1
Introduction

This chapter presents the motivation that led to the development of this thesis, well as the main objectives and contributions of the work.

1.1
Motivation

A large number of applications require continuous and timely processing of high-volume of data originated from many distributed sources to obtain real-time notifications/alerts from complex queries over the steady flow of data items [1] [2]. This has led to a new computing model called *data stream processing* [1]. This model is focused on sustained and timely analysis, aggregation and transformation of large volume of data streams that are continuously updated [2].

The need to process data streams comes from several application areas such as network monitoring, traffic engineering systems, intelligent routing of cars in metropolitan areas, sensor networks, telecommunication systems, financial applications, meteorology and *Onboard Car Survivability Assistance*. Applications of intelligent routing of cars in metropolitan areas, such as Waze [3], collect data from many distributed mobile devices to infer the actual condition of the route (e.g. streets and roads) and guide the driver (user) through the best way. This kind of application requires not only the data correlation of a large set of mobile devices but also the processing of much data to infer more complex situations (e.g. traffic condition and the better path to the driver´s destination). The network monitoring applications should inspect the packets as they flow through the routers [4]. McGregor and Kannan [4] explain each router may be able to forward up to a billion packets per hour. These packets should be analyzed and correlated with others in order to detect attacks such as intrusion and Denial of Service (DoS).

Onboard Car Survivability Assistance systems are systems that alert the car driver or medical/police teams about the occurrence – or possibility – of accidents or danger situations. More and more embedded electronic devices are used in cars

[5] [6] [7] [8], as well as sensors that feed automotive systems with context data about environment, driver and driving conditions, for instance. Such data will be an important data source for complex analysis and reasoning for *Onboard Car Survivability Assistance* systems. The system may infer from simple events such as strong deceleration and acceleration, accentuated turn left and right, high engine RPM (Revolutions Per Minute), unknown voice in cabin and high noise – or voice – level in cabin that the driver was kidnapped.

Such applications share the requirement of real-time processing of large streams of sensor data produced by huge sets of client nodes, which may be vehicles, aircrafts, mobile users, computing devices or smart objects, with embedded sensors. Although some kind of data processing can be performed locally at the client nodes, e.g. simple transformations or classification of sensor data, most other information about the monitored system as a whole requires parallel data processing by dedicated machines, which increases the need for load balancing solutions [9] [10] [11].

In order to cope with the high processing demand, stream processing applications typically employ SIMD (Single Instruction, Multiple Data) parallelism and use multiple processing units, where each unit is responsible for processing a subset of data items independently of the remaining data items, and hence without need to manage communication or synchronization among those units. Since the stream processing applications typically use multiple processing units, these kind of application has a native need for load balancing solutions.

1.2 Problem Statement

One of the most promising communication infrastructures for the aforementioned stream processing applications is OMG´s Data Distribution Service (DDS), which is the first open international middleware standard directly addressing publish-subscribe communications for real-time and embedded systems [12]. The DDS standard was designed with the intention of supporting scalable communication and computing [13].

The major advantages of DDS for data stream processing are its provision of real-time asynchronous communication and its peer-to-peer architecture, which in turn support a scalable deployment of data stream processing nodes. In spite of

the advantages of using DDS as the basic infra-structure for stream processing, it lacks support for load balancing among these processing nodes. Thus, currently it is the responsibility of the developer of the DDS-based application to implement the load balancing for these nodes, which is completely orthogonal to the application logic, but significantly increases the complexity of the application's implementation. In order to compensate this limitation of DDS and facilitate the development of scalable stream processing applications based on DDS, in this thesis we investigate load balancing for DDS-based applications, seeking for an application-transparent and autonomous re-distribution of data streams among DDS nodes responsible for the stream processing.

Actually, one of the main problems related to load balancing for stream processing is to ensure that each data item of the stream is processed once, and at most once. When considering specifically DDS-based applications, this translates into the problem of guaranteeing that the data item filters of the processing nodes (i.e. the data item subscribers) involved in the load re-distribution are updated in a coordinated form and that no data item arriving during this process is lost or delivered to more than one processing node. Hence, a load balancing solution for DDS-based systems should have the following main features:

- **Transparent:** Publishers and subscribers should be completely unaware of how load balancing works and the system should not experience interruptions in service while load balancing is being carried out. Moreover, the load balancing operation should be completely autonomous, meaning that it should not require any human interaction;
- **Generic:** The solution should be application independent, which means that it should not assume any characteristics of the data items, nor incorporate any specificity related to the application's data processing.
- **Heterogeneous:** The solution should be able to work with different nodes´ resource capacities such as different operating system, memory sizes, CPU (*Central Processing Unit*) speeds or network bandwidths.
- **Adaptive:** It should be possible to use several load balancing algorithms, and, the load balancing algorithms should be capable of considering (and optimizing) different metrics (e.g. CPU and memory).

- **Dynamic:** Load balancing should be invoked whenever an uneven load distribution among nodes is detected, rather than at predefined time intervals;
- **Reliable:** The solution should ensure that all data items are processed exactly once, as discussed before;
- **Lightweight:** The load balancing solution should have minimum impact on the performance of data stream processing application.

1.3 Objective and Contributions

Stonebraker *et al* [1] outlines eight requirements that systems should meet to be capable of real-time stream processing, and among these, one requirement is to support partition and scale applications automatically, i.e. that it should be possible to split an application over multiple machines for scalability purposes, and to distribute the load among them, without the developer having to write low-level code.

Aligned with this high-level "automatic scalability and partition" requirement, the main goal of this thesis is the design and implementation of a generic and dynamic load balancing solution to enable scalable stream processing on DDS-based systems. More specifically, the main contributions of this thesis include:

- A generic load balancing software layer that is independent of the data items/objects processed by the applications;
- A load balancing solution that can easily incorporate new load balancing algorithms.
- An approach that transparently controls how input data is routed to and delivered to DDS nodes to enable data stream processing on DDS-based systems;
- Experiments that demonstrate the performance and the overhead of the load balancing solution.

1.4 Organization

The remainder of this thesis is organized as follows:

- Chapter 2 presents an overview of the key concepts and technologies in the areas of DDS, load balancing algorithms and autonomic computing that are used throughout this work;
- Chapter 3 delves into details the proposed load balancing approach, which it is called Data Processing Slice Load Balancing (DPSLB);
- Chapter 4 describes the prototype that implements the Data Processing Slice solution and the simple load balancing algorithm developed to validate the prototype;
- Chapter 5 explains the performance evaluation of the implemented load balancing solution using a prototype DDS application, and discusses the performance results.
- Chapter 6 reviews the related work on load balancing for Publish/Subscribe systems, including DDS;
- Finally, chapter 7 summarizes and discusses the central ideas presented in this thesis, and proposes lines of future work on the subject.