Autonomous Load Balancing of Data Stream Processing and Mobile Communications in Scalable Data Distribution Systems

Rafael Oliveira Vasconcelos and Markus Endler
Department of Informatics
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
rvasconcelos@inf.puc-rio.br, endler@inf.puc-rio.br

Berto de T. P. Gomes and Francisco J. da Silva e Silva
Graduate Program Electric Engineering (PPGEE)
Federal University of Maranhão (UFMA)
São Luís, Brazil
bertodetacio@ifma.edu.br, fssilva@deinf.ufma.br

***Abstract*—A huge number of applications such as network monitoring, traffic engineering systems, intelligent routing of cars, sensor networks, mobile telecommunications, logistics applications and air traffic control require continuous and timely processing of high volume of data originated from many distributed sources as well as mobile communication and monitoring. The deployment and operation of infrastructures enabling such mobile communication and data stream processing have two key requirements: they must be capable of handling large and variable numbers of wireless connections to the monitored mobile nodes regardless of their current use or locations, and must automatically adapt to variations in the volume of the mobile data streams. This article describes the design, implementation, and evaluation of an autonomic mechanism for load balancing data streams and mobile connections. The autonomic capability has been incorporated into a scalable middleware system based on a Data Centric Publish Subscribe approach - using the OMG Data Distribution Service (DDS) standard - and aimed at real-time and adaptive handling of mobile connectivity and data stream processing for large sets of mobile nodes. Several performance evaluation experiments of the proposed infrastructure are presented, demonstrating its viability and the advantages arising from the use of an autonomic approach to handle the requirements of high variability and scalability.**

***Keywords*-Load balancing, Data Stream Processing, Autonomic computing, DDS, Mobile Communication Middleware.**

I. Introduction

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 from complex queries over the steady flow of data items [1] [2] [3] [4]. This has led to a new computing model called Data Stream Processing [3], focused on sustained and timely filtering, aggregation, transformation and analysis such data streams.

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 and meteorology. Crowd-sourcing applications such as Waze [5], collect data from many distributed mobile devices to infer the actual condition of the routes (e.g., streets and roads) and guide their users - the drivers - towards the best route. This kind of application requires not only the data fusion from a huge set of mobile devices, but also the processing of this data to infer more complex situations (e.g., the local traffic condition and alternative routes to the driver´s destination).

Such applications share the requirement of real-time processing of large flows of sensor data (i.e., context data) produced by hundreds of thousands of client nodes, which may be vehicles, aircrafts, mobile devices, 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 context information about the monitored system as a whole requires parallel data processing by sets of dedicated machines (e.g., clusters of processing nodes). This, in turn calls for load balancing solutions [6] [7] [8] for these clusters.

In order to handle large volumes of data streams in a scalable and self-manageable manner, such systems have to be distributed and autonomous. However, using software and hardware that manages itself requires self-management autonomic capabilities to detect and independently react to run-time problems [9]. Thus, there are several challenges in the field of self-managed and adaptive Data Stream Processing for large-scale mobile systems, since it involves timely communication, scalable processing and context-awareness.

A pilar to build self-manageable systems is Autonomic Computing (AC). The main goal is to build computing systems and applications able to manage themselves, thus minimizing human intervention [10] [11] [12] [13]. In order to accomplish the AC challenges, scientific and technological advances in a wide range of fields and system architectures are required, as well as new programming paradigms [14]. On the other hand, today's mobile communication and data stream processing systems lack autonomic features that are necessary to support the large and variable amounts of data flows envisioned by the massive and ubiquitous dissemination of sensors and mobile devices in our modern society. In particular, the deployment and 24x7 operation

of such mobile data stream processing and communication infrastructures pose two intrinsic technical challenges: they must be capable of (i) handling huge and variable numbers of mobile data connections, and (ii) of automatically adapting to variations in the volume of the mobile data streams.

In order to address these challenges, we have developed a scalable middleware system that supports efficient and adaptive handling of mobile connectivity and data stream processing for thousands of mobile nodes. In this paper, we specifically explain the autonomic load distribution mechanisms implemented in the middleware, and discuss their potential benefits. Experiments with large data stream have demonstrated the low overhead, good performance and the reliability of the proposed solution.

The remainder of this paper is organized as follows: Section II presents an overview of the key concepts and technologies which are used throughout this work: the Data Distribution Service (DDS) for Real-Time Systems standard, the MAPE-K reference model for autonomic systems, and load balancing approaches in middleware. Section III presents the Scalable Data Distribution Layer (SDDL), used as the middleware for mobile communications and the MAPE-SDDL extension, which adds autonomic capabilities to the SDDL middleware. Section IV delves into the proposed *Data Processing Slice Load Balancing* approach for mobile data streams and explains how it was implemented, while Section V describes the detailed evaluation of the implemented system using a prototype application. Section VI reviews related work on load balancing for Publish/Subscribe systems, including DDS, while Section VII discusses the advantages of using an Autonomic Computing for load balancing and the benefits of the load balancing solution proposed by this work. Finally, Section VIII contains some concluding remarks about the central ideas presented in this work and mentions some possible lines of future work on the subject.

II. Background

This section presents the main concepts and technologies that are used in our work.

A. Data Distribution Service (DDS)

One of the most promising communication infrastructures for the aforementioned data stream processing applications is the Data Distribution Service (DDS) for Real-Time Systems. DDS is an OMG [15] (Object Management Group) standard for Publish-Subscribe communication that aims to provide an efficient and low-latency data distribution middleware for distributed applications [16] [17]. DDS promotes a fully decentralized P2P (Peer-to-Peer) and scalable middleware architecture based on the Data-Centric Publish-Subscribe (DCPS) model. It also supports a large array of Quality of Service (QoS) policies for communication (e.g., best effort, reliable, ownership, several levels of data persistency, data flow prioritization and several other message delivery options) [18] [19]. Unlike traditional Publish-Subscribe middleware, DDS can explicitly manage network resources through fine-tuning of its Network Services and use of QoS policies such as Deadline, Latency Budget, Transport Priority, etc, that are critical for implementing real-time and soft real-time systems.

Publishers and Subscribers of a DDS Domain (the collection of nodes pertaining to a single application), which are named Participants, are containers for Data Writers and Data Readers, respectively, which exchange typed data through a common Topic [17]. Pardo-Castellote, Farabaugh and Warren [20] explain that Data Writers and Data Readers are the primary point for a Participant to publish data into a DDS Domain or to access data that has been received by a Subscriber. The DCPS makes it possible to organize its Topics in a relational model, providing support for identity and relations, i.e., for each Topic it is possible to define one or more primary keys, and any number of foreign keys representing, respectively, relationships with other Topics.

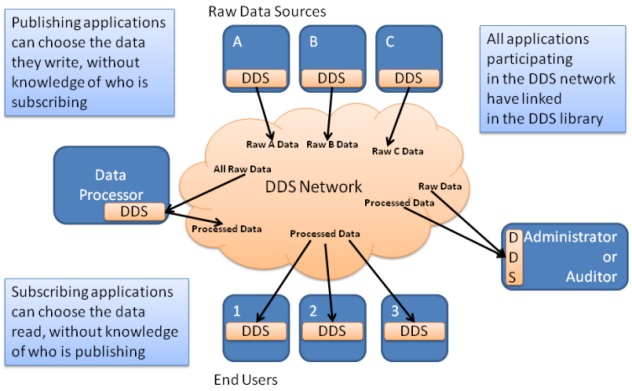

Figure 1. DDS System Architecture [17]

Figure 1 illustrates a hypothetical system that uses DDS as its data distribution middleware. This hypothetical application has some sources of "Raw Data", a Data Processor that performs some processing on the "Raw Data" to produce "Processed Data", some End Users that consume the processed data, and an Administrative User performing auditing functions, for instance. DDS supports not only Topic subscriptions, but also content-based subscriptions. The latter are enabled by DDS *Content Filtered Topics*, which holds a *Filter Expression* formed through SQL92 (Structured Query Language). This Filter Expression defines a selective information subscription, i.e., only the topic data that match the Filter Expression are delivered to the Data Reader. An use example of Content Filtered Topic is shown in Figure 2, where a Filter Expression (Value > 260) is applied upon the "Value" field.

The DDS enables applications to filter data based on the content of the data either at the Publisher side (Data

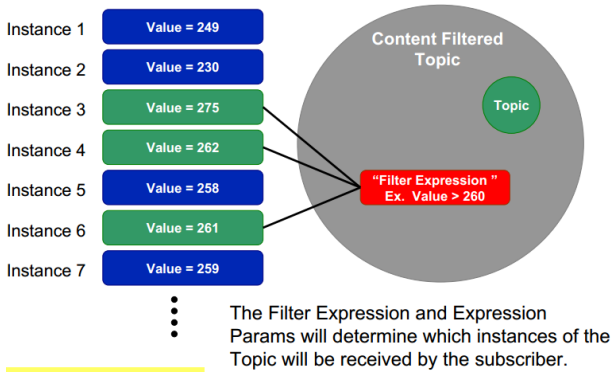

Figure 2. Content Filtered Topic example [21]

Writer) or Subscriber side (Data Reader). By applying filters at the Publisher, some applications can conserve significant network bandwidth by avoiding the network transmission of irrelevant data [22]. Although this capability, for some kinds of application – such as those that have a dynamic and unpredictable number of Publishers and Subscribers – the filtering at the Subscribers is the best choice. In DDS, topics are defined using DDL (Data Definition Language). This language is very similar to the OMG IDL for describing data types. A compiler is then used to translate the DDL type definitions into specific programming language code that is included into an application.

B. MAPE-K Autonomic Architecture

The MAPE-K [23] (*Monitoring, Analysis, Planning, Executing and Knowledge*) model, illustrated in Figure 3, is a general architecture for the development of autonomic software components, which was originally proposed by IBM [9]. This model defines the tasks and the interactions among the main architectural components of autonomic systems. According to MAPE-K, each element in such a system is divided into an autonomic manager and a managed resource.

