over multiple collaborating computing nodes. More specifically, we use distributed processing of continuous streams of sequencing data. Moreover, our solution maintains compatibility with the already established tools by generating output the standard format CRAM – which is particularly space-efficient and thus well suited to large-scale applications. The software we created for this work builds on Apache Flink and Apache Kafka. Apache Flink is a framework for distributed (i.e., multi-node) stream data processing. With stream processing data is processed as soon as it arrives at the step – rather than waiting for the previous step to finish. Data flows between successive steps of a pipeline without intermediate files. On the other hand, Apache Kafka is a connector/queue service that allows us to connect multiple processes – e.g., multiple Flink programs – to each other.

The Flink-Kafka-based pipeline is implemented as two modules. The first module (*preprocessor*) reads as input raw Illumina data and performs BCL conversion, filtering and demultiplexing. The second module implements the alignment step in Flink, using the Read Aligner API (RAPI [5]) which provides Java bindings for the BWA-MEM aligner. The two modules are connected by a Kafka broker. The output

consists of aligned reads in CRAM format – though more complex pipelines could be formed by chaining additional Kafka brokers and processing modules.

We evaluated the performance and scalability of our Flink-Kafka pipeline on a human genome dataset, running on the Amazon Elastic Compute Cloud (EC2) and comparing it with the conventional pipeline.
Hardware. We ran our experiments using up to 12 instances of type i3.8xlarge that are equipped with: 32 virtual cores (Xeon E5-2686 v4, 45 MB cache); 244 GiB of RAM; 4 x 1.9 TB NVMe SSDs; 10 GbE.
Input dataset. We used 1/4 of the output from an Illumina HiSeq 3000 run, with 12 human genomic samples and using a single multiplexing tag per fragment. Data size: 47.8 GB of raw data scattered over 47,050 *gzip-compressed* BCL files plus 224 filter files (QC pass/fail).

We ran our pipeline with varying number of nodes n, from 1 to 12. The setup included a single Kafka broker while all other nodes ran both preprocessing and alignment modules. The baseline workflow was implemented with bcl2fastq2 → bwa-mem → samtools (the latter for conversion of SAM to CRAM). The exact scripts are available at `https://github.com/crs4/2017-flink-pipeline`. As mentioned previously, the conventional pipeline only exploits shared-memory parallelism and hence ran on a single node.

Results

The runtimes measured in our experiments (Table 1) show that on a single node our pipeline is 11% slower than the baseline due to the overheads caused by Flink and the communication layer. However, as the number of nodes increases, our pipeline achieves near optimal scalability, as shown in Figure 1. The relative scalability (i.e., compared to our runtime on a single node) is 10.6 on 12 nodes, whereas the absolute one (i.e., compared to the faster baseline) is 9.5 on 12 nodes. This result is particularly remarkable since the runtime on 12 nodes is below 15 minutes, and the constant costs of the frameworks begin to have a significant impact on the total runtime.

Table 1: Running times of our test pipelines.

nodes	time (minutes)
baseline	137
1	152
2	77
4	39.6
8	20.4
12	14.3

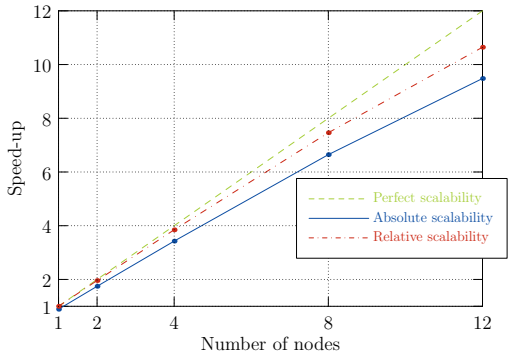

Figure 1: Strong scaling of our Flink/Kafka pipeline, compared with baseline.

References

[1] Adam Cornish and Chittibabu Guda. “A Comparison of Variant Calling Pipelines Using Genome in a Bottle as a Reference”. In: *BioMed Research International* 2015 (2015), p. 11. DOI: 10.1155/2015/456479.

[2] Heng Li. *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM*. 2013. arXiv: 1303.3997v1 [q-bio.GN].

[3] Matthew Ruffalo, Thomas LaFramboise, and Mehmet Koyutürk. “Comparative analysis of algorithms for next-generation sequencing read alignment”. In: *Bioinformatics* 27.20 (2011), pp. 2790–2796. DOI: 10.1093/bioinformatics/btr477.

[4] Ola Spjuth et al. “Experiences with workflows for automating data-intensive bioinformatics”. In: *Biology Direct* 10.1 (2015), pp. 1–12.

[5] Francesco Versaci, Luca Pireddu, and Gianluigi Zanetti. “Scalable genomics: From raw data to aligned reads on Apache YARN”. In: *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*. 2016, pp. 1232–1241. DOI: 10.1109/BigData.2016.7840727.