Architecture of processing and analysis system for big astronomical data

I.Yu.Kolosov, S.V.Gerasimov, A.V.Meshcheryakov *Faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University, Moscow, Russia*

There is need for computing solutions that are capable of processing large amounts of data produced by modern sky surveys and can span multiple processing steps. In this work, we explore the use of big data analytics technologies deployed in the cloud for processing of astronomical data. We have applied Hadoop and Spark to the task of co-adding a large volume of astronomical images. We compared the overhead presented by these frameworks and execution time on different workloads. From our experiments, we conclude that

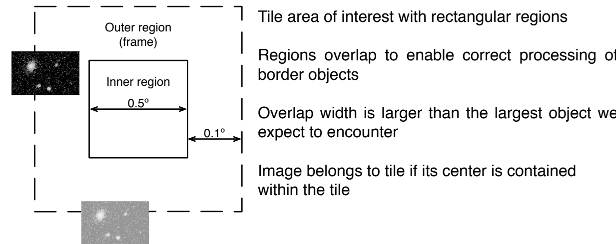
a) Performance of both frameworks is generally on par
b) Spark is faster on the workloads we used for testing. On the other hand, its configuration is less flexible than that of Hadoop, which can lead to performance problems.
c) The approach to distributed image coaddition that we used requires disk space for storing intermediate output (approximately 2 times the input size). Thus, it might be necessary to split large workloads into several jobs.

Technologies used

We used **SWarp** by Emmanuel Bertin to do image reprojection, background subtraction and coaddition.

- **Node configuration**: 12 D12 worker nodes. (28 GB RAM, 8 cores, 200 GB SSD)
- **Storage**: Azure blob storage acessed via HDFS interface. **No data locality**

Computational strategy

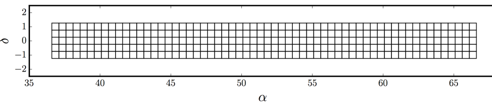


Tile area of interest with rectangular regions

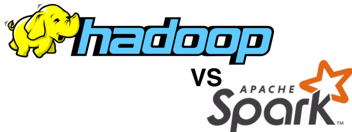
Regions overlap to enable correct processing of border objects

Overlap width is larger than the largest object we expect to encounter

Image belongs to tile if its center is contained within the tile



We used ~900 GB of Stripe82 r-band data with tiles arranged as shown on the graph



Both frameworks have the following features:

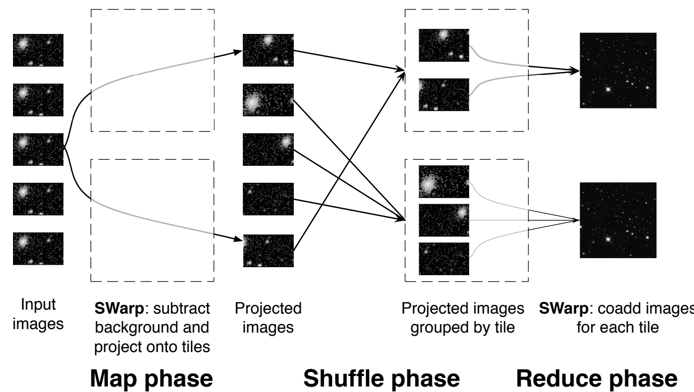
- Fault-tolerance
- Ability to exploit data locality by moving computations to data
- Parallel I/O using Hadoop Distributed File System (HDFS)
- Supported by major cloud platforms

On top of that, Spark offers:

- Support for in-memory computations
- Flexible API that is better suited to chaining operations

We used **Java** and **Python** for our Hadoop and Spark image coaddition implementations respectively.

Distributed image coaddition with MapReduce



Cluster configuration

Hadoop

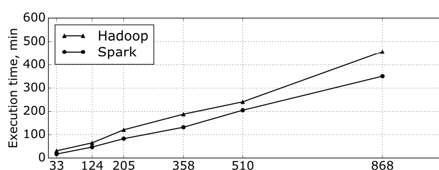
- Hadoop uses a dedicated container for each task
- Can use different container configurations for different kinds of tasks
- 3 GB Mapper containers
- 4 GB Reducer containers

Spark

- Spark starts a predefined number of executors and distributes tasks between them
- Executor is re-used for multiple tasks — can't use different configurations for different kinds of tasks
- 4736 MiB container for each executor
- 60 executors

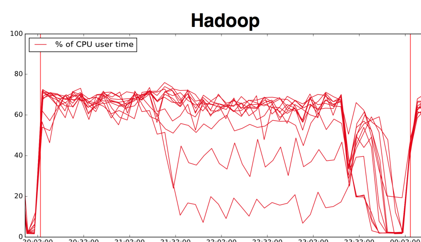
Evaluation

Execution time

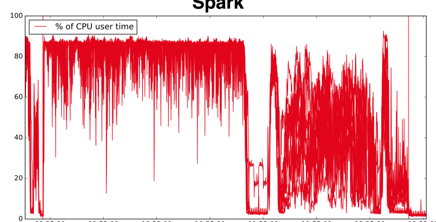


% of user CPU time

Hadoop



Spark



Disk space requirements

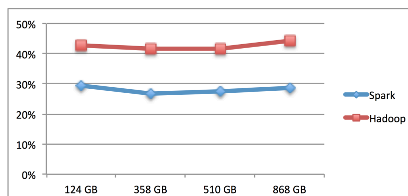
There may be multiple projections for each input image, so lots of disk space is needed for intermediate output

- Map output to be read by reduce tasks
- Files created by SWarp (deleted after it finishes)

We had to split jobs in half for bigger experiments (868 GB)

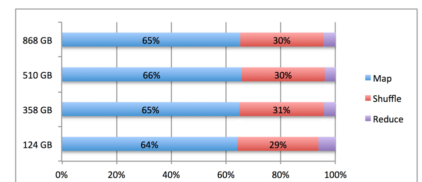
Framework overhead

We measured overhead by comparing total time across all tasks to time spent executing user code.

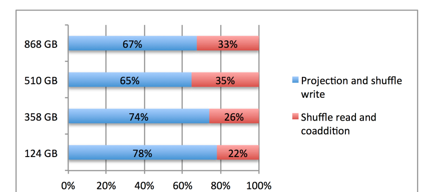


Aggregate time by stage

Hadoop



Spark



Conclusions

We applied Hadoop and Spark to image coaddition and compared the performance of the two frameworks. From our research, we conclude that Spark is faster and has less overhead. Spark also has more flexible API and easier configuration. The downside of Spark is that it doesn't let the user specify different executor configurations for different execution stages because the stages are defined by Spark at runtime. This can make configuring executors harder when dealing with heterogeneous tasks, as in our case where projection tasks consume more CPU time and coaddition tasks consume more memory.

Acknowledgements

This work is supported by Russian Foundation for Basic Research grants 14-22-03111-ofi-m and 14-22-03111-ofi-m. We also thank Microsoft Azure for Research for providing us with computing resources.