Contract Research Organization
for Data Science

Centre Régional Informatique et d'Applications
Numériques de Normandie

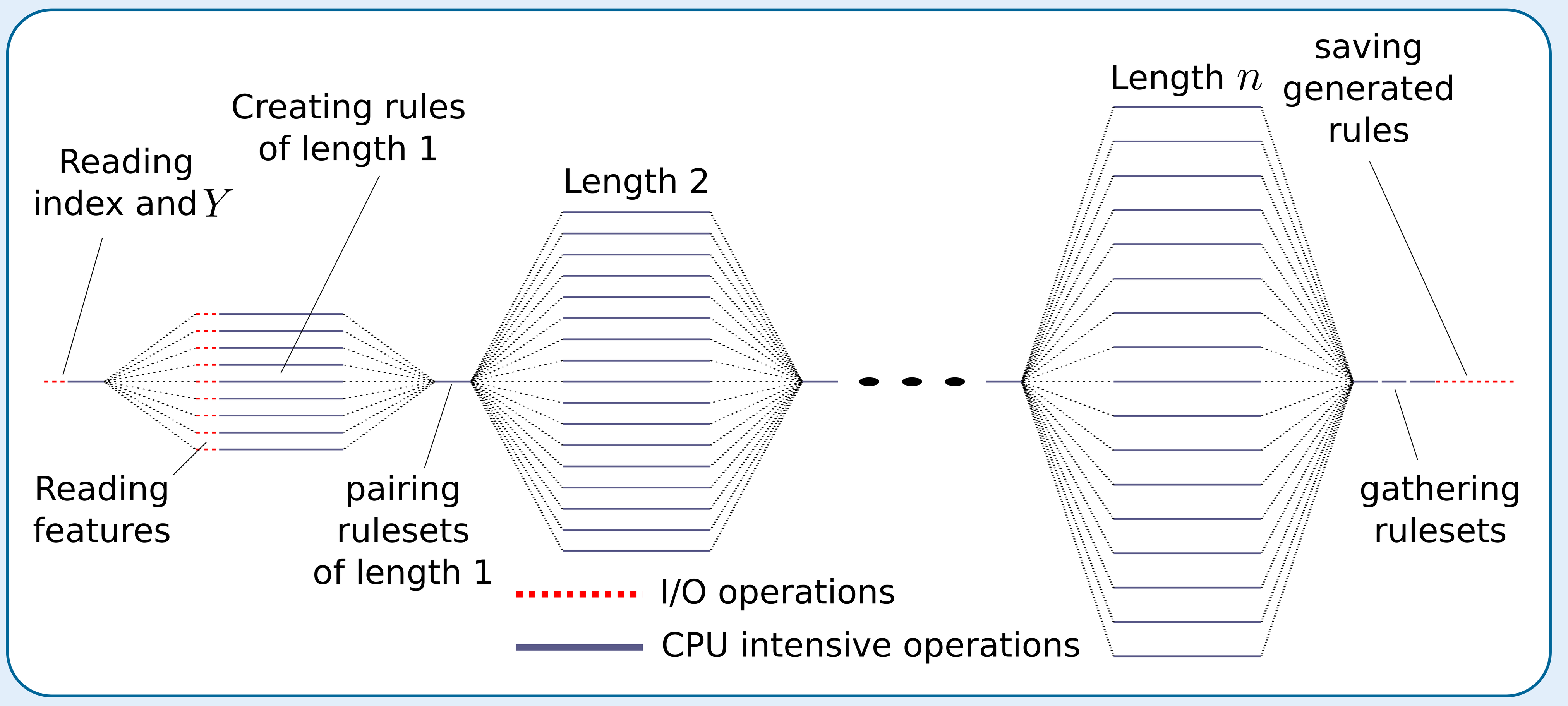
Acceleration of a learning algorithm in Python on a HPC cluster using *mpi4py*

Advestis' algorithm based on systematic exploration produces rules for various use cases, including investment recommendations. Advestis collaborated with the CRIANN to refactor and run the code on HPC infrastructures. The first goal is to see whether a very high number of cores and a better memory management could allow for a deeper exploration of the data compared to a local machine, where the hardware limitations prevent the exploration of all rules, hopefully resulting in better predictions. The first goal is to compare the speed and cost of running the code on a private cloud provider, here Google Cloud Plateform (GCP), with the public cluster "Myria", managed by the CRIANN. We found that an increase in the number of explored rules results in a net increase in the predictive performance of the final ruleset, and that Myria is twice faster and cheaper than GCP.

