Abstracting Big Data Processing Tools for Smart Cities*

Fernanda de Camargo Magano, Kelly Rosa Braghetto
Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo
Rua do Matão, 1010 – 05508-090 – São Paulo – Brazil
Email: {nanda,kellyrb}@ime.usp.br

***Abstract*—Large volumes of data from various sources are generated continuously in cities. The processing and analysis of these data play a key role in the implementation of initiatives for smart cities. In order to process urban Big Data, it is essential to use high-performance tools to accelerate processing and provide quick answers. However, this use is not trivial because Big Data tools are not interoperable and require from their users knowledge of parallel and distributed computing and databases. In this work, we compare popular open-source Big Data processing frameworks and propose a software system to abstract and facilitate their use in smart city applications. The architecture of the system is composed by an interface to specify dataflow models as well as services to interpret these models and instantiate them in different Big Data tools. An implementation of the system on top of a smart city platform is also addressed.**

I. Introduction

The evolution of the Internet of Things (IoT) infrastructure and the falling costs of technology are causing an impressive increase in the number of electronic devices with sensing capabilities pervaded in the urban environment, allowing to monitor temperature, traffic, air quality, flooding, among others. Moreover, social networks can be combined with mobile devices and sensor networks in order to get contextual information from users, supporting applications' awareness of their location, preferences and relationships. The processing and analysis of these data are fundamental to the implementation of smart city initiatives, since they provide a better understanding of what happens in cities. They make it possible to identify problems and their probable causes, supporting decision making. This impacts citizens quality of life.

The volume of urban data is quite large: in São Paulo bus fleet there are almost 15,000 buses distributed in more than 1,300 lines, 29 terminals and 18,800 stopping points, according to data from São Paulo Transports [1]. Furthermore, the buses position can be different at each instant of time. Consequently, the bus location data is updated several times per minute. The characteristics of urban data, combined with the heterogeneity of their sources and their economic and social value, enable us to classify them as *Big Data*.

Big Data processing is done in batches or in real-time [2]. In batch processing, previously collected and stored data is processed, which may take hours if batches are large. In real-time processing, the data is processed as it arrives at the application, generating results with low latency. Some applications require the combined use of batch and real-time processing. This can be useful, for example, in a traffic monitoring system: to identify and report real-time crashes quickly, as well as to predict the most dangerous areas and to avoid new situations of risk by querying historical information.

The use of Big Data processing tools to handle collected data is critical to provide rapid response. There are several tools with parallel and distributed computing capabilities to process Big Data [2] and, albeit powerful, these tools are not trivial to be used since they require from their users knowledge in programming, parallel and distributed computing, besides databases. Moreover, each of them has its particularities in the way it receives, stores and processes data. Even though they generally deal with open data formats, they do not use standardized languages for specifying processing models and, consequently, are not completely interoperable.

In this work, we propose a software system that integrates a smart city platform with Big Data processing tools. The system abstracts the specificities of the Big Data tools processing models, making it easier their use in the development of applications for smart cities. The architecture of the system is composed of an interface (API) for the specification of *dataflow models* for real-time and batch data processing, and services which interpret dataflow models and instantiate them in different Big Data tools. The interface synthesizes the data routing and processing features most often found in Big Data tools, providing a standardized representation for them.

Dataflow models are directed acyclic graphs (DAGs) where nodes represent data processing activities and edges represent the flows of data among them [3]. This model is very expressive and can describe both batch and real-time (stream) processing. For this reason, dataflows can be used to create an abstraction layer for Big Data tools. The interpretation services map the dataflow models defined through the API to specific models for Big Data tools. The instantiation refers to a particular execution of a dataflow already mapped to a tool.

We also present a proposal to implement the system on top of the free software platform for smart cities [4] of InterSCity[1],

*This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq, proc. 465446/2014-0, CAPES proc. 88887.136422/2017-00, and FAPESP, proc. 2014/50937-1. Fernanda de Camargo Magano is supported by CNPq (National Council for Scientific and Technological Development).

