timi vs Spark

THE ULTIMATE SHOWDOWN

Introduction - What's the objective ?

We'll compare the execution time for running common data transformations tasks for two data management tools: TIMi and Spark. We'll also review the strategy used in each tool to distribute the workload amongst many different servers (for distributed computation).

Why?

"Data scientists" are actually spending more than 80% of their time to do simple data management tasks (e.g. research, unification and data cleaning) (Source: IDG). This fact is also sometimes called the "*80/20 data science dilemma*" (google it!). Therefore, a fast and user-friendly solution for high-velocity data transformation is an important element for the comfort, the efficiency and the productivity of any data scientist. For example, if it were possible to reduce to only 8% these 80% of time devoted to data management, the productivity of your data scientists would be multiplied by a factor of three!

How?

We'll measure the execution time of a series of 22 data transformations that exactly replicates the 22 SQL queries included in the famous TPC-H benchmark. This workload is universally recognized as "interesting and representative" of the data transformations that are commonly used in the everyday life of all data scientists. We'll also briefly review the results obtained during the "Big Data Bench": This is another, more recent benchmark, with a broader scope than the TPC-H.

About the two contestants

Spark is a well-known platform that is used in many commercial product as the underlying computing engine for "big data" transformations. Spark is thought to be the fastest data transformation tool in the Hadoop ecosystem (e.g. Spark is recognized to be several order of magnitude faster than Hive). Spark is a distributed "in-memory" tool that will typically uses all the combined RAM memory of all the servers inside the PC cluster to performs the requested data transformations. For this benchmark, we coded the data transformations in Scala. Scala is the native, fastest language available on Spark. Spark is created using the Java language.

The data management tool included inside TIMi is named Anatella (v1.94). Anatella is created using the C and assembler language. Anatella is an out-of-memory tool. It means that the data that you can manipulate is not limited to your (small) RAM size.

The Setup

To run this benchmark, we used Spark v2.3 and Anatella v1.93 with all defaults settings. As hardware, we used the "*LDLC PC10 WANOMAN*". This is a standard configuration available on ldlc.com. The specs are:
CPU: Intel Core i7-8086K (4.0 GHz) . It's a 6 cores –12 threads CPU.
RAM: 16 GB (DDR4—3GHz—CL15)
Storage: Samsung 870 SSD - NVMe - 2TB
The total price of this server is less than 3K€.

The workload

We run the 22 queries from the TPC-H on 4 different database sizes:

Unit: millions of rows	1GB	10GB	100GB	1TB
#Customers	0.15	1.5	15	150
#Purchases	6	60	600	6000

The data is stored inside .parquet files (for Spark) and .gel files (for TIMi). All the files are stored on a SSD drive (i.e. storing data on HDFS causes a huge speed penalty for Spark). We executed all queries in a non-interactive session ("as if" the queries were running during the night). This makes a big difference for Anatella since Anatella possesses an "interactive" mode that allows near instantaneous computation of the most complex queries (thanks to a unique advanced data-cache system).

The results

The timing results are given in the Table 1 next page. In this table, some columns have been computed using the Amdahl's law. The Amdahl's law is illustrated in Figure 1 (bottom left of the page), it's based on the following simple principle: In every distributed process, there exists an "incompressible" time that is equals to "$s\,t_1$", with:

- "t_1" the total runtime on 1 CPU.
- "s" the percentage of the process that is not parallelizable/distribuable. It's also named "incompressible time". It's given in the column "R" in the table next page. Please note that, in the result table, The Spark incompressible time "s" is never below 20% .
- "$s\,t_1$" the initialization time in [sec]. It's given in column "N" in the table next page.

