Flink in Genomics

Integrating Flink and Kafka in the standard genomic pipeline

F. VERSACI L. PIREDDU G. ZANETTI

– Flink Forward 2017 –

13 September 2017

CRS4

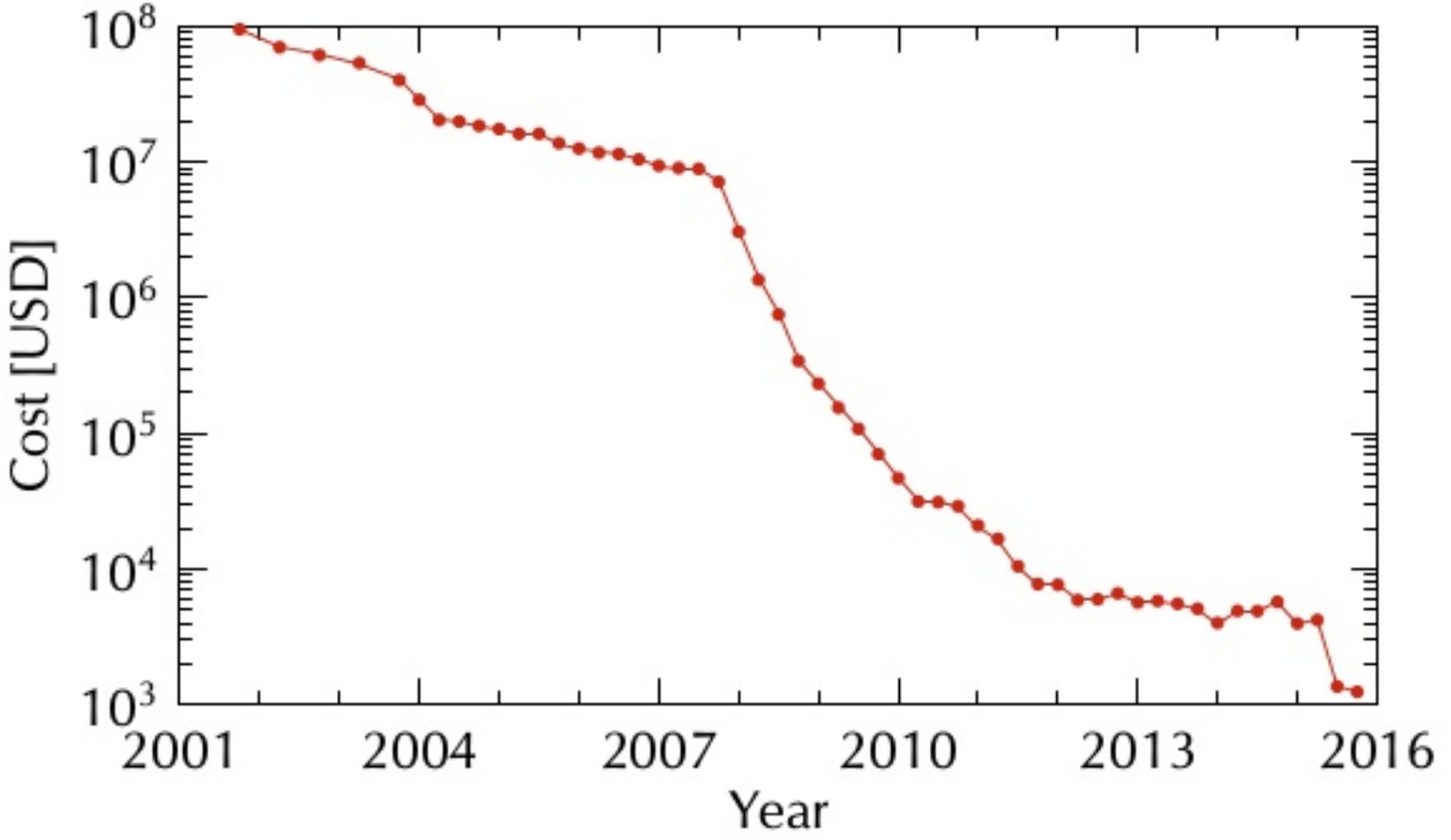
Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna

- Research center in Sardinia, Italy
- Focus on big data, biosciences, HPC, visual computing, energy and environment

Outline

Next-Generation Sequencing
Cost

- Genome sequencing is now much cheaper than in the past
- About 1000 euros per whole human genome



(Data from `https://www.genome.gov/sequencingcosts/`)