[1] http://interscity.org/

a research project hosted by the National Institute of Science and Technology (INCT) of the Internet of the Future for Smart Cities, funded by FAPESP, CAPES, and CNPq. Applications developed on this platform are expected to benefit from the data processing efficiency provided by the Big Data tools without the burden of having to deal with them directly.

II. Related Works

The most relevant works related to the approach presented in this paper address real-time and batch processing [5], [6], [7], dataflow models [8], [9] and abstraction layers [10].

Taneja et al. [5] presented the SMASH cloud platform, to support data processing in transportation domain. SMASH processes data in near real-time, since it uses micro-batches and tools like Apache Spark[2] and Apache Hadoop[3]. The platform was applied to process traffic data in Australia.

Dissanayake and Jayasena [7] proposed a platform which combines batch and real-time processing, being capable of analyzing large volumes of data from Internet of Things. The technology used for batch distribution is Hadoop Distributed File System (HDFS). For the real-time processing layer, the system uses Apache Storm[4], not benefiting from features of other frameworks. This also occurs in the work of Taneja et al. [5], then both platforms are limited to a specific tool.

JSFlow [8] framework integrates real-time and batch processing into a single system that abstracts the data and provides a programming model that uses a JSON-based dataflow algebra. For this, it extends Jaql, a language which provides several processing methods (e.g. *filter*, *join*, *sort*, and *group by*) converted to Hadoop and Spark. JSFlow includes operators for real-time processing in Jaql (e.g., *window*, *tostream* and *append*). A prototype was built to evaluate the framework using Apache Spark. However, the solution is still a prototype and was not evaluated with other frameworks besides Spark.

Misale et al. [9] characterized the dataflow model used in Big Data frameworks from a more theoretical perspective. They also used the model to analyze some frameworks, such as Spark and Storm, and their user APIs. However, they did not addressed the implementation of the theoretical model.

In this work, we propose a software abstraction layer for Big Data processing tools in order to homogenize their form of usage and facilitate the implementation of smart city applications. Similar approaches were already developed for other application domains. For example, Crankshaw et al. [10] presented CLIPPER, a modular architecture that isolates user applications from the diversity of machine learning frameworks, offering a common interface (API) to access them. In CLIPPER, new frameworks can be added to the abstraction without changing the final user applications.

III. Comparison of Big Data Frameworks

A survey on Big Data processing tools was carried out in order to characterize and compare them. We have considered the most popular free software tools to real-time processing and whose development community is still active: Flink[5], Storm, Spark, Samza[6] , and Apex[7] – all from the Apache Foundation. Based on this study, we have also identified the common features of the tools, which were considered in the definition of the abstraction system proposed in this work.

Table I
Frameworks comparison: processing type, latency, throughput and consistency guarantees

Frameworks/ Features	Real-time processing	Latency	Throughput	Consistency guarantees
Apache Flink	Native	Low	High	Exactly-once
Apache Storm	Native Micro-batches with Storm Trident	Very low	High	Exactly-once (only for Trident)
Apache Spark	Micro-batches	Not proper for low latencies	High	Exactly-once
Apache Samza	Native	Low	High	At least once
Apache Apex	Native	Low	High	Exactly-once

Table II
Frameworks comparison: APIs and connectors

			Connectors	
Frameworks/ Features	**Written in**	**APIs**	**Integration with Kafka and Hadoop**	**Integration with RabbitMQ**
Apache Flink	Java, Scala	Declarative	Yes	Yes
Apache Storm	Java, Clojure	Compositional	Yes	No
Apache Spark	Java, Scala, Python, R	Declarative	Yes	Yes
Apache Samza	Java, Scala	Declarative	Yes	No
Apache Apex	Java, Scala	ApexStream - declarative DAG API - compositional	Yes	Yes

Table I characterizes the tools in terms of their data processing models, latency, throughput, and consistency guarantees. All tools have native support for real-time processing except for Apache Spark, which uses a micro-batch model. For this reason, Spark is not the most suitable to process data with low latencies. For consistence, the exactly-once guarantee is the strongest since it prevents data loss and duplication. Apache Samza does not support this guarantee, while Apache Storm only provides it in its Trident API, which uses micro-batches.

