Performance Differences Using Serial and Parallel Processing Methods in Big Data Analytics at AFC Ajax

Karim Chouaten
k.chouaten@outlook.com
University of Amsterdam

ABSTRACT

As data is increasing in terms of volume, velocity and variety, it eventually can result into technical challenges if the infrastructure of the organisation is not set up properly. This study investigated how Amsterdamsche Football Club Ajax (AFC Ajax) might mitigate this challenge, by evaluating how parallelization can be used to improve the performance of the current data processing method of the player tracking and athletic performance datasources. This data process is being used by AFC Ajax to calculate the physical load per player, for each match and training session. The research has been implemented using *Python*, *Apache Spark SQL* and *Azure Databricks*, where the performance (defined by the median runtime in seconds) is measured of both the current and new situation. The current situation includes a serial processing method using one worker on the computing cluster, while the new situation is using a parallel processing method, which is facilitated by twelve workers. What can be concluded is that parallelization optimizes the current situation with roughly 80.2%, resulting in a significant runtime decrease from 95.4 seconds to 18.9 seconds. Future research needs to be done to clarify related aspects that fall outside the direct scope of this study.

1 INTRODUCTION

Elite soccer clubs nowadays are collecting, processing and analyzing large-scale amounts of player and ball tracking data, in order to improve their competitive position with regard to other competing soccer clubs [2, 5]. Looking from the technological perspective, this (business) goal can rather be challenging to achieve if the infrastructure layer is not set up properly. This is especially the case when over time the volume, velocity and/or variety of the data is increasing, which eventually can result into the abundance of data, also called "big data".

One of the soccer clubs which can run into this problem in the (near) future is Amsterdamsche Football Club Ajax (AFC Ajax N.V.). The researcher of this study is currently employed by this soccer club in the Data Intelligence team, and is therefore the main source of knowledge for the internal organisational and technological information used for this study.

The reason why AFC Ajax might run into the aforementioned problem, is because the soccer club is planning to combine many different player tracking and athletic performance datasources of partner soccer clubs into one centralized dataset, to eventually develop high-end coaching services and advanced player analytics for itself and its partner soccer clubs [6]. With the expected explosive increase in volume, velocity and variety of the data, this eventually may result into big data challenges.

In the current situation, all player tracking and athletic performance data are processed using traditional Extraction, Transform and Load (ETL) methods and are stored in SQL Server Databases. The currently deployed queries for performing ETL are not well suited for handling big amounts of data. This is because some of these queries have inefficient methods of processing the data, which eventually results into relatively high runtimes and CPU usage, which can be a bottleneck for facilitating (realtime) big data analytics in the future. All in all, the current data infrastructure of the soccer club is not yet suitable to handle big data in an effective and efficient way.

This study therefore investigated how the processing flow of the player tracking and athletic performance data of AFC Ajax players can be improved using *parallelization*, to eventually facilitate big data analytics processes. Due to time constraints, the scope of this study is set to only investigate the performance improvements of the dataflows concerning the player tracking and athletic performance datasources, addressing specifically the *volume* component of big data.

Furthermore, the defined problem statement sounds: *The current technological infrastructure of AFC Ajax is not yet set up in a way to optimally facilitate real-time big data analytics in an effective and efficient way.*

In order to (partly) mitigate this problem, the following research questions have been taken into consideration during this study: 1) *How is player tracking and athletic performance data currently processed?* And 2) *How can parallelization be used to improve the current performance?*

The tools used for the implementation of this study are *Python*, *Apache Spark SQL* and *Azure Databricks*. The motivation behind the selection of the mentioned tools is due to the fact that AFC Ajax currently is using Microsoft Azure Cloud Services for nearly all of its data related processes

(the researcher confirms this as he is working there with these tools), making the implementation of the potential solution produced by this study relatively less complicated. Furthermore, the implementation of parallelization using cloud services with Apache Spark SQL and Databricks in combination with relatively big datasets is more feasible, in comparison with hosting all of the data on a single laptop device.