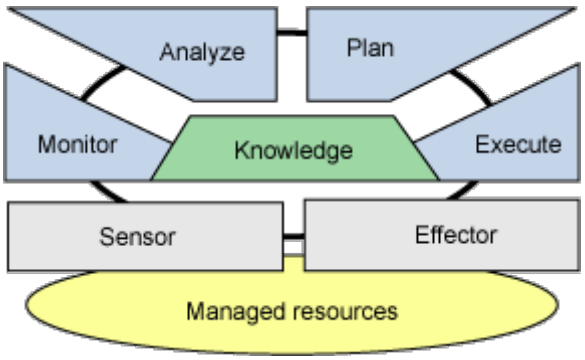

Figure 3. MAPE-K Autonomic Architecture [24]

The managed resource corresponds to the system or some system component providing the business logic that is to be dynamically adapted as the computing environment changes. The managed resource can be, for instance, a Web server, a database, a software component in a given application (e.g., the query optimizer in a database), an operating system, etc. The autonomic manager performs all the functions comprising the adaptation logic on the managed resource: monitoring, analysis, planning, and adaptation execution. MAPE-K defines two types of access points with the managed resource: sensors and effectors, which are the only means of direct interaction with the managed resource. Sensors are responsible for collecting information from the managed resource. For example, if the managed resource is an application server, this could be for instance, the response times of remote requests from a client. The information collected by the sensors reaches the monitors, where they are interpreted, classifieds and transformed into higher level information, such as the mean response time distribution for different sorts of requests. This information is then sent to the next stage of the cycle, the analysis and planning phases. This stage produces an action plan, which consists of a set of adaptation actions to be performed by the executor. The effectors are the access points that allow the autonomic managers to perform adjustments or adaptations at the managed resources, such as allocating more/less buffer space or setting some flow control parameter. The decision of which adaptation actions must be applied in a given situation requires a knowledge representation of the computing system, its states and its environment. This knowledge can be represented and processed in different ways (e.g., Ontologies, basic ECA-Rules, machine learning, etc.) and must be shared among the monitoring, analysis, planning and executing components of the autonomic manager.

C. Load Balancing in Middleware

With the widespread and rapid development of Cloud Computing and mobile devices, computational resources have become ubiquitous. Despite the recent technology developments, mobile devices still have more restrictive processing and memory capacity and stringent energy limitations than stationary machines. On the other hand, for several applications one has the option to move some processing tasks from the mobile client side to the server/cluster/cloud side. This shift has several advantages for the application, but also increases the demand for load balancing mechanisms [6] [7] [8] [25], especially at the middleware layer used for communication among the servers and/or cluster nodes.

Load balancing strategies have been classified under a loosely unified set of terms and according to [25], the first classifications came from [26] [27]. Figure 4 depicts the classification proposed by [27]. Following the proposed taxonomy, a load balancing algorithms can be either local or global. Local solutions deal with a single processing node, while global algorithms deal with more than one processing node [25]. A global solution may be divided into static,

when the load balancing algorithm is executed only when there is a new task, and dynamic, which runs the algorithm continuously or periodically. At an operational level, an algorithm may be classified as physically distributed (distributed) or physically non-distributed (centralized). Unlike the centralized approach, in physically distributed load balancing algorithms, the decisions are taken by several nodes. And this decision can be made cooperatively, or non-cooperatively. In the former, the algorithm requires a common agreement among the nodes, while in the latter each node makes a selfish decision. Moreover, according to [28] in the global solution, decisions are made by a single node such as the physically non-distributed defined by [25].

Delving into more details of the load balancing process, a dynamic load balancing algorithm have four main elements that are: (i) Initiation, (ii) Load Balancer Location, (iii) Information Exchange and (iv) Load Selection, shown in Figure 5 [29].

The Initiation policy defines how the current load information is exchanged among the nodes. While in a periodic strategy, information is exchanged at predefined time intervals, an event-driven initiation strategy is based on the local load observation. According to [29], the later strategy better handles load imbalance and has a lower overhead than the periodic strategy when the system load is already balanced.

A designer of load balancing algorithm should choose one of two strategies for the location of the load balancer, i.e., the node in charge of analyzing the system load and deciding whether a load redistribution among the nodes is required. Load Balancer location strategies can be centralized or distributed. Unlike the centralized strategy where a single node evaluates the load of the entire system, a distributed approach has some, or possibility all, nodes responsible for made load balancing decisions.

Because the remaining sub-strategies of the taxonomy shown in Figure 4 and Figure 5 are not of much relevance for this work, we refer to [29] for an in-depth discussion about the characteristics of the other policies. Moreover, the objective of this work is not to propose specific load distribution algorithms, but rather provide general mechanisms that support the implementation of several distributed load balancing algorithms.

III. The SDDL Middleware

A. Overview of the SDDL

The Scalable Data Distribution Layer (SDDL) [19] [30] [31] [32] [33] is a communication middleware that connects stationary nodes running in a DDS Domain and deployed in a cloud to mobile nodes that have an IP-based wireless data connection, as illustrated in Figure 6. Some of the stationary nodes are data stream processing nodes, while others are gateways for the communication with the mobile nodes (MNs). Gateways use the Mobile Reliable UDP (MR-UDP) [19] [32] protocol to maintain a virtual connection with each MN. The MR-UDP protocol was developed to be robust to short-lived wireless disconnections, IP address changes of the MNs and capable of Firewall/NAT traversal. One of the nodes in the DDS Domain, the Controller, is also a Web Server that can be accessed by a Web browser, for displaying all the MN's current position (or any other node specific information) and for send unicast, broadcast, or groupcast message to the mobile nodes. Figure 6 shows other nodes in SDDL that are Load Balancer, PoA-Manager and Processing Nodes. All nodes showed in Figure 6 will be explained throughout this work.

Taking advantage of DDS' distributed P2P architecture and its highly optimized Real-Time Publish Subscribe wired protocol, SDDL is naturally scalable, i.e., new processing nodes or Gateways can be dynamically added to SDDL's core whenever more MNs have to be served, or new data flow processing is required. In regard to the connections with the MNs, whenever some Gateway is overloaded the data flow to and from a large set of MNs, SDDL is capable of seamlessly migrating a fraction of this set of MNs to a underloaded Gateway. This is possible through a SDDL-internal management node, called the PoA-Manager, which continuously monitors the load of each Gateway - in terms of the number of served MNs - and a Client communication library (CNClib) at the MNs, which accepts both updates of alternative Gateway addresses and/or commands to reconnect to a new Gateway address, from the PoA-Manager. In spite of the unavoidable mobile disconnection, these handovers between Gateways are very fast and completely transparent to the client applications running on the mobile nodes. On the one side, the messages from the MN are buffered in the CNClib until the new connection is established, and on the other side, messages addressed to the MN are also temporarily intercepted by a SDDL node and then re-routed to the new Gateway, as soon as it signals the connection establishment.

B. MAPE-SDDL

In order to address general dynamic adaptivity requirements for the SDDL middleware, we decided to extend it with autonomic capabilities. This extension, inspired by the MAPE-K loop, is called MAPE-SDDL. The goal is to support resource monitoring, as well as analysis, planning and execution of dynamic reconfigurations on components of the SDDL middleware. The MAPE-SDDL architecture (at a high level of abstraction) is illustrated in Figure 7. It comprises four services: Monitoring Service (MS), Local Event Service (LES), Analysis and Planning Service (APS), and Control and Executing Service (CES).

1) Monitoring Service (MS): The MS collects data from any SDDL managed resources, such as Gateways and Processing Nodes. The monitoring is applied to properties from these resources, such as: CPU load, amount of memory available, network bandwidth and latency, number of served

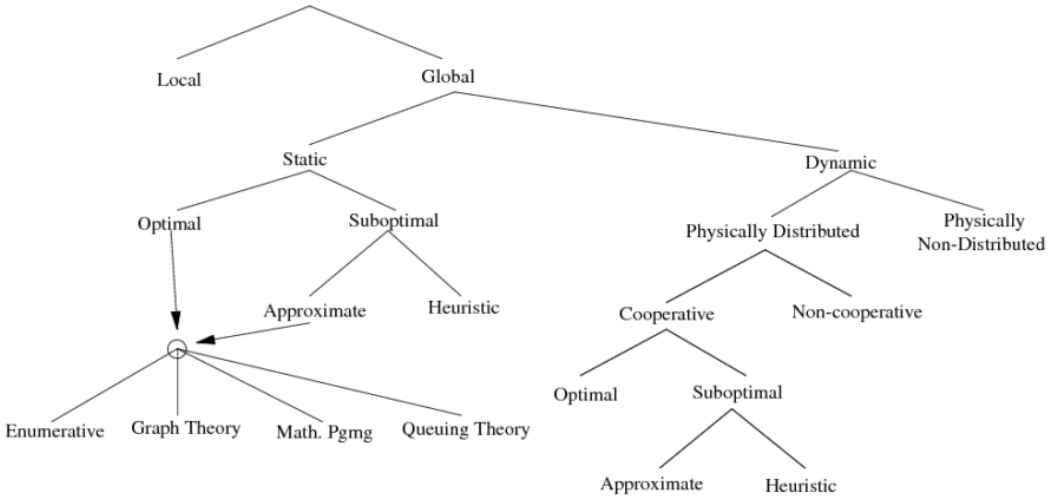

Figure 4. Load balancing hierarchy [27]

