The Journal of Systems and Software 000 (2016) 1–18

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Research paper

Proactive elasticity and energy awareness in data stream processing

Tiziano De Matteis, Gabriele Mencagli*

Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, I-56127, Pisa, Italy

ARTICLE INFO

Article history:
Received 29 September 2015
Revised 19 June 2016
Accepted 7 August 2016
Available online xxx

Keywords:
Data stream processing
Elasticity
Model predictive control
Frequency scaling

ABSTRACT

Data stream processing applications have a long running nature (24 hr/7 d) with workload conditions that may exhibit wide variations at run-time. *Elasticity* is the term coined to describe the capability of applications to change dynamically their resource usage in response to workload fluctuations. This paper focuses on strategies for elastic data stream processing targeting multicore systems. The key idea is to exploit *Model Predictive Control*, a control-theoretic method that takes into account the system behavior over a future time horizon in order to decide the best reconfiguration to execute. We design a set of energy-aware proactive strategies, optimized for throughput and latency QoS requirements, which regulate the number of used cores and the CPU frequency through the *Dynamic Voltage and Frequency Scaling* (DVFS) support offered by modern multicore CPUs. We evaluate our strategies in a high-frequency trading application fed by synthetic and real-world workload traces. We introduce specific properties to effectively compare different elastic approaches, and the results show that our strategies are able to achieve the best outcome.

1. Introduction

Data Stream Processing (Andrade et al., 2014) (hereinafter DaSP) is a computing paradigm enabling the online analysis of live data streams processed under strict Quality of Service (QoS) requirements. These applications usually provide real-time notifications and alerts to the users in domains like environmental monitoring, high-frequency trading, network intrusion detection and social media.

Elasticity in DaSP is a vivid and recent research field. It consists in providing mechanisms to adapt the used resources in cases in which the workload fluctuates intensively. Such mechanisms are able to scale up/down the used resources on demand, based on the actual monitored performance (Gedik et al., 2014). This problem has been studied in the last years, with works proposing elastic approaches both for single nodes and distributed environments (Gulisano et al., 2012; Fernandez et al., 2013; Heinze et al., 2014). A review of these solutions is described in Section 6.

This paper provides advanced strategies that fill missing aspects of the existing work. Most of the elastic supports are *reactive* (Gulisano et al., 2012; Fernandez et al., 2013; Heinze et al., 2014; Kumbhare et al., 2014), i.e. they take corrective actions based on the actual QoS measurements. In this paper we present *predictive* strategies that try to anticipate QoS violations. Furthermore, most of the existing approaches (see Section 6) are *throughput-oriented* and do not take into account explicitly the processing *latency* as the main parameter to trigger reconfigurations. In this paper we propose strategies that address both throughput and latency constraints. Finally, the existing approaches do not face *energy/power consumption* issues. In this paper we tackle this problem by targeting multicore CPUs with *Dynamic Voltage and Frequency Scaling* (DVFS) support.

The proactivity of our approach has been enforced using a control-theoretic method known as *Model Predictive Control* (Camacho and Bordons, 2007) (MPC), in which the system behavior over a future time horizon is accounted for deciding the best reconfigurations to execute. As far as we know, this is the first time that MPC has been used in the DaSP domain.

A first version of this work has been published in Ref. De Matteis and Mencagli (2016). This paper extends this preliminary work by presenting two energy-aware strategies with different resource/power usage characteristics: the first targets *high-throughput*, while the second is oriented toward *low-latency* workload. Furthermore, we provide a detailed analysis of our runtime mechanisms for elasticity and a comparison with state-of-the-art techniques. Finally, in this paper we specifically study the complexity issues related to the online execution of our adaptation strategies by presenting a Branch & Bound approach to deal with this problem.

* Corresponding author.
E-mail addresses: dematteis@di.unipi.it (T. De Matteis), mencagli@di.unipi.it (G. Mencagli).

http://dx.doi.org/10.1016/j.jss.2016.08.037
0164-1212/

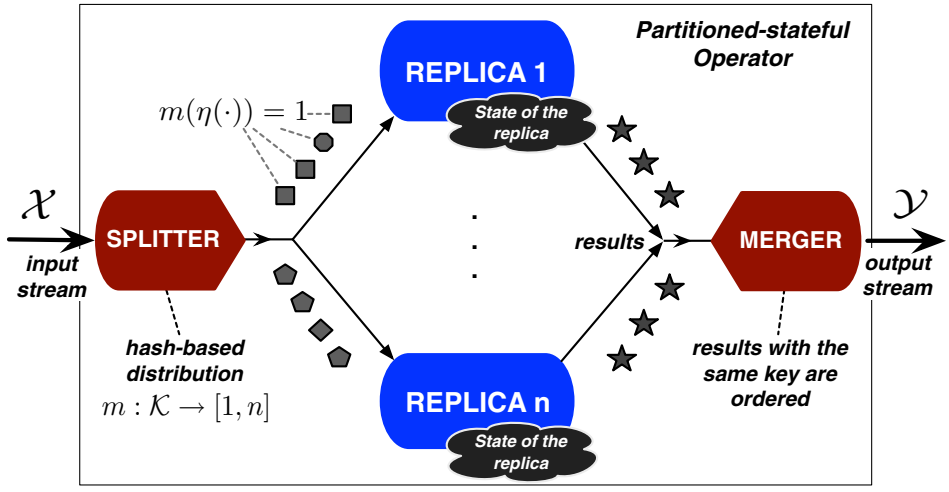

Fig. 1. Parallel partitioned-stateful operator: all the tuples with the same key are routed to the same replica.

