Abstracting Big Data Processing Tools for Smart Cities

Fernanda de Camargo Magano
and Kelly Rosa Braghetto

Workshop on the Distributed Smart City (WDSC’2018)
Computer Science Department – IME USP

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.

October 2nd, 2018
Index

1 Introduction

2 Concepts
   Big Data Processing
   Big Data Tools’ APIs

3 Related Works

4 Comparison of Big Data Frameworks

5 An Architecture to Abstract Big Data Tools
   Integration with a Smart City Platform
   API for Dataflow Specification
   Validation and Analysis

6 Conclusion Remarks

7 References
Context

Urban Big Data

- Evolution of Internet of Things and cheaper technology
- Participatory sensing (mobile phones, social network, among others)
- Large volumes of data from heterogeneous sources
- Important role of data processing and analysis for smart cities
Problem

Big Data tools

- Have good resources, but are **hard to be used** by data scientists or developers beginners to these frameworks
- Require from their users **knowledge** in programming, parallel and distributed computing
- Have not standardized languages and, therefore, are not completely **interoperable**
Goals

The main goal of this work is to make the use of Big Data processing frameworks easier for smart cities applications, by abstracting the specificities of these tools. For this, we propose:

- An interface (API) to specify dataflows for processing data in real time and batches
- A software system that integrates a smart cities platform with Big Data processing frameworks, using the proposed API and developing mappers for different tools
Big Data Processing Tools and the Dataflow Model

- Several tools share almost the same basic concepts
- **Apache tools**: Storm, Spark, Apex, Flink and Samza
  - **Open source**, with **active communities** and widely used
- They use the **Dataflow Model**
  - Expressive model: describes **batches, micro-batches and streams**
  - **Directed graph** to represent data dependencies
Real-time Dataflow Example

Figure 1: Best route selector application for smart cities
User APIs: Declarative and Topological

Declarative:

- High level
- Expressed as methods of objects representing collections
- Advanced operations (e.g., state and windows managing)

Topological or compositional:

- Programs expressed using graphs
- Explicit connection among nodes
- Specification of the code executed by the nodes
# Related Works

### Cho, Shiokawa, and Kitagawa (2016)

- **JSFlow** framework: uses dataflow algebra based in JSON
- Extends Jaql – a declarative and functional language
- **Disadvantage:** prototype only uses Spark

### Misale et al. (2017)

- Describes the dataflow model and user APIs
- **Disadvantage:** theoretical work
Comparison of Big Data Frameworks

- The **features** were chosen based on smart cities applications
- The **comparison** led to the proposed abstraction and API

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

**Table 1:** Tools comparison - processing and consistency guarantees
### Comparison of Big Data Frameworks

#### Table 2: Tools comparison - APIs and connectors

<table>
<thead>
<tr>
<th>Frameworks/Features</th>
<th>Written in</th>
<th>APIs</th>
<th>Connectors</th>
<th>Integration with RabbitMQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Flink</td>
<td>Java, Scala</td>
<td>Declarative</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Apache Storm</td>
<td>Java, Clojure</td>
<td>Compositional</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Apache Spark</td>
<td>Java, Scala, Python, R</td>
<td>Declarative</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Apache Samza</td>
<td>Java, Scala</td>
<td>Declarative</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Apache Apex</td>
<td>Java, Scala</td>
<td>ApexStream - declarative DAG API - compositional</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks Architecture Structure

- Frameworks have a **core** layer
- Provide user **APIs**
- Offer **libraries** (for IO operators, SQL, ML)
- Can run above **Hadoop** ecosystem

![Figure 2: Example of framework architecture – Apache Apex](http://dt-docs.readthedocs.io/en/latest/rts/)

---

Apache Apex - Dataflows representation

**Code 1: Word count - DAG API**

```java
LineReader lineReader = dag.addOperator("input", new LineReader());
Parser parser = dag.addOperator("parser", new Parser());
UniqueCounter counter = dag.addOperator("counter", new UniqueCounter());
ConsoleOutputOperator cons = dag.addOperator("console", new ConsoleOutputOperator());

dag.addStream("lines", lineReader.output, parser.input);
dag.addStream("words", parser.output, counter.data);
dag.addStream("counts", counter.count, cons.input);
```
Apache Apex - Dataflows representation

Code 2: Word count - ApexStream API

StreamFactory.fromFolder("/tmp")
    .flatMap(input -> Arrays.asList(input.split(" ")),
        name("Words"))
    .window(new WindowOption.GlobalWindow(),
        new TriggerOption().accumulatingFiredPanes()
            .withEarlyFiringsAtEvery(1))
    .countByKey(input -> new Tuple.PlainTuple<>(new
        KeyValPair<>(input, 1L)), name("countByKey"))
    .map(input -> input.getValue(), name("Counts"))
    .print(name("Console"))
    .populateDag(dag);
Our Proposed Architecture

- The architecture meets the **goals** of this project by including an interface (**API**) and a **software system**.

![Diagram of proposed microservices architecture]

**Figure 3: Proposed microservices architecture**
Integration with InterSCity Platform

Figure 4: Smart-cities platform architecture

\(^2\text{Image from https://gitlab.com/interscity/interscity-platform}\)
Our API for Dataflow Specification

![UML class diagram of the proposed API (simplified)](image)

**Dataflow<T>**
- prop: Properties
- Dataflow(IO<T> io, DataStream<T> data, DataTransformation<T> trans): void
- setEnv(Properties prop): void

**IO<T>**
- type: String
- read(): Data<T>
- write(Data<T> d): void

**Data<T>**
- format: String
- type: char
- Data(T obj): void

**DataTransformation<T>**
- format: String
- input: Data<T>
- output: Data<T>
- select(): Data<ResultSet>
- join(): Data<ResultSet>
- count(): Data<ResultSet>
- filter(String condition): Data<ResultSet>

Figure 5: UML class diagram of the proposed API (simplified)
Abstraction - Input and Output

Figure 6: Classes for input and output connectors
Abstraction - Data Transformation

Figure 7: Classes for data transformations
Validation

- Urban mobility **case study**
- Compare **implementation codes** with and without using the abstraction (directly done using the Big Data tools)
- Use of **metrics** to measure API usability (Scheller e Kühn, 2015)
Case Study - Application to Predict Bus Arrival Time

Figure 8: Dataflow of the server system
Conclusion Remarks

Our contributions in this work are

- A **comparison** among different Big Data tools
- The proposal of an **API** to support the specification of dataflows
- 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
- The **evaluation** of the system by means of a smart city application which processes urban mobility data

Abstracting Big Data Processing Tools for Smart Cities

Fernanda de Camargo Magano, Kelly Rosa Braghetto
fernanda.magano@usp.br kellyrb@ime.usp.br

Open source code at GitLab:
https://gitlab.com/interscity/abstraction-layer

InterSCity website:
http://interscity.org