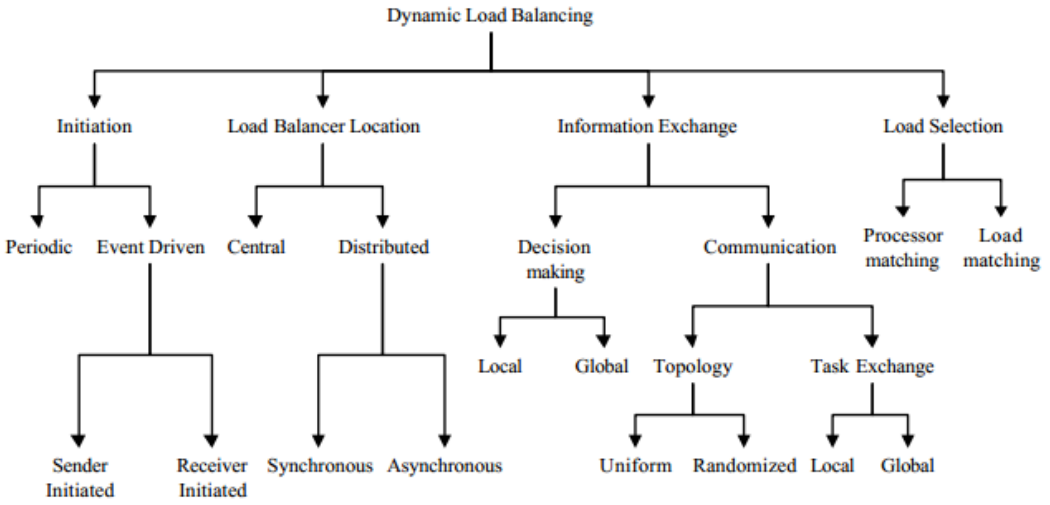

Figure 5. Taxonomy of dynamic load balancing algorithms [29]

MNs (by each Gateway) or number of DPSs assigned to each Processing Node (see DPS concept in Section IV-A). Each Monitor is responsible for a single property. Each property is associated with a set of operation ranges, which are defined by the framework user. For example, one could use the following operation ranges for monitoring the CPU load usage: [0%,30%], [30%,70%] and [70%,100%]. The MS then notifies the LES (Local Event Service) whenever the monitored property switches its operation range, which might indicate a significant change on resource usage. MS and LES are components that run in separate processes, but they are located on the same node. Therefore, unlike other cases in which communication occurs through DDS topics, communication between MS and LES occurs through standart JavaRMI [34].

2) Local Event Service (LES): The LES receives these range change events from the MS and publishes event notifications to subscribed components. Events are occurrences that indicate that a resource availability condition extended itself throughout a specified amount of time, i.e., its duration time. Event evaluation is based on regular expressions written by application developers or operators, as part of each event definition. For an event notification to be triggered, the corresponding expression must remain valid during the specified duration time. This avoids the generation of events when short-lived situations occur (e.g., a CPU load peak on a Processing Node during a few seconds). The publication topic of event notifications is the `Event Notification Topic`, illustrated in Listing 1. This topic is defined by two data structures: the data structure `Property` (line 1) contains the monitored property name (line 3), the

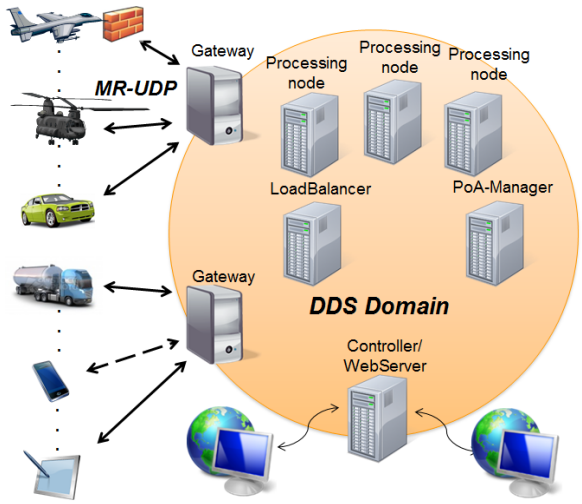

Figure 6. SDDL Architecture

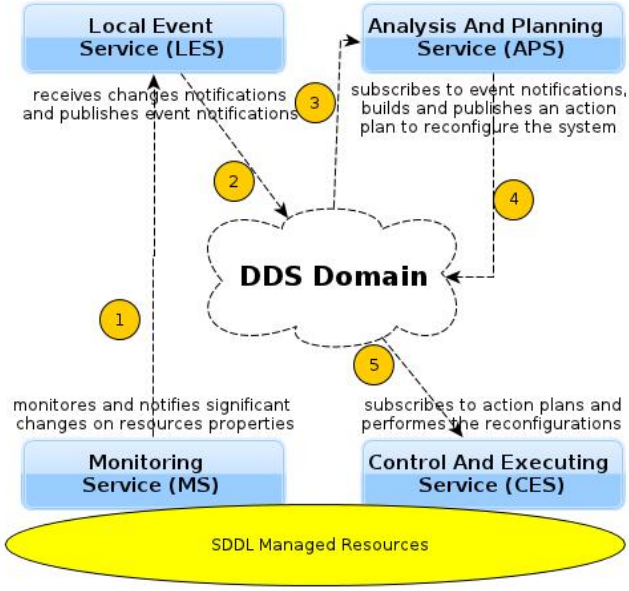

Figure 7. MAPE-SDDL Architecture

property value (line 4), and the timestamp at which the measurement occurred (line 5); the data structure `Event Notification` (line 8) corresponds to the event itself. The `Event Notification` contains the event identifier (line 10), the node identifier that triggered the event (line 11), and the set of monitored properties (with their values) that composes the notified event (line 12).

Listing 1. Event Notification Topic (DDL syntax)

```
1  struct Property
2  {
3      string name;
4      double value;
5      string timeStamp;
6  };
7
8  struct EventNotification
9  {
10     string eventID;
11     string sourceID;
12     sequence<Property> propertiesSeq;
13 };
```

3) Analysis and Planning Service (APS): The APS subscribes to the `Event Notification` topic and analyzes the received notifications identifying eventual problems that requires reconfiguration actions. Mobile connection overload on the Gateways, and unbalanced load between Processing Nodes are examples of problems that are already being diagnosed by the MAPE-SDDL APS. After diagnosis, the APS will compose the dynamic reconfiguration actions to resolve the identified problem, and then build an appropriate action plan. The decision-making algorithm for building the plan is based on user defined rules and uses a rule processing engine. The action plan is a sequence of reconfiguration actions to be executed on SDDL components. The action plan for mobile connectivity management, for instance, takes the form of a mandatory handover request to several mobile nodes (with a new Gateway address list) that is generated by the PoA-Manager, an instance of the APS. Action plans are sent to the CES component through the `Action Plan Topic`. This topic is defined by two data structures. The data structure `Action` (line 1) contains the action identifier (line 3), the node identifier that will perform the action (used by CoreDX to filter the DDS message delivery designed for each Processing Node - line 4), and a set of arguments required to perform the action (line 5). In Java, the arguments corresponds to a byte array in order to allow the sending of any serializable objec. The data structure `ActionPlan` (line 8) corresponds to the plan itself, containing the plan identifier (line 10), and the set of actions that comprises this plan (line 11).

Listing 2. Action Plan Topic (DDL syntax)

```
1  struct Action
2  {
3      string actionID;
4      string executorID;
5      sequence<octet> args;
6  };
7
8  struct ActionPlan
9  {
10     string planID;
11     sequence<Action> actionsSeq;
12 };
```

4) Control and Executing Service (CES): Finally, the CES implements the adaptation engine that applies the corresponding reconfiguration actions to SDDL components in response to their availability/load changes. CES is divided into two components: Controller and Executor. The Controller is responsible for managing the execution of the action plan, including the execution order of the reconfiguration actions that must be applied as defined by the APS component. The Executor is responsible for actually executing a given reconfiguration action. Among the dynamic reconfiguration actions currently supported, is the ability of moving DPSs from a Processing Node to another (cf. Section IV-A). The ability of migrating sets of MNs from one Gateway to another (cf. Section III-A) is also implemented by CES, which in this case resides in the mobile Client Lib, that performs the disconnection from one Gateway and the reconnection to the new Gateway.

IV. Load Balancing of Mobile Data Streams

A. Proposed Autonomous Approach

This work proposes a load balancing solution for DDS-based systems named Data Processing Slice Load Balancing (DPSLB), illustrated in Figure 8. The key concept of the proposed solution is the Data Processing Slice (DPS), which is the basic unit of load for balancing among server nodes. These nodes will be called Processing Nodes (PNs) throughout the text. The general idea is that each PN has some DPS assigned to it, and that load balancing is equivalent to a redistribution of the total number of DPS among the PNs according to their current load (which is indicated by several metrics, such as CPU and memory utilization).

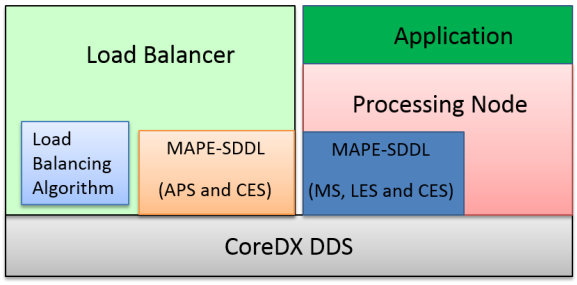

Figure 8. Implementation architecture

The types of DDS nodes that compose the DPSLB approach, showed in Figure 8, are: PNs, which execute the MS, LES and the CES Executor; and the Load Balancer, which executes the APS and the CES Controller of the MAPE-SDDL architecture. The Load Balancer is responsible for monitoring the load of PNs, generating the actions to redistribute the system's workload when an unbalance is detected and controlling the actions executed by PNs to move DPSs between them. The Load Balancer has a module, Load Balancing Algorithm, which may execute any global algorithm that analyzes the load of the PNs, decides if the system is unbalanced and performs the corrective actions.

The DPSLB solution was designed to Pub/Sub systems that supports content-based subscriptions and are brokerless, i.e., Pub/Sub systems that do not have brokers and employ a fully decentralized P2P architecture such as the DDS standard. The data items produced by the Publisher Client Nodes are delivered directly to the Processing Nodes without the need of centralized elements such as brokers. Thus, the Processing Nodes (PNs) are the Subscribers in charge of processing the data items instead of brokers that route the data items to other elements. It is expected that the proposed solution will be deployed in systems with thousands of Processing Nodes and hundreds of thousands of Client Nodes and a data production rate estimated of dozens of gigabits per second.

In its current conception, the DPS Load Balancing supports applications where each data item is processed independently of any other item. This limitation comes from the way that processing load is distributed among PNs: through the application of disjoint subscription filters. Since a Processing Node does not receive all data items published on the DDS Domain, PN may be unable to process a data item "A" that depends on data item "B" delivered to and processed by another Processing Node. Hence, the proposed load balancing solution is tailored for data-parallel applications, i.e., where each data item is processed independently of other items, and data items can be processed out of order by any Processing Node.

