Future Generation Computer Systems 43–44 (2015) 1–11

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

High frequency batch-oriented computations over large sliding time windows☆

Leonardo Aniello *, Leonardo Querzoni, Roberto Baldoni

Cyber Intelligence and Information Security Research Center, University of Rome "La Sapienza", Via Ariosto 25, 00185, Rome, Italy
Department of Computer, Control, and Management Engineering Antonio Ruberti, University of Rome "La Sapienza", Via Ariosto 25, 00185, Rome, Italy

HIGHLIGHTS

- We present a batch-oriented approach for time window computations.
- We analyze the impact of input data organization on computation performances.
- We define a set of strategies for smartly organizing input data.
- We implement these strategies on Hadoop/HDFS framework.
- We present experimental evaluations about the effectiveness of this solution.

ARTICLE INFO

Article history:
Received 26 April 2013
Received in revised form
10 September 2014
Accepted 19 September 2014
Available online 10 October 2014

Keywords:
Event processing
Batch processing
Time window based computations
Data analytics
Big data

ABSTRACT

Today's business workflows are very likely to include batch computations that periodically analyze subsets of data within specific time ranges to provide strategic information for stakeholders and other interested parties. The frequency of these batch computations provides an effective measure of data analytics freshness available to decision makers. Nevertheless, the typical amounts of data to elaborate in a batch are so large that a computation can take very long. Considering that usually a new batch starts when the previous one has completed, the frequency of such batches can thus be very low.

In this paper we propose a model for batch processing based on overlapping sliding time windows that allows to increase the frequency of batches. The model is well suited to scenarios (e.g., financial, security etc.) characterized by large data volumes, observation windows in the order of hours (or days) and frequent updates (order of seconds). The model introduces multiple metrics whose aim is reducing the latency between the end of a computation time window and the availability of results, increasing thus the frequency of the batches. These metrics specifically take into account the organization of input data to minimize its impact on such latency. The model is then instantiated on the well-known Hadoop platform, a batch processing engine based on the MapReduce paradigm, and a set of strategies for efficiently arranging input data is described and evaluated.

1. Introduction

Event processing is a constantly evolving research area which keeps growing to adapt to emerging technologies and paradigms [1]. With event processing, events produced by possibly different sources are usually collected in batches delimited by time windows and then elaborated to produce other events as output. Several real applications require, indeed, to recognize particular patterns within specific time lapses or to produce periodic reports on what happened in precise time ranges. A relevant example for the first case is represented by Intrusion Detection Systems (IDSs), which keep monitoring network traffic data searching for known malicious signatures, trace them and raise alerts whenever too many suspect activities occur within a defined time interval. Port scan detection techniques based on the activities observed in specific time windows are investigated in [2] and [3]. In this scenario, where a large number of network probes can produce high rate event streams, it is advisable to adopt large time windows (hours, days) to catch *slow attacks*; at the same time, it is necessary

☆ This is an extended version of a paper from the same authors appeared at SAC 2013.

* Corresponding author at: Department of Computer, Control, and Management Engineering Antonio Ruberti, University of Rome "La Sapienza", Via Ariosto 25, 00185, Rome, Italy. Tel.: +39 3485668071.
E-mail addresses: aniello@dis.uniroma1.it (L. Aniello), querzoni@dis.uniroma1.it (L. Querzoni), baldoni@dis.uniroma1.it (R. Baldoni).

http://dx.doi.org/10.1016/j.future.2014.09.008
0167-739X/

to have frequent (every few seconds) analysis results to promptly react to alarms. An example of the second case is represented by algorithmic trading, an application scenario where result latency is critical to profitability. Such scenario is characterized by large data volumes (millions of trades per day), medium to large observation windows (hours, days) and frequent (down to a second) updates to quickly catch highly volatile financial opportunities. We refer to this kind of event processing as Time Window Based Computations (TWBCs).

Event processing engines managing TWBCs must cope with an ever increasing number of event sources and continuously growing data rates. To keep up with this trend, processing engines must be able to manage *huge input data volumes*. Output of such computations is often used to take the best decision about some next action to be performed. Therefore, it is crucial to get these results as soon as possible, otherwise they are likely to become obsolete before they can be actually used. To cope with this requirement, processing engines must be *timely* in the production of their output.

Event processing engines can adopt two possible approaches for TWBCs with respect to the relationship between when events arrive and when they are elaborated. If events are processed as soon as they enter the engine, we talk about *online* event processing. Conversely, if events are first stored and then periodically processed in batches, we talk about *batch* event processing. This latter approach is usually preferred when timeliness requirements are not that strict; an inner characteristic of batch processing resides, indeed, in the delays to be paid in order to get updated results, because of its periodical nature. On the other hand, running computations periodically allows to cope with load spikes, failures and imbalances much more easily than the online approach does. Furthermore, a batch approach enables the decoupling of data loading and data elaboration, which provides higher flexibility to accommodate for possible distinct requirements.

