Big DataBase Management System

Alberto Abelló[1], Sergi Nadal[1]

[1]Universitat Politècnica de Catalunya-BarcelonaTech

Abstract

A Big Data system is a tiny fraction of analytical code surrounded by a lot of "plumbing" devoted to manage the generated models and the associated data. Hence, we can consider that plumbing to be mimicking a DBMS, which is indeed a complex system that actually has to serve different purposes and hence provide multiple and independent functionalities. Thus, it can neither be studied nor built monolithically as an atomic unit. Oppositely, there are different software inter-dependent components that interact in different ways to achieve the global purpose. Similarly to DBMS, in a Big Data system, we have to understand among other issues how our system is going to collect data; how these are going to be used; where they are going to be stored; how they are going to be related to the corresponding metadata; if we are going to use any kind of master data, where these will come from and how they will be integrated; how are the data going to be processed; how replicas are going to be managed and their consistency guaranteed; etc. In this paper, we briefly discuss the difficulties to build such system, paying special attention to how metadata can help storage and processing.

Keywords

Big Data architecture, DBMS

1. Functionalities of a DBMS

In order to study the functional components required to manage data, it is essential to first understand the required functionalities. A database management system (DBMS) is a software system that provides the functionalities to manage large, shared and persistent data collections, while ensuring reliability and privacy. Out of the many functionalities provided by a DBMS, we highlight: Storage, Modeling, Ingestion, and Querying/Fetching.

Nowadays, a new kind of data-intensive systems that gather and analyse all kinds of data has emerged bringing new challenges for data management and analytics. These are today referred as Big Data systems. Thus, we can see a Big Data Management System as a DBMS that has to provide the previously highlighted functionalities adapted to the new scenarios posed by Big Data [1]. The development of a cohesive and integrated Big Data system is, however, a challenging task that requires to understand how the required functionalities can be performed by different, independent components. Hence, it is crucial to understand and establish how they interact.

We highlight some of the challenges such new systems face. On the one hand, the "Velocity" dimension of Big Data identifies the need of managing and processing data streams which are generated at a very large pace. Nevertheless, this does not mean everything must be done in real time. Indeed, some execution flows do not feel such pressure and do not require such low latency. Hence, it is the case that in many applications both batch and stream processing have to coexist in the same architecture. This, however, makes the architecture more complex in terms of number of components, also hindering their communication and data sharing, as well as the consistency of independent processing branches. On the other hand, the "Variety" dimension refers to the complexity of providing an on-demand integrated view over an heterogeneous and evolving set of data sources such that it conceptualizes the domain at hand. An example of it is BigBench [2], which defines a benchmark representative of real use cases of Big Data. We can observe that the main differences with traditional data-intensive applications are (i) the presence of external sources and (ii) the relevance of non-structured data (i.e., not typed and not tabular, which today represents the majority of data being generated). It is clear, thus, that to make use of rapidly generated, external and non-structured information sources, we need specific and specialized architectural components that interact with many others to transform such complex data into actionable knowledge.

2. Big Data Architectures

The previously identified challenges require a complete reconsideration of classical DBMS architecture and components that date back to the 70s. Yet, the approach so far adopted by the data management community has been that of developing components addressing each of the required functionalities (i.e., Storage, Modeling, Ingestion, Processing, and Querying/Fetching) as efficiently

Published in the Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK

aabello@essi.upc.edu (A. Abelló); snadal@essi.upc.edu (S. Nadal)

https://www.essi.upc.edu/dtim/people/alberto (A. Abelló); https://www.essi.upc.edu/dtim/people/snadal (S. Nadal)

0000-0002-3223-2186 (A. Abelló); 0000-0002-8565-952X (S. Nadal)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3135/dataplat_keynote1.pdf

as possible. Indeed, current technological stacks for the management and processing of data-intensive tasks are composed of independent components (commonly those in the NOSQL family) that generally work in isolation and are orchestrated together to map to what would be equivalent to different functionalities of a DBMS. In complex applications, where many of these tools need to interact, it is definitely not wise to do it arbitrarily. Indeed, such erroneous approach yields what is coined as a "pipeline jungle" [3]. The alternative is, thus, to reconsider some of the architectural patterns they implement.

