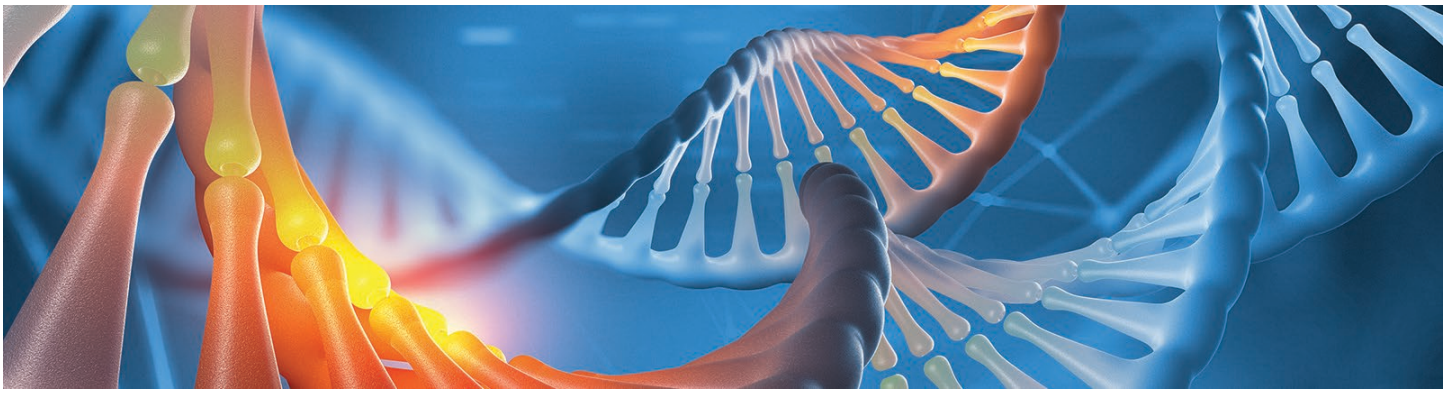
ACCELERATING CLOUD-BASED GENOMICS PIPELINES THROUGH I/O PROFILING FOR ANALYSIS OF MORE THAN 3,000 WHOLE GENOME PAIRS ON AWS

Dr. Rosemary Francis, Chief Scientist, Altair / Keiran Raine, Cancer Researcher, Wellcome Sanger Institute / January 27, 2021



Making a Cancer Pipeline Portable

This paper presents an overview of the work by the Wellcome Sanger Institute to make one of their cancer pipelines portable and to tune it for cloud deployment using the Altair Breeze™ and Altair Mistral™ I/O profiling tools. **With the insight from the tools we were able to tune the cloud configuration to boost speed by 20% with a cost reduction of 10%.**

As an extension to the work, we profiled the containerized workload on AWS and determined that the default storage option is the best value for money over faster, more expensive options. This cost saving is possible due to the effort that has been put in by the scientists at the Wellcome Sanger Institute to make the pipeline fast, agile, and cloud-ready.

Cancer, Ageing and Somatic Mutation Group at the Wellcome Sanger Institute

The research of the Wellcome Sanger Institute's Cancer, Ageing and Somatic Mutation (CASM) Group focuses on understanding the mutational processes that lead to various age-related diseases and cancer. This is a diverse and rapidly changing field using both model organisms and patient samples.

CASM-IT is a dedicated and motivated team of bioinformaticians and software developers who support scientific staff, simplifying the use of tools, incorporating them into large-scale pipelines, and presenting data in more easily digestible forms. The team develops many of their own tools.

Migration to Containers and Cloud Compute

In 2013, two of the largest funders of genomic research agreed to one of the most ambitious data analysis projects at that time: the ICGC/TCGA PanCancer Analysis of Whole Genomes (PCAWG). The goal of the project was to analyze 2,000 donors, generating uniform and comparable data that was to be processed at many sites.