The Amdahl's law states the following:

$$Speedup \text{ on } n \text{ CPU} = \frac{Runtime \text{ on } 1 \text{ CPU}}{Runtime \text{ on } n \text{ CPU}} = \frac{t_1}{s t_1 + (1-s) t_1 / n} = \frac{1}{s + (1-s)/n} \quad \text{when } n \to \infty$$

The maximum speed-up that any distributed process can achieve (when $n \to \infty$) is thus:

$$MaximumSpeedUp = 1/s$$

Since "s" is never below 20%, the maximum speed-up reachable for Spark is $1/20\% = 5$
This means that it's enough to achieve a speed-up above 5 with Anatella (...and we have that: See column O: O is always >5) compared to Spark to get a tool that Spark will never beat whatever the data size or the size of the Spark Cluster. In other words: We observed that, for each TPC-H query, the total Anatella runtime is largely below the Spark incompressible time "s" (i.e. we always have column S<R or column G<N). This basically means Anatella already finished all computations since a long time while Spark is still busy trying to complete its initialization phase. This means that **one Anatella server is always faster than a Spark cluster whatever the size of the Cluster.** There is no doubt that, once the distributed phase (illustrated in green on Figure 1, on the left) is running, the Spark's engine speed will be almost impossible to beat by a single server. ...But