Problematic

We aim at predicting a random variable $Y \in \mathbb{R}$, a financial return, given a random vector $X \in \mathbb{R}^d$, made, in this example, of financial and extra-financial scores.
The learning sample is composed of n pairs of observations. X and Y are indexed by a multi-index with two levels : Date $\times$ Stock.
The learning algorithm is rule-based. A rule of length k is a If-Then statement on k features, predicting a value for Y when true.
To allow better interpretability and simpler computation, X is discretized in m bins. The algorithm generates all rules of lengths 1 to l and runs in $\mathcal{O}(n(dm)^l)$ time. The number of rules thus generated from our dataset (10 years of data on 200 stocks and 1300 features) with m=5 and l=2 is $\sim 10^9$. With 1s per rule, this is ~30 years. Two things are used to reduce this run time: Thresholds and candidates. Thresholds, that filter out bad rules on the fly, are chosen based on mathematical and application-dependent criteria. But keeping only the N best candidate rules for each length k is purely artificial, to allow the code to run on a desktop computer. The parallelisation process and run on HPC infrastructures aim at using thresholds only (giving ~120k rules) to explore more rules and enhance the performance. A rule can be defined by its activation vector of length , containing 1 when the rule is true and 0 when it is not. Doing so saves a lot of time, but those vectors can not always all hold in memory depending on the run configuration.

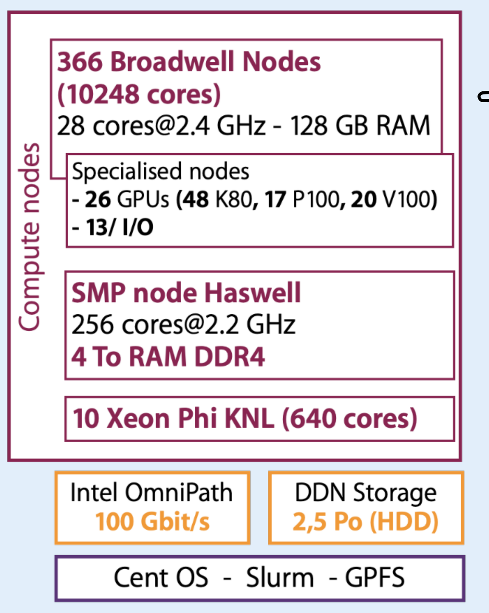
Algorithmic structure



In the algorithm, we define a ruleset as the set of all rules of same length using the same feature(s). This definition allows for a natural choice of strategy: one process per ruleset. The figure shows the parallelisation of the algorithm.
One can see that there are bottlenecks: we need to gather all rules of length n-1 to generate rules of length n, which are all the possible combinations of rules of length n-1 and 1.
Those bottlenecks could be bypassed by starting the computation of a ruleset of length n as soon as two rulesets of length n-1 are available, but that would imply a lot of refactor. And as one can see on the graph below, the code is already 96% parallel, so the speed increase would not be much.

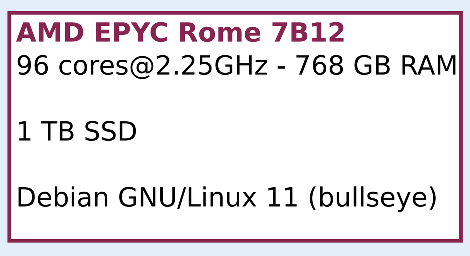
Infrastructure

Supercomputer Myria (CRIANN)



Cost evaluation:
CPU use $\times$ cost/hour.core

Google Cloud VM



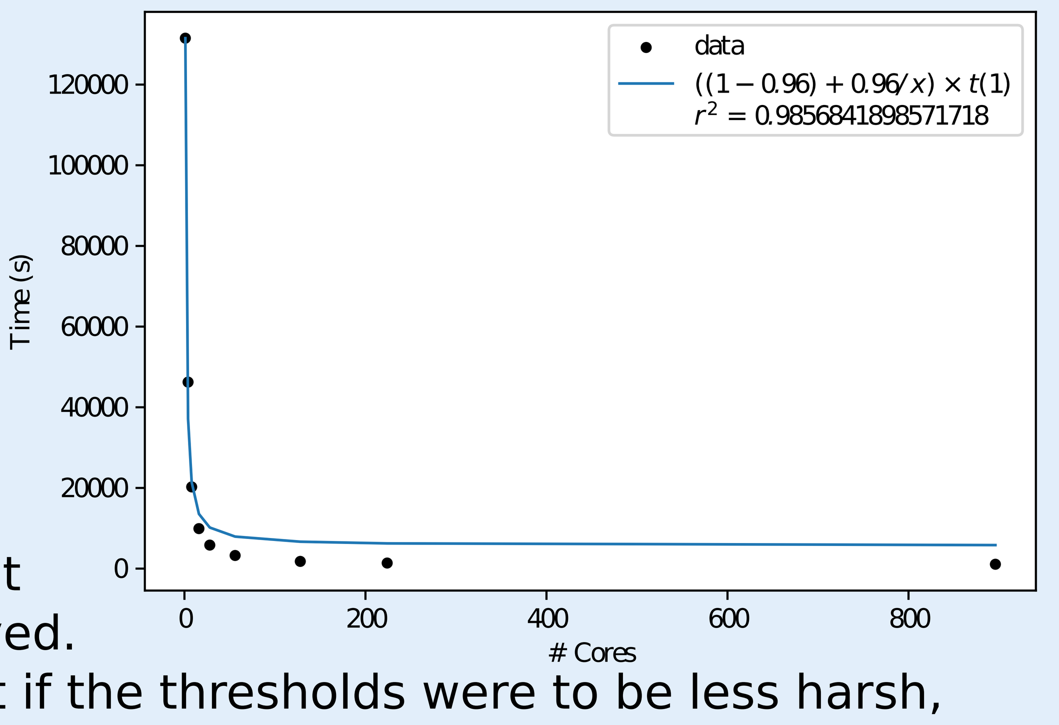
Cost evaluation:
Biling comparison between the beginning and the end of a run.

