Scaling a Variant Calling Genomics Pipeline with FaaS

9th International Workshop on Serverless Computing

Part of ACM/IFIP International Middleware Conference, December 11-15, 2023 in DAMSLab, Department of Arts, University of Bologna, Italy

UNIVERSITAT ROVIRA i VIRGILI

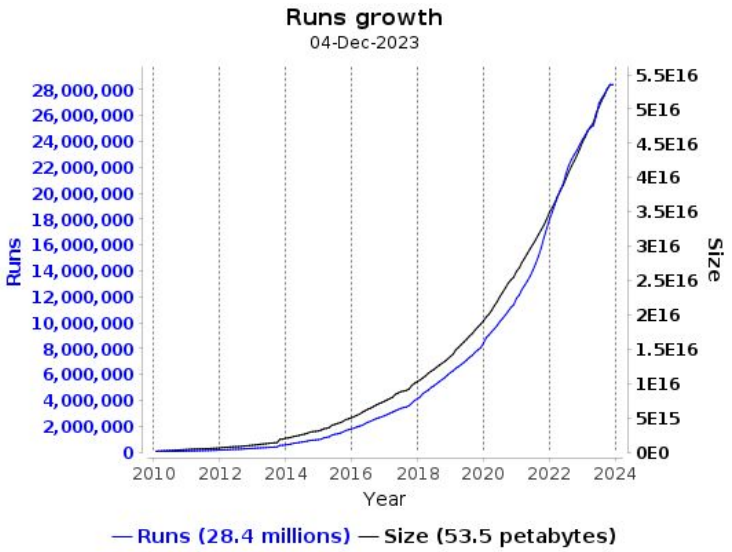
BioSS

Aitor Arjona[a]
Arnau Gabriel-Atienza[a]
Sara Lanuza-Orna[a]
Xavier Roca-Canals[a]
Ayman Bourramouss[a]
Tyler K. Chafin[b]
Lucio Marcello[b]
Paolo Ribeca[b]
Pedro García-López[a]

a. Universitat Rovira i Virgili, Tarragona, Spain
b. Biomathematics and Statistics Scotland, Edinburgh, United Kingdom

Genomics Workloads

ENA
European Nucleotide Archive



- ➔ Genomics is a **compute- and data intensive-** task.
- ➔ Exponential growth in data size and complexity.
- ➔ Biomedical institutions with HPC **struggle to keep up**.

Cloud IaaS for Genomics

The Cloud elasticity is key for scaling genomics workloads using short-term resources.

nextflow

APACHE Spark™

aws

Azure

Google Cloud

Cloud IaaS for Genomics

nextflow

APACHE Spark™

aws

Azure

Google Cloud

The Cloud elasticity is key for scaling genomics workloads using short-term resources.

➔ **Complexity for bioinformatic users**

- Capacity/VM size for processing X GB of data?
- Auto-scaling?
- Hidden costs?

Configuring, deploying and scaling genomics workloads is challenging for bioinformatics users.

Going serverless

Serverless (FaaS)

1. **Pay only for resources used** at millisecond granularity, scale down to zero when not used
2. **Instant scalability** (~200 ms cold start, thousands of parallel functions)
3. **Completely managed:** Scaling, security…

➔ Why serverless for genomic pipelines?

- No servers to manage!
- *Less* friction to the Cloud for *less* experienced (bioinformatic) users
- Allows to **massively and effortlessly scale highly-parallel genomics workloads**.