Earlier studies related to this research are outlined below. Firstly there is the work by Bialkowski et al. [1] who has done a study regarding the large-scale analysis of soccer matches using spatiotemporal tracking data, where a big amount of data (400,000,000 data points) is used to accurately detect and visualize formations, as well as analyze individual player behaviour. The research of Bialkowski et al. shows that it is technically possible to aggregate big amounts of data for soccer analytical purposes, which are similar to the data aggregation process discussed in the research of this paper, but only with many more data points than the current situation of AFC Ajax.

Other related studies involve real-time soccer analytics using a sensor system, soccer analytics annotations system and video processing system using a video camera array, resulting into a distributed processing implementation [11]. The research of Rein & Memmert [8] discusses how big data and modern machine learning technologies may help to address the technical challenges of a continuous increase in data volume, this to aid in developing a theoretical model for tactical decision making in team sports. Finally, the research of Parhami [7] illustrates how parallelization can be applied to the handling of extremely large data sets. The research of Parhami provides the theoretical basis for the study of this paper, assuming that parallelization might improve the performance of the current situation.

The execution of this study is done by firstly investigating the player tracking and athletic performance datasources, in order to create a fair representation of a big data situation. This is done by scaling up the datasources by 20 times using duplication. The motivation behind this number (20) is that the resulting data volume is both not too much or too small, making it a reasonable representation of a big data situation. This to make it possible for this study to evaluate the impact of parallelization, in a situation where big data is present. Furthermore, before scaling up the datasources, the data also has been anonymized. This anonymization is performed to comply with the privacy restrictions of the General Data Protection Regulation (GDPR) [12], as the data concerns player-related health, psychological and physical performance data of their performance on the field, which cannot be shared with external individuals or organisations.

After generating a representing and anonymized dataset of reasonable size, the current flow of processing the player tracking and athletic performance data has been investigated. After the dataflow has been elaborated, its technical performance was measured. This was done by executing the current SQL query multiple times and measuring the runtime in seconds to process the generated dataset. Finally, the median of the execution attempts is taken, to extract a representative metric to evaluate the differences between the current and new situation. The median has been chosen as the performance metric, to minimize the effect of any fluctuations that might occur during the multiple execution attempts. According to Laerd Statistics [10], the median is also the best measure of central tendency in the case of a skewed dataset with ratio data [13]. This can be similar to the situation of this study, where skewed fluctuations in the performance measurements might take place.

Then, parallelization has been implemented using Spark SQL and Python within the Azure Databricks environment of Microsoft Azure Cloud Services, after which the performance was evaluated in the same way the current dataflow is measured (by taking the median). This in order to provide AFC Ajax with insights to improve its infrastructure, so it can facilitate big data analytics for player tracking and athletic performance in the (near) future.

2 IMPLEMENTATION

The implementation is done by firstly generating representative datasets by scaling up the original datasources of AFC Ajax, which are then being processed by the current SQL query to calculate the load of each player per match and training session (aggregation of both tracking and performance data). The datasources used by the current SQL query are shown below in table 1. Additional details regarding the consulted datasources can be found in Appendix B of this paper.

Datasource	Columns	Records (scaled)
Inmotio	134	6,665,169
Catapult	85	1,122,324
Readiness	18	1,090,089
Players Ajax	24	1,497 (unscaled)
Teams Ajax	9	50 (unscaled)
Load extern	46	153,636

Table 1: Consulted datasources during this study.

What can be noticed is that the datasources *Players Ajax* and *Teams Ajax* have not been scaled. This is done to prevent errors when the different datasources are being joined by the SQL query. The scaling results into more available records per player and team, which will increase the computational load in general because there is more data to process. Also scaling up the player and team datasources using duplication, will result in player and team records to not be unique

anymore, while it is a condition to be able to calculate the load per player without errors.

After generating datasets which are representative for the datasources mentioned above, the player load is calculated using a SQL query. The current SQL query is normally being executed within the Microsoft SQL Server Management Studio (SMSS) environment, but to be able to compare the performance with the new situation, it is important to use the same platform for execution. For this reason, the environment Azure Databricks hosted on the Microsoft Azure Cloud Services is used for both the current as new situation.