As mentioned, the proposed solution relies on the concept of DPS, or simply, Slice, which represents a percentage of the total system workload being processed by the PNs. Every data item of the data stream (e.g., produced by a mobile node) must be assigned to a single DPS, in order to be processed by some PN. If a data item has no associated Slice, it will not be delivered to a PN for processing. Each Slice is logical identified by a unique numeric ID (identifier), commonly in a range between zero and the total number of defined Slices, minus one. Thus, the DDS Topic carrying application data produced by the publishing nodes has a specific numeric field holding the Slice-ID assigned to each data item.

Unlike Virtual Servers [35] [36] [37], Slices do not behave as new PNs – as this would increase the system overhead since each Virtual Server is a node monitored and managed by the Load Balancer, which increases the CPU, memory and network overheads because more software components are instantiated – but only as a logical partition of the global data volume. The arbitrary assignment of data items to slice IDs enables the choice of load distribution with different granularities (i.e., coarse-grained or fine–grained load distribution). Because the global data item space is partitioned into the set of Slices, a higher amount of Slices allows to split the workload in smaller portions (fine-

grained), and a small amount of Slices means coarse-grained workload distribution. This problem, "balls into bins", is better explained by Martin Raab and Angelika Steger [38]. The number of DPS is also a upper bound for the maximum quantity of PNs, since each PN needs at least one Slice to get involved into the DPSLB.

The Attribution Function is responsible for choosing a valid DPS for each produced data item. The DPSLB solution requires the Attribution Function to be a very low cost function, since it has to compute/choose a DPS for each produced data item, and this data will probably be produced at a very high rate. This function may be a hash function applied to a field of the data item, to the data producer's ID, or a random value. A good candidate function for this is modulo operator (remainder of division). Attribution Function does not have to ensure that the data items are uniformly distributed over the total set of Slices since the workload can be balanced by re-arranging the number of Slices assigned to each PN.

Figure 9 illustrates an Attribution Function that consists of a hash function that applies the modulo operator to the Sensor ID field, in a configuration with ten Slices. Hence, each data item computed by the Attribution Function is assigned to one Slice. As shown, Sensor IDs 21, 1, 52 and 19 are mapped to Slice IDs 1, 1, 2 and 9, respectively. The Attribution Function must be called before the data item is published in the DDS Domain.

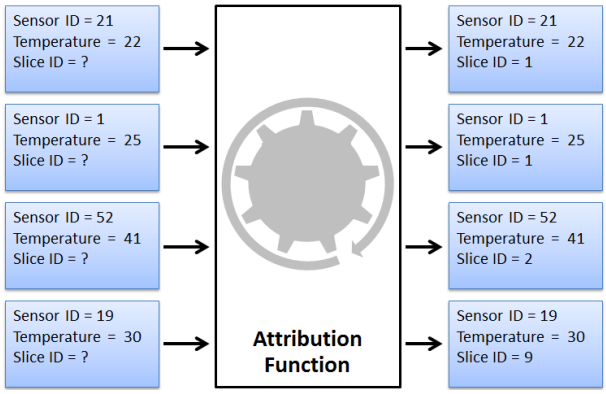

Figure 9. An example of Assignment Function applied upon data item

In our context, Load Balancing is the process of moving Slices from a PN to another. The process is started when the Load Balancer detects a load unbalance of the system and decides that some DPS should be moved to a different PN to reach a better performance. During this process both PNs involved, i.e., the Slice-giving and the Slice-taking PN, must work in a coordinated manner so to guarantee that all data items are processed, and only by one of the PNs.

The Load Balancer plays the role of coordinator of the reconfiguration actions to be executed in the Load Balancing Process, which are effectively executed by the CES `Executor` component running in the overloaded and the underloaded PNs. The algorithm within the Load Balancer has to inform which are the Slice-giving and the Slice-taking PNs and how many Slices should be moved among these PNs, thus starting the Load Balancing Process. The Load Balancing Algorithm is a generic module that can be implemented using many algorithms. This module is notified about new PNs that are able to join the DPSLB solution and called when the Load Balancer needs to analyze the system workload. After being called, the algorithm has to classify the PNs and inform how many Slices should be moved from overloaded nodes (Slice-giving) to underloaded nodes (Slice-taking). With this information, the Load Balancer is able to generate and send the corresponding commands to PNs.

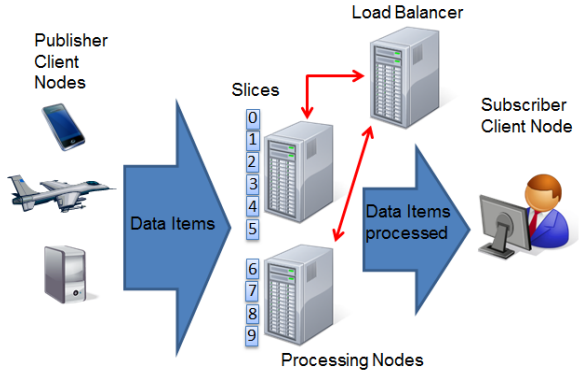

Figure 10. Interactions between clients, PNs and Load Balancer

Figure 10 illustrates the interactions between the nodes that compose the DPSLB Solution. Data items produced by Publisher Client Nodes are processed by PNs and Subscriber Client Nodes receive the processed data from PNs. The Load Balancer interacts only with PNs: both to gather their current workload and to send the load distribution actions to the corresponding PNs (depicted as red arrows in Figure 10). Figure 11 shows the redirection of the data stream when DPS-5 is moved from PN `A` to PN `B`. During the Load Balancing Process both PNs receive the data items of DPS-5, but initially none of them will process the data from this DPS. Instead, they store these received data in their local caches. Then, PN *A* sends its cached items to *B*. After receiving *A*'s cached items, PN *B* has to identify the data items that appear in both caches and then generate a `Merged Cache`, which contains all data items of DPS-5 without duplicates. Finally, DPSLB layer on B is able to notify application about the data items in the `Merged Cache`. The actions executed during the Load Balancing Process ensure that there is neither data item loss nor data item processed more than once.

If there are more than two PNs involved in the Load Balancing Process, the Load Balancer starts one Load Balancing Session for each pair of Slice-giving and Slice-taking PNs.

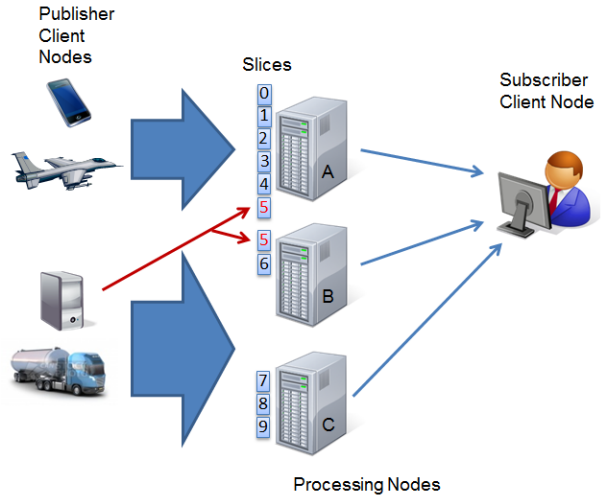

Figure 11. Data flow during Load Balancing Process

For instance, if there are one Slice-giving PN (PN A) and two Slice-taking PNs (PNs B and C), the Load Balancer starts one Load Balancing Session for PNs A and B and after this Load Balancing Session finishes, it starts the second one with PNs A and C. Each Load Balancing Session involves only two PNs. The Load Balancing processes permits not only use new PNs to increase the system´s resources but also to reduce it when some PNs are idle, which enables the system have an elasticity of resources. To do so, all Slices assigned to an idle PN should be moved to another PN before the idle PN can leave the system.

B. Implementation

1) Load Balancer: Roughly speaking, the Load Balancer receives event notifications through MAPE-SDDL, analyzes them, possibly generates an action plan and sends the corresponding commands through MAPE-SDDL to the Processing Nodes involved in the defined reconfiguration actions. An action plan is generated and sent in response of a detection of unbalance load.

The Load Balancing Algorithm executes the logic for analyzing the system load, deciding which will be the Slice-giving and Slice-taking PNs and how many Slice should be moved from the first to the latter. The Load Balancing Algorithm must implement the LoadBalancingAlgorithm interface that consists of two methods: `onNewProcessingNode()` and `analyzeLoad()`. The method `onNewProcessingNode()` is called when the Load Balancer detects that a new PN arrived in the system and `analyzeLoad()` is called every time that a new event notification is received from a PN. This last method returns a collection of Slice-Movement objects, which contains the Slice-giving and Slice-taking PN and how many Slices should be moved to the Slice-taker.

2) Processing Node: The PN is the managed resource from the perspective of the MAPE-K model used in the developed DPSLB prototype. In addition to the data processing, which is intrinsically determined by the application build upon DPSLB, each PN periodically verifies its monitored properties and, depending of their operation ranges, notifies the LES that evaluates these values against the specified expression. Hence, LES eventually sends a event notification, which holds all monitored data, to the Load Balancer through APS. The current version of this prototype periodically checks the PN´s monitored properties every two seconds and then sends a event notification to the Load Balancer.

When a PN receives a data item (or a sample in DDS jargon), it checks whether the data item is assigned to a Slice that it is responsible for. If this is the case, the PN notifies the application through the `onNewData()` method.

3) CES and Load Balancing Process: CES is the adaptation engine that enables PNs to receive actions for moving Slices as a consequence of a load redistribution action plan. The actions supported by the PN are: `addSlice`, `removeSlice`, `updateSliceState` and `sendCacheToNode`. `RemoveSlice` is used to set a Slice to the `Not In Use` state, which means that data items assigned to it can be discarded by the PN because another PN is processing these data items. On the other hand, `addSlice` changes a Slice to the `Available` state, meaning that the PN is responsible for processing the data assigned to the Slice. Therefore, the `addSlice` and `removeSlice` actions do not actually add or remove a Slice, but only change the Slice state. The action `updateSliceState` changes the Slice state to `In Load Balancing Session`, which will be hereafter explained.

