Autonomic Parallel Data Stream Processing

Tiziano De Matteis
Department of Computer Science, Universitá di Pisa
Largo B. Pontecorvo 3, I-56127 Pisa, Italy
Email: dematteis@di.unipi.it

Dissertation Advisor: Marco Vanneschi

DOCTORAL DISSERTATION COLLOQUIUM

EXTENDED ABSTRACT

***Abstract*—Data Stream Processing (DaSP) is a recent and highly active research field, applied in various real world scenarios. Differently than traditional applications, input data is seen as transient continuous streams that must be processed "on the fly", with critical requirements on throughput, latency and memory occupancy. A parallel solution is often advocated, but the problem of designing and implementing high throughput and low latency DaSP applications is complex per se and because of the presence of multiple streams characterized by high volume, high velocity and high variability. Moreover, parallel DaSP applications must be able to adapt themselves to data dynamics in order to satisfy desired QoS levels. The aim of our work is to study these problems in an integrated way, providing to the programmers a methodological framework for the parallelization of DaSP applications.**

***Keywords*—Parallel computing, Autonomic Computing, Data Stream Processing, Data Parallelism, Quality of Service, Structured Parallelism**

I. Introduction

In recent years, our ability to produce information has grown steadily, driven by an ever increasing computing power, communication rates, hardware and software sensors diffusion. The ability to gather and analyze this *continuous flow* of data, in order to extract insights and detect patterns, becomes a valuable chance. As a consequence, an increasing number of applications requires continuous processing of data flowing from several sources to obtain timely responses to complex computations. Examples of such applications emerge in various fields, such as financial applications, network management, social networks analysis; all of them have in common the need of real-time processing, high volume input streams with unlimited length and variable arrival rates. Similar requirements led to the definition of a new line of research, referred to as *Data Stream Processing* (DaSP) [1]:

- differently than traditional applications, data is seen as transient continuous streams rather than being modelled as traditional permanent, "in memory" data structures or relations;
- continuous streams cannot be efficiently stored before processing them. This has led to the use of *windows*, *approximate answering* and, more in general, *on-the-fly* processing as solution to this problem.

Since supporting high throughput and low latency is a critical requirement for DaSP applications, they necessitate taking advantage of parallel hardware and distributed systems. Traditional high-performance solutions, or their simple variants, are not always sufficient to solve this problem. Stream processing has been intensively studied and applied in parallel computing, however by assuming that operations are independently applied to distinct elements of a stream, or to single corresponding elements of multiple streams in a data-flow fashion, while in DaSP problems the results of the computation may be obtained by processing a set of (or a subset of) the input elements that belong to one or more input streams. Therefore, typical computational patterns of DaSP applications require *quite novel parallelism models* and related design and implementation techniques. Second, in this landscape parallel solutions must be designed with *autonomic* properties in mind. In order to maintain *Quality of Service* (QoS) specifications, novel parallelism models and implementations should tolerate run-time fluctuations in the workload, in the arrival rate from input streams and combinations of these events that are inevitable in long-running situations. The aim of our research is to provide a methodological framework for approaching these issues in an integrated way. Parallelism models, and their performance cost models, should guide the definition of related adaptation strategy and reconfiguration mechanisms.

The remainder of the paper is organized as follows. Section II presents uses cases and the state of the art. Section III outlines our proposed approach. Finally, Section IV concludes this dissertation.

978-1-4799-5313-4/14/$31.00

II. USE CASES AND STATE OF THE ART

The topic of data stream processing is a very recent one and there is a large volume of literature which has been published in the field over the last years. A number of systems, specifically designed according to this model, has been developed in both the academia and the industry, highlighting that this topic, besides being a fascinating and challenging research one, can be of strategic importance for data and communication intensive applications. *Financial applications* require to continuously analyze market data feeds in order to identify trends and opportunities: usual tasks regard complex correlations between different streams or patterns recognition over stock prices to quickly predict their future development, [2]. In this world, achieving and maintaining latencies under the millisecond may produce a significant advantage over competitors. *Network operators* and *service providers* need to monitor data traffic to identify bottlenecks, to analyze the provided service level or to detect possible attacks, generating alerts when something unexpected happens [3]. Wire-speed processing is required and slowly reacting or, even, not noticing a threat can lead to considerable damage in both economic and security terms. *Social media* have become ubiquitous and important for social networking and content sharing. Information broadcasted by users can be used to understand new phenomena: for example Twitter's stream can be processed in order to identify events of a particular type such as earthquakes and typhoons or breaking news [4]. In all the aforementioned scenarios, apart from handle high volumes of data and maintaining low computation latencies, it is required to face important variations in the arrival rate, satisfying the imposed performance requirements.