The current SQL query consists of seven separate *UNION* elements, which combines the player performance and tracking data into one dataset using the six datasources mentioned above. The resulting player load dataset (represented as a SQL view) originally consists of 54 columns and 430,132 records (after scaling 9,032,765 records). This player load dataset is then used for soccer analytics and different kinds of reporting, used in specific soccer software and dashboards. The data processing flow of the current situation is visualised below in figure 1.

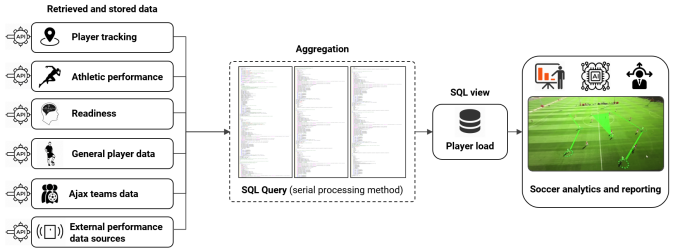

Figure 1: Current data processing flow

In order to measure the performance of the current situation, which is using serial data processing, only *one* worker is used on a Databricks computing cluster of worker type Standard DS3 v2 with 14.0 GB memory, 4 cores and 0.75 Database Units (DBU). For each attempt (total of ten), the runtime in seconds is recorded in a database using Microsoft Excel. The median of all those ten attempts have been calculated afterwards. In order to measure the performance of the new situation, which is using parallel data processing, *twelve* workers have been used by adjusting the amount of workers on the same cluster as the current situation.

In the current situation, each separate sub-query (consisting of one UNION element) is awaiting the previous sub-query to finalize, to eventually merge the results of each sub-query into one collective dataset. This delay might cause an increase in runtime, as each sub-query is awaiting the preceding query to finalize. This results into an unnecessary bottleneck, especially because each sub-query should be able to be executed independently from the others.

In the new situation, all sub-queries are executed simultaneously using the parallel processing, which is different than the current situation where serial processing is applied. In this study, *Apache Spark SQL* has been used to facilitate the parallelization process. Parallel processing with Spark SQL consists of a driver program that runs the user's main function, and executes various parallel operations on a cluster [9, 14]. Spark provides a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel [14]. In theory, the new situation should be nearly equal to the longest sub-query, as each sub-query is being executed simultaneously. Meaning that the total runtime of the new situation should be equal to the runtime of the sub-query that ends as last. Looking from the other side, the current situation should equal the sum of the runtime of each separate sub-query, this due to the fact that serial processing is being used. The Evaluation chapter further discusses the findings of this analysis.

After the implementation of parallelization and evaluating its impact on the performance in comparison with serial processing, the earlier mentioned datasources have been converted into delta lakes (or delta tables), as this was highly recommended by the Databricks platform to further improve the technical performance. A delta lake is an open source storage layer that brings reliability to regular data lakes [3]. Delta lakes provide Atomic, Consistent, Isolated and Durable (ACID) transactions, scalable metadata handling, and unifies streaming and batch data processing. It runs on top of existing data and is fully compatible with Apache Spark APIs.

In order to convert the regular datasources into delta tables, the different datasources first need to be configurized in how to partition and optimize each datasource. Each partition has been made by using the unique *player ID* parameter, while the optimization has been done by taking a datetime field, which differs in each datasource. There are different ways to configurize the method of partitioning and optimizing the delta tables. This study however does not discuss the most optimal way of doing this, as it is not included in the scope.

The conversion into delta lakes has been done to evaluate to what extent the performance can be optimized further, by combining the benefits of both parallel processing and an optimized way of storing datasets in the form of delta lakes.

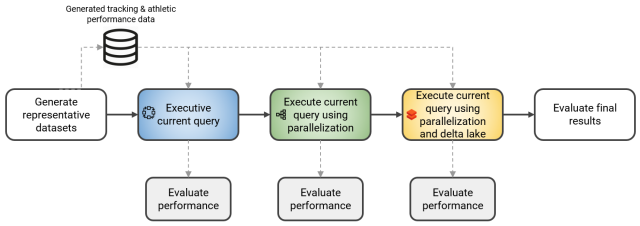

Figure 2: Implementation process flow

The performed implementation steps are visualized above in figure 2.