Table II characterizes the tools in terms of their APIs, connectors, and languages. All the analyzed tools can be integrated with Kafka[8] (a message broker) and HDFS. In the context of this work, RabbitMQ support is an important feature as this make the integration of the tool with the InterSCity platform easier, since RabbitMQ is the broker used by the platform. The tools with support to RabbitMQ are Flink, Spark, and Apex.

Apache Flink, Spark and Samza have declarative APIs, whereas Storm has a compositional API, which is lower level comparing with the declarative ones. In compositional APIs, the dataflows DAGs need to be explicitly defined, by declaring the processing nodes and their connection channels. Apex has a declarative (higher level) and also a compositional (lower

[2]https://spark.apache.org/
[3]http://hadoop.apache.org/
[4]http://storm.apache.org/
[5]https://flink.apache.org/
[6]http://samza.apache.org/
[7]https://apex.apache.org/
[8]https://kafka.apache.org

Figure 1. Microservices architecture

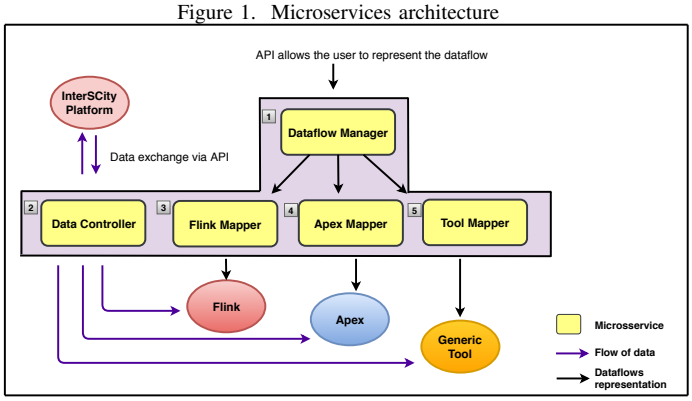
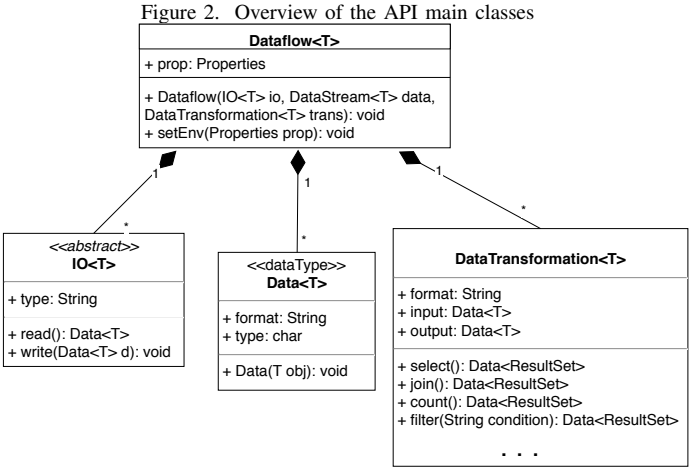


Figure 2. Overview of the API main classes



level) API, hence these two different degrees of abstraction can be applied by a user of this framework.

Apache Flink has two main declarative user APIs called DataStream and DataSet, applicable to streams and batches, respectively. Therefore, in some frameworks these two types of data processing are treated separately.

IV. An Architecture to Abstract Big Data Tools

The abstraction system proposed in this work provides an API for the specification of dataflow models. The basic building blocks in this interface are *activities*, *streams*, *data batches*, and *channels* for data transfers (between activities, from data sources to activities or from activities to sinks). An activity is a processing unit which receives input data and generates output data. The *dataflow process networks* are used as underlying theoretical model for the API building blocks, as proposed by Misale et al. [9].

The abstraction system also offers microservices that map dataflow models (specified through the API) into specific source codes to be executed in Big Data tools. More details about these microservices are presented in Section IV-A.

A. Integration with a Smart City Platform

The architecture of the abstraction system is illustrated in Figure 1. The architecture is defined on top of the smart city platform of the InterSCity project [4]. This platform has a microservice architecture. To facilitate the integration, the abstraction system proposed here is being created as new microservices in the platform.