2.1. New storage patterns

Big Data analysis is an exploratory task, thus integrating and structuring the data *a priori* entails a large overhead. Alternatively, the *Data Lake* approach promotes to maintain raw data as they are in the sources as a collection of independent files. Then, once data scientists require some of these data for a concrete purpose, it is when the task of integrating, cleaning and structuring into the right format and schema for the problem at hand is performed. This referred as the "Load-first, Model-later" approach.

The risk with this approach is that files can be simply massively accumulated without any order, resulting in what is called a "Data Swamp", where just finding the relevant data would be a challenge. The solution for this is creating an organization of files and semantically annotate into a metadata catalog conceptualizing the domain (e.g., implemented via graph-based formalisms). Thus, to the already existing mappings in the catalog, we should add links from each file to such graph containing the relevant concepts for our business. In this way, users would be able to perform guided searches over the metadata instead of blindly navigating the files. If properly done, a semantic approach can even facilitate automation of integration and queries [4].

2.2. New processing patterns

Descriptive analytics study how the business performs at different levels of granularity (e.g., regions, cities or districts), and how it evolves over time. Timeliness of data is usually not an issue for long term trends, and days or even weeks are acceptable for the current data to be processed and made ready for the analysis. Oppositely, predictive analytics aim to foresee how a given entity (e.g., customer) is going to behave in the near future. Obviously, since the purpose of a prediction is to react or at least be ready to take some action, data freshness and response time is typically crucial in this case.

Consequently, since time requirements are contradictory, we have to distinguish both precessing flows, giving rise to what is known as λ-Architecture. This, consists of two execution branches fed from the same sources. One focuses on batch processing, and the other on stream processing. Nevertheless, maintaining such potentially redundant flows generates some management risks. For the prediction to be accurate, the new arriving tuples have to go through the same transformation and preparation tasks as the training data (otherwise, the validity of the prediction would be compromised). The λ-architecture evolved into κ-architecture, as a simplification with a single execution engine (hence a single implementation of the transformations).[1] The batch processing is replaced by playing the data through the streaming system quickly. If for any reason, we require different versions of the transformations for different predictive models, we can keep all of them in the same system and choose the most appropriate one at every moment, independently of whether it is for training or production.

3. Conclusion

The current problem in Big Data is not how to make a more accurate predictive model, but how to manage the data needed for its training. The difficulty is amplified by having independent components that need to interact without a solid backbone that removes the burden of their connectivity from the shoulders of developers. If we pay attention to either Velocity or Variety, we can conclude that metadata is crucial for such backbone and the governance of Big Data. However, current state of the art has not reached the required level of maturity to give the view of a single homogenous system coordinated through those metadata instead of adhoc scripts or specifically programmed APIs and connectors.

References

[1] P. Jovanovic, S. Nadal, O. Romero, A. Abelló, B. Bilalli, Quarry: A user-centered big data integration platform, Inf. Syst. Frontiers 23 (2021) 9–33.

[2] T. Rabl, M. Frank, M. Danisch, H. Jacobsen, B. Gowda, The vision of bigbench 2.0, in: Proceedings of the 4th Workshop on Data analytics in the Cloud (DanaC), ACM, 2015, pp. 3:1–3:4.

[3] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, D. Dennison, Hidden technical debt in machine learning systems, in: Advances in Neural Information Processing Systems, volume 28, Curran Associates, Inc., 2015.

[4] S. Nadal, O. Romero, A. Abelló, P. Vassiliadis, S. Vansummeren, An integration-oriented ontology to govern evolution in big data ecosystems, Inf. Syst. 79 (2019) 3–19.

[1] https://www.oreilly.com/radar/questioning-the-lambda-architecture