Finally, the researcher of this study and the stakeholders of AFC Ajax have agreed to a performance improvement threshold of **50%** in runtime in seconds. In other words: the stakeholders are satisfied if the new situation results into a runtime in seconds which at least is twice as fast as the current situation. The next chapter discusses the findings of implementing the procedure illustrated in the Implementation chapter.

3 EVALUATION

Note: all produced code, datasets and additional products are available using a GitHub Repository found in Appendix A.

The results of measuring the performance of the current situation (serial processing using one worker), new situation (parallel processing using twelve workers) and the delta tables, are analyzed and visualised using a Microsoft Power BI dashboard. During the implementation, also the runtime of eight and sixteen workers have been measured, to evaluate if increasing the amount of workers result into a linear or even exponential improvement of the performance. The dashboard is shown below in figure 3.

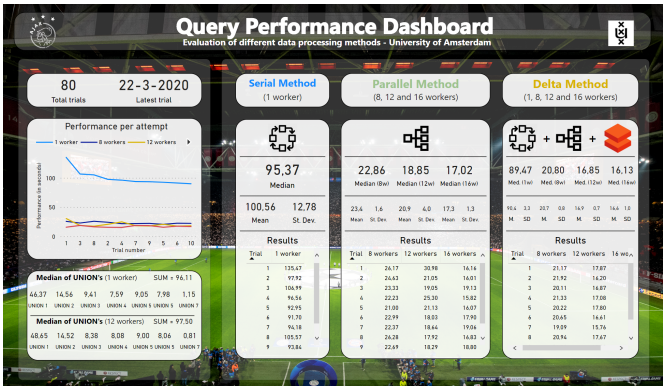

Figure 3: Descriptives of the implementation.

After using the developed dashboard to extract the calculated descriptives of the implementation, table 2 has been filled in to clearly structure the findings.

When looking at the findings, it can be derived that the median of the serial method equals 95.4 seconds and with delta tables 89.5 seconds, while the median of the parallel method equals 18.9 seconds and with delta tables 16.9 seconds.

What can be concluded now, is that in this research we have shown that parallelization optimizes the current situation with roughly 80.2%, resulting in a significant runtime decrease from 95.4 seconds to 18.9 seconds. Also, converting the original datasources into delta tables, as recommended by the Databricks platform, results in an additional improvement of 10.6% when used in combination with the parallel processing method. This might be caused due to the fact that delta tables are being optimized using a specific algorithm, which makes it possible to retrieve the in the cloud stored data in a relatively more effective and efficient way [3], resulting in runtime performance improvements.

Method	Workers	Median	Mean	St. Dev.
Serial	1	95.37	100.56	12.78
Parallel	8	22.86	23.40	1.60
	12	18.85	20.90	4.00
	16	17.02	17.30	1.30
Delta Table	1	89.47	90.40	3.30
	8	20.80	20.70	0.80
	12	16.85	16.90	0.70
	16	16.13	16.60	1.00

Table 2: Runtime in seconds per method.

In theory, an explanation for the parallel method being relatively better than the serial method, might be caused due to the availability of having multiple workers. This in turn makes it possible to execute the different sub-queries simultaneously, while the serial method only has one worker to execute each sub-query one after one in a sequence. If this is the case, the parallel method should be approximately equal to the longest sub-query, while the serial method should be equal to the sum of each sub-query.

The median runtime of each of the seven sub-queries is also visualised in the dashboard (figure 4) in the left-bottom corner. These have been calculated by executing each sub-query independently, and measuring the runtime in seconds multiple times.

After evaluating the runtime of each sub-query, it can be proven that the runtime of the serial method indeed is equal to the sum of each sub-query executed independently using both one and twelve workers (*serial method median = 95.4 sec, sub-query sum using 1 worker = 96.1 sec and 12 workers = 97.5 sec*). It unfortunately cannot be proven that the parallel method is approximately equal to the longest sub-query (*median parallel method with 12 workers = 18.85 sec and longest sub-query = 48.65 sec*). Further research needs to be done to confirm (or reject) this claim.