In order to participate, the "Sanger" pipeline (CASM-IT) needed to be usable at many sites. CASM-IT started working to solve the tie-ins. In 2015, the decision was made to switch to Docker and for significant processing to be carried out using Amazon Web Services (AWS). The project went well and the initial deployments were handled without containers. Final versions of all analysis flows exist on dockstore.org.

The result of this migration to containers and the cloud is that tools are now far more portable. More users can access the tools and it is easier for the IT team to get buy-in for changes to working practices. Tools are now being employed in a PanProstate analysis project, which is a similar size to PCAWG but restricted to a single cancer type.

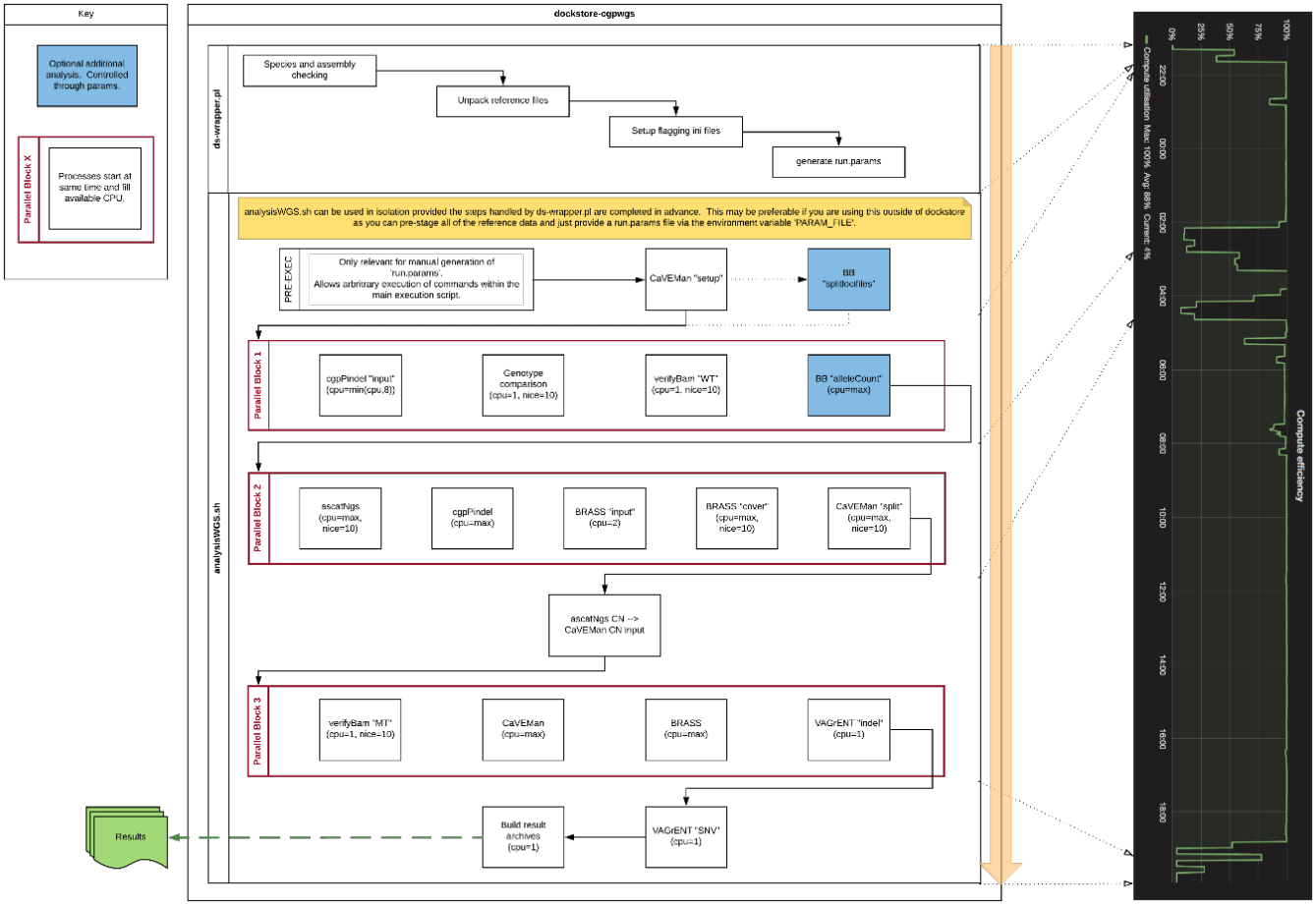
Containers Aren't Always as Simple as They Seem

