Demo: Diligent - An OSN Data Integration system based on reactive microservices

Alexandros Tsilingiris
Aristotle University of Thessaloniki
alextsil (at) csd.auth.gr

Ilias Dimitriadis
Aristotle University of Thessaloniki
idimitriad (at) csd.auth.gr

Athena Vakali
Aristotle University of Thessaloniki
avakali (at) csd.auth.gr

Georgios Andreadis
Aristotle University of Thessaloniki
andreadi (at) eng.auth.gr

***Abstract*—This demo showcases some of the capabilities of Diligent, a platform for collecting and analysing data from Online Social Networks and is still under development. Diligent relies on microservices and reactive streams, which optimize the time spent (t), to the resources used (r), ratio (t/r). The proposed demo will present:**

- **The vast hardware utilization margins produced by using both blocking and reactive I/O approaches.**
- **The performance gap between using blocking I/O and Reactive I/O clients.**

Both experiments highlight the added benefits of using reactive approaches in online social network data processing systems.

I. Introduction

To achieve the best and fastest results in social media data integration, system architecture should be put into great consideration, since the challenging aspects of big data streams [1] are becoming more and more significant with the rapid accumulation of online data.

Diligent is an end-to-end software solution that takes care of the complete data integration process [2], from collecting raw data from online streams on one end, to providing freshly integrated data through an API on the other end. Data integration provides a unified view and management of the data, which in turn opens up new opportunities for data analytics applications and insights, thus increasing the value of the underlying datasets. These datasets can be used either for research, eg. sentiment analysis [3] or discovering patterns in smart cities [4]. The pipeline of the system's inner workings can be seen in figure 1.

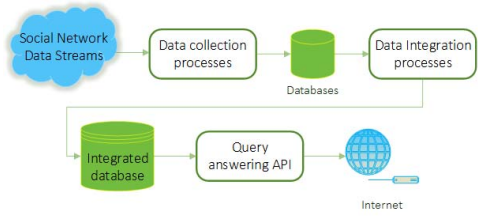

Fig. 1. Diligent's architecture pipeline

The users of the system will be able to access it through its web based front-end and log in using their OSN accounts in order to allow third party access to their data. The platform will then schedule data gathering and the integration pipeline will eventually complete, thus making freshly integrated data available for any interested parties.

All sub-systems are implemented using the latest techniques in software engineering and distributed systems design, and can be scaled independently, in real-time, based on current system load or upon command of load forecasting models using machine learning. The system can operate on commodity hardware and does not require any initial special setup for its nodes to register into the swarm, since the Spring framework ecosystem takes care of this abstraction layer. The only requirement is the Java Runtime Environment (JRE). This allows system administrators to simply scale the system horizontally or vertically without the burden of additional configuration. Geographical distribution is also possible, allowing higher quality service for clients that are closer to the nodes and minimizing unnecessary, large cross-continental data transfers. The system makes extensive use of the following latest approaches and battle tested frameworks:

- Microservices [5] architecture, built on top of Spring Boot as core, Spring Cloud as a configuration manager and Netflix Eureka as a service discovery agent, to allow for infinite on-demand scaling and endless distribution possibilities, ensuring maximum service availability.
- Reactive network I/O, using Spring WebFlux, to provide access to its data in a streaming fashion and make sure response times are minimized, therefore increase service quality.
- Reactive database I/O, using the official MongoDB libraries, to ensure that the minimum accomplishable amount of hardware resources are used, hence minimize operational costs.

Diligent's architecture blueprint can be seen in figure 2. The system consists of the following independent subsystems:

- Data collection: This sub-system is responsible for watching and persisting the vast amount of available social network data streams into our databases. Spring boot was used to design microservices in a modular way that could support any type of data from any online source. The

978-1-5386-4705-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SMARTCOMP.2018.00059

main target is online social network streams, but future expansions to include blogs, fora and other sources can be done seamlessly.

