Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 49-30 (2016) 114–119

Strategies for Big Data Analytics through Lambda Architectures in Volatile Environments

Veith, Alexandre da Silva * **Anjos, Julio C. S. dos** * **de Freitas, Edison Pignaton** * **Lampoltshammer, Thomas J.** ** **Geyer, Claudio F.** *

* *Federal University of Rio Grande do Sul (UFRGS), Porto Alegre -RS - P.O.Box 15064 - Brazil (e-mail: alexandre.veith@ufrgs.br and {jcsanjos, edison.pignaton, geyer}@inf.ufrgs.br)*

** *Danube University Krems, Department for E-Governance and Administration, Dr.-Karl-Dorrek-Str. 30, 3500 Krems, Austria (e-mail: thomas.lampoltshammer@donau-uni.ac.at)*

Abstract:
Expectations regarding the future growth of Internet of Things (IoT)-related technologies are high. These expectations require the realization of a sustainable general purpose application framework that is capable to handle these kind of environments with their complexity in terms of heterogeneity and volatility. The paradigm of the Lambda architecture features key characteristics (such as, robustness, fault tolerance, scalability, generalization, extensibility, ad-hoc queries, minimal maintenance, and low-latency reads and updates) to cope with this complexity. The paper at hand suggest a basic set of strategies to handle the arising challenges regarding the volatility, heterogeneity, and desired low latency execution by reducing the overall system timing (scheduling, execution, monitoring, and faults recovery) as well as possible faults (churn, no answers to executions). The proposed strategies make use of services such as migration, replication, MapReduce simulation, and combined processing methods (batch- and streaming-based). Via these services, a distribution of tasks for the best balance of computational resources is achieved, while monitoring and management can be performed asynchronously in the background.

Keywords: Internet of Things (IoT), Scheduling, Batch processing, Stream processing, Cloud computing, Grid computing

1. INTRODUCTION

The concept of the Internet of Things (IoT) can be described as the seamless fusion of virtual environments and contained objects with their real-world counterparts (Uckelmann et al., 2011). In return, this makes the creation of robust, flexible, and dynamic applications imperative, in order to handle heterogeneous and volatile environments. A major aspect in this regard is represented by the challenge to handle vast amounts of data, including all relevant processing steps, in particular data analytics. Due to this fact, the area of big data analytics has attracted high levels of attention of industry and academia alike. This fact is represented by the total increase of data-driven projects by 125% during the period 2014-2015. [1] So far, the majority of big data deployments were initially using batch processing-oriented approaches (i.e. the entire amount of data is gathered, stored, and afterwards processed step-by-step) (Hu et al., 2014). However, *batch processing* has no support for low-latency scenarios. Thus, a new model called *stream processing* or *oriented-to-events processing* has witnessed a huge increase in volume and availability (Tudoran et al., 2014). The handled events are usually characterized by a small unit size (in the order of kilobytes), but have overwhelming collection rates, due to the continuous data flow. To overcome this issue, new *stream processing* frameworks have emerged, such as Apache Storm, Spark, Flink or S4.

Over time, *stream processing* and its associated processing engines have evolved up to the point of the introduction of the Lambda Architecture (LA) paradigm (Marz, 2013). Lambda Architectures are designed to handle vast amounts of data in conjunction with both *batch* and *stream processing* methods. While *batch processing* helps to reduce latency, to improve data transfer, to provide fault-tolerance, as well as a comprehensive and accurate view upon the data, *stream processing* provides capabilities to deal with real-time data. Thus, the rise of LAs are directly related to the rapid growth of Big Data real-time analytics.

According to Ewen et al. (2013), the Lambda Architecture paradigm is the starting point of the so-called 4^{th} *Generation of Data Processing Engines* that comprise several features regarding the design and implementation of processing engines for massive data, such as robustness,

[1] IDG - `http://www.idgenterprise.com/`

10.1016/j.ifacol.2016.11.138

fault tolerance, low latency of reading and updating, scalability, generalization, extensibility, ad-hoc queries, and minimal maintenance. To achieve these properties, the architecture foresees a Big Data system to be constructed in several layers. Anjos et al. (2015) presented the SMART platform, which is a modular framework for Big Data analysis. SMART considers a large variety of data sources, such as distributed datasets and social networks, where there is a clear need for standardization. The Dispatcher module (DM) in the SMART platform is an orchestration system that needs several policies to the managing data and tasks. This paper aims at the improvement of the decision-making engine of the Dispatcher module based on the computational capacity of the machines via scheduling strategies and advanced execution setups regarding data streams to achieve an optimal distribution of tasks for the best balance of computational resources.