InterSCity platform receives data from external systems, sensors, and other kind of devices through a microservice called `Resource Adapter`. Therefore, the abstraction system does not need to provide support for data collection, since the platform already supports it. To obtain data, the system needs to subscribe to a RabbitMQ topic created in the platform or request data through a microservice called `Data Collector`, which manages historical data storage. Thus, the system microservice `Data Controller` (in Figure 1) communicates with the platform's microservices to get the data to be processed.

The microservice `Dataflow Manager` is responsible for receiving dataflows provided by users, specified through the API. This microservice does a preliminary "generic" mapping before calling other microservices (e.g. `Flink Mapper`, `Apex Mapper`, and `Tool Mapper`) responsible for the specific mappings. These microservices convert the dataflow into input formats readable by the Big Data tools. The microservice `Tool Mapper` illustrates the flexibility of the proposed abstraction, which can be extended to other existing Big Data frameworks or new ones that may arise.

B. The API for Dataflow Specification

Big Data processing tools need to support transformations on data. For this, they provide some built-in operators and support to user defined functions (UDF). In the sequence, we describe the operators identified as the most common in these tools and, as consequence, chosen to be included in the API to support the specification of dataflows.

Seeing that the tools use the MapReduce model or another model based on it, the operators to perform mapping and reduction are fundamental. Additionally, operations used in databases are also present in Big Data frameworks. Aggregation functions that transform input data, grouping the data in such a way as to summarize them considering some criterion, are an example of this kind of operations. Other operations such as selection, join, grouping, and filters are also available in the Big Data frameworks.

Data processing include reading data, applying transformations on them, and writing the results in the output. These elements are modeled in the UML class diagram of the proposed API, shown in Figure 2 [9].

Users interact with the API through the class `Dataflow<T>`. An instance of `Dataflow<T>` created by a user is composed of input and output (instances of implementations of `IO<T>`), data (instances of implementations of `Data<T>`) and their transformations (instances of `DataTransformation<T>` class). The

[9]In Table II, it is possible to note that the Java language is used in the implementation of all Big Data tools evaluated in this work. Moreover, new updates for the tools become available first in this language. This is why we have chosen Java for the implementation of the abstraction system.

Figure 3. Classes for input and output connectors

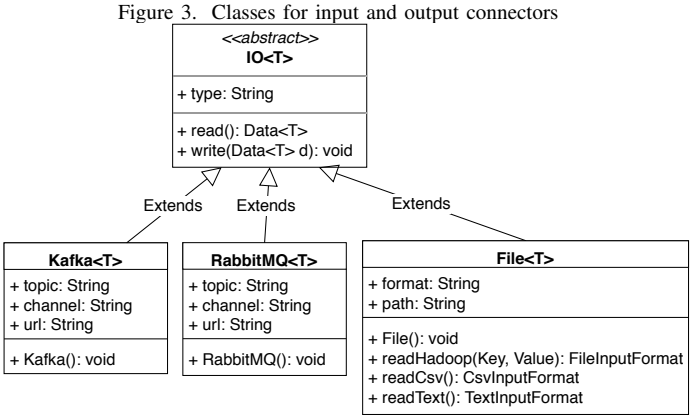
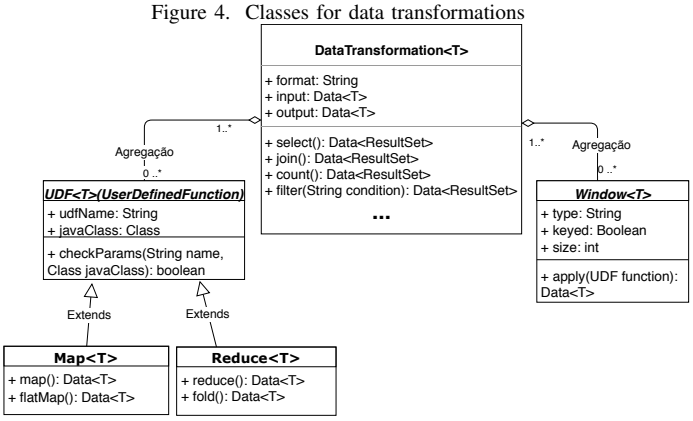


Figure 4. Classes for data transformations



execution environment of a dataflow can be configured through the `setEnv` method.