It is often a general assumption that containers just run. However, many variables can mean that a container will not perform as well as it should, or not at all. Problems particularly arise when the software that is being containerized has not been written by a software developer. It can be poorly defined or unstable, with lots of dependencies that are not clear and can be left behind when the container is generated.

The ideal situation for containerizing a tool is for it to be a well-understood tool with no other dependencies and in-house support. Ideally it will have been designed to be species-agnostic and for I/O to have been taken into consideration.

An example of the ideal scenario is dockstore-cgpwgs, the CASM-IT whole genome sequencing (WGS) analysis container. The team knows the order of events and can easily map them back to a CPU utilization trace. It is clear where CPU usage is expected to be low.

The Mistral and Breeze tools were able to identify why the team was hitting I/O bottlenecks in areas where they expected full CPU utilization. This is where the tools are very useful.



I/O Profiling Tools

The Breeze and Mistral tools are designed to give application-level I/O data, trading off between detail and scalability. Breeze analyses one application at a time to debug dependency issues and I/O problems. It can tell you about every program, file, and network connection so you can easily solve user issues, containerize and migrate applications to the cloud, and check I/O performance. The Breeze Healthcheck report is a simple way to check application performance against dozens of common I/O issues.

In contrast, Mistral is an I/O telemetry tool that is lightweight enough to be run in production to locate bad I/O patterns. IT managers can set up system checks and users can opt in to get more information through I/O profiling as a service.

When optimizing the WGS pipeline, the CASM IT team could make initial performance gains using their knowledge of the tools in use. However, the team knew where to look and the characteristics of the individual processes. The Breeze and Mistral tools helped direct the team to the next set of easy gains, items that hadn't been considered because they were used to using high-performance file systems such as Lustre.

Milestone	Runtime (Wall Clock)	Comments
Proof of concept	32h	
Reorganize I/O	30h	Tune using knowledge of pipeline
Altair profiling	29h	Tune using Breeze to identify obvious I/O in algorithms and for light compression of intermediate data
More profiling	28h	Push more processes to named pipes
Hardware upgrade	18h	Latest tuning effort on production hardware

Combining improved hardware (the data above was taken from scavenged pilot systems) and further enhancements of algorithms, this example data now completes in 18 hours.

Profiling dockstore-cgpwgs

To create the I/O profile, our team first had to set up and run the WGS pipeline on AWS. This was easy because it had been packaged for cloud deployment.

This is the script we ran, using chromosome 21:

```
ds-cgpwxs.pl -r /data/step2/input/core_ref_GRCh37d5.tar.gz -a
/data/step2/input/VAGrENT_ref_GRCh37d5_ensembl_75.tar.gz -si
/data/step2/input/SNV_INDEL_ref_GRCh37d5-fragment.tar.gz -t /data/step2/input/COLO-829.bam -tidx
/data/step2/input/COLO-829.bam.bai -n /data/step2/input/COLO-829-BL.bam -nidx
/data/step2/input/COLO-829-BL.bam.bai -exclude
"1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,22,X,Y,MT,NC_007605,hs37d5,GL%" -outdir
/data/step2/output
```

The team ran the pipeline on the older magnetic EBS storage and the fastest SSD for an initial comparison. NVMe differs from the other types of SSD available in that it is connected via a motherboard PCI slot instead of being network attached. The m5d is needed for the NVMe storage and for everything else the team used an m5.large instance.