Batch processing is heavily employed within business workflows of many medium to large companies for periodical ETL (Extract, Transform, Load) operations where large data sets produced daily up to hourly have to be moved, analyzed and archived so as to provide the proper means for enforcing specific business intelligence strategies. These scenarios are representative examples of the challenges and opportunities that the emerging BigData trend is fostering. Some well known companies that are employing this kind of approach are Oracle [4], Dell [5], MicroStrategy [6] and Cisco [7].

In this paper we focus on the batch approach and investigate what are its pros and cons in order to assess whether batch-oriented computation frameworks can properly meet TBWC requirements. More specifically, *we study how to reach the desired frequency of batch computations in a given application scenario. This is done through the introduction of a model based on overlapping sliding time windows. The model points out metrics that, if well-tuned, can reduce the output latency* (i.e., the time between the end of a computation time window and the availability of computation results). We also carry out an analysis of the impact of input data organization on these metrics showing how a smart subdivision of incoming events in data batches can help in minimizing the output latency. Finally, an implementation of the model in the well-known Hadoop framework is proposed. The implementation includes several input data organization strategies aiming at reducing output latency.

The rest of the paper is structured as follows: Section 2 discusses the related work, Section 3 introduces batch processing in TWBCs and our model, discussing the related performance metrics and two possible strategies for arranging input data; Section 4 describes the instantiation of the model in the Hadoop batch processing framework and an ad-hoc strategy for improving performances; Section 5 discusses the experimental evaluation results; finally, Section 6 concludes the paper.

2. Related work

The developments in the area of distributed event processing happened during last decade have been based mostly on the concept of continuous queries, which run unceasingly over streams of events provided by external sources. These queries are compiled in a network of processing elements that can be distributed over available resources. Several projects explored this line, although the structures used to model the compiled query are named differently. Among the most cited, we find InfoSphere [8,9] (networks of InfoPipes), Aurora [10,11] (networks of processing boxes), TelegraphCQ [12] (networks of dataflow modules), STREAM [13] (query plans composed by operators, queues and synopses), Borealis [14] (networks of query processors), and System S [15] (Event Processing Network (EPN) of Event Processing Agents (EPA)).

In this paper we adopt the jargon introduced by the latter. The reconfiguration of an EPN at runtime introduces several issues. The main one is the rebalancing of the load among nodes.

Shah et al. [16] define a dataflow operator called flux, which is integrated in an EPN and takes care of repartitioning stateful operators while the processing is running. Its limitations concern the dependence on configuration parameters that need to be tuned manually and the lack of fault tolerance mechanisms.

Gu et al. [17] propose a mechanism to process Multiway Windows Stream Joins (MWSJs) which distributes tuples to distinct nodes to allow for parallel processing. Their algorithm is specific for MWSJs. Xing et al. [18] describe an algorithm for placing operators such that no replacement is required at runtime. They deem that operators cannot be moved at all. Xing et al. [19] introduce a load distribution algorithm for minimizing latency and avoiding overloading by minimizing load variance and maximizing load correlation. Liu et al. [20] propose a dynamic load balancing operator for stateful algorithm, which spills state to disk or moves the operators to other nodes to resolve imbalances. Lakshmanan et al. [21] present a stratified approach where the EPN is horizontally partitioned in strata and operators can be moved within a single stratum only.

Stateful operators in a continuous query pose the question of addressing the problem of memory constraints. The most studied case is the one considering joins over distinct event flows or data streams, which requires the usage of some time or count based window in order to avoid maintaining the whole history of input data. Time windows cannot guarantee a consequent bound on required memory because of the variability of input event rate. Employing load shedding as a solution [22–25] could not be feasible in several scenarios where the accuracy of the processing is a main requirement, for example decision support, intelligence or disaster recovery. In this case, the employment of some disk-based storage is required, as described in several works [26–29] which however deal with the processing of finite data sets.

There exist some projects addressing the topic of distributed processing of data stored to secondary storage without employing continuous queries.

DataCutter [30] is a middleware which breaks down on-demand clients' requests into processing filters in charge of carrying out required computations. It has been devised to carry out complex processing over large distributed data sets stored to disk.

The MapReduce paradigm [31] implemented in Google and its open source implementation Hadoop [32] have received a great interest by the community and a lot of related projects [33–35] have been developed adopting a similar approach.

Dryad [36] is a project developed by Microsoft which organizes the processing as a dataflow graph with computational vertices and communication channels. The computation is batch and can elaborate files stored to a distributed file system.