During a Load Balancing Process, both Slice-giving and Slice-taking PNs should update the state of the involved Slices to `In Load Balancing Session`. After the Slice-giving PN updates and removes the Slices, the Slice-taking PN can proceed with the update action. The specific sequence of actions sent by the Load Balancer to move a DPS between two PNs are: (i) Update the DPS's state at `A` to `In Load Balancing Session` in order to cache the new data items received, (ii) Add the DPS at `B` with `In Load Balancing Session` state in order to start caching the data items, (iii) Remove DPS at `A` to inform that `A` can discard the new data items received, (iv) Update DPS's state at `B` to `Available` to inform that `B` can process the new data items and (v) Send cache from `A` to `B`. After this, `B` will generate and process the `Merged Cache`, and `A` will continue to process the data of its other Slices. The `add` and `remove` actions determine if the corresponding data items are delivered or not, respectively, to a node in a DDS Domain. This is possible by a dynamic adjustment of the subscriber filters.

The data items of a Slice cache are sent through a DDS

Topic named `CacheTopic`. The `CacheTopic` carries fields to inform the Slice ID, Slice-giving PN ID, Slice-taking PN ID and the data items, which are serialized into a `byte array`. The Slice-taking PN, after receiving a CacheTopic sample, deserializes the data items and gets its Slice local cache. With both local and remote caches, the Slice-taking PN uses a Java Set in order to generate the Merged Cache, which is a set that has no duplicated data items. After generating the Merged Cache, the Slice-taking PN removes it from the local cache and delivers the data items to the application through the data items' `application data reader listeners`.

V. Evaluation

Initial dynamic adaptation tests performed with the MAPE-SDDL middleware have already shown encouraging performance results. Regarding MAPE-SDDL's connectivity load balancing, we did the following test: We initially connected 600 simulated mobile nodes (MNs) to one Gateway, and then activated a new 'empty' Gateway. After a while, the PoA-Manager identified a load unbalance, and requested half of the MNs to migrate simultaneously to the new Gateway. At this bulk handover, all 300 MNs were able to reconnect at the new Gateway in less than 750 ms and none of the data items produced regularly (every 10 seconds) by each of the MNs was lost.

In order to evaluate the DPSLB solution and its implementation, we also developed a prototype application that utilizes the DPSLB prototype for balancing of its data processing load. This prototype application consists of clients that publish color images into the DDS domain, and PNs that receive the images, convert them to grayscale and, thereafter inform the corresponding client about completion of the image processing. Both communication paths happen trough two DDS Topics.

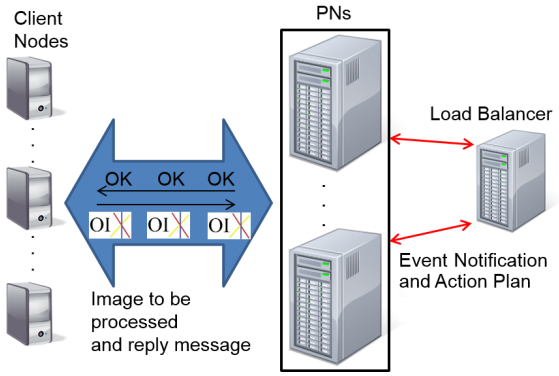

Figure 12. Deployment of the evaluation application

Figure 12 illustrates the deployment of the prototype application used for evaluation. Clients publish images through the `ClientTopic` and PN servers reply with completion notifications published into the `ServerTopic`, which are shown in Figure 13. The ClientTopic has the fields: `sliceId` (required by DPSLB to produce the merged cache); `id` of the data item; `senderId` to identify the client; `timestamp` to inform the data item creation time and `message`, that carries the serialized image. The `ServerTopic` holds fields: the data item `id`, `timestamp`, `senderId` and `message`, which carries the reply message, a serialized Java String, such as "Processed". Although this message could as well carry the result image (grayscale), this application prototype sends only a "OK" message, since the content and size of the reply message is irrelevant for evaluating the DPSLB solution. The Load Balancer analyzes the load of PNs and, transparently to the application, balances their image processing workload. It is important to stress that there is no communication, neither directly nor indirectly, between clients and the Load Balancer. Hence, the load generated by clients does not affect the Load Balancer, only the PNs.

```
struct ClientTopic
{
    long sliceId;
    DDS_KEY long long id;
    long senderId;
    long long timestamp;
    sequence<octet> message;
};

struct ServerTopic
{
    DDS_KEY long long id;
    long long timestamp;
    long senderId;
    sequence<octet> message;
};
```

(a) IDL Topics

```
public class ClientTopic {
  public int    sliceId;
  public long   id;
  public int    senderId;
  public long   timestamp;
  public byte[] message;
};

public class ServerTopic {
  public long   id;
  public long   timestamp;
  public int    senderId;
  public byte[] message;
};
```

(b) Java Topics

Figure 13. Evaluation application topics

Since the image processing done by this evaluation has no restrictions with the delivered order and dependency between each image that is processed (i.e., there is no relationship between the images published by the same client), this application may not be classified as data stream processing. However, the processing done by the application layer is totally independent of the Processing Node layer since it just delivers the data items to the application layer. Moreover, this processing task demands high CPU utilization and serves to validate that the DPSLB solution is able to effectively distribute the load among the PNs without result in data item loss/duplication.

DPSLB prototype was tested with data/image publication rates starting from 160 (1.4 MB/s) up to 1,365 (10 MB/s) data items per second. The Attribution Function of choice was the modulo operator applied on the id field, and the number of available slices was chosen to be 10. The setup used for the experiment, as ilustrated in Figure 14, was the following: 5 PNs, one Load Balancer and a Client simulator deployed on three physical machines (PMs) executing in a

LAN with bandwidth of 100 Mbps. Each PN executed on a dedicated virtual machine (VM) running the Ubuntu 12.04 32-bit Operating System, configured to use one CPU core and 512 MB. The three physical machines had following configurations: Intel i5 4 x 2.66 GHz, 8 GB DDR3 1333 MHz running Windows 7 64-bit; Intel i5 4 x 3.1 GHz, 8 GB DDR3 1333 MHz running Fedora 15 64-bit; and Intel Dual-Core 4 x 2.66, 8 GB DDR2 667 MHz running Mac OS X 10.7.5. The chosen virtualization product was Oracle VM VirtualBox since it is free and has cross-platform support.

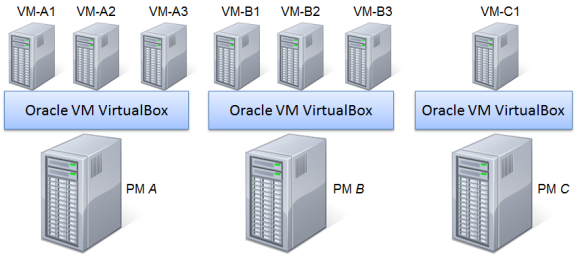

Figure 14. Deployment of the Virtual Machines

The Load Balancer and PN on VM-A1 were initiated before the evaluation starts. At initial time 0s (second zero) the Client Node was started with a data production rate of 1.4 MBs. After 18 seconds the second PN on VM-A2 was added to the system; at 25s the third PN on VM-B1 was detected by the Load Balancer; at 35s the fourth PN on VM-B2 arrived and at 45s a fifth PN on VM-B3 joined the system. Finally, at 59s the data produced by the Client Node was increased from 1.4 MBs to 10 MBs. The evaluation was finalized at time 85s.

A. Throughput

The throughput metric – expressed in data items per second (DI/s) – was used to demonstrate that an increase of the set of PNs leads to an increase of the system´s processing capacity, as expected. This metric was collected at the client side and the throughput in an instant of time represents the amount of reply messages received at the specified instant of time from all PNs.

Figure 15 shows that the system throughout increases by a nearly equal amount whenever a new PNs arrives at the system. The vertical red lines indicate the point of time when a new PN joined the system, and the green arrow indicates the time when the data production rate was increased to 10 MB/s. The throughput started from 40 DI/s and reached up to 323 DI/s at 72s. With five PNs and a data production rate of 1.4 MBs, the system was able to process up to 245 DI/s and the mean was 240 DI/s. Finally, when the client node augmented its production rate to 10 MB/s, at 59s, the throughput experienced a fall to 163 DI/s and after 6s reached 283 DI/s. From 66s until the end of the evaluation, the throughput had an average of 317 DI/s. Immediately after

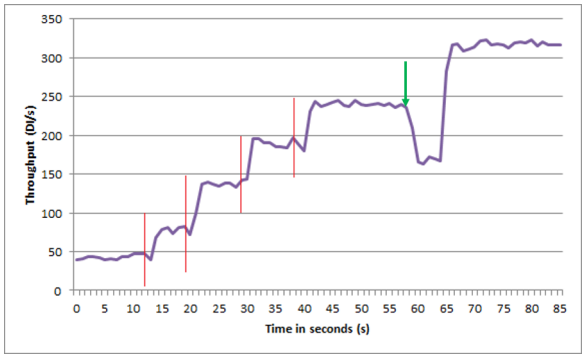

Figure 15. Throughput over the time of the experiment (DI/s X seconds)

a new PN joins the system, the throughput suffers a small retraction and after 1 second the system achieves a higher throughput level.

The decrease of the throughput at 59s may be explained by to the fact that the network and the DDS middleware had to deal with a sudden burst of the data production rate. In order to achieve better throughput and reduce the CPU and network overheads, DDS can aggregate many samples (a.k.a. data items) into a single packet and send this single packet, instead of sending many small data samples, which helps to increase the latency to send the data items and consequently decrease the throughput. Another noteworthy issue is that in this test the data production rate almost reached the theoretical network bandwidth of 100 MBs.

It is important notice that the throughput grows almost proportionally to added processing capabilities of the PNs. Specifically in this evaluation, the major capability is CPU speed, as image processing requires most resources in CPU throughput.

B. CPU Usage

The CPU usage shows that, using the modulo operator as Attribution Function, the DPSLB solution effectively achieves an even distribution of the data items over the PNs and that this data flow drives to an equal increase of the CPU usage (expressed in percentage (%)). The CPU usage was collected at each PN.