Also, the sum runtime of each sub-query executed using the parallel method (*12 workers*) is even 1.4 seconds longer then the serial method (*parallel method = 96.11 sec versus serial method = 97.50 sec*). An explanation for this unfortunately could not be found in the literature, but it could be related to the starting mechanism of Spark SQL in combination with Databricks, which might cause additional delay as the deployed workers need to be coordinated during the process.

Another interesting finding is that increasing or decreasing the amount of workers using the parallel method, does not result in a linear (or even exponential) increase or decrease in the performance. For example, eight workers result in a performance improvement of respectively 76.0%, twelve workers of 80.2% and sixteen workers 82.2%. What can be understood is that using more than one worker indeed improves the performance compared to using only one worker, but by for example doubling the amount of workers from eight to sixteen workers, does not even come near the doubling of the performance improvements.

This makes sense, because this else would mean that for any situation where it is possible to increase the performance (measured in runtime) by implementing parallelization, it also would be possible to decrease the runtime to nearly zero seconds (because adding more workers will then result in a linear or exponential runtime decrease), but this is not the case. Future research however needs to be done to substantiate this finding by looking at the technical specifications of this process.

4 DISCUSSION

All in all, this research has proven that the parallel processing method is relatively better than serial processing method in the situation of AFC Ajax, when looking at the performance defined by the runtime in seconds. What also could be proven is that delta tables result in a better performance compared to the default datasource type of Azure Databricks. With a performance improvement of 80.2% (using twelve workers) it implicates that the earlier defined threshold of 50% is surpassed with roughly 30%, meaning that the stakeholders of AFC Ajax, along with the researcher, are satisfied with the produced result.

However, even though the stakeholders of this study are satisfied with the achieved result, there are still some points that need to be discussed. These are outlined below.

Firstly, Spark SQL using Databricks has some inconsistencies looking at the execution timespan, which can influence the actual measured runtime per method. This for example resulted in the same Spark SQL code and processing method measuring 135 seconds for the first attempt, while the third attempt resulted in 104 seconds runtime. For this reason, the median has been taken to minimize the effect of those fluctuations. The impact of those fluctuations however cannot be eliminated completely by using the median. This limitation therefore has to be mentioned and taken in consideration.

Also, the developed solution unfortunately cannot be implemented 1-on-1 in AFC Ajax, as the SQL code slightly got adjusted to be able to execute it using Spark SQL in Databricks. Slight adjustments need to be made to the SQL code in order to be able to implement the produced solution into the soccer club. Furthermore, as mentioned earlier, in order to convert the regular datasources into delta tables, there were several configurations that needed to be done before the conversion could be carried out. There is a possibility that the configurations chosen in this research (see Implementation chapter) have not been set up optimally. This can be an interesting topic to research further.

Another relevant and interesting topic for future research includes the investigation of whether there are significant differences between the serial and parallel method, looking at the load on the used Central Processing Unit (CPU) of the hardware. The results of such study can be interesting for AFC Ajax (and other organisations that are being in similar scenario's), especially if a trade-off needs to be made whether the new situation results into a much higher financial cost, as the CPU load can be linked directly to the occured costs made by the cloud computing services platform (Microsoft Azure Cloud Services in this research's case).

Wrapped up, the results of this research are satisfactory for the stakeholders, while there are a few limitations that need to be taken into consideration. Future research needs to be done to clarify related aspects that fall outside the scope of this study.

REFERENCES

[1] Alina Bialkowski, Patrick Lucey, Peter Carr, Yisong Yue, Sridha Sridharan, and Iain Matthews. 2014. Large-Scale Analysis of Soccer Matches Using Spatiotemporal Tracking Data. (12 2014).

[2] Idea Connection. 2018. The Big Data Revolution in Soccer - Open Innovation success stories. https://www.ideaconnection.com/open-innovation-success/The-Big-Data-Revolution-in-Soccer-00700.html

[3] Databricks. 2020. Delta Lake Documentation. https://docs.databricks.com/delta/index.html. Accessed: 2020-04-03.