The area of Big Data processing has been recently dominated by *MapReduce* programming model. Hovewer, MapReduce systems typically operate over static data for batch processing [5], characteristic that is difficult to reconcile with a DaSP scenario, whose main challenges are related to input data that have high and, possibly, variable arrival rates and real-time processing requirements. Trying to tackle these problematics, several traditional software technologies have been repurposed and remarketed to address this application space. *Data Stream Management Systems* (DSMSs) are derived from traditional DBMS and are specialized in dealing with transient data. In particular they are able to execute continuous queries which run continuously and provide updated answers as new data arrives. Typical query operators of a DSMS are similar to relational algebra operators such as *Map*, *Filter*, *Aggregates*, *Join* and others. Various DSMS software infrastructures have been proposed over the last years, such as *TelegraphCQ* [6], *Borealis* [7] and *StreamCloud* [8]. *Complex Event Processing* systems, are instead born as an adaptation of Pattern Matching Engines. They view input data as *primitive events* that are analyzed for detecting particular patterns that represent *complex events* whose occurrence has to be notified to the interested parties. Several CEP systems are available today, such as *Sase+* [9], *Oracle CEP* [10] and *Esper* [11]. Compared to DSMS and CEP systems, a more prominent and flexible solution seems to be constituted by *Data Stream Processing Systems* (e.g. *Yahoo! S4* [5], *Twitter Storm* [12]), that are conceived for the implementation of generic DaSP applications by means of an imperative language rather than a declarative one. In this case, the computation is expressed by a graph of data stream operators that programmers can define and compose in an arbitrary way, usually deployed in a large scale scenario, such as a cluster.

Although some of these systems provide a complete framework for the development, deployment and maintenance of a DaSP application, they still lack of effectiveness for what concern parallelism exploitation and autonomic management. In the case of DSMS or CEP systems, parallelism is obtained by implementing ad-hoc solutions (e.g. for a particular relational operator) or between different sub-computations (e.g. between different operator of the same query). In the case of Data Stream Processing Systems, instead, parallelism is expressed using keyed input elements. Stream operators can be duplicated, assuring that input elements with the same key are processed by the same operator instance. Therefore, while the former solution lacks of generality, the latter can led to poor performances in the case in which elements are not uniformly distributed on the keys or it is not possible to identify meaningful keys on the input data.

In DaSP applications, the resource requirements to maintain the desired level of QoS is often dynamic and not easily predictable, due to the continuously evolving nature of their input streams and workload. For this reason, applications should be able to self-adapt at runtime; the paradigm enabling this behaviour is known as *Autonomic Computing* [13]. The key aspect of this paradigm is the synergic interaction between: *(i)* proper application-level and run-time support mechanisms in charge of applying reconfigurations and *(ii)* a decision-making strategy. The first point, concerns how reconfigurations are defined and executed, since they represent intrusive actions often inducing performance degradation. The work in [14] provides an overview of reconfiguration issues for traditional structured parallelism patterns. On the other hand, also the definition of adaptation strategies is a complex issue which leads to interdisciplinary researches. Proposed solution consists in the use of policy logic rules [15], or in the application of Control Theory concepts [16].

III. A PARALLEL STRUCTURED APPROACH

Without loss of generality, we can express DaSP applications by means of a structured composition of *autonomic parallel modules*. Modules express sub-computations, at least one, in which the application can be decomposed and edges represent internal data streams of data, on which the computations are applied. If the actual computation is not able to sustain the current arrival rate and/or to provide acceptable levels of computation latency, the computation graph has to be restructured, understanding which modules are bottlenecks and parallelizing them according to some parallelism paradigms.

A good starting point in defining proper parallel patterns can be Data Parallelism, which is a powerful paradigm able to optimize the throughput, latency and memory occupancy. Data parallelism is based on the *replication* of functions among a set of functionally identical executors (*Workers* in the following), on the *partitioning* of the input-output data (a partition of each data structure is assigned to each Worker), and possibly on the *replication* of some data in all the Workers. It can be used to compute single stream items in a stream parallel program, but it is not able to deal with some peculiarity of DaSP computations, such as the presence of *windows*. Windows clauses define a continuous segmenting of the input data streams: at any instant, they define the set of input elements that must be considered in order to produce the results. Windowing methods can be characterized according to a number of parameter (e.g. *time* vs. *count* based, *sliding* vs. *tumbling*) and are used mainly for two reasons:

- unblocking operators that would otherwise need to read the whole input before producing the results;
- focus the computation on recent data, possibly more interesting for the problem that has to be resolved.