Analyzing Figure 16, it is possible to notice that as soon as the PN becomes active, its CPU usage goes up to a value higher than 90% and all PNs have a small CPU usage difference from each other. As expected, the system's load (mean load) is increased whenever a new PN arrives in the system and starts processing data items. When the data production rate was increased, at instant 59s, the system load fell from 92% to 85%, together with the throughput, but after 6s went again up to 97%.

This momentary decrease on the CPU usage probably shares the same explanation given for the throughput dip: a sudden burst of data traffic. The fall on the CPU usage

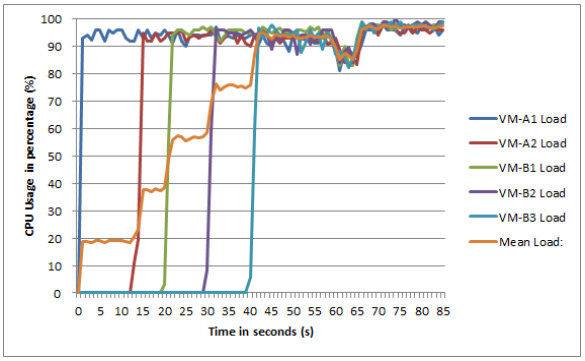

Figure 16. CPU usage over the time of the experiment (% X seconds))

suggests that it was caused by a bottleneck at the network and DDS communication layer.

C. Round-trip Delay

The round-trip delay (RTD), or Round-trip Time, is measured in seconds (s), and encompasses the time interval from the instant a client sends a data item until it receives an acknowledgment informing that the data item was successfully processed by a PN. The RTD was collected at the client side and the RTD in an instant of time that represents the mean RTD of all data items processed by all PNs at the specified instant of time. An increase of the RTD may indicate that the system is receiving more data items than it is able to process.

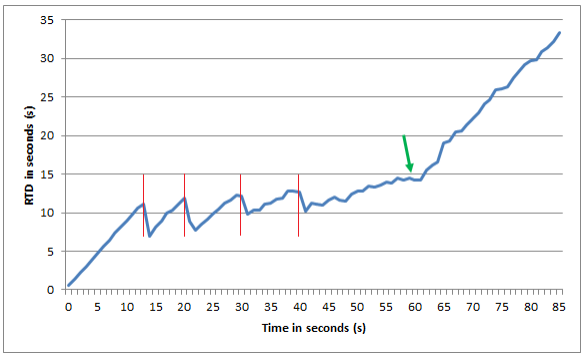

Figure 17. Round-trip Delay over the time of the experiment (RTD in seconds X time in seconds)

The RTD during the experiment is shown in Figure 17, where the vertical red lines indicate when a new PN joined the system, as those in Figure 15, and the green arrow indicates the time when the data production rate was increased to 10 MB/s. This chart reveals that the data production rate is higher than the processing capacity of the system since the RTD increases. It also shows sudden drops of the RTD whenever new PNs arrived on the system. This phenomenon can be explained by the fact that a new PN has no data items on its queue, so that the first data items it processes have a low RTD, which in turn helps to reduce the mean RTD. But after a while the data items are queued also at the new PN because it is not able to process them at the rate that they are delivered, and thus, the RTD keeps increasing.

In spite of the steady increase of the mean RTD, it is possible to observe from Figure 17 that, after instant 40s, the RTD begins to have a smoother increase: i.e., from 0s to 5s, where there was one PN, the RTD increased by approximately five seconds, while between 40s and 60s, when all 5 PNs had joined the system, the RTD increased by less than five seconds. But starting at 59s, as a result of the increase of the rate of published data items, the RTD started again rising faster than in the interval between 40s and 60s, which again is due to the insufficient processing capacity of the system against the high rate of data item production.

D. Overhead

To assess the Load Balancing overhead, we compared the throughput and the mean round-trip delay of the same image processing application in two configurations: using the DPSLB solution and without Load Balancing support. The overhead of the DPSLB solution was expressed by percentages (%) of the throughput loss, and the mean RTD increase, respectively. To evaluate the DPSLB overhead, 10,000 data items were produced with a data production rate of 1,150 data items per second (DI/s).

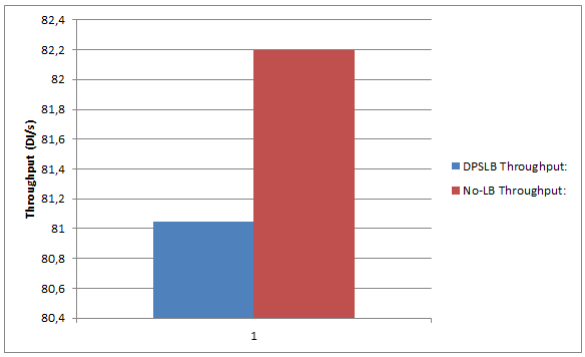

Figure 18. Mean throughput (DI/s) comparison among DPSLB solution and another without Load Balancing)

The application using DPSLB was able to process 81.044 DI/s and the application without any load balancing support, was able to process 82.194 DI/s, as shown in Figure 18. These numbers show an overhead of 1.4% introduced by the DPSLB implementation. Regarding to the RTD, shown in Figure 19, the application using DPSLB had a mean RTD of 60.45 seconds, while the application without DPSLB delivered a mean RTD of 59.51 s. This difference represents an increase of 1.58% on the RTD.

The mean time required to complete a Load Balancing Process with a data production rate of 10 MB/s and ten slices was 454 ms. While in Load Balancing Process, the DPSLB

Table I
IMPACT OF THE LOAD BALANCING PROCESS ON RTD AND THROUGHPUT

	2 PNs	3 PNs	4 PNs	5 PNs
RTD before	10.610 s	11.159 s	12.335 s	12.856 s
RTD during	11.115 s	11.944 s	12.145 s	12.652 s
RTD after	6.940 s	8.856 s	9.838 s	10.227 s
Throughput before	48 DI/s	83 DI/s	142 DI/s	189 DI/s
Throughput during	40 DI/s	72 DI/s	143 DI/s	180 DI/s
Throughput after	68 DI/s	100 DI/s	195 DI/s	231 DI/s

prototype resulted in a mean CPU overhead of 1.4% when analyzing the CPU usage on the PN that gives some slices to another. We believe that the overhead introduced by the DPSLB solution has a low cost when compared with the benefits that it introduces. Both throughput loss and RTD increase are lower than 1.6%, which seems a reasonable overhead in return of Load Balancing support.

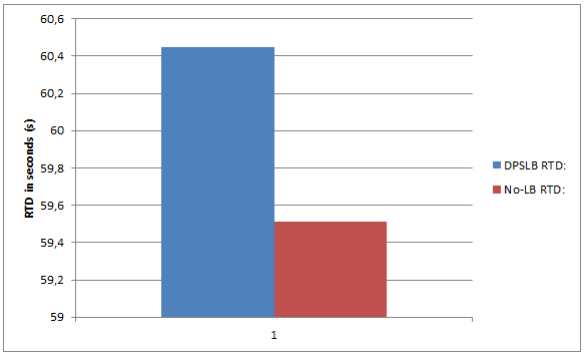

Figure 19. Mean Round-trip Delay comparison among DPSLB solution and another without Load Balancing)

The Load Balancing Process Overhead tries to capture the impact of the Load Balancing Process on the system´s throughput and mean RTD. The RTD and throughput before, during and after the Load Balancing Process are shown in Table I. The columns 2, 3, 4 and 5 PNs show the number of PNs participating in the Load Balancing Process.

When the second, third, fourth and fifth PNs arrived, the RTD was increased by 4.76% and 7.035% and decreased by 1.54% and 1.58%, respectively. The mean of the RTD overhead for all these four load balancing situations was therefore 2.167%. When a Slice-Taking PN receives Slices from two Slice-Giving PNs, the Slice-Taking PN has a Load Balancing Session for each Slice-Giving PN, which are sequentially executed. Thus, as soon as a Load Balancing Session is over, the Slice-Giving PN keeps running normally and the Slice-Taking PN is able to process data items that are assigned to the Slices received from the Slice-Giving PN. This behavior allows the PNs to start processing data items as soon as Load Balancing Session is completed and, hence, do not contribute to an increase the RTD.

Table II
LOAD BALANCING PROCESS OVERHEAD FOR DIFFERENT THE NUMBERS OF SLICES AND DATA ITEM PRODUCTION RATES

	10 Slices	100 Slices	1,000 Slices
1.4 MB/s	401 ms	422 ms	432 ms
4 MB/s	406 ms	433 ms	450 ms
10 MB/s	454 ms	479 ms	491 ms

Analyzing the throughput versus the arrival of new PNs, the throughput was decreased by 16.667%, 13.253% and 4.762% when the second, third and fifth PNs joined the system, and increased by 0.704% when the fourth PN joined, which represents a mean overhead of 8.494%. However, there is a trend towards lower overheads as more PNs join the system since the overhead starts by 16.667% till 4.762% when the second and fifth PNs joined the system, respectively. The higher throughput when the fourth PN arrived may have occurred because the Load Balancing Process involved only a single PN that was already active, which could help to maintain the throughput almost stabilized.

In order to measure the influence of the number of Slice and the data production rate on the Load Balancing Process performance, the number of Slices available was increased from 10 to 100 and 1,000 and the data production rate from 1.4 MB/s to 4 MB/s and 10 MB/s. From Table II it is possible notice that the data production rate has a higher impact on the overhead than the number of Slices. When the data production rate was increased by a factor of 10, the time required to complete the Load Balancing Process increased by in 13.217%. On the other hand, by increasing 10 and 100 times the number of Slices, this only augmented the Load Balancing Process time by 5.237% and 7.73%, respectively. This behavior suggests that the network saturation has a greater impact on the Load Balancing Process overhead than the increase of the number of Slices.

VI. RELATED WORK

There is much research and development of autonomic load balancing in middleware for distributed systems, but to the best of our knowledge, there is no other work that leverages the benefits of the MAPE-K model for dynamic adaptiveness in DDS-based systems, and more specifically, proposes a load balancing approach for mobile data stream processing that is reliable, efficient and flexible.