Next-Generation Sequencing
Applications

High-throughput DNA sequencing has many applications, including

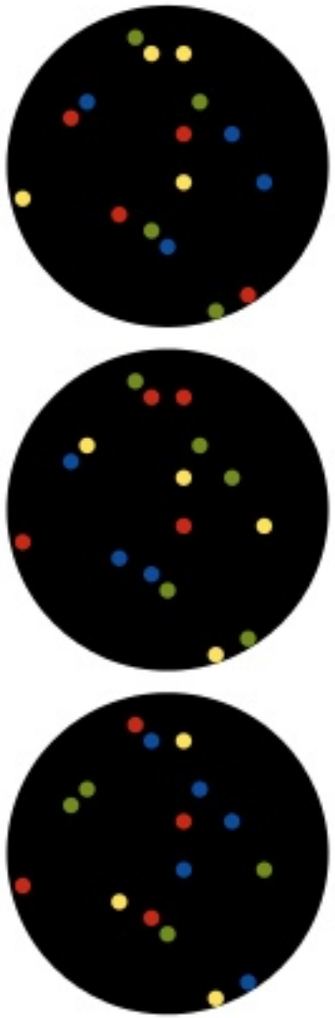
- Research into understanding human genetic diseases
- Medicine, e.g., oncology, clinical pathology, ...
- Human phylogeny
- Personalized diagnostic applications

Huge amount of data

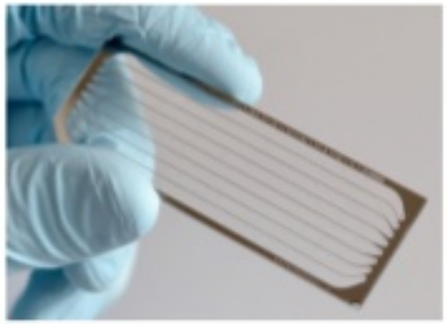
A single sequencer can produce 1 TB/day of data

- Which need to be converted, filtered, aggregated, reconstructed, analyzed, ...
- A single genomic center can have hundreds of sequencers

Shotgun genome sequencing



- The DNA is a sequence of four bases: Adenine, Cytosine, Guanine and Thymine (A, C, G and T)
- The genome is broken up into short fragments (reads)
- The fragments are attached to a support (tile)
- Fluorescent molecules are iteratively attached to the bases
- At each cycle, the machine acquires (optically) a single base from all the fragments



(CC-BY-3.0 image from `https://goo.gl/9dqN8g`)

Data processing in NGS
First steps and standard pipeline

Pre-processing Reads need to be reconstructed from the raw, cycle-based, sequencer's output (BCL files)

Alignment The short reads are aligned to a reference genome

When using Illumina sequencers, the standard pipeline starts with two programs:

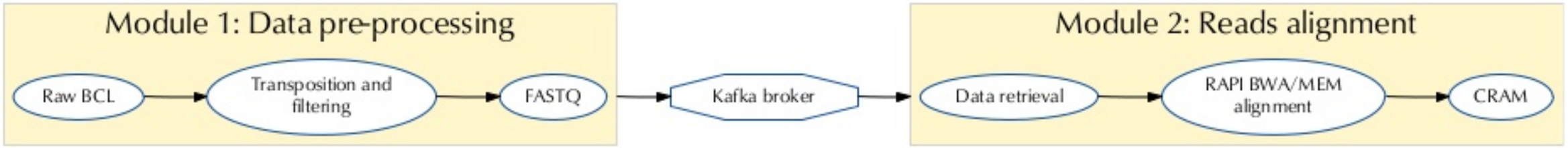
bcl2fastq2 Proprietary, open-source tool by Illumina to convert raw BCL data to FASTQ format

BWA-MEM Free (GPLv3) aligner to reconstruct the full genomic sequence based on the short reads generated by the sequencer

Problems

- Parallel tools, but shared-memory (single node)
- Communication via intermediate files:
  - Different steps must be run sequentially
  - New data requires re-computing the workflow from scratch

Our approach



Data pre-processing

- Flink program, written from scratch (see FlinkForward 2016 and BigData 2016[1])
- Modified to sends its output to Kafka
- It creates many (thousands) topics

