Towards Autonomous Load Balancing for Mobile Data Stream Processing and Communication Middleware Based on Data Distribution Service

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*—Intelligent Transportation Systems, Fleet Management and Logistics, and integrated Industrial Process Automation share the requirement of remote monitoring and high performance processing of huge data streams produced by large sets of mobile nodes. The deployment and operation of infra-structures enabling such mobile communication and data stream processing have two key requirements: they must be capable of (a) handling large and variable numbers of wireless connections to the monitored mobile nodes regardless of their current use or locations, and (b) automatically adapt to variations in the volume of the mobile data streams. This article describes a mechanism for load balancing of mobile data streams based on the autonomic reference model MAPE-K. The autonomic capability has been incorporated into a scalable middleware system based on the OMG DDS standard and aimed at real-time and adaptive handling of mobile connectivity and data stream processing for great sets of mobile nodes. Besides explaining the autonomic extension of this middleware (MAPE-SDDL) and the generic load balancing approach implemented, we also present results of initial tests and discuss the potential benefits of using this model for general dynamic adaptivity in distributed middleware.**

***Keywords*-Load balancing; Data Stream Processing; Autonomic computing; DDS; mobile communication middleware.**

I. INTRODUCTION

As the public and private sectors are increasingly investing in Intelligent Transportation Systems for smarter cities, Logistics, and integrated Industrial Process Automation that depend on monitoring of the real world objects (and people) through wireless interfaces, systems for mobile tracking, communication and coordination are becoming commonplace. Such applications share the requirement of remote monitoring and high performance processing of huge data streams produced by large sets of mobile nodes, which may be vehicles, people, or other smart objects with embedded sensors. Although some sorts of data processing may be performed locally at the mobile nodes (i.e., simple sensor data encoding or interpretation), most other application-relevant information about the monitored mobile system as a whole that requires execution of complex analysis and correlation functions over these mobile data streams, in real-time, has to be done by dedicated machines in a cluster or cloud. For example, it may be necessary to identify all the nodes that are located inside a region affected by some problem/accident and which may imply some danger or cost to the corresponding nearby mobile users. Moreover, these systems must also support timely and reliable communication with the monitored mobile nodes, in order to send instructions, share alternative routes, make enquiries or disseminate alerts, either to nodes individually, or to groups of nodes [1].

However, today's mobile communication and data stream processing systems lack important features that are necessary in order to support the large amounts of data flows envisioned by the massive and ubiquitous dissemination of sensors and mobile devices in industry and city-scale applications. In particular, the deployment and 24x7 operation of such mobile data stream processing and communication infra-structures pose two intrinsic technical challenges: they must be capable of (a) handling huge and variable numbers of connections to the monitored mobile nodes regardless of their current locations, and (b) automatically adapt to variations in the volume of the mobile data streams, either because of sudden increase in the set of nodes (e.g. a popular event happening in a region), intensified message exchange among the mobiles, or because new sensed data is required for a more precise analysis of a regional problem, such as road defect or an potential accident. 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.

The remainder of this paper is organized as follows: Section II gives an overview of the enabling technologies, which are the Data Distribution Service for Real-Time Systems standard and the MAPE-K reference model for autonomic systems; 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 shows initial performance results of the implemented system using a prototype application. Section V reviews related work on load balancing for Publish/Subscribe systems, including DDS. Section VI discusses the advantages of using an autonomic approach for load balancing and the benefits of the load balancing solution proposed by this work. Section VII contains with some concluding remarks about the central ideas presented in this work and with probable lines of future work on the subject.

II. Enabling Technologies

The efficiency, scalability and adaptiveness of our middleware system builds upon a combination of the following two key technologies.

A. Data Distribution Service for Real-Time Systems

The Data Distribution Service for Real-Time Systems (DDS) [2] is an Object Management Group (OMG) standard which specifies a publish/subscribe communication infrastructure aimed at high performance and real-time distribution of critical information in distributed systems. This specification was designed with the intention of obtaining a high scalability, portability and interoperability [3]. The DDS was conceived around a