Use case

The use case, which was quite small, could hold all the activation vectors in RAM, but we still ran with activation vector on disk too, for the sake of comparison.

Run time vs #Cores on Myria

This graph shows how much the algorithm is parallel using Amdahl's law fitted on the run time on Myria from 1 to 896 cores. We deduced a high value of 96% of parallelisation, and see that above 100 cores (4 nodes), no significant increase in run time is achieved.



One should keep in mind that if the thresholds were to be less harsh, this number could greatly increase, because individual rulesets would generate more rules and thus take more time.

Cost comparison between GCP and Myria

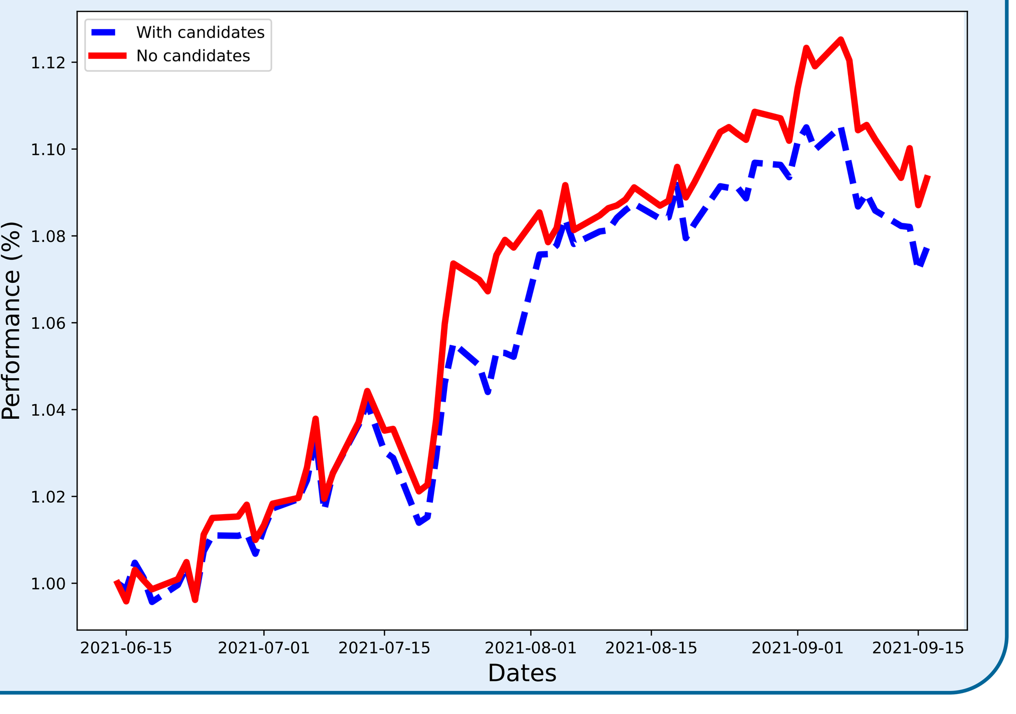
The table shows the run times and relative prices of the algorithm on Myria and on GCP, with 96 cores, with activation vectors in RAM or on SSD.

96 Cores	Time (s)	Price (AU)
Myria (AV in RAM)	975 (100 %)	1
Myria (AV on HDD)	1422 (143 %)	1.52
GCP (AV in RAM)	2785 (286 %)	2.76
GCP (AV on SSD)	3004 (308 %)	3.15

The price of writing activation vectors to disk is a slight increase in run time due to additional I/O. We can also see from the table than the Myria cluster is both cheaper and faster than the GCP VM.

Increase in predictive performances

The graph compares the predictive performance of the algorithm with and without candidates. Running without candidates on a local computer would require more than a day and could result in memory errors and crashes. We see a net gain by exploring more rules by using no candidates.



Philippe COTTE - Advestis, Benoist GASTON - CRIANN | JCAD 2022 | 10 - 12 octobre | Dijon