A common load balancing solution applied on Web Servers, cloud computing and clusters is based on centralized dispatcher [39] [40] [41] [42] [43] [44] [45] where all data stream or requests go through the dispatcher, which chooses one server node to process a set of the data stream or to accept the client request. It is important stress that this approach has a centralized load balancer that is a bottleneck and is not reasonable on Pub/Sub systems.

While in Pub/Sub systems nodes with the same subscription receive the same data, in the proposed load balancing solution PNs – which are homogeneous and have the same subscription – do not receive the same data, instead each data is delivered to a single PN in order to produce a data stream flow. To do so, the proposed solution manages how data are routed by DDS to PNs, which is simpler than routing problem in Pub/Sub systems. However, the novel proposal, [46] [47] [6] propose load balancing mechanisms for distributed systems, either for the routing layer or the data processing layer.

The work by Cheung et al. [46] has developed a load balancing mechanism to balance the subscription load among brokers on the Padres Pub/Sub system [48], where publishers or subscribers may freely migrate among brokers. While [46] focuses on the routing layer for a broker-centered Pub/Sub system and clients (publishers or subscribers) are impelled to change their *brokers* for data flow load balancing, DPSLB solution is based on DDS' P2P architecture for balancing the load among *PNs*.

REVENGE [47] is a DDS-compliant infrastructure for news dispatching among mobile nodes and that is capable of transparently balancing the data distribution load within the DDS network. In the same way, [47] only load balances the routing substrate, while MAPE-SDDL is able to load balance the mobile connections via PoA-Manager and PNs via Load Balancer in the DPSLB.

In [6], a non-coordinated load balancing approach that relies on *magnetic fields* is proposed: the idea is that underloaded nodes attract data from overloaded nodes. In an completely opposite way, the Load Balancer in DPSLB performs the MAPE-K tasks of Analysis, Planning and Execution, and carefully synchronizes the re-allocation of Data Processing Slices from one PN to another. This has the advantage of a more efficient and reliable load balancing, but the drawback of the dependability of the Load Balancer.

One of the most remarkable differences between [46] and this work is that Cheung and Jacobsen work with heterogeneous client nodes that have to receive all data that match their subscriptions. In a appositive way, the DPSLB works with homogeneous server nodes that have the same subscription but should not receive the same data. While [46] focuses on balancing the Brokers´s load by migrating clients (publishers or subscribers) to other Brokers, the proposed solution by this work relies on DDS' P2P architecture for balancing the data flow processing load among PNs rather than balancing the subscription and dissemination loads, which is transparently done by DDS and SDDL.

Similarly to Cheung and Jacobsen's work, in REVENGE [47] load balancing is focused only in the routing of subscriptions and notifications, rather than load balancing the data processing load. Unlike REVENGE, this work proposes a solution to balance the load on PNs so as to enable the deployment of new services that require a great amount of computational resources. To achieve fault tolerance and a better load balancing on the routing layer, REVENGE works with the concept of multi-domain communication and "hot copies" of the routing substrate. Our work neither works with multi-domains nor have capabilities to support fault tolerance on its load balancing.

Magnetic Field [6] approach is a decentralized and not coordinated load balancing where there is not a Load Balancer. Thus, the nodes communicate with each other to build the magnetization network. Our work, on the other hand, is a coordinated and global load balancing where Load Balancer is in charge of gathering load information about nodes and making and managing the load redistribution actions among nodes. Differently from the message attractions in magnetic fields, our approach selects a single PN that must process each data (message). To do so, the proposed solution manages the data routing done by DDS.

While Calsavara and Lima Jr's approach [6] relies on the attraction of messages through the magnetization relationship, this works proposes an approach that relies on slices, which is expected to provide an efficient load balancing system that is able to directly delivery messages (data) to the appropriate nodes on DDS-based systems.

The main advantage of employ coordinated and global load balancing is that better analyzes and decisions can be done since the Load Balancer is able to gather information about all PNs after take any decision. Since the Load Balancer act also as a coordinator for the PNs during a Load Balancing Process, there is no need for complex autonomic algorithms and leader election at the PNs to decide which actions should be executed by each PN to realize the load balancing without conflicts. The clear disadvantage of this approach is that the Load Balancer may be a point of failure and bottleneck for the system´s scalability when there are hundreds of thousands of PNs since the Load Balancer has to analyze the load of all PNs.

Although there are many other load balancing approaches that are found both in academia and industry, none of them explores the capability of the node to receive all data published in a DDS Domain without the need of Brokers or a central dispatcher. In order to effectively realize a processing load balancing in a DDS Domain, a possible approach is simply managing the subscription filters so to control the data stream processing. Therefore, all data routing and its optimization is responsibility of DDS.

VII. Discussion

This paper proposed a novel approach to load balancing mobile connections and data streams based on the MAPE-K model, which entails several advantages that go far beyond a simple boost of performance. Most current load balancing methods are quite inflexible, since they always make same sorts of decision, without considering that the system may require different load distribution approaches depending

on the current state of data stream processing (high/low load) or the state of the infra-structure (e.g., node failures, communication failures, re-organization, etc.). Moreover, by being disassociated from any Autonomic architecture, traditional load balancing mechanisms fail to incorporate self-monitoring, self-analysis and self-adaptation behavior as response to changes in the execution environment.

Since our load balancing mechanism is structured according to the MAPE-K model it is able to deliver sustained load balancing performance under various conditions. For example, based on the information collected by MS, CES is capable of adjusting parameters of the load balancing algorithm directly (i.e., parametric adaptation). For other changes of conditions, CES may even substitute the load balancing algorithm by a more effective one. Adaptation can also be used to circumvent failures of PNs and Gateways. For these cases, the APS can choose the best parameters, algorithms or techniques for handling outage of failed elements, and recovery actions. Finally, it is also possible to implement a machine learning technique in the APS, which would allow the load balancing mechanism to anticipate future change demands, and thus react in a more effective and efficient way. This knowledge about past behavior and adaptation performance of the system would have to be represented and analyzed by the adaptation logic, which is a feature made possible by the MAPE-K model.

Furthermore, our load balancing approach for Data Stream Processing is targeted at DDS-based systems, which support fully decentralized system architectures. It is a generic solution, transparent to the SDDL applications, able to route data streams to PNs with low overhead and is inherently scalable, i.e., it supports large numbers of nodes and large-volume data streams. Also, since PNs can dynamically join or leave the system during operation, DPSLB supports seamless variations of computational resources.

The DPSLB Solution works with any type of application object/message and is totally transparent to the application developers, who must only inform which DDS Topics are subject to load balancing by DPSLB. They can still customize their applications with the DDS QoS policies of their choice. For example, fault tolerance can be achieved by deploying replicas of a PN responsible for some data slices, and using the *Ownership* DDS QoS policy to dynamically switch between the redundant output flows produced by the PN replicas.

Finally, the DPSLB also supports customization since several load balancing algorithms (i.e., implemented in APS of MAPE-SDDL) can be applied in the Load Balancer. For example, one could deploy an algorithm which seeks the uniform load distribution, or otherwise, use another algorithm which tries to minimize the system's global energy consumption.

VIII. Conclusion and Future Work

The need for remote monitoring and high performance processing of large mobile data streams in a timely manner is becoming common to many systems such as Intelligent Transportation Systems, Fleet Management and Logistics, and integrated Industrial Process Automation.

The main contribution of this work is the development of a novel approach to load balancing that has two main novelties: its autonomic behavior based on MAPE-K model and the use of DDS as its communication infra-structure. The underlying middleware, MAPE-SDDL, supports not only load balancing of mobile connections among different Gateways but also node balancing of data stream processing across multiple PNs. To the best of our knowledge MAPE-SDDL is the first middleware that has developed an autonomous load balancing approach tailored to DDS-based systems. Preliminary performance evaluations have shown encouraging results what motivate us to continue the development of SDDL and its autonomic extensions.

By being disassociated from any autonomic reference model, traditional load balancing mechanisms fail to incorporate self-* properties, which are the pillars for the development of more adaptive and scalable systems. Moreover, most of the traditional load balancing approaches are not well suited for high-throughput mobile communication and data stream processing systems, as they are not based on a communication layer with real-time communication capabilities. On the other hand, our load balancing approach was specially designed for decentralized systems based on the DDS standard, and hence is capable of fulfilling application requirements such as real-time and high throughput data communication and processing, scalability and fault tolerance.

The evaluation has yielded encouraging performance result, which motivate us to proceed with the development of SDDL's adaptivity and load balancing capabilities. In particular, we could check that during the Load Balancing Process there was neither any data item loss nor duplication. It could also be noticed that the addition of new PNs effectively enhances the system´s processing capacity and does not drive to an rise of the overhead. For example, the overall throughput could be augmented from 40 DI/s for one PN, to 323 DI/s, when five PNs are used. While new PNs help to increase the throughput, more PNs reduce the RTD – or at least to reduce the growth – because the load is divided across the available PNs since the system is able to promptly process more data items in the same unit of time. Most importantly, the proposed DPSLB solution allows a stream processing system to scale in the number of PNs with acceptable overhead. The overhead introduced by the DPSLB prototype represents only 1,4% and 1,58% of the throughput and RTD, respectively, which is a low cost compared to the benefit of having a load balancing mechanism.

With a data stream rate of 10 MB/s and 1.000 Slices, the DPSLB prototype was able to move 500 Slices from a single PN to a second PN in less than 500 ms (milliseconds), which is fast enough for many stream processing applications. And regarding load balancing of mobile node (MN) connections, MAPE-SDDL is able to migrate 300/600 MNs from one Gateway to another in less than 750 ms.

As future work, we are planning the design, implementation and evaluation of other Attribution Functions and load balancing algorithms, both for data stream processing and connectivity load distribution, as well as developing support for general state transfers among the managed resources (PNs or Gateways) during Load Balancing Process. New Attribution Functions would be valuable to study how the DPSLB solution behaves in face of uneven Attribution Function and heterogeneous PN resource capacities. In particular, we believe that collecting statistical data about the data items received in each Slice, and the corresponding workload associated to each Slice will certainly enable much better Load Balancing Algorithms. Thereby, they would be able to take into account the different data stream rates assigned to each Slice, and to decide which specific Slices, not only how many, should be moved among the PNs.

