Future Generation Computer Systems 52 (2015) 22–36

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Improving the predictability of distributed stream processors

P. Basanta-Val [a,*], N. Fernández-García [b], A.J. Wellings [c], N.C. Audsley [c]

[a] Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Avda de la Universidad no 30, Leganes, 28911, Madrid, Spain
[b] Centro Universitario de la Defensa. Escuela Militar Marin, Universidade de Vigo, Spain
[c] Department of Computer Science, University of York, York, YO10 5GH, UK

HIGHLIGHTS

- Model combining stream processing technology and real-time.
- Extensions to the Storm processor.
- Performance evaluation of the extension on a cluster.

ARTICLE INFO

Article history:
Received 14 October 2014
Received in revised form
1 February 2015
Accepted 27 March 2015
Available online 14 May 2015

Keywords:
Real-time
Distributed stream processing
Predictable infrastructure

ABSTRACT

Next generation real-time applications demand big-data infrastructures to process huge and continuous data volumes under complex computational constraints. This type of application raises new issues on current big-data processing infrastructures. The first issue to be considered is that most of current infrastructures for big-data processing were defined for general purpose applications. Thus, they set aside real-time performance, which is in some cases an implicit requirement. A second important limitation is the lack of clear computational models that could be supported by current big-data frameworks. In an effort to reduce this gap, this article contributes along several lines. First, it provides a set of improvements to a computational model called distributed stream processing in order to formalize it as a real-time infrastructure. Second, it proposes some extensions to Storm, one of the most popular stream processors. These extensions are designed to gain an extra control over the resources used by the application in order to improve its predictability. Lastly, the article presents some empirical evidences on the performance that can be expected from this type of infrastructure.

1. Introduction

Current trends in computational infrastructures look at the Internet as a low-cost distributed-computing platform for hosting legacy and next generation applications in the cloud [1–3]. The benefits offered by the use of an infrastructure such as the Internet are diverse: increased computational power, higher availability in 24 × 7 periods, and reduced energy consumption [2,4,5]. One type of application that may potentially benefit from this low-cost execution platform is big-data systems.

A big-data system processes a collection of data that is difficult to process using traditional techniques and, thus, requires specific processing tools. According to [4], the main reasons of this difficulty can be characterized using three V's: The first V is for volume of processed data, the second V is for variety in the data (that is, it can come from different sources and be represented using heterogeneous formats), and the last V refers to velocity (meaning that applications work with data produced at high rates). Typically, big-data applications run analytics on clusters of machines connected to the Internet [6–11]. Each big-data analytic refers to how the data is analyzed to produce a desired output. Typical application domains include meteorology, genome processing, physical simulations, biological research, finance, and Internet search.

Nowadays, to develop these big-data applications, practitioners use different software frameworks, including Hadoop [12], Storm [13,14], and Spark [15] which emphasize different programming aspects of the big-data ecosystem. Hadoop is focused on batch processing of large data sets on commodity machines. The typical deadline for Hadoop applications ranges from minutes to weeks; often processing Petabytes [16] of data. Apache Storm works on a streaming data model and it is targeted to online applications with sub-second deadlines. Lastly, Spark offers optimized

* Corresponding author.
E-mail addresses: pbasanta@it.uc3m.es (P. Basanta-Val), norberto@cud.uvigo.es (N. Fernández-García), andy@cs.york.ac.uk (A.J. Wellings), neil@cs.york.ac.uk (N.C. Audsley).

http://dx.doi.org/10.1016/j.future.2015.03.023
0167-739X/

I/O access, reducing computational times in comparison with Hadoop but with heavier computational costs than Storm [16].

Another feature of big-data applications is that they often have real-time requirements that need to be met in order to provide applications with timely response [8,17]. For instance, the Hadron collider outputs a 300 Gb/s stream that has to be filtered to 300 Mb/s for storage and later processing. Some data mining applications, like, for instance, those used for on-line credit card fraud detection, and contextual selection of web advertisements may benefit from real time techniques. In these applications having shorter response-time usually has an economic impact. This is also the case of high frequency trading systems [3,18]. Another simple example of big-data analytic that faces temporal restrictions is the trending topic detection algorithm used in Twitter to show a dynamic, up-to-date, list with the most popular hashtags. The Twitter trending topic detection application is the case of a big-data scenario characterized by the "velocity", as tweets are small pieces of data (140 characters), but are produced at a high rate. In all these cases the response has to meet application deadlines. Furthermore, in many cases big-data applications requirements are complex and may demand the coexistence of several quality-of-service restrictions on a single big-data application.

