Ειδικά Θέματα Βάσεων Δεδομένων
Skyline Project Report

Michail Theologitis

April, 2023

The three algorithms, namely, *MR-DIM*, *MR-GRID* and *MR-ANGLE* are configured by a hyper-parameter $\mathtt{p}$; it defines the number of partitions (local skylines) to be computed and hence the parallelism we can achieve. Before showing the results it is important to explain what that hyper-parameter actually means for each algorithm.

- MR-DIM : We partition the first dimension in $\mathtt{p}$ disjoint partitions. Hence, we compute $\mathtt{p}$ number of local skylines and the most CPU cores we can utilize is $\mathtt{p}$.

- MR-GRID : We partition each dimension in $\mathtt{p}$ disjoint partitions. Hence, the total number of partitions is $\mathtt{p}^d$ where d is the dimension of our data. As the paper notes, we have a few partitions that are dominated from the start, that is, if a single data-point exists in one partition the other can safely be thrown away. We actually know beforehand the dominated partitions in $\mathbb{R}^d$. We compute exactly $\mathtt{p}^d - (\mathtt{p} - 1)^d$ local skylines (requires some thought to see why this is the case). Thus, the most CPU cores we can utilize is exactly $\mathtt{p}^d - (\mathtt{p} - 1)^d$.

- MR-ANGLE : We partition each angular coordinate dimension in $\mathtt{p}$ disjoint partitions. Hence, we compute exactly $\mathtt{p}^{d-1}$ local skylines (there are $d-1$ angular coordinates). Thus, the most CPU cores we can utilize is exactly $\mathtt{p}^{d-1}$.

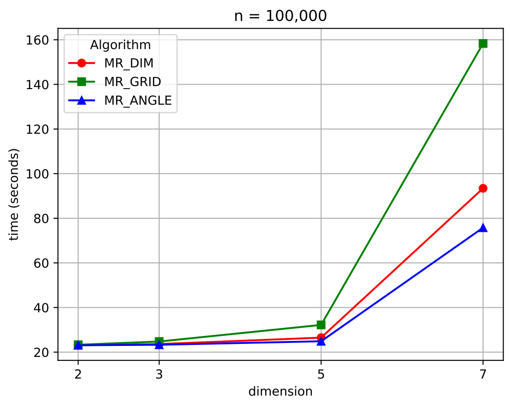
Even if obvious, it is important to note the architecture of the skyline computation. The algorithms are meant to be implemented in the *MapReduce* model, where the map stage consists of computing the local skylines and the reduce stage combines those local skylines to find the single global skyline. At first glance, it would be logical to assume that increasing the partitions of the map stage would decrease the overall computation time. But, as we will see this is not the case. The reason is simple, as the dimension of the data points d increases, the size of the local skylines increases tremendously and the *driver* that runs the reduce stage is unable to combine such huge amounts of data.

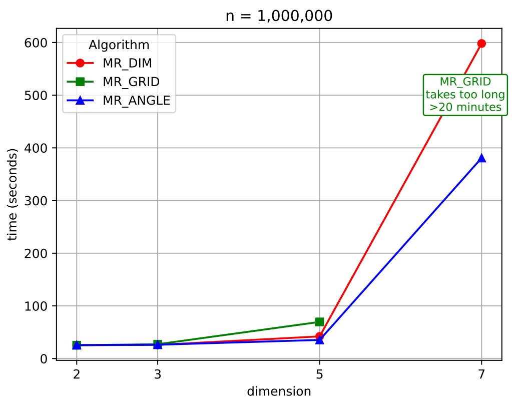
Since the MR-DIM and MR-GRID algorithms take no precaution against this, they are hopeless as d increases (for $d \geq 10$). We will comment more on this after the plots.

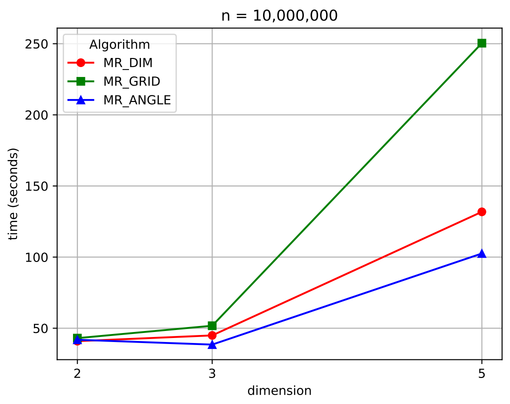
The tests were run in a cluster managed by YARN, with the driver program running in client mode on the local machine. This setup was chosen for development and testing purposes. We had three workers with 4 CPU cores and 4 GB RAM each.