- Databases: MongoDB[6] databases are leveraged in order to allow each data source to maintain its original schema prior to integration in order to maximize the information availability. NoSQL databases are ideal for storing social media data [7] and other forms of semi-structured data, since a common schema should not be deducted at this stage of the pipeline.
- Data integration: In this sub-system, all the data is aligned, linked and finally fused together. The task of the data alignment phase is to understand which records have the same semantics and which do not. The task of data linkage is to detect which records refer to the same entity and which do not. Every data source contains its own naming conventions for what might be the same kind of information. The final task of data integration, data fusion, is to determine which values reflect the real world and which do not. Since social media data sources are not controlled by the system, the cleanness cannot be assumed. Parallelization can be leveraged across these stages via the microservices architecture. Spring Boot was used to build those microservices, along with a variety of RabbitMQ queues, in order to allow task separation and prioritization among all microservices. Finally, this stage will be able to support both offline and online algorithms. When clients require instantaneous answers from the system, the already integrated results from its database will be provided. If data freshness is more relevant than response times, the system provides online answers where the most fresh data - taken directly from live data streams - are chosen to be processed and integrated in real-time. This method uses reactive I/O, thus allowing the client to subscribe to the source and be provided with the results as long as they are available in real-time.
- Integrated database: This database will host the data after they have been integrated by the previous stage and will mostly serve for instant query answering. Since MongoDB will be used to persist the already integrated - but still semi-structured - data, replication comes out of the box and allows greater availability and increased parallelism. MongoDB sharded clustering was used for this purpose. With multiple copies of data on different database servers. Replication will also provide increased read capacity as clients can send read operations to different servers. Maintaining copies of data in different data centers can increase data locality and availability for this distributed system. Additional copies will also be maintained for dedicated purposes, such as disaster recovery, reporting, or backup.
- Query answering API: A REST API using Spring Boot will be created in order to allow access to Diligent by humans and machines. Any device that can connect to the Internet will be able to browse the provided endpoints and resources since the JSON file format is used everywhere in the system to allow interoperability. The API will offer access to already integrated data that will reside in the systems integrated database, data that will be ready for consumption by the clients. The same API will also provide results that will be produced on-the-fly by online algorithms on the data integration stage, thus partially or fully bypassing the integrated database and returning fresh integrated results to the client. The API will make extensive use of the reactive model meaning that the requested resources will immediately start streaming to the client as long as they are available, without the need to wait for query completion, allowing for new online data analytics opportunities - Virtually any data analytics scenario - both online and offline - can be performed.

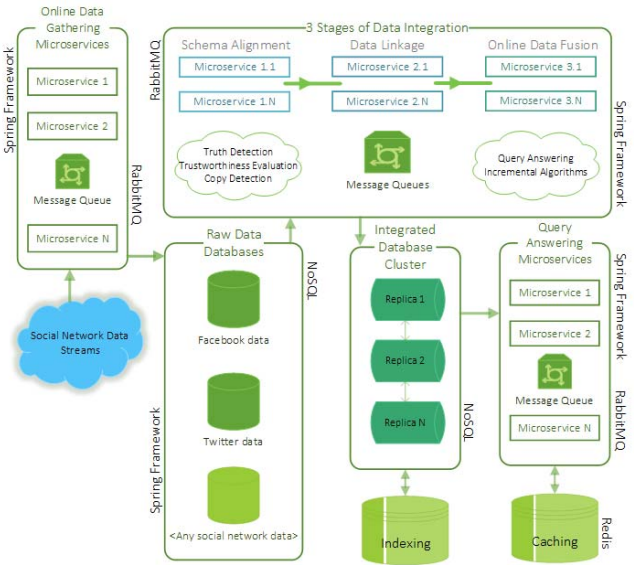

Fig. 2. Diligent's architecture blueprint

By utilizing this design and implementation approach, both small and large teams can achieve better system stability, maximum service availability and seamless scalability. The addition of the reactive model to the microservices architecture provides an extra layer of response time improvements and hardware utilization savings, thus making distributed OSN data integration systems viable even for small organizations with limited funds. Large deployments will also benefit from that added layer on their massive systems since, when applied at scale, the results are more inducing.

II. Demo part I - Hardware utilization

In large distributed systems, cross-communication via the network is very common. The target of these communications is to transfer as most possible data over the wire as possible, with the minimal resource cost. We will demonstrate that by using a reactive I/O approach, we allow our systems to do other things while waiting for I/O completion and therefore they can do more processing work with the same resources. This allows for better, faster and cheaper data integration since