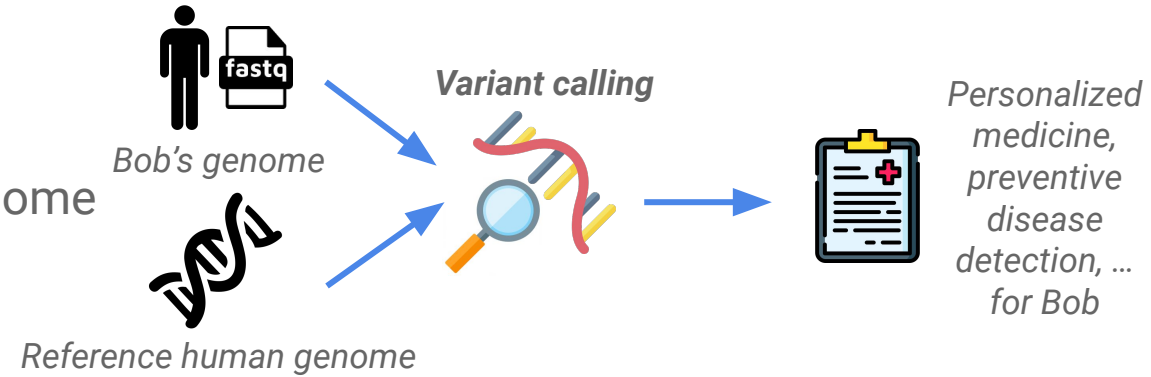
Serverless Genomic Variant Calling

Objective: Adapt a **single-node HPC** variant calling genomics application to **serverless** in order to **scale in parallelism**, **process larger datasets** and **decrease runtime**.

Serverless Genomic Variant Calling

Objective: Adapt a **single-node HPC** variant calling genomics application to **serverless** in order to **scale in parallelism**, **process larger datasets** and **decrease runtime**.

- **Variant Calling:** detect differences (mutations, variants) in a sampled genome compared to a reference genome.



Distributing a Variant Calling pipeline

Sampled Genome (FASTQ)

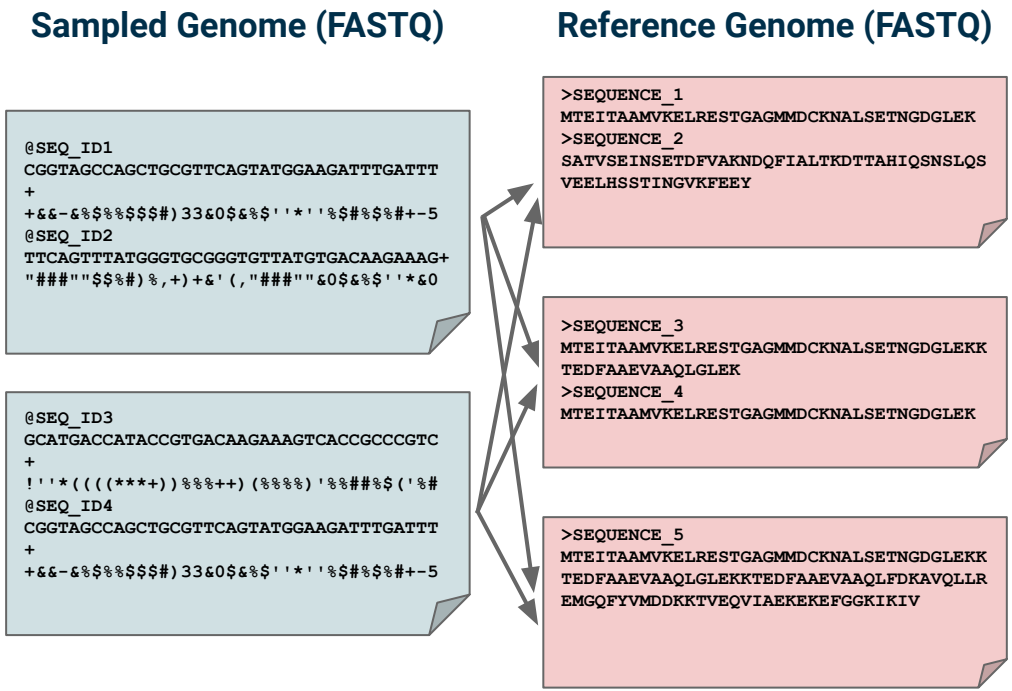
```
@SEQ_ID1
CGGTAGCCAGCTGCGTTCAGTATGGAAGATTTGATTT
+
+&&-&%$%%$$$#)33&0$&%$''*''%$#%$%#+-5
@SEQ_ID2
TTCAGTTTATGGGTGCGGGTGTTATGTGACAAGAAAG+
"###""$$%#)%,+)+&'(,"###""&0$&%$''*&0
@SEQ_ID3
GCATGACCATACCGTGACAAGAAAGTCACCGCCCGTC
+
!''*((((***+))%%%++)(%%%%)'%%##%$('%#
@SEQ_ID4
CGGTAGCCAGCTGCGTTCAGTATGGAAGATTTGATTT
+
+&&-&%$%%$$$#)33&0$&%$''*''%$#%$%#+-5
@SEQ_ID5
TTCAGTTTATGGGTGCGGGTGTTATGTGACAAGAAAG+
"###""$$%#)%,+)+&'(,"###""&0$&%$''*&0
```