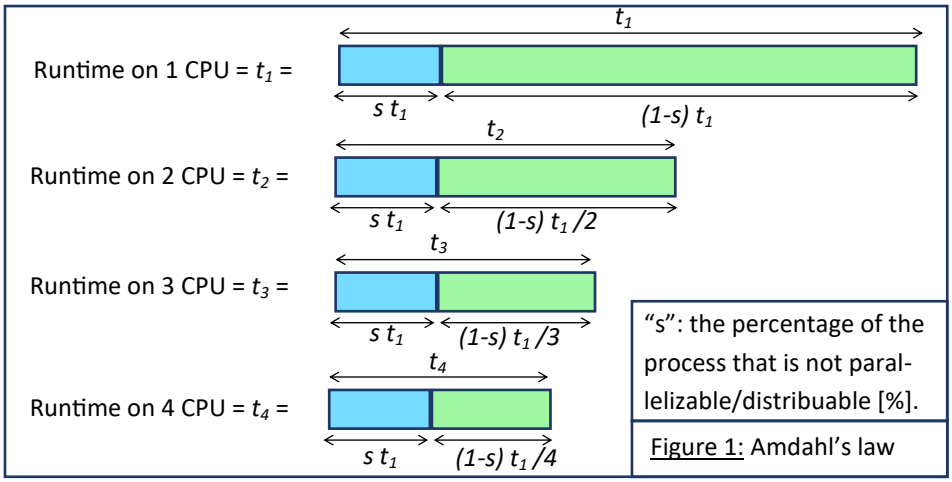

Figure 1: Amdahl's law

Table 1

	1 GB database			10 GB database			100 GB database													1 TB database	
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
TPC-H Query	Anatella runtime [sec]	spark 6CPU runtime [sec]	Anatella speed-up	Anatella runtime [sec]	spark 6CPU runtime [sec]	Anatella speed-up	Anatella runtime [sec]	spark 1CPU runtime [sec]	spark 2CPU runtime [sec]	spark 3CPU runtime [sec]	spark 4CPU runtime [sec]	spark 5CPU runtime [sec]	spark - 6CPU runtime [sec]	Spark incom-pressible time "s t_1" [sec]	Anatella Speedup vs Spark 1 CPU (=J/I)	Anatella Speedup vs Spark 6 CPU (=O/I)	Anatella Speedup vs Spark infinite CPU (=P/I)	Spark incom-pressible time "s" (=P/J) [%]	"Anatella Time" / "Spark Time 1CPU" (=I/P) [%]	Anatella runtime [sec]	Anatella RAM usage [MByte]
Q1	0.72	17	23	3.70	22	6	27.1	184	99	80	74	63	59	37.4	6.8	2.2	1.4	20.4 %	14.8 %	260	204
Q2	0.16	27	176	0.70	218	310	5.7	956	792	751	700	688	686	649.0	167.2	120.0	113.5	67.9 %	0.6 %	61.5	800
Q3	0.70	19	27	3.72	126	34	34.5	929	732	727	651	643	932	571.9	26.9	27.0	16.6	61.6 %	3.7 %	360	10053
Q4	0.72	20	28	3.70	37	10	33.7	830	436	410	349	350	738	229.5	24.6	21.9	6.8	27.7 %	4.1 %	337	160
Q5	0.70	160	228	4.70	234	50	43.7	2275	1208	1123	994	994	1516	673.6	52.0	34.7	15.4	29.6 %	1.9 %	509	2045
Q6	0.22	14	64	1.20	17	14	6.2	102	55	46	39	37	35	20.7	16.4	5.6	3.3	20.3 %	6.1 %	65.3	154
Q7	0.70	231	329	3.70	214	58	45.6	1113	955	879	852	831	810	761.7	24.4	17.8	16.7	68.4 %	4.1 %	760	8828
Q8	0.70	190	270	4.22	208	49	49.3	1621	1312	1266	1147	1131	1124	1009.3	32.9	22.8	20.5	62.3 %	3.0 %	511	1576
Q9	2.70	283	105	17.73	111	6	200.0	2059	2064	1524	1389	1348	1366	1227.4	10.3	6.8	6.1	59.6 %	9.7 %	2668	7392
Q10	1.22	194	159	4.20	76	18	38.9	1035	849	805	756	766	758	698.6	26.6	19.5	17.9	67.5 %	3.8 %	394	2169
Q11	0.14	91	645	0.72	44	61	4.2	441	365	359	338	320	329	303.9	104.2	77.7	71.8	68.9 %	1.0 %	32.8	2192
Q12	0.70	20	28	3.22	20	6	47.7	454	349	334	306	301	299	262.8	9.5	6.3	5.5	57.9 %	10.5 %	284	1161
Q13	2.20	138	62	13.22	27	2	105.6	377	256	204	203	185	182	142.8	3.6	1.7	1.4	37.9 %	28.0 %	1109	1186
Q14	0.16	15	95	0.39	16	40	3.2	373	317	322	284	295	286	275.4	115.9	88.8	85.5	73.8 %	0.9 %	37.3	257
Q15	0.14	18	126	1.20	21	18	9.7	error	error	593	error	568	563	error				error		112	2528
Q16	0.39	173	442	3.20	84	26	31.4	839	698	671	647	643	637	587.3	26.7	20.3	18.7	70.0 %	3.7 %	280	11636
Q17	0.39	20	52	2.70	27	10	26.7	1255	972	889	862	779	763	664.8	46.9	28.5	24.9	53.0 %	2.1 %	646	525
Q18	0.72	21	29	4.20	33	8	36.9	1135	943	857	814	802	785	717.4	30.7	21.3	19.4	63.2 %	3.3 %	408	8672
Q19	0.70	15	21	4.70	17	4	44.1	972	331	312	295	290	287	119.2	22.0	6.5	2.7	12.3 %	4.5 %	492	188
Q20	0.39	41	105	1.70	44	26	21.7	972	803	744	732	737	698	643.2	44.7	32.1	29.6	66.2 %	2.2 %	314	885
Q21	1.70	578	339	12.72	204	16	127.7	3815	2976	2912	2629	2611	2469	2235.4	29.9	19.3	17.5	58.6 %	3.3 %	330	329
Q22	0.72	17	23	4.20	24	6	44.5	206	153	153	140	130	128	112.3	4.6	2.9	2.5	54.5 %	21.6 %	595	2027

