Systems for Data Science

CS-449

CS-449 Project Milestone 3: Optimizing, Scaling, and Economics

Author:
Kamil Seghrouchni

Professors:
Erick Lavoie
Athanasios Xygkis

Spring Semester 2021

Contents

3 Optimizing with Breeze, a Linear Algebra Library

3.1 Overview of some Breeze Operations

3.2 Questions

3.2.1 Predictor of Milestone 2 using the Breeze library

Using Breeze library, the predictor of the previous milestone was optimized in term of execution time, the same values for the mean absolute error were found and reported in the table bellow.

Table 1: mean absolute error for cosine based prediction

		k=100	k=200
MAE	0	0.7563	0.7485

3.2.2 Computing time for all k-nearest neighbours :

After optimizing with Breeze library, the computation time reported in table 3 were obtained for similarities execution averaged over 5 runs.

Table 2: Computation time for k-nearest neighbors (in seconds)

	min(s)	max(s)	average(s)	stddev(s)
K-nearest neighbors	0.3527	0.4974	0.3958	0.0519

3.2.3 Time for computing all 20,000 predictions:

After optimizing with Breeze library, the computation time reported in table 3 were obtained for all 20'000 prediction sexecution averaged over 5 runs .

Table 3: Computation time for all 20'000 predictions (in seconds) (on local machine)

	min(s)	max(s)	average(s)	stddev(s)
All predictions	3.0138	3.2491	3.1239	0.0783

3.2.4 Comparing computing time with previous milestone :

Table 5 reports the time for computing similarities measured in seconds in the previous milestone. Note : optimization in previous milestone was thought to compute only the superior triangle of the similarity matrix, meaning half of the similarities computed in the current milestone.

Table 4: Duration in seconds for computing all similarities in milestone 2 (on local machine)

Time duration	min(s)	max(s)	average(s)	stddev(s)
DurationInMicrosecForComputingSimilarities	12.176	13.19	12.634	0.424

Table 5: Comparing duration in seconds to compute all similarities between the two milestones

Time duration	min(s)	max(s)	average(s)	stddev(s)
Time old/ Time new	34.524	26.518	31.918	8.179

As one can observe, the speedup was of on average roughly 32 for computing similarities between the two milestones. The reason behind such speedup is that, in previous milestone, data structures like map had to be employed for similarity computations as nested spark RDD was not a solution. The cost of such conversion and implementation choice was higher as the number of similarity computations increased. Here, the use of mature linear algebra libraries with efficient multi-core vectorized implementations, allowing storing the computations in cache memory, with fast retrieval renders the overall computations much less slower.

4 Scaling with Spark

4.1 Questions

4.1.1 Testing spark implementation:

Distributed version of the program gave the following result for k=200 on ra.test data set (see table 6)

Table 6: Mean absolute error for k=200, ra.test dataset:

		k=200
MAE	0	0.7346

4.1.2 Comparing execution time :

Running manually the scaled version of our implementation on a single executor multiple times (cf table 7) allowed us to compare the results with the optimized implementation (cf table 1) on the same data set (ra.test). In it is clear that these results are not similar. The execution time for computing the k-nearest neighbors in the scaled version is nearly four time the one of the optimized version for a single executor. In turn, this high overhead impacts the overall prediction times (which takes into account the similarity computation).

Table 7: Execution time for computing k-nearest neighbors and predictions on ra.test with scaled version, using single executor (in seconds)

.

	min(s)	max(s)	average(s)	stddev(s)
k-nearest neighbors	66.103	75.627	71.274	3.161
prediction	124.378	140.531	130.853	5.5454

Table 8: Execution time for computing k-nearest neighbors and predictions on ra.test with optimized version (in seconds)

.

	min(s)	max(s)	average(s)	stddev(s)
k-nearest neighbors	15.5037	16.6584	16.1046	0.4201
predictions	61.4959	65.1065	63.2359	1.3036

4.1.3 Comparing execution time varying executors :

Looking at figure 1 and 2, it the speed up in execution time with increasing number of executors can be clearly observed. Aside form the higher magnitude increase doubling the number of executors (from to 2)

Table 9: Execution time for computing k-nearest neighbors on ra.test with scaled version, varying number of executors (in seconds)

.