Reference Genome (FASTQ)

```
>SEQUENCE_1
MTEITAAMVKELRESTGAGMMDCKNALSETNGDGLEKK
TEDFAAEVAAQLGLEKKTEDFAAEVAAQLFDKAVQLLR
EMGQFYVMDDKKTVEQVIAEKEKEFGGKIKIV
>SEQUENCE_2
SATVSEINSETDFVAKNDQFIALTKDTTAHIQSNSLQS
VEELHSSTINGVKFEEYLKSQIATIGENLVVRRFATLK
AGANGVVNGYIHTNGRVGVVIAAACDSAE
>SEQUENCE_3
MTEITAAMVKELRESTGAGMMDCKNALSETNGDGLEKK
TEDFAAEVAAQLGLEKKTEDFAAEVAAQLFDKAVQLLR
EMGQFYVMDDKKTVEQVIAEKEKEFGGKIKIV
>SEQUENCE_4
SATVSEINSETDFVAKNDQFIALTKDTTAHIQSNSLQS
VEELHSSTINGVKFEEYLKSQIATIGENLVVRRFATLK
AGANGVVNGYIHTNGRVGVVIAAACDSAE
>SEQUENCE_5
MTEITAAMVKELRESTGAGMMDCKNALSETNGDGLEKK
TEDFAAEVAAQLGLEKKTEDFAAEVAAQLFDKAVQLLR
EMGQFYVMDDKKTVEQVIAEKEKEFGGKIKIV
```

- Process: **String similarity search** - Compare all sample genome to all reference genome

Distributing a Variant Calling pipeline



- Process: **String similarity search** - Compare all sample genome to all reference genome
- How to distribute the workload?
  - Partition the dataset and perform an all to all comparison
  - Cartesian product → **Lots of parallel tasks**
- Partitioning the dataset implies:
  - Partial correction process
  - Merging all partial results to produce the final output

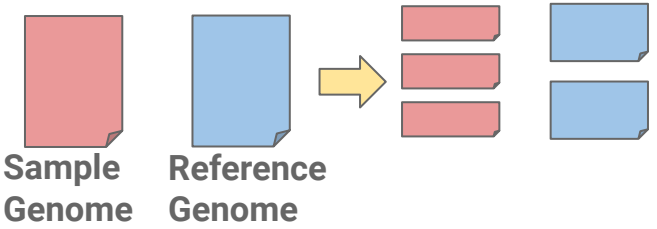
Serverless Variant Calling Pipeline Architecture

Pre-Processing
Partition input data to be distributed between lambdas.

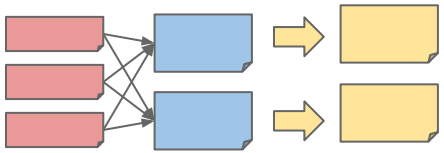
Map
Genome alignment (String similarity search) between sample and reference sets.