The common denominator in all these applications is that they may benefit from techniques included in real-time systems to increase the predictability of the infrastructure and also reduce the number of required resources. These types of techniques are well known [19] and may be integrated into the computational infrastructures to have fine-grained control on the number of machines used by an application and in determining an execution timeline for applications with different types of quality.

Most popular frameworks like Hadoop and Storm were designed for the general purpose domain and are inefficient for real-time application development. They provide simple models with minimum parameters that enable simple forms of parallel computation, relying on the map/reduce and stream programming models. They do not explicitly assign different parts of the application to different physical nodes, which is typically required in real-time applications. They also lack the notion of scheduling parameters enforced in different computational nodes. In their current forms, both technologies are inefficient when worst-case analysis is used to determine the worst-case computation times. Fortunately, both technologies may increase their predictability by integrating predictable computational models within their cores. But although some steps have been taken in increasing predictability in big-data infrastructures [20], there are still some important pending issues in the path towards commercial implementations.

In this context, the main goal of this article is to increase the predictability of the stream processing model, in which Storm is inspired, including real-time capabilities. The approach follows the guiding principles that inspired real-time Java [21,22], keeping backward compatibility with plain Storm, allowing plain and predictable coexistence in a single machine. Thus, the real-time Storm is able to run traditional and real-time applications in the same computing cluster (see Fig. 1).

The main improvement in the predictability of Storm in this article is the characterization of the streams of Storm as real-time entities that can be scheduled using existing real-time scheduling theory. This characterization allows reasoning about the characteristics of a real-time application in terms of deadlines, and determining the number of machines required for its implementation.

The rest of the article is organized as follows. Section 2 explores other similar frameworks and their relationships with the real-time stream processing model from the perspective of general purpose and real-time systems. Section 3 defines a simple computational model for real-time stream processing, which is mapped to the Storm framework later in Section 4. Section 5 shows an application developed with this new infrastructure. Section 6 provides practical evidence on the performance one may expect from this type of infrastructure, showing how to deduce the deadlines of the applications from their real-time characterization. Lastly Section 7 highlights the main contributions and our most related ongoing work.

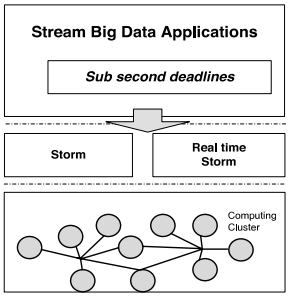

Fig. 1. A framework for real-time big-data stream processing.

2. Related work

2.1. Stream processing technology

Processing high-volume streams with low-latency has traditionally required the development of ad-hoc solutions [23]. Usually, these solutions are expensive to implement, difficult to maintain and are tailored to particular application scenarios, which limit their reusability [24]. To address these limitations and support the development of stream processing applications, several proposals of Stream Processing Engines (like Aurora [25], STREAM [26], Borealis [27], IBM's Stream Processing Core [28], and SEEP [29]) appeared in the state-of-the-art.

However, the emergence of new application scenarios, like high frequency trading, social network content analysis, sensor-based monitoring and control applications, and other low-latency big-data applications, has greatly increased the demands on this kind of stream processing platforms. As indicated in [30–32], there is a demand of general-purpose, highly scalable stream computing solutions that can quickly process vast amounts of data.

Taking this into account, it is not surprising to find in the recent state-of-the art several proposals for low-latency big-data stream processing systems, like S4 (Simple Scalable Streaming System) [13], Storm [33], or Spark Streaming [34,35], some of them backed by important Internet companies like Yahoo (S4) or Twitter (Storm). These systems are designed as general-purpose platforms that can be run on clusters of commodity hardware. Using specific programming interfaces, developers can implement scalable stream processing applications on top of them, taking advantage of the functionalities provided by the platform: information distribution, cluster management, or fault-tolerance.

All these major approaches have been designed with the general purpose performance in mind and do not provide facilities for real-time performance, like increased architecture awareness, low-level access facilities, and deadline characterization. However, most of them may be easily extended via new programming interfaces developed to increase their scalability and fault tolerance. For instance, Storm processor includes the pluggable scheduler concept that may be extended to include different scheduling policies [36] increasing adaptability. Similar architectural benefits have been exploited in this article to pin the execution graphs of the streams to specific machines of a cluster.

2.2. Real-time support for stream processing

Big-data infrastructures like Hadoop and Storm lack efficient implementations to run multiple concurrent real-time applications with different quality-of-service requirements, because they