To exploit intra-operator parallelism, windows must be replicated or partitioned among Workers. Such distributions can be critical, since windows are data structures that change over time, in terms of elements, cardinality or both of them.

The abstract scheme of a DaSP module (reported on Figure 1) can be expressed as composed by three functionalities: *Distributor*, *Workers* and *Collector*. In the general case a module will have a non deterministic semantic, with the execution enabled if at least one input stream contains a value.

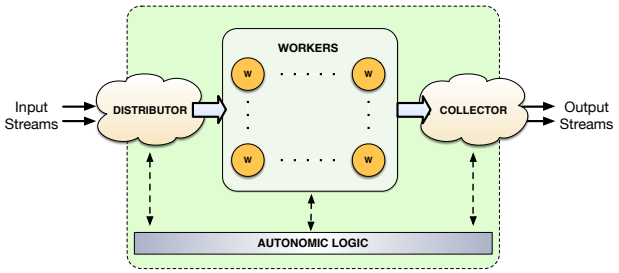

Figure 1. Abstract scheme of a generic parallel DaSP module

Concerning the data distribution and windowing management, we can consider two models:

a) **centralized model**, also called *Agnostic Workers*;
b) **distributed model**, also called *Active Workers*.

In the centralized model, the Distributor is in charge of: *(i)* performing all the processing and storing actions needed to build and to update the data window for each stream; *(ii)* distributing data windows to the Workers, *partitioning* or *replicating* them according to the particular computation, in general taking into account load balancing too. In this model, Workers are *agnostic* of the window management and data distribution, i.e., they are just in charge of applying the computation to the received window data. In the distributed model, window management and data distribution are partially or entirely delegated to the Workers. They receive elementary data values, or window segments, from the Distributor and are in charge to perform actions to build and to update the data window. Workers are *active* on windows management too, thus their code is very similar to the sequential version. In every case, the Collection logic is responsible to transmit the results onto output streams and to perform specific actions required by the operator, e.g. sorting of the results. The application of the centralized and distributed model depends on the computation algorithm and on the windowing modes.

In order to respect QoS specifications, a DaSP module must be able to reconfigure itself at runtime. For this reason, a module is characterized by the presence of a local *controller* (*Autonomic Logic* in Figure 1) that monitors the current module execution and implements an adaptation strategy that drives the reconfiguration selection. Reconfigurations can regard changes in the parallel implementation of the module (e.g. changes in the number of Workers, in their interconnection, dynamic replacement of the running implementation with another one featuring different QoS) but can also impact functional aspects of the computations such as changes in the windowing method (e.g. window length, expiration policy, etc.). Since the reconfigurations adopted by a module may have effects on the QoS variables of the other modules, controllers are coupled with each other in order to exchange information and negotiate to reach agreements. In this context, the aspects that deserve to be investigated, regard:

- the reconfiguration mechanisms: some reconfigurations involve intrusive actions on the computation that require a clear understanding of the parallelism pattern. Their overhead and impact on computation correctness must be predictable and known a priori, in order to be properly evaluated during the decision phase;
- the adaptation strategies, that is "how to choose" and "when trigger" a suitable reconfiguration. Control methodologies, with expected behaviors in terms of optimality, stability and efficiency [16], are of special interest, and their application in the Data Stream domain is quite new with respect to the traditional autonomic systems studied in the past.

A. Application to Parallel Stream Join

We will briefly discuss the application of our Data Stream Parallel approach to the parallelization of a Stream Join operator. The module has in input two streams X and Y and its output will be another continuous stream consisting of pairs (x, y), where x is a tuple from stream X and y is a tuple from stream Y such that *(i)* x and y satisfy the join predicate, and *(ii)* x was in the active window of stream X at the same time that y was in the active window of stream Y.