d	MR-DIM	MR-GRID	MR-ANGLE
2	12	7	12
3	12	4	3
5	12	2	2
7	12	2	2

Table 1: Optimal Hyperparameter p for our environment







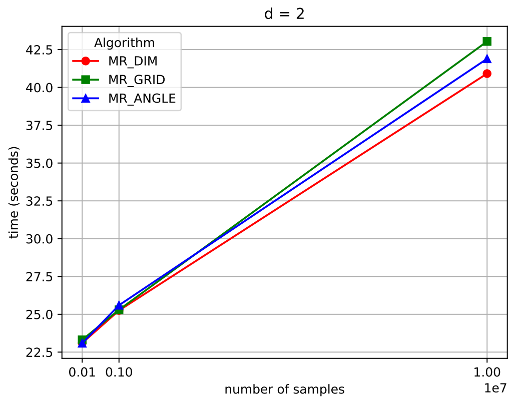
We have to note that in the $n = 10,000,000$ case when $d = 7$ the MR-ANGLE test took 47 minutes to complete so there was no point trying the other two algorithms (the plot would look very messy too).

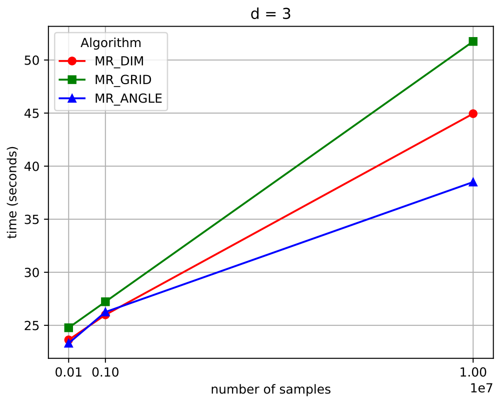
Based on the plots above we could claim that as d increases the MR-ANGLE algorithm is twice as fast as the other two. This conclusion is far from the truth. What the plots fail to show us is how much of the total time is due to computation of local skylines (with parallelism - map stage) and how much is due to the single driver trying to combine the local skylines (reduce stage). Sadly, I do not have the relevant plots that show this (required a lot of time with Spark history server set-up on Google Cloud or Grid etc.), but I can offer insight from older testing. For MR-DIM and MR-GRID as d increases above 7 the overwhelming percentage of computation time is due to the driver trying to combine the huge local skylines. In fact, with the suggested implementation, MR-GRID will never finish (remember that the map stage produces $\mathtt{p}^d - (\mathtt{p} - 1)^d$ local skylines!).

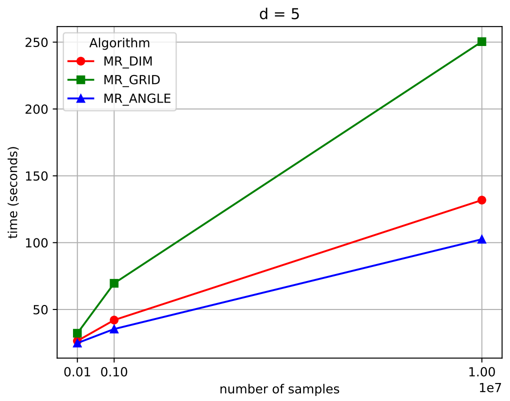
With this information in mind, we can articulate the problem that the MR-GRID and MR-DIM algorithms have: As we try to scale (to deal with larger data sets) we logically increase the partitions to achieve higher parallelism - add nodes/executors - subsequently, though, the number of local skylines increases and it becomes impossible for the driver to combine them. We simply can not scale MR-DIM and MR-GRID. Increasing parallelism actually hurts these two algorithms which is something totally counterproductive.

The idea for a solution is simple: find a way to increase parallelism (partitions - local skylines) in order to deal with larger data but at the same time reduce the amount of points in each local skyline considerably so the driver is able to combine them. The MR-ANGLE algorithm does exactly that. The plots do not do justice to the algorithm (they do not show local/global skyline compuation times) so I will offer insight from older tests. For $d = 7, 8, 10$ one third of the time is spent on the driver (reduce stage) and most of the the time is spent on the computation of the local skylines. The MR-ANGLE algorithm is in fact scalable; as input data increases we can increase the partitions (parallelism) - add executors and nodes - with beneficial results. The current bottleneck of MR-ANGLE in our testing enviroment is the low parallelism our resources can achieve (12 total CPU cores) something that can be easily dealt with in a real-world scenario.

Lastly, we show the plots for fixed d. The results are somewhat expected since the real problem lies in the dimension of the input data.


d = 2
Algorithm
MR_DIM
MR_GRID
MR_ANGLE
time (seconds)
number of samples
1e7


d = 3
Algorithm
MR_DIM
MR_GRID
MR_ANGLE
time (seconds)
number of samples
1e7


d = 5
Algorithm
MR_DIM
MR_GRID
MR_ANGLE
time (seconds)
number of samples
1e7