How long it took versus cost:

1 x 150GB NVMe SSD	51m 27s	$191.79 /month
Magnetic EBS HDD	1h 01m 44s*	$174.43 /month

Comparing run times, the initial conclusion is that SSD is worth it: it was 20% faster for a 10% cost increase. Both the cost difference and the performance difference were lower than expected. Of course, the tools also add a small overhead, but this was low for both: on magnetic storage the overhead was 0.3% for Breeze and 0.08% Mistral over the total run time, and 0.73% for Breeze and 0.2% for Mistral during the period of intensive I/O.

Profile Results: Good and Bad I/O Patterns

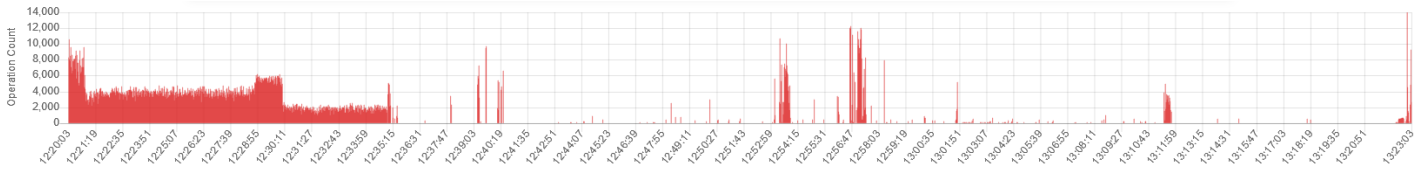
To explain the performance results, we took a closer look at the I/O profile. Breeze can characterize the amount of time spent in good and bad I/O. Bad I/O patterns are activities that are obviously a waste of time such as failed I/O, as well as patterns that can cause performance issues such as random I/O.

On the HDD, small-read operations took 12m27s, small writes took 4s and there was 12m3s of good streaming I/O. By comparison, on the SSD small reads took 12s, small writes took 4s, and there was 53s of good streaming I/O. In addition to the SSD having much faster I/O, the proportion of time doing small I/O operations was much smaller, which shows the impact those operations can have on performance.

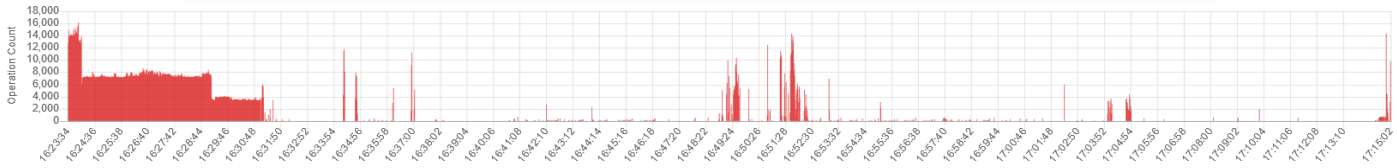
On both storage platforms there were some large sync operations: 13s on the HDD and 43s on the SSD. This is colored yellow on the Breeze profile results chart because sync operations are sometimes bad practice, but in this case Breeze showed that the sync operations were to write out large memory mapped files at the end of the pipeline so were a good design and would result in efficient I/O.

IOPs Over Time

Most of the I/O was performed in the first ten minutes with short bursts after that. As a result, the run times were similar, despite very different I/O performance profiles.