The abstract classes in Figure 2 were designed to allow the API to be expanded in a transparent way to end users. For example, to include a new type of data source, one can create a new implementation of the abstract class `IO<T>`. The same applies to data output. The API must have connectors for the most commonly used types of IO, such as Kafka, RabbitMQ, HDFS, and files (JSON, text, CSV), as shown in Figure 3.

There are two main categories of data transformations, as shown in the class diagram of Figure 4. The first one is the user defined functions (instances of the class `UDF<T>`) which consist in functions implemented by the user of the abstraction. Map and Reduce functions are examples of user defined functions. The second category is the window operators (class `Window<T>`), generally used to group data according to temporal windows in stream processing.

C. Validation and Analysis

The implementation of the proposed abstraction system is a work in progress[10]. As proof of concept, mapper microservices for two different Big Data frameworks are being implemented. An application for smart cities in the domain of urban mobility will be used as case study. The application will process real data from the city of São Paulo, Brazil. Through its implementation and execution, it will be possible to evaluate the API and the abstraction system.

[10]The source code of the abstraction system is available at https://github.com/nandacamargo/abstraction-layer, under the Apache License.

The evaluation of the Big Data tools (Section III) has shown that the best options to be considered in the proof of concept are Flink and Apex. Both frameworks can handle real-time and batch data. In addition, Flink architecture based on layers with different levels of abstraction, as well as the Apex Malhar operators library facilitate the mapping implementation.

V. Conclusion and Future Works

The ever-increasing volume of data collected in cities and the need for rapid responses resulting from the processing of these data make clear the importance that Big Data frameworks have in contemporary society. Aiming the use of a variety of frameworks with a smaller learning curve for their users, it is essential to develop an abstraction that homogenizes the access to the different features provided by the tools.

In this work, we compared popular open-source Big Data tools to identify their common features and, based on that, we defined the main operators the abstraction should have. We proposed an API to support the specification of dataflows with these operators. Moreover, we presented a microservices architecture on top of a smart city platform, to map the dataflows to different Big Data frameworks.

Our ongoing work includes the implementation of mapper microservices and the evaluation of the system by means of a smart city application which processes real data from urban mobility of São Paulo, Brazil.

References

[1] SPTrans, "SPTrans: São Paulo bus fleet data," http://www.sptrans.com.br/rede-onibus/, June 2018.

[2] Y. Zhang, T. Cao, S. Li, X. Tian, L. Yuan, H. Jia, and A. V. Vasilakos, "Parallel processing systems for big data: a survey," *Proceedings of the IEEE*, vol. 104, no. 11, pp. 2114–2136, 2016.

[3] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *ACM Sigmod Record*, vol. 34, no. 3, pp. 44–49, 2005.

[4] A. D. Esposte, F. Kon, F. M. Costa, and N. Lago, "InterSCity: A scalable microservice-based open source platform for smart cities," *Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems*, 2017.

[5] Y. Gong, P. Rimba, and R. Sinnott, "A big data architecture for near real-time traffic analytics," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. ACM, 2017, pp. 157–162.

[6] K. Taneja, Q. Zhu, D. Duggan, and T. Tung, "Linked enterprise data model and its use in real time analytics and context-driven data discovery," in *Mobile Services (MS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 277–283.

[7] D. Dissanayake and K. Jayasena, "A cloud platform for big iot data analytics by combining batch and stream processing technologies," in *Information Technology Conference (NITC), 2017 National*. IEEE, 2017, pp. 40–45.

[8] H. Cho, H. Shiokawa, and H. Kitagawa, "Jsflow: Integration of massive streams and batches via json-based dataflow algebra," in *Network-Based Information Systems (NBiS), 2016 19th International Conference on*. IEEE, 2016, pp. 188–195.

[9] C. Misale, M. Drocco, M. Aldinucci, and G. Tremblay, "A comparison of big data frameworks on a layered dataflow model," *Parallel Processing Letters*, vol. 27, no. 01, p. 1740003, 2017.

[10] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system." in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 613–627.