CS-449 Project Milestone 2 : Optimizing, Scaling, and Economics

Ecole Polytechnique Fédérale de Lausanne

2021-2022

Braz Lucas - Durand Pierre-Alain

SCIPER : 343141 - 344313

Date : 20/05/2022

Table des matières

1 Our setup

We used the same machine for all the Project, including Milestone 1. Our setup specification :

CPU Model	CPU speed	RAM	OS	Scala version	JVM version
Intel i5-9300H	2.40GHz	8Gio	Windows 10	Scala 3.1.1	JVM 1.8.0_292 64-Bit

Our code implementations for each part can be found in the *src/* folder.

2 BR - Optimizing

For this section, to get the prediction kNN model we use this code line :

```
val (kNN_model, suvPerUser) = kNN_builder(train, conf_k)
```

Then, as for two next sections, the similarity coefficient go in the function :

```
val suv = addAutoSimilarityZero(suvPerUser)
```

Which return the similarity coefficient with the right coefficient (0.0) for auto-similarity.
And we compute the MAE with :

```
val kNN_MAE = computeMAE(test, kNN_model)
```

2.1 BR.1

Our result, using our new optimized implementation, with k=10 :

```
"BR.1": {
   "1.k10u1v1": 0,
   "2.k10u1v864": 0.24232304952129619,
   "3.k10u1v886": 0,
   "4.PredUser1Item1": 4.319093503763853,
   "5.PredUser327Item2": 2.6994178006921192,
   "6.Mae": 0.8287277961963542
}
```

These results are consistent with those found in milestone 1.

2.2 BR.2

Our timing results for $k = 300$:

```
"BR.2": {
   "average (ms)": 967.6141333333334,
   "stddev (ms)": 244.57493440683757
}
```

With our last, slow, implementation, we got, with same k and same dataset : 140778.73679999998 ms.
The speedup is therefore a ratio of $\frac{140778}{967} = 145.5$. Our implementation was probably really not optimized, which should be taken into account.

3 EK - Distributed Exact

For this section, to get the prediction kNN model we use this code line :

```
val (kNN_model, suvPerUser) = kNN_builder_parallel(train, conf_k, sc)
```

3.1 EK.1

Our result, using our new optimized and distributed implementation, with k=10 :


```
"EK.1": {
   "1.knn_u1v1": 0,
   "2.knn_u1v864": 0.24232304952129619,
   "3.knn_u1v886": 0,
   "4.PredUser1Item1": 4.319093503763853,
   "5.PredUser327Item2": 2.6994178006921192,
   "6.Mae": 0.8287277961963542
}
```


These results are still consistent with those found in BR part.

3.2 EK.2

Our timing results for $k = 300$ with 1 worker :


```
"EK.2": {
   "average (ms)": 2699.582766666667,
   "stddev (ms)": 444.9510487546343
}
```


Our timing results for $k = 300$ with 2 workers :


```
"EK.2": {
   "average (ms)": 2105.1248,
   "stddev (ms)": 391.2953743800796
}
```


Our timing results for $k = 300$ with 4 workers :


```
"EK.2": {
      "average (ms)": 1542.2791666666665,
      "stddev (ms)": 359.66786222413907
}
```


Doubling the number of workers seems to decrease linearly the computation time. The improvement is therefore logarithmic.
We don't observe a speedup compared to the BR part, which is quite surprising. Maybe there is a time required to parallelize which make the parallelization usefull for longer computations (on 1M dataset for example).

4 AK - Distributed Approximate

For this section, to get the prediction kNN model we use this code line :

```
val (kNN_model, suvPerUser) = kNN_builder_parallel_approx(train, conf_k, sc,
    partitionedUsers)
```

4.1 AK.1

Our result, using our new optimized, distributed and approximate implementation, with k=10, 10 partitions and 2 replications :


```
"AK.1": {
   "knn_u1v1": 0,
   "knn_u1v864": 0,
   "knn_u1v344": 0.23659364388510976,
```

```
    "knn_u1v16": 0,
    "knn_u1v334": 0.19282239907090362,
    "knn_u1v2": 0
}
```


4.2 AK.2

Our MAE, using our new optimized, distributed and approximate implementation, with k=300, 10 partitions and 2 replications :


```
"AK.2": {
    "mae": 0.7584399718717879
}
```


By varying the level of replication, but with k=300 and 10 partitions, we obtain the following table giving the MAE according to the replication :

Replication	1	2	3	4	6	8
MAE	0.8056	0.7563	0.7457	0.7410	0.7391	0.7391

The minimum level of replication such that MAE is still lower than the baseline predictor of Milestone 1 (MAE of 0.7604), with k=300 and 10 partitions is : 2.

This reduce the number of similarity compared to an exact k-NN. Indeed by splitting we calculate a ratio of $\frac{N_{replications}}{N_{partitions}}$ of the exact k-NN required computations.

4.3 AK.3

Our timing results for $k = 300$ and 8 partitions with a replication factor of 1 and 1 worker :


```
"AK.3": {
    "average (ms)": 1311.5390333333335,
    "stddev (ms)": 360.8011644630538
}
```


Our timing results for $k = 300$ and 8 partitions with a replication factor of 1 and 2 workers :


```
"AK.3": {
    "average (ms)": 1164.2723333333333,
    "stddev (ms)": 350.42248729324183
}
```


Our timing results for $k = 300$ and 8 partitions with a replication factor of 1 and 4 workers :


```
"AK.3": {
    "average (ms)": 1104.0921999999998,
    "stddev (ms)": 439.6526489481668
}
```


We clearly see a speedup compared to the distributed exact version.

5 E - Economics

5.1 E.1


```
"E.1": {
    "MinRentingDays": 1893
}
```

The minimum number of days of renting to make buying the ICC.M7 less expensive is 1893 days

5.2 E.2


```
"E.2": {
   "ContainerDailyCost": 0.540864,
   "4RPisDailyCostIdle": 0.072,
   "4RPisDailyCostComputing": 0.096,
   "MinRentingDaysIdleRPiPower": 1507,
   "MinRentingDaysComputingRPiPower": 1130
}
```


5.3 E.3


```
"E.3": {
   "NbRPisEqBuyingICCM7": 355,
   "RatioRAMRPisVsICCM7": 0.5408450704225352,
   "RatioComputeRPisVsICCM7": 0.034178403755868544
}
```