Reads alignment

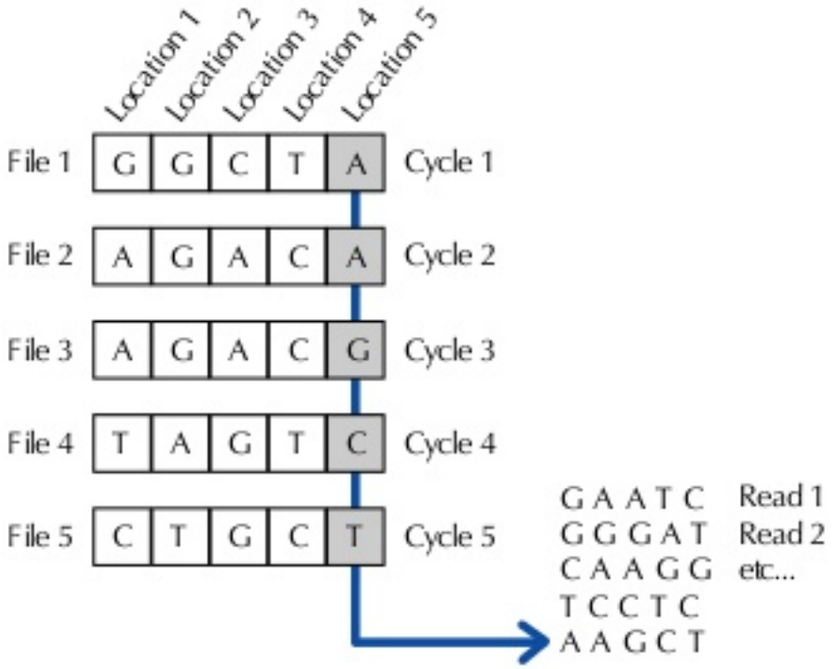
- Flink program, powered by RAPI and BWA/MEM
- Reads from Kafka
- Writes to Kafka or HDFS

[1] Francesco Versaci, Luca Pireddu, and Gianluigi Zanetti (2016). "Scalable genomics: From raw data to aligned reads on Apache YARN". In: *IEEE BigData 2016*, pp. 1232–1241

Outline

Data pre-processing

- Bases and quality scores are decoded
- A data transposition is performed to obtain the reads
- For each tile several new Kafka data topics are created
- The list of the new topics is sent to a special control topic
- The reads are written into the data topics
- Finally, an EOS marker is appended to each data topic



Sending finite streams

- Each data topic will contain a finite amount of data
- Kafka streams are potentially infinite
- Hence we need to send a EOS when a data topic is completed

Reads alignment

- The control topic is polled every 3 seconds, to receive notice of newly opened data topics
- The data topics are grouped into Flink jobs and aligned in parallel
- For each data topic, the reads are chunked together and aligned
- The aligned reads are sent to the output sink (e.g., written as CRAM files).

- The number of data topics is not known in advance
- When a data topic is completed the corresponding DataSource needs to be terminated
- To improve the aligner performance the chunks should be the same size

Handling finite streams
Our experience

- Finite streams do not work as (we) intended with count windows (last window is not evaluated)
- There is an easy work-around which uses event time windows

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
val consumer = new FlinkKafkaConsumer010[MyType](
  topicname, new MyDeserializer, properties)
  .assignTimestampsAndWatermarks(new MyTStamper)
val ds = env.addSource(consumer)
val out = ds.timeWindowAll(Time.milliseconds(1024))
  .apply(new MyAllWindowFunction)
// do something with the output
// ...
```

Handling finite streams
Deserialization

```
class MyDeserializer extends DeserializationSchema[MyType] {
  override def getProducedType =
    TypeInformation.of(classOf[MyType])
  override def isEndOfStream(el : MyType) : Boolean = {
    return el.isEOS
  }
  override def deserialize(data : Array[Byte]) : MyType = {
    // deserialization code here
  }
}
```

Handling finite streams
Timestamping