1. Alignment
2. Partial index correction
3. Assemble alignment

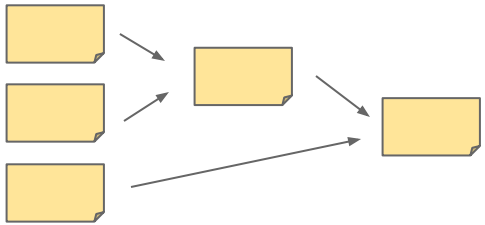
Reduce
Reduce and merge partial outputs from map phase.



How to distribute the workload?
→Partition the dataset

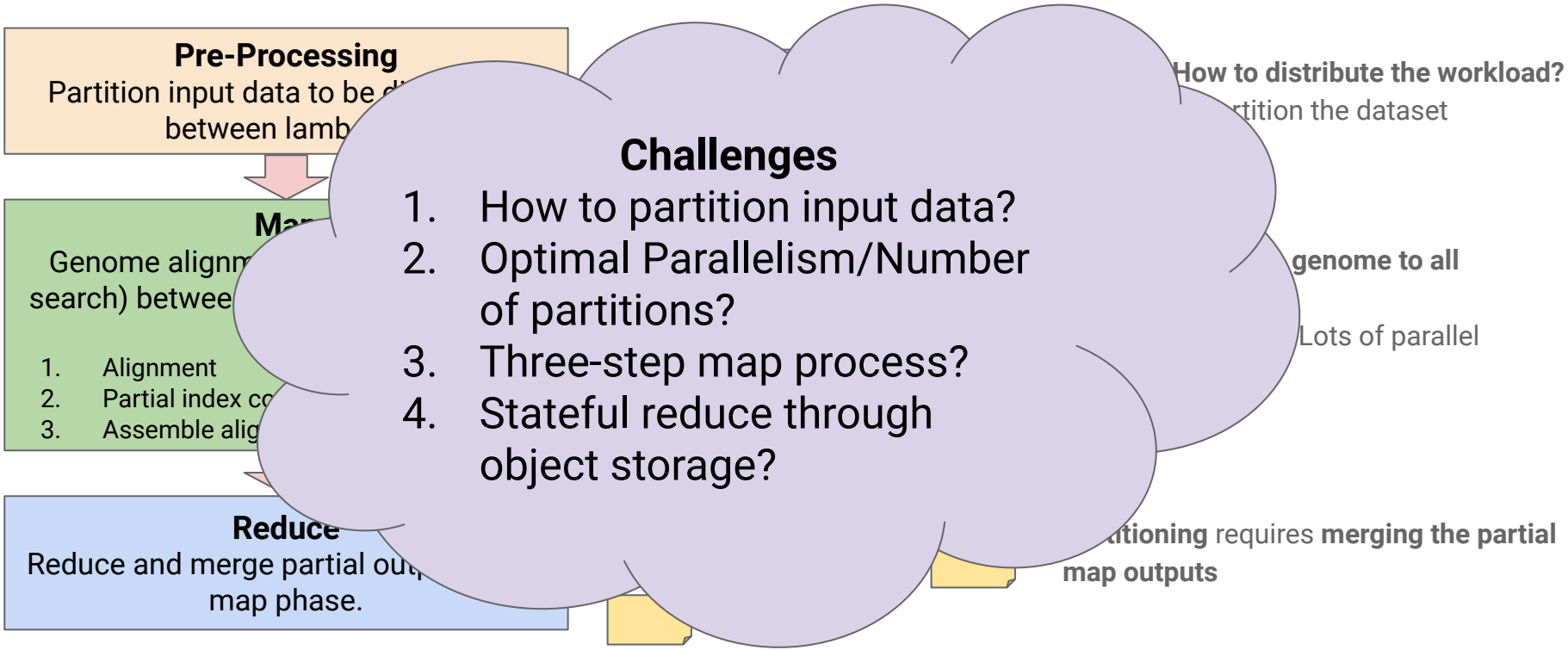


Compare all sample genome to all reference genome
→Cartesian product: Lots of parallel tasks