The outline of this paper is the following. Section 2 provides a brief overview of DaSP. Section 3 describes our strategies. Section 4 shows the details of the reconfiguration mechanisms. Section 5 analyzes our strategies and compares them with the state-of-the-art. Finally, Section 6 reviews similar research works and Section 7 concludes this paper.

2. Overview of data stream processing

DaSP applications are structured as data-flow graphs (Andrade et al., 2014) of core functionalities, where vertices represent *operators* connected by arcs modeling *data streams*, i.e. unbounded sequences of data items (*tuples*).

Important in DaSP are stateful operators (Andrade et al., 2014) that maintain an internal state while processing input tuples. A typical case is represented by *partitioned-stateful operators* (Andrade et al., 2014), which are applied when the input stream conveys tuples belonging to different logical substreams. In that case, the operator can maintain a different internal state for each substream. Examples are operators that process network traces partitioned by IP address, or market feeds partitioned by a stock symbol attribute.

Owing to the fact that the significance of each input tuple is often time-decaying, the internal state can be represented by the most recent portion of each substream stored in a *sliding window* (Andrade et al., 2014). The window boundaries can be (*time-based*) or (*count-based*).

2.1. Intra-operator parallelism

In this work we study elasticity for parallel partitioned-stateful operators, which represent the target of the most recent research (Gedik et al., 2014; Gedik, 2014). A parallel operator is composed of several functionally equivalent *replicas* (Andrade et al., 2014) that handle a subset of the input tuples, as sketched in Fig. 1.

The operator receives a stream of tuples $\mathcal{X} = \{x_1, x_2, \ldots\}$ and produces a stream of results $\mathcal{Y} = \{y_1, y_2, \ldots\}$. For each tuple $x_i \in \mathcal{X}$, let $\eta(x_i) \in \mathcal{K}$ be the value of a partitioning key attribute, where $\mathcal{K}$ is the domain of the keys. For each key $k \in \mathcal{K}$, $p_k \in [0, 1]$ denotes its relative frequency. The replicas are interfaced with the input and the output streams through the *splitter* and the *merger* functionalities. The first is responsible for routing each input tuple to the corresponding replica using a (hash) *routing function* $m : \mathcal{K} \rightarrow [1, n]$, where n is the number of replicas. The merger collects results from the replicas and transmits them onto the output stream.

All the tuples with the same key are processed sequentially by the same replica in the arrival order. Therefore, no *lock* is needed to protect the state partitions since each partition is accessed exclusively by the same replica. Furthermore, this solution allows the ordering of results within the same group to be preserved (this can be a necessary property depending on the application semantics).

2.2. Motivations for elastic scaling

Real-world stream processing applications are characterized by highly variable execution scenarios. The dynamicity can be described in terms of three different factors:

1. *(D1) variability of the stream pressure*: the input rate can exhibit large up/down fluctuations;
2. *(D2) variability of the key distribution*: the frequency of the keys $\{p_k\}_{k \in \mathcal{K}}$ can be time-varying, making load balancing impossible to be achieved statically;
3. *(D3) variable processing time* per input tuple that may change during the application lifetime. This is possible for different reasons like the current system availability (sharing between applications), or for endogenous causes related to the elastic operator, e.g., the processing time may be dependent on the number of tuples maintained in the window to be processed.

Applications must tolerate these variability issues in order to keep the operator QoS optimized according to some user criteria. Our strategies will be designed to optimize two performance aspects: *i) throughput*, i.e. the number of results delivered per time unit; *ii) latency* (or response time), i.e. the time elapsed from the reception of a tuple triggering the operator internal processing logic and the delivering of the corresponding result.

To achieve the needed QoS, one could think to configure the operator in such a way as to sustain the peak load (e.g., the highest expected arrival rate) by using all the available resources at the maximum CPU frequency supported by the hardware. However, this solution may be very expensive both in distributed environments (number of machines turned on) and on single nodes (too high power consumption). The goal of any elastic support is to meet the application-dependent QoS specifications with high probability by keeping the operating cost within an affordable range. To this end, we target strategies able to modify the following configuration parameters of an elastic operator: *i)* the number of cores

Please cite this article as: T. De Matteis, G. Mencagli, Proactive elasticity and energy awareness in data stream processing, The Journal of Systems and Software (2016), http://dx.doi.org/10.1016/j.jss.2016.08.037