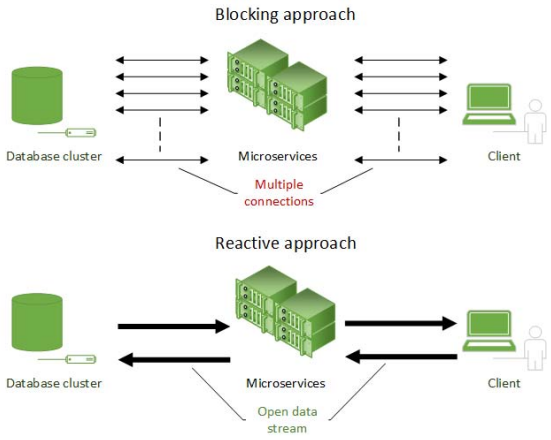

Fig. 3. Illustration of blocking and reactive I/O

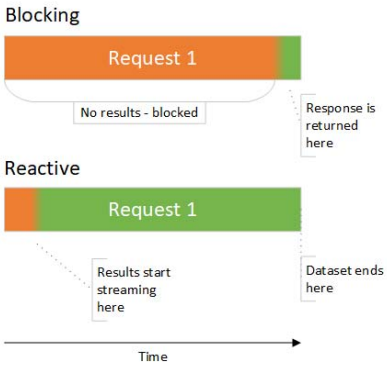

Fig. 4. Dataset retrieval in blocking and reactive approaches

data will flow in greater rates and more resources will be available for the underlying processing.

We devised the following experiments to demonstrate the vast hardware utilization margins produced by using both blocking and reactive I/O approaches on both local and remote database deployments. We also highlighted the benefits of streaming the results (Demo II).

Four tests were performed where we used Twitter stream API in order to measure the difference when inserting all tweets directly from the stream to a database. We used JVM profiling software to measure thread usage per test. The 4 cases we covered are the following:

- Blocking I/O live stream inserts in a local database (same machine).
- Reactive I/O live stream inserts in a local database (same machine).
- Blocking I/O live stream inserts in a remote database (hosted by mLab).
- Reactive I/O live stream inserts in a remote database (hosted by mLab).

The purpose of this experiment is to demonstrate resource wastage when blocking I/O is used, and how a significant amount is preserved when reactive I/O is implemented. As illustrated in figure 3, multiple expensive connections are created for each request all across the full-stack of the system. When the reactive approach is used, each part of the system maintains a single open data stream to serve all data transfer needs across the nodes. This approach saves resources that are wasted when a system is repeatedly opening, waiting on and closing connections. During our experiments we observed thread usage improvements up to a factor of 10. The results will be presented in a short video, where thread usage for all cases will be compared side-by-side.

III. Demo part II - Service quality

In this experiment we performed two (2) tests, one using blocking I/O and a blocking client, and another one using reactive I/O and a reactive client. We used Google Chrome as a client for both cases. We populated our remote database with random tweets with a total size of 50Mb and created two clients for each of the following cases:

- Batch retrieval using blocking I/O and a blocking client.
- Stream reception using reactive I/O and a reactive client.

The purpose of this experiment is to highlight the difference in retrieving a blocking batch of results versus receiving a live stream. In the latter case, as seen in figure 4, results start flowing in the system at the moment they are available, thus increasing service quality in a data integration system by keeping the data as fresh as possible. This allows the system to perform faster online analysis and even provide responses in a streaming fashion. An illustration of this procedure can be seen in figure 3, where both data streams are kept open and continuously forward data. The client can now start processing data immediately or even cancel the request if the initial set of results is not relevant: In a way, the client can peek into the dataset without having to download it first. The results will be presented in a short video, where two Google Chrome windows will be compared side-by-side during data retrieval.

Acknowledgment

This work has been supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreements No. 780121 and No. 710659.

References

[1] Mahanti, Rupa (2016). Big Data Fundamentals. Vol. 18. 4, p. 55. ISBN: 9780133443059.
[2] Dong, Xin Luna and Divesh Srivastava (2015). Big Data Integration. doi: 10.1109/ICDE.2013.6544914.
[3] Chatzakou, Despoina, and Athena Vakali. "Harvesting opinions and emotions from social media textual resources." IEEE Internet Computing 19.4 (2015): 46-50.
[4] Gkatziaki, Vasiliki, et al. "DynamiCITY: Revealing city dynamics from citizens social media broadcasts." Information Systems 71 (2017): 90-102.
[5] Newman, Sam. Building microservices: designing fine-grained systems. "O'Reilly Media, Inc.", 2015.
[6] Banker, Kyle. MongoDB in action. Manning Publications Co., 2011.
[7] Gyorodi, Cornelia et al. (2015). A comparative study: MongoDB vs. MySQL. In: 2015 13th International Conference on Engineering of Modern Electric Systems, EMES 2015, pp. 05. doi: 10.1109/EMES.2015.7158433