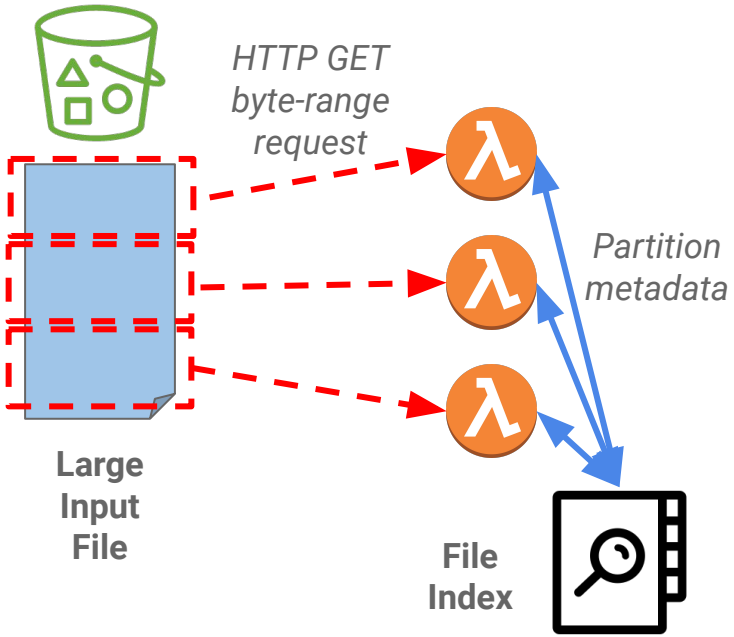


Partitioning requires **merging the partial map outputs**

Serverless Variant Calling Pipeline Architecture



Challenge 1
Input Data Partitioning



- **Objective**: **partition** input data
- Partial reads with HTTP GET byte-range requests

- Arbitrary **byte-ranges** breaks the genome file
- We require more **metadata** for each partition (sequence identifier and offset)

- Indexing to **locate and identify** each sequence for **any arbitrary byte-range**
- Lookup index for partition metadata

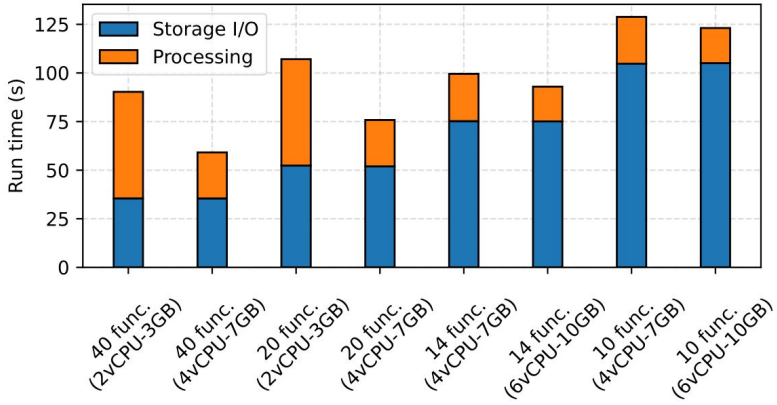
Challenge 2
Data dependencies from synchronous HPC code

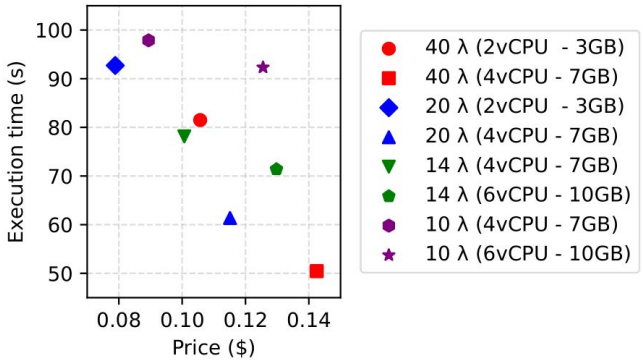
- **Data dependencies→Functions calling functions:** Stop the process, call another task, synchronize (wait), get result, then resume.