```
class MyTStamper extends AssignerWithPeriodicWatermarks[MyType] {
  var cur = 0l
  override def extractTimestamp(el: MyType, prev: Long): Long = {
    val r = cur
    cur += 1
    return r
  }
  override def getCurrentWatermark : Watermark = {
    new Watermark(cur - 1)
  }
}
```

Outline

Experimental setup

- Experiments run on the Amazon Elastic Compute Cloud (EC2)
- Up to 12 instances of i3.8xlarge machines

  CPUs 32 virtual cores (Xeon E5-2686 v4, 45 MB cache)

  RAM 244 GiB

  Disks 4 x 1.9 TB NVMe SSD

  Network 10 Gb Ethernet

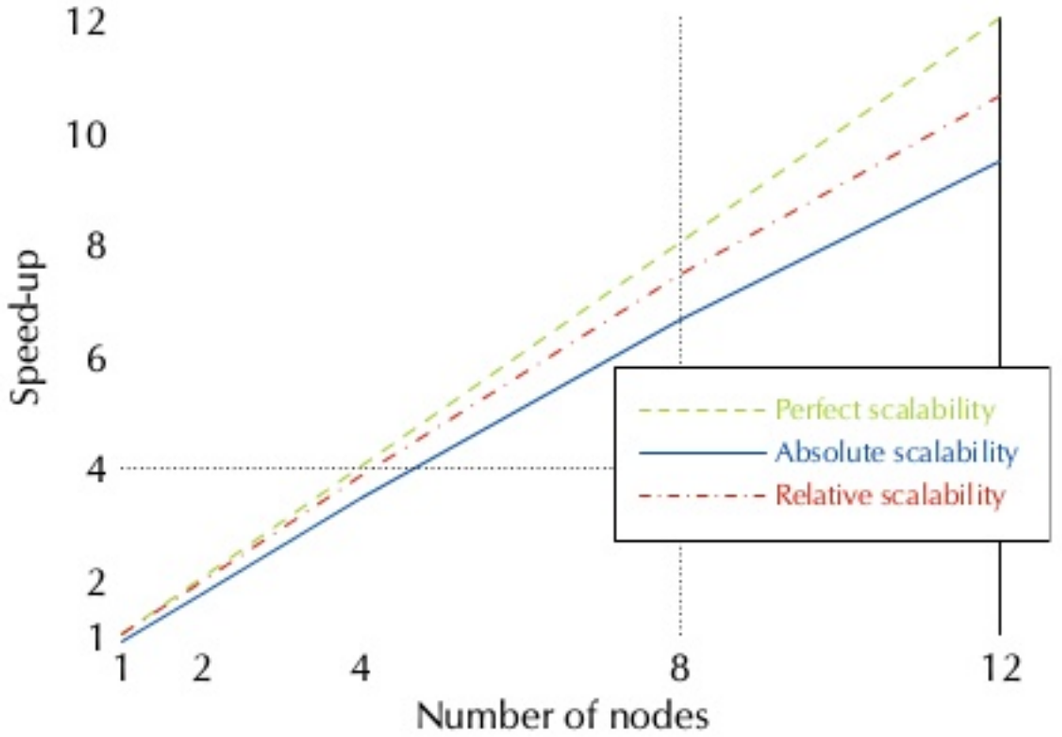
  OS Ubuntu 16.04.2 LTS (Linux kernel 4.4.0-1013-aws)

Input dataset

Sequencer	Illumina HiSeq 3000
Size of data files	47.8 GB (gzip-compressed)
Number of files	47,040
Number of samples	12
Number of reads	$3.38 \cdot 10^8$
Number of DNA bases	$3.38 \cdot 10^{10}$

Results
Strong scalability

Nodes	Time (minutes)
Baseline	137
1	152
2	77
4	39.6
8	20.4
12	14.3



- Same input, variable number of nodes
- On single node: 11% slower than the baseline
- On more nodes: almost linear scalability

Load on Kafka

We also tested the Kafka broker under stress, using 4 nodes

Node-1 Running the Kafka broker and Flink Job-Manager

Node-[2-4] Running the preprocessor at full speed (32 concurrent jobs per node)

- Data flow of 450 MB/s towards the broker for about 3 minutes
- 2688 topics created
- 340 million messages sent
- The CPU load was about 900%
- Assuming linearity, the network bandwidth is saturated at about 25 cores (10 Gb/s = 1250 MB/s)

Limits to scalability

- An aligning node at maximum power (32 cores at 100%) requires 10 MB/s of data
- A single Kafka broker can sustain up to 125 nodes

Conclusion and future work

- We have presented a fully scalable streaming pipeline to process raw Illumina NGS data up to the stage of aligning reads
- To facilitate the adoption of scalable stream processing in genomics...
- ...we aim at integrating our pipeline with the upcoming Genome Analysis Toolkit 4 (GATK4, a Spark-based genomic analysis tool)

Conclusion and future work

- We have presented a fully scalable streaming pipeline to process raw Illumina NGS data up to the stage of aligning reads
- To facilitate the adoption of scalable stream processing in genomics...
- ...we aim at integrating our pipeline with the upcoming Genome Analysis Toolkit 4 (GATK4, a Spark-based genomic analysis tool)

Thanks for your attention!