We also plan to design and implement a new component in MAPE-SDDL, called Distributed Event Service (DES), which will allow the detection of composite events made of basic events from different event sources (e.g., distributed PNs). DES is required for cases where the decision to reconfigure the system must consider the combination of events detected by several LES at distributed resources. For example, it could be used to detect the overload in a group of PNs, instead of the overload detection at individual PNs. The DES will thus enable the load balancing mechanism to be driven by a global perspective on a distributed group of Nodes.

ACKNOWLEDGMENT

"This work is partly supported by project Mobile InfoPAE, CNPq scholarships nº. 310253/2011-0 and 140966/2013-7, and FAPEMA."

REFERENCES

[1] R. O. Vasconcelos, M. Endler, B. d. T. P. Gomes, and F. J. d. S. e. Silva, "Towards Autonomous Load Balancing for Mobile Data Stream Processing and Communication Middleware Based on Data Distribution Service," in *ICAS 2013: The Ninth International Conference on Autonomic and Autonomous Systems*, Lisbon, 2013, pp. 7–13.

[2] R. O. Vasconcelos and M. Endler, "A Dynamic Load Balancing Mechanism for Data Stream Processing on DDS Systems," M.Sc Thesis, Departamento de Informatica, PUC-Rio - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2013.

[3] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, Dec. 2005.

[4] A. Margara and G. Cugola, "Processing flows of information," in *Proceedings of the 5th ACM international conference on Distributed event-based system - DEBS '11*. New York, New York, USA: ACM Press, 2011, p. 359.

[5] Waze, "Free GPS Navigation with Turn by Turn - Waze," 2013. [Online]. Available: http://www.waze.com/. [Accessed Dec. 18, 2013].

[6] A. Calsavara and L. A. P. Lima Jr., "Scalability of Distributed Dynamic Load Balancing Mechanisms," in *ICN 2011 The Tenth International Conference on Networks*, no. C, 2011, pp. 347–352.

[7] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing," in *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2010, pp. 551–556.

[8] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[9] IBM, "An architectural blueprint for autonomic computing," *IBM White Paper*, 2006.

[10] M. C. Huebscher and J. a. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Computing Surveys*, vol. 40, no. 3, pp. 1–28, Aug. 2008.

[11] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu, "The Autonomic Computing Paradigm," *Cluster Computing*, vol. 9, no. 1, pp. 5–17, Jan. 2006.

[12] R. Sterritt, "Autonomic computing," *Innovations in Systems and Software Engineering*, vol. 1, no. 1, pp. 79–88, Mar. 2005.

[13] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[14] M. Parashar and S. Hariri, "Autonomic computing: An overview," in *In Proceedings of the 2004 international conference on Unconventional Programming Paradigms (UPP'04)*, J.-P. Banâtre, P. Fradet, J.-L. Giavitto, and O. Michel, Eds. Le Mont Saint Michel: Springer-Verlag, Berlin, Heidelberg, 2005, pp. 247–259.

[15] OMG, "Object Management Group," 2013. [Online]. Available: http://www.omg.org/. [Accessed Dec. 18, 2013].

[16] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," *ICDCSW '03 Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[17] C. Tucker, "What can DDS do for You? Learn how dynamic publish-subscribe messaging can improve the flexibility and scalability of your applications." *OMG Whitepapers: Data Distribution Service Portal*, 2013.

[18] M. Xiong, J. Parsons, J. Edmondson, H. Nguyen, and D. C. Shmidt, "Evaluating the Performance of Publish/Subscribe Platforms for Information Management in Distributed Real-time and Embedded Systems," *OMG Whitepapers: Data Distribution Service Portal*, 2010.

[19] L. David, R. Vasconcelos, L. Alves, R. André, and M. Endler, "A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes," *Journal of Internet Services and Applications (JISA)*, vol. 4, no. 1, p. 16, 2013.

[20] G. Pardo-Castellote, B. Farabaugh, and R. Warren, "An introduction to DDS and data-centric communications," 2005. [Online] Avaliable: http://www.omg.org/news/whitepapers /Intro_To_DDS.pdf. [Accessed Dec. 18, 2013].

[21] G. Pardo-Castellote, "DDS Tutorial – Part II - Hands On," 2009. [Online]. Available: http://www.omg.org/news/ meetings/GOV-WS/pr/rte-pres/DDS_Tutorial_RTEW09.pdf. [Accessed Dec. 18, 2013].

[22] T. O. Computing, "Learn About How it Works: Take the CoreDX DDS Tour Twin Oaks Computing, Inc," 2012. [Online]. Avaliable: http://www.twinoakscomputing.com/coredx/dds_tour. [Accessed Dec. 08, 2013].

[23] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer.*, vol. 36, pp. 41–50, January 2003.

[24] I. Naick, "Make autonomic computing a reality with IBM Tivoli. Using IBM Tivoli Provisioning Manager and IBM Tivoli Intelligent Orchestrator to create an on demand environment," *IBM White Paper*, 2004.

[25] A. K. Y. Cheung, "Dynamic Load Balancing in Distributed Content-based Publish/Subscribe," Ph.D. dissertation, M.Sc Thesis, Graduate Department of Electrical and Computer Engineering, University of Toronto, 2006.

[26] N. Shivaratri, P. Krueger, and M. Singhal, "Load distributing for locally distributed systems," *Computer*, vol. 25, no. 12, pp. 33–44, Dec. 1992.

[27] T. Casavant and J. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141–154, 1988.

[28] D. Grosu and A. Chronopoulos, "Algorithmic Mechanism Design for Load Balancing in Distributed Systems," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 1, pp. 77–84, Feb. 2004.

[29] A. Osman and H. Ammar, "Dynamic load balancing strategies for parallel computers," in *International Symposium on Parallel and Distributed Computing (ISPDC)*, 2002.

[30] R. O. Vasconcelos, L. Silva, L. Alves, and M. Endler, "Scalable Data Distribution Layer - Overview, Use Instructions and Download," 2012. [Online]. Avaliable: http://www.lac-rio.com/sddl/. [Accessed Dec. 18, 2013].

[31] R. O. Vasconcelos, L. David, L. Alves, R. André, and M. Endler, "Real-time Group Management and Communication for Large-scale Pervasive Applications," Rio de Janeiro, 2012, Monografias em Ciência da Computação - MCC 05/2012, Dep. de Informática, PUC-Rio, ISSN 0103-9741.

[32] L. David, R. Vasconcelos, L. Alves, R. André, G. Baptista, and M. Endler, "A Large-scale Communication Middleware for Fleet Tracking and Management," in *Salão de Ferramentas, Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2012)*, Ouro Preto, 2012.

[33] M. Endler, R. O. Vasconcelos, L. David, R. André, and L. Alves, "A DDS-based middleware for scalable tracking and communication of wireless-connected mobile nodes in a WAN," Rio de Janeiro, 2012, Monografias em Ciência da Computação - MCC 06/2012, Dep. de Informática, PUC-Rio, ISSN 0103-9741.

[34] Oracle, "Getting Started Using Java RMI," 2013. [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/ guides/rmi/hello/hello-world.html. [Accessed Dec. 18, 2013].

[35] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proceedings of the eighteenth ACM symposium on Operating systems principles - SOSP '01*. New York, New York, USA: ACM Press, 2001, p. 202.

[36] A. Rao, K. Lakshminarayanan, and S. Surana, "Load balancing in structured P2P systems," in *Proceedings of IPTPS*, 2003, pp. 68–79.

[37] L. Xia, H. Duan, X. Zhou, Z. Zhao, and X.-W. Nie, "Heterogeneity and load balance in structured P2P system," in *2010 International Conference on Communications, Circuits and Systems (ICCCAS)*. IEEE, Jul. 2010, pp. 245–248.

[38] M. Raab and A. Steger, ""Balls into Bins"—A Simple and Tight Analysis," *In Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98)*, pp. 159–170, 1998.

[39] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. Greenberg, "Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services," *Performance Evaluation*, vol. 68, no. 11, pp. 1056–1071, Nov. 2011.

[40] C.-C. Yang, C. Chen, and J.-Y. Chen, "Random Early Detection Web Servers for Dynamic Load Balancing," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*. IEEE, 2009, pp. 364–368.

[41] V. Suresh, D. Karthikeswaran, V. Sudha, and D. Chandraseker, "Web server load balancing using SSL back-end forwarding method," *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, pp. 822–827, 2012.

[42] Z. Zhang and W. Fan, "Web server load balancing: A queueing analysis," *European Journal of Operational Research*, vol. 186, no. 2, pp. 681–693, Apr. 2008.

[43] D. C. Shadrach, K. S. Balagani, and V. V. Phoha, "A Weighted Metric Based Adaptive Algorithm for Web Server Load Balancing," in *2009 Third International Symposium on Intelligent Information Technology Application*. IEEE, 2009, pp. 449–452.

[44] T. C. Chieu, A. Mohindra, A. a. Karve, and A. Segal, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in *2009 IEEE International Conference on e-Business Engineering*. IEEE, 2009, pp. 281–286.

[45] A. Corsaro, "DDS in SCADA, Utilities, Smart Grid and Smart Cities," 2012. [Online]. Available: http://www.slideshare.net/Angelo.Corsaro/dds-in-scada-utilities-smart-grid-and-smart-cities

[46] A. K. Y. Cheung and H.-A. Jacobsen, "Load Balancing Content-Based Publish/Subscribe Systems," *ACM Transactions on Computer Systems*, vol. 28, no. 4, pp. 1–55, December 2010.

[47] A. Corradi, L. Foschini, and L. Nardelli, "A DDS-compliant infrastructure for fault-tolerant and scalable data dissemination," in *The IEEE symposium on Computers and Communications*. IEEE, June 2010, pp. 489–495.

[48] G. Li and H.-A. Jacobsen, "Composite subscriptions in content-based publish/subscribe systems," in *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware (Middleware '05)*, G. Alonso, Ed. Grenoble, France: Springer-Verlag New York, Inc., 2005, pp. 249–269.