This paper is structured as follows: Section 2 sets out the state-of-the-art for data-intensive computation, establishes the framework in this landscape, compares it to others and demonstrates how it works in relation to others considering heterogeneous infrastructures, hybrid infrastructures, and hybrid engines. Section 3 presents the SMART platform and its main modules. Section 4 details the Dispatcher Module and discusses the strategies required to overcome the environmental limitations of this module. Section 5 concludes the paper and reports opportunities for future work.

2. RELATED WORK

2.1 Heterogeneous Infrastructures

JetStream is a set of strategies for efficient transfers of events between cloud data centers (Tudoran et al., 2014). *JetStream* is self-adapting regarding streaming conditions. It aggregates the available bandwidth and enables the routing of data through cloud sites. The study by Tudoran et al. (2014) focuses on event transfers between inter- and intra-nodes. The authors propose an adaptive cloud batching in form of an algorithm that aggregates the streams in batches, resulting in latency reduction. However, it just considers the latency and not the volatility. The work concentrates on environments where the computational resources can break away unexpectedly and the scheduling policies must be adapted to it.

SMART (Anjos et al., 2015) is a platform that offers an efficient architecture for Big Data analysis applications for small and medium-sized organizations. Its implementation considers heterogeneous data sources and aims at data analysis scenarios in geo-distributed environments, considers cost, fault tolerance, network overhead, I/O throughput, as well as the minimization of data transfers between computational resources. Yet, these parameters are not enough to work with volatile environments, especially regarding *stream processing*. To overcome these environment limitations, it is necessary incorporate information from physical components, such as memory, CPU speed, and storage. Thus, the overall capacity impacts the overall performance. In addition, regarding the volatility, replication must be incorporated as a fault control mechanism.

Similarly, Pham et al. (2016) purpose a generic, extensible, scalable, fine-grained, and re-configurable multi-cloud framework. It is based on a lightweight kernel and provides a hierarchical Domain Specific Language (DSL). The DSL allows for a fine-grained level of administration. However, the proposed solution does not control the workload at the nodes and possible faults, as the Deployment Manager is just an interface to set the devices and the Virtual Machines (VMs).

2.2 Hybrid Infrastructures

BIGhybrid summarizes the main features of a Hybrid MR environment based on the merge of two environments, namely a Cloud (MR-BlobSeer) environment and a Desktop Grid (BitDew-MapReduce) environment (Anjos et al., 2016). The Global Dispatcher located outside the Desktop Grid (DG) as well as of the cloud environment features middleware functionality for handling task assignments and input data from users. It is a distributed data storage system that manages policies for data splitting and distribution in batch applications such as MapReduce. The working principle is similar to the publish/subscribe service, where the system obtains data and publishes the computed results. This approach has several drawbacks, if applied to *stream processing*, due to delays regarding the processing of responses.

HybridMR is a model for the execution of MapReduce on hybrid computation environments (Cloud and DG) developed by Tang et al. (2015). Two innovative solutions are proposed to enable such large-scale data-intensive computation: (i) HybridDFS, which is a hybrid distributed file system. HybridDFS features reliable distributed storage that alleviates the volatility of desktop PCs (i.e., fault tolerance and file replication mechanism); and (ii) a Node priority-based fair scheduling (NPBFS) algorithm has been developed to achieve both data storage balance and job assignment balance by assigning each node a priority through quantifying CPU speed, memory size, as well as input and output capacity. The NPBFS approach is very interesting because it uses some miscellaneous environment variables to schedule the tasks. Although regarding *stream processing*, its application is not possible due to the high flow latency. *HybridMR* just uses the flow rate to deploy the tasks, however, the rate does not consider particular task information.

2.3 Hybrid Engines

Apache Spark is a framework introduced by Zaharia et al. (2012) that uses resilient distributed datasets (RDDs) and enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that are designed to allow users to keep intermediate results in memory, control their partitioning to optimize data placement, and manipulate them through a valuable set of operators. Liao et al. (2015) presented some scheduling inefficiencies related to the time window that constructs the RDD. The batch interval needs to be dynamically adjusted, so that fluctuations within the data rate can be handled in a production environment and the total delay of every event can be controlled within a certain range for real-time scenarios.