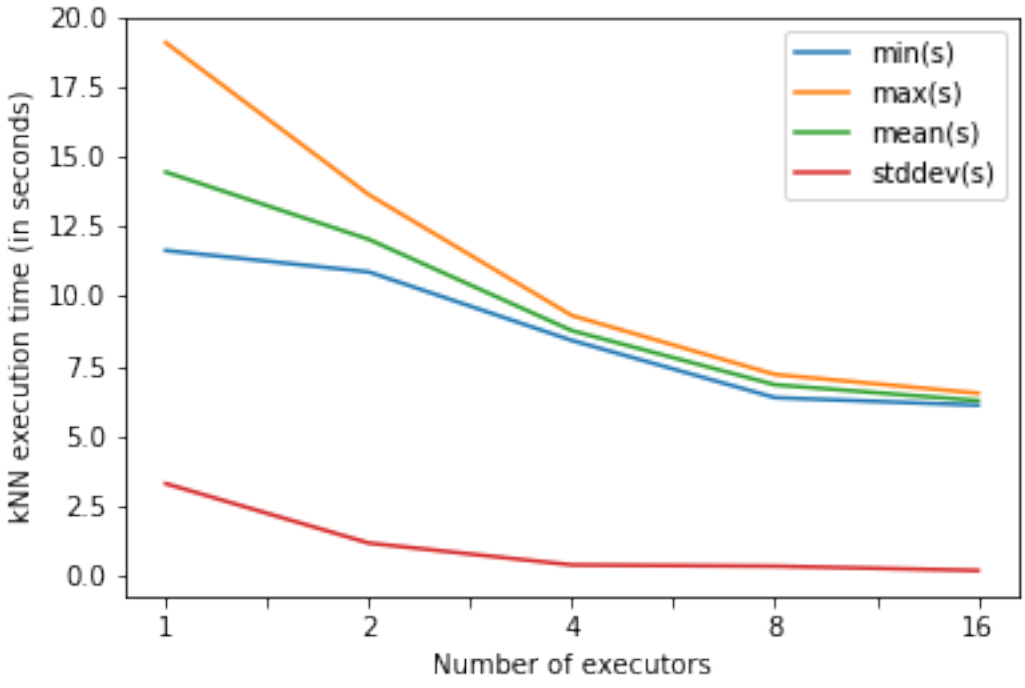
	min(s)	max(s)	mean(s)	stddev(s)
1	11.6444	19.0839	14.4522	3.2996
2	10.8765	13.6430	12.0434	1.1702
4	8.4246	9.3157	8.7747	0.3880
8	6.3762	7.2050	6.8370	0.3447
16	6.1072	6.5357	6.2696	0.1897

Table 10: Execution time for computing predictions on ra.test with scaled version, varying number of executors (in seconds)

.

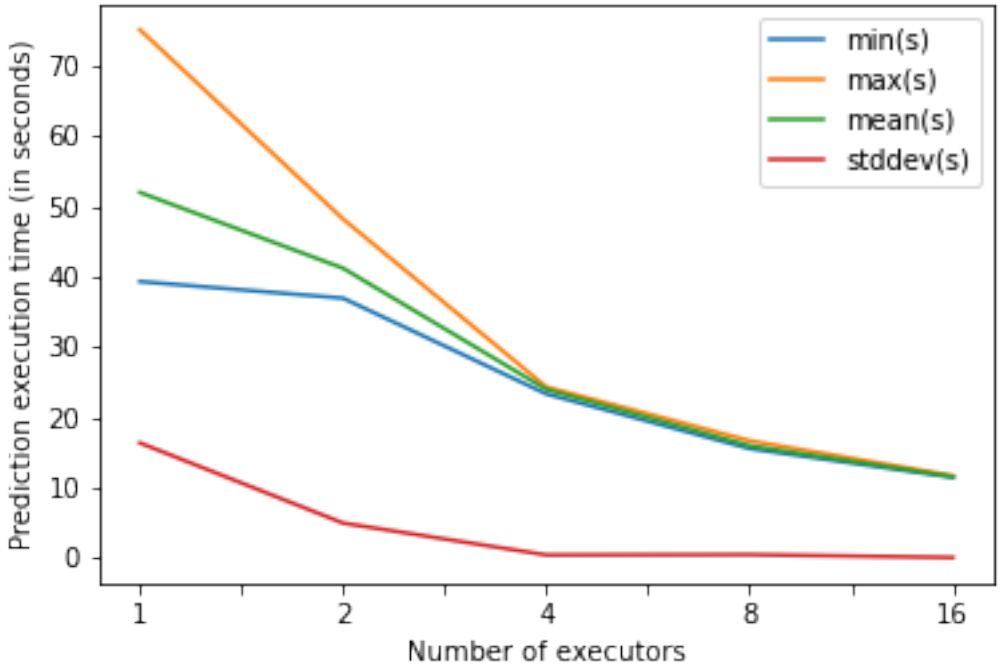
	min(s)	max(s)	mean(s)	stddev(s)
1	39.3395	75.1563	52.0100	16.3914
2	36.9709	48.2282	41.2319	4.9862
4	23.3515	24.3339	23.9976	0.4570
8	15.5807	16.6987	15.9968	0.4991
16	11.4901	11.7272	11.5979	0.0980

Figure 1: Visualizing the execution time for computing k-nearest neighbors on ra.test with scaled version, varying number of executors (in seconds)



.