[4] Rosalie Gearhart, Fredric L. Goss, Kristrin M Lagally, John M Jakicic, Jere Gallagher, and Robert J. Robertson. 2001. Standardized scaling procedures for rating perceived exertion during resistance exercise. *Journal of strength and conditioning research* 15 3 (2001), 320–5.

[5] Analytics Insight. 2020. What Impact is Big Data Having on Soccer? https://www.analyticsinsight.net/what-impact-is-big-data-having-on-soccer/

[6] AFC Ajax N.V. 2020. Internal Documents.

[7] Behrooz Parhami. 2018. Parallel Processing with Big Data. (2018), 1–7. https://doi.org/10.1007/978-3-319-63962-8_165-1

[8] Robert Rein and Daniel Memmert. 2016. Big data and tactical analysis in elite soccer: future challenges and opportunities for sports science. *SpringerPlus* 5, 1 (Aug. 2016). https://doi.org/10.1186/s40064-016-3108-2

[9] Apache Spark. 2020. Spark Programming Guide. https://spark.apache.org/docs/2.1.1/programming-guide.html. Accessed: 2020-04-03.

[10] Laerd Statistics. 2019. Mean, Mode and Median - Measures of Central Tendency - When to use with Different Types of Variable and Skewed Distributions | Laerd Statistics. https://statistics.laerd.com/statistical-guides/measures-central-tendency-mean-mode-median.php

[11] Håkon Kvale Stensland, Østein Landsverk, Carsten Griwodz, Pål Halvorsen, Magnus Stenhaug, Dag Johansen, Vamsidhar Reddy Gaddam, Marius Tennøe, Espen Helgedagsrud, Mikkel Næss, Henrik Kjus Alstad, Asgeir Mortensen, Ragnar Langseth, and Sigurd Ljødal. 2014.

Bagadus: An Integrated Real-Time System for Soccer Analytics. *ACM Transactions on Multimedia Computing, Communications, and Applications* 10, 1s (Jan. 2014), 1–21. https://doi.org/10.1145/2541011

[12] European Union. 2016. Regulation (EU) 2016/679 Of The European Parliament And Of The Council - General Data Protection Regulation (GDPR). https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

[13] Wikipedia. 2020. Central tendency. https://en.wikipedia.org/w/index.php?title=Central_tendency&oldid=946612661 Page Version ID: 946612661.

[14] Matei Zaharia, N. M. Mosharaf Chowdhury, Michael Franklin, Scott Shenker, and Ion Stoica. 2010. *Spark: Cluster Computing with Working Sets.* Technical Report UCB/EECS-2010-53. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-53.html

A GITHUB REPOSITORY

The generated datasets, Python/SparkSQL code, Power BI dashboard, PowerPoint presentation and YouTube video can be found in the following GitHub repository:
https://github.com/karimchouaten/UvA-BigDataResearch

B DETAILS OF USED DATASOURCES

The consulted datasources of this study, along with the amount of columns, (scaled) amount of records and a description have been illustrated below.

- Inmotio (134 x 6,665,169): Player performance data registered by Inmotio sensors during matches and training sessions. 31 columns and roughly all records have been used.
- Catapult (85 x 1,122,324): Tracking data registered by Catapult sensors during matches and training sessions. 32 columns and roughly all records have been used.
- Readiness (18 x 1,090,089): Measuring the player readiness through a standardized form which is being filled in by the player after each training session (i.e. mood, energy level, muscle soreness and amount of sleep) [4]. Only the column *TotalScore* and all records have been used.
- Players Ajax (24 x 1,497): Contains all registered players at Ajax. This dataset is being used to join different datasets and retrieve a player's fullname based on his player ID.
- Teams Ajax (9 x 50): Contains all registered teams at Ajax and is being used to join different datasets and retrieve the team name the players are playing in, based on the team ID.
- Load extern (46 x 153,636): Player load per match and training session, registered through external systems which are neither Inmotio nor Catapult. 41 columns and all records have been used.