Table 2

TPCx-BB (Big-Bench) Query	SF1000 incom-pressible time s" [%]	SF3000 incom-pressible time "s" [%]
Q1	57 %	43 %
Q2	21 %	20 %
Q3	34 %	30 %
Q4	22 %	22 %
Q6	30 %	20 %
Q8	42 %	29 %
Q10	89 %	75 %
Q11	44 %	35 %
Q12	38 %	26 %
Q13	32 %	24 %
Q14	61 %	37 %
Q15	77 %	57 %
Q16	23 %	16 %
Q17	87 %	74 %
Q18	85 %	56 %
Q19	95 %	84 %
Q21	59 %	33 %
Q22	67 %	41 %
Q24	42 %	31 %
Q29	24 %	15 %
Q30	17 %	16 %

s_{min}

we are not even trying to do that: With our single Anatella server, we are just "competing against" the Spark incompressible time "s". ...and, unfortunately for Spark, this time "s" is just terribly BIG! What's even worse is that, for most of the queries (see the cells in **red** in column R), the Spark incompressible time "s" is above 50%! Meaning that **the maximum speed-up for Spark is 2, whatever the size of your cluster**. In comparison, the average speed-up achieved with Anatella compared to Spark (i.e. the average of column O) is 39.4.

Summary on the Spark incompressible time "s"

To summarize, the main finding of this white paper is the catastrophic large value of the Spark incompressible time "s" (almost always above 50%!). **This makes the whole Spark system nearly unusable** since the major Spark promise (i.e. horizontal scalability: to deliver higher-speed on a larger infrastructure) is not achieved: it's a catastrophic failure.

The methodology to compute "s" (i.e. to compute the column N or R) is based on fitting a line obtained by applying the Amdahl's law through the data points collected in columns H to M. You'll find more info about this subject on the Github repository: https://github.com/Kranf99/TPC-H-Benchmarck-Anatella-Spark

You can intuitively compute yourself "s": Just compute the value of the column N, based on an extrapolation of what you see in the columns H to M. The "s" value (column R) is then "column N" divided by "column H".

Furthermore, many independent researchers observed the same phenomenon about "s": For example, we are reproducing in the Table 2 (above on the right) the results obtained inside Figure 3 of the scientific paper named "*Amdahl's Law in Big Data Analytics: Alive and Kicking in TPCx-BB (BigBench)*"). You can see that there are nearly no queries that have s<20% (in **green**) and many of them have s>50% (in **red**). The best speedup that you can expect (using more CPU) is thus $\frac{1}{s_{min}} = \frac{1}{0.15} = 6.6$

Scalability

Let's now assume that you have an unlimited budget to build your computing cluster. This means that you can buy 22 servers and run each of the 22 TPC-H queries simultaneously, one query per server. Anatella is built for that because:

- Anatella has a system that distributes the queries to execute on many nodes, one query per node.
- This ways of distributing the workload has an initialization time "s" that is almost zero: i.e. Anatella just need to send the data transformations to execute on each node: This is almost instantaneous. Since the initialization time "s" is zero, this means that Anatella is truly infinitively scalable and can achieve speed-up ratios over 10000 (while Spark is limited to a speed-up of 5).
- Anatella can also execute one unique query using many servers (in the same way as Spark is doing) but this is usually not very efficient (it's still more efficient than Spark but we are still badly "hurt" by the incompressible time "s" that is non-null in this case).
- With an infinite budget, Anatella is still 17.5 times faster than Spark.
- Since Anatella is not an "in-memory" tool, one Anatella server can process nearly any data size. In opposition, Spark is not designed to run "one query per node" (because Spark needs to use all the RAM of the whole cluster to run efficiently: i.e. it's an "in-memory" tool). This means that, if you attempt to use Spark to run "one query per node" (as Anatella is doing by default), you'll get many failures and instabilities.

Given the fact that Anatella is the only tool that can reliably use a large cluster of PC's to achieve speed-up ratios above 10000, we can really say that Anatella is the only truly scalable solution.

Total cost of ownership

Adopt TIMi now and benefit from infrastructure costs that are divided by 22.2 on average! $\left(22.2 = \frac{\text{sum column J}}{\text{sum column I}}\right)$ What do you think about reducing your half-a-million Amazon/Azure bill to only 22.5K€? That's great! 👍👍😄