Our proposed parallelization is inspired by the distributed

model previously depicted. The Distributor dispatches the incoming tuples to a bi-dimensional topology of *Active Workers* (Figure 2). A tuple x_i received from stream X is forwarded

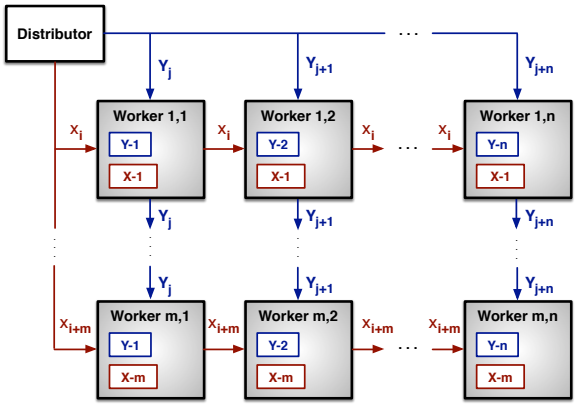

Figure 2. General layout of Workers using m rows and n columns

to a specific row of Workers, $W_{j,*}$, and symmetrically, Y tuples are forwarded to the various columns. Therefore, at each moment a Worker $W_{j,k}$ has a partition of X's window and a partition of Y's window. It will be activated non-deterministically upon the reception of a tuple from one of the two streams and will apply the original sequential computation, removing expired tuples independently. The Collector, is simply in charge of collecting the results from the various Workers and produce them in output.

The detailed study of the problem, implementation, experimentation and results are reported in [17], here omitted for the lack space. However, it is worth noticing that our parallelization is able to achieve higher throughput and lower latency (in some cased of some order of magnitude) compared to the state-of-the-art. An autonomic version of this module is currently under study, taking into account reconfigurations that regard the Workers layout. This can be modified by adding rows/columns of new Workers if needed, or adjusting its shape. For example, if X and Y have the same window size, a square layout will maintain the same partitioning of Y among columns and X among rows; passing to a rectangular layout, but also a linear one, can be interesting in the case of asymmetric streams (i.e. having different inter-arrival times) because it is able to balance the partition size of the two windows per Worker. In all these cases, the run-time support must be able to rearranging the windows partitions and changing the communication channels to reflect the new topology.

IV. Final considerations and ongoing research

With this work we pointed out which are our first ideas toward the definition of a framework for efficiently dealing with the parallelization of Data Stream Processing problems. The proposed parallel paradigm has been evaluated on the parallelization of a stream join operator, but clearly needs further development and evaluation through concrete experimentation. Hand in hand with parallelism models, autonomic management of DaSP modules must be properly studied in terms of strategies and reconfiguration mechanisms. This should guide the definition and implementation of efficient support mechanisms through which realize a real development environment for parallel DaSP computations.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. PODS*, ACM, 2002, pp. 1-16.

[2] Investopedia. Analyzing Chart Patterns: Head And Shoulders. [Online]. http://www.investopedia.com/university/charts/charts2.asp.

[3] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Proc. of RAID*, Springer-Verlag, 2000, pp. 85-103.

[4] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes Twitter users: real-time event detection by social sensors," in *Proc. of WWW*, ACM, 2010, pp. 851-860.

[5] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Proc. ICDMW*, IEEE, 2010, pp. 170-177.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah, "Telegraphcq: Continuous dataflow processing for an uncertain world," in *Proc. of CIDR*, ACM, 2003, pp. 668-668.

[7] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in *Proc. of CIDR*, ACM, 2005, pp. 277-289.

[8] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," *IEEE TPDS*, 2012, pp: 23(12):2351–2365.

[9] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proc. of SIGMOD*, ACM, 2008, pp. 147-160.

[10] Oracle. Cep. [Online]. http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html

[11] Esper. [Online]. http://esper.codehaus.org/

[12] N. Marz. Twitter Storm. [Online]. http://storm-project.net

[13] M. C. Huebscher and J. A. McCann, "A survey of Autonomic Computing - Degrees, Models, and Applications," *ACM Compt. Survey*, 2008, pp. 7:1–7:28.

[14] M. Vanneschi and L. Veraldi, "Dynamicity in distributed applications: issues, problems and the ASSIST approach," *Journal of Parallel Computing*, 2007, pp. 822-845.

[15] J.O. Kephart and W.E. Walsh, "An Artificial Intelligence Perspective on Autonomic Computing Policies," in *Proc. of POLICY*, IEEE, 2004, pp. 3-12.

[16] G. Mencagli, M. Vanneschi, and E. Vespa, "A Cooperative Predictive Control Approach to improve the Reconfiguration Stability of Adaptive Distributed Parallel Applications," *ACM Transactions on Autonomous and Adaptive Systems*, To appear.

[17] D. Buono, T. De Matteis, G. Mencagli, and M. Vanneschi, "Tiowards a methodology for parallel data stream processing: application to parallel stream join," Dept. of Computer Science, Universita' di Pisa, TR-13-18, 2013.