Number of I/O Operations Over Time on Magnetic HDD

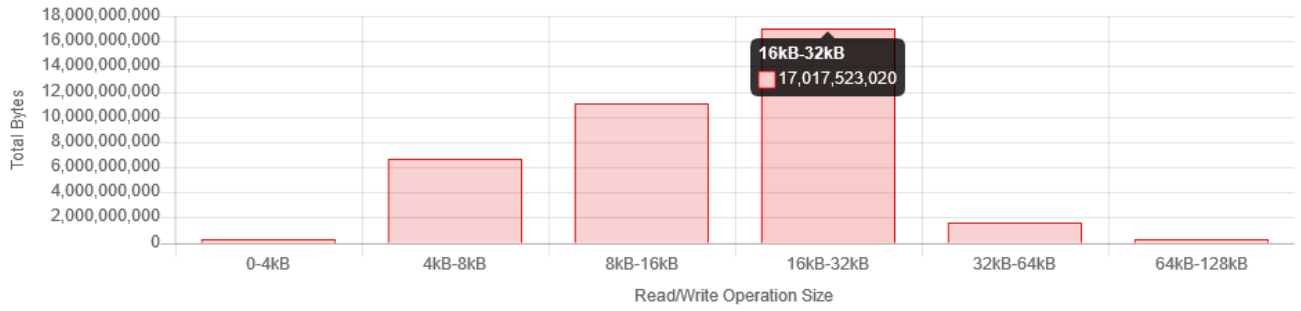


Number of I/O Operations Over Time on NVMe SSD

Results Analysis

Genome pipelines always have lots of very small I/O operations, but there has obviously been a lot of work to minimize that in this pipeline. Effort has been made to read larger blocks when the sections of data required are close. This means that the size of the I/O patterns are not as small as we would normally expect from a genome pipeline. This is likely to have made a lot of difference to the I/O performance.

Number of Read/Write Operations by I/O Size



Amount of Data Read/Written by I/O Size

Comparing Different Storage Products on AWS

Taking the analysis a step further, we ran the pipeline on a range of other storage options with the M5.large compute instance: GP2, 100 provisioned IOPS SSD, 500 provisioned IOPS SSD, and throughput optimized HDD (500GB). We chose 100GB for each SSD test.

GP2 storage is the AWS default option and is now always SSD. The IOPS it can deliver depends on the size of the disk. The provisioned IOPS guarantee 100 or 500 IOPS respectively where an IOP size is 256KB. It does therefore reward random access. The magnetic disks have an IOP size of 1,024KB so reward streaming reads and writes.

The following is a comparison of running the pipeline on the different storage options with the run time and monthly cost normalized against the GP2 option:

	Time*	Normalized Time	Cost Per Month	Normalized Cost
GP2 (AWS default SSD)	52m 23s	100%	174.11	100%
Magnetic EBS	1h 01m 44s	118%	174.43	100%
Provisioned 100 IOPS	1h 42m 01s	195%	184.61	106%
Throughput-optimized HDD	1h 19m 32s	152%	189.01	109%
150GB NVMe	51m 27s	98%	191.79	110%
Provisioned 500 IOPS	54m 22s	104%	215.01	123%

The throughput-optimized HDD performed very badly with the small reads. A workload with more streaming I/O would have done better, but in this case the bursty small reads were very slow making the optimized HDD option slower as well as more expensive than GP2.

The provisioned IOPS SSDs didn't have enough IOPS for the bursts so also didn't deliver better performance despite costing more. Workloads with very even I/O demands would do better on those.

NVMe is the fastest, but only 2% faster than GP2 for a 10% price increase. This leaves GP2 as the best, but this tradeoff is tight and would be different if the pipeline had more I/O. Pipelines that have not been optimized or that have a longer I/O-intensive period at the start may perform better on faster storage, but it is easy and quick to check.

Conclusion

The work done to optimize I/O and reduce the number of read operations made a big difference when choosing the type of storage for this pipeline because it enabled a cheaper option to win overall. When scaling up to full genomes run in parallel, those cost savings really start to matter. Although a lot of effort was put into tuning the pipelines and making them portable and easy to run, there were also a lot of easy wins that could only have been identified by measuring with the right tools.

All the tests for the storage sizing work cost less than $20 and took two days, even though the team wasn't familiar with the pipeline before work started. This work saved 10% of the cost of running the pipeline on the cloud by making the right storage choice. It is very easy and low-cost to run the experiment, so it is practical to rerun the experiment across different storage options regularly to ensure that the architecture still represents the best value.

As the Breeze and Mistral tools showed, improving runtime doesn't have to require extensive rewrites. Knowing where to look is key.