- **No preemption** in serverless →Blocked tasks **occupy concurrency slot**
- Can provoke deadlocks and limit scalability

- Blocking code must be **split** into many **asynchronous tasks** that can be **scheduled independently**
- **Data dependencies** must be passed through **object storage**

Challenge 3
Optimal degree of parallelism





- How many tasks to launch?

- **More tasks** →More parallelism →**Less scalable** (concurrency limit)
- **More tasks** → Larger data chunks → Less efficient

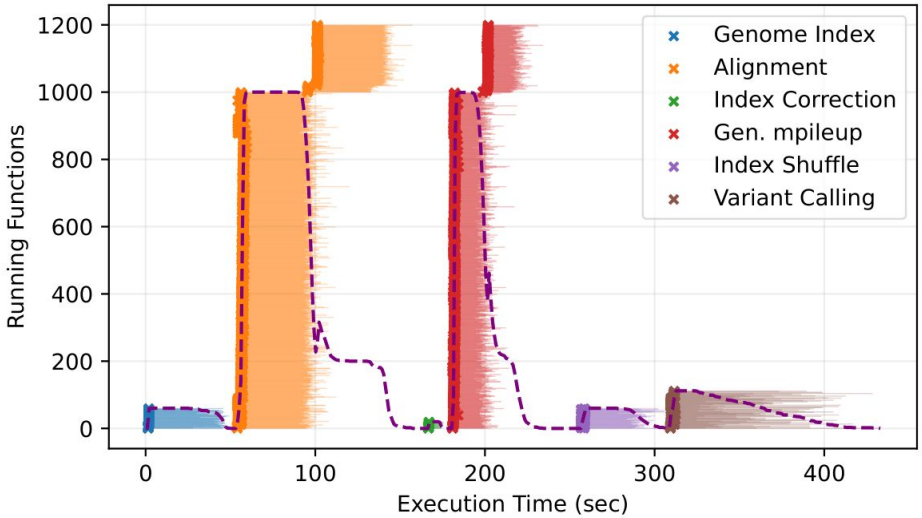
- Leverage **intra-function parallelism** to launch **less tasks** without sacrificing **performance**

Evaluation

HPC vs Serverless

Stage	*HPC*	*Serverless*
Genome Indexing	0 min 14.20 s	0 min 9.81 s
Alignment	0 min 14.20 s	0 min 48.10 s
Index correction	-	0 min 7.63 s
Generate mpileup	51 min 15.79 s	1 min 6.55 s
Index shuffle	-	0 min 10.73 s
Variant Calling	54 min 5.04s s	0 min 27.82 s
Total	106 min 8.21 s	2 min 50.64 s

Large scale execution



Insights and conclusions

1. **Optimizing performance-cost** with **multi-CPU** functions requires **balancing intra- and inter-function parallelism**
2. Effective **data partitioning** is vital to exploit the massive parallelism offered by FaaS.
3. **Asynchronous and non-blocking** code is strictly necessary to guarantee scalability and to **avoid the concurrency limits** of serverless platforms.
4. Although **object storage** offers scalability for intermediate workflow data storage, it suffers from **slow performance and elevated costs**.
5. **Serverless data processing services** for object storage, like S3 SELECT, provide a nice **synergy** with serverless FaaS workloads.

Thank you!
Any questions?

You can find me at aitor.arjona@urv.cat

Challenge 4
Stateful data movements

- Reduce stage → Stateful data movements through object storage

- **I/O time** from Lambda to S3 is **expensive**
- We want to **simplify** the pipeline

- Delegate **shuffle logic** to S3 SELECT
- **S3 SELECT** allows to define simple SQL queries over semi-structured data
- **Cheaper, more simple, and less error-prone** than doing **ad-hoc shuffling**