Figure 2: Visualizing the execution time for computing predictions on ra.test with scaled version, varying number of executors (in seconds)



5 Economics

5.1 Questions

5.1.1 Icc M7 Profitability

In order to define the number of days making buying an ICC M7 server profitable, we divide the purchasing price by the renting price per day. To obtain the number of years, we just divide by the number of days in year (365).

$$\frac{\text{purchasing}_{\text{ICCM7}}}{\text{renting}_{\text{ICCM7}}} = 1715.6863 \text{ days} = 4.7005 \text{ years} \tag{1}$$

5.1.2 Container vs Icc M7

To determine the cost of a container with the same capabilities as an ICC M7 server, we multiply the renting price per RAM and CPU with the equivalent quantity factor. Then we sum them up as in the following :

$$\text{RamIccM7} * \text{ramRenting} + \text{CPUIccM7} * \text{CPURenting} = 24 \cdot 64 \cdot 0.012 + 2 \cdot 14 \cdot 0.088 = 20.896 \tag{2}$$

This leads to a ratio of :

$$\frac{\text{renting}_{\text{ICCM7}}}{\text{renting}_{\text{container}}} = \frac{20.40}{20.896} = 0.9762 \tag{3}$$

The difference between the two renting price being negligible (order of 5%), one can says that there are no difference between the two from that point of view.

5.1.3 Container vs 4 Raspberry PI

To determine the cost of a container with the same capabilities as 4 Raspberry PI, we multiply the renting price of the container per RAM and CPU with the equivalent quantity factor. Knowing that 1 CPU is equivalent to 4 RPIs, we then we sum them up as in the following :

$$\text{Ram4RPI} * \text{ramRenting} + \text{CPU4RPI} * \text{CPURenting} = 8 \cdot 4 \cdot 0.012 + 1 \cdot 0.088$$

(4)

Then since the renting price for RPIs is withing a given range, we compute the RPI - container for the two bounds :

$$\max\left(\frac{\text{renting}_{\text{4RPis}}}{\text{renting}_{\text{container}}}\right) = \frac{0.054 \cdot 4}{8 \cdot 4 \cdot 0.012 + 1 \cdot 0.088128} = 0.4576 \tag{5}$$
$$\min\left(\frac{\text{renting}_{\text{4RPis}}}{\text{renting}_{\text{container}}}\right) = \frac{0.0108 \cdot 4}{8 \cdot 4 \cdot 0.012 + 1 \cdot 0.088128} = 0.0915$$

Once again the container is not cheaper than the equivalent 4RPI service.

5.1.4 Container and 4 Raspberry PI Profitability

In order the determine the number of days upon which it is more profitable to buy and run 4RPIs than renting a container, we apply the simple following formula :

$$\begin{aligned} \text{days} \cdot \text{renting}_{\text{container}} &> 4 \cdot \text{purchase}_{\text{RPi}} + \text{days}\,(4 \cdot \text{renting}_{\text{RPi}}) \\ \text{days} &> \frac{4 \cdot \text{purchase}_{\text{RPi}}}{r_{\text{container}} - \text{renting}_{\text{RPi}}} \end{aligned} \tag{6}$$

$$\tag{7}$$

Using the minimum and maximum renting price, this yields respectively to 885 and 1482 days.

5.1.5 ICC M7 vs Raspberry PI

$$\frac{\text{purchasing}_{\text{ICCM7}}}{\text{purchasing}_{\text{RPIs}}} = \frac{35000}{94.83} = 369 \tag{8}$$

Then, to determine the throughput and ram ratio we do the following :

$$\text{Throughput ratio} = \frac{\#\ \text{RPI} \cdot \text{CPU}_{\text{RPI}}}{\text{CPU}_{\text{ICCm7}}} = \frac{369 \cdot 0.25}{2 \cdot 14} \tag{9}$$

$$\text{Ram ratio} = \frac{\#\ \text{RPI} \cdot \text{RAM}_{\text{RPI}}}{\text{RAM}_{\text{ICCm7}}} = \frac{369 \cdot 8}{24 \cdot 64} \tag{10}$$

5.1.6 User storage

In order to determine the number of users that can be held per GB, RPI, ICCM7, we must first determine the storage load it represents. To store a user, we must store his ratings (100 of them) as well as his similarities (200 of them). More over, for the system to be running, we shall not occupy more than 50% of the storage capacities. Thus, given the memory load of ratings, similarities and the storage capacity of each options, we did the following:

$$\text{number of user} = \frac{\text{memory option}(bytes)}{\text{bytes per rating} \cdot 200 + \text{bytes per simi} \cdot 100} = \frac{\text{memory option}(bytes)}{900} \tag{11}$$

This yields to 555555 users per GB, 444444 per RPI and 8.533E8 per ICCM7.

5.1.7 Option choice

The choice of option will largely depends on the usage and durability of my application. In fact, given large data set having 25 million ratings per user, all options have the means to store roughly 250'000 users. Yet the ICCM7 would represent the safest option in terms of higher loads incoming but the centralized aspect of this solution makes it a single point of failure and can jeopardize the whole service. On the other hand, buying RPI would rapidly be profitable compared to renting cloud services. This argument is even more relevant if we plan on using this application for the years to come. But that same argument involves a higher human implication in running and maintaining the hardware. My choice would go for the cloud option, as the scalability and adaptability offered by the renting options are not permitted by the two others. More over, as we are plan on recommending movies, our application must adapt to the ever changing will and likes of users and a data turnover of 2-3 years does not seem dangerous. It is also a safer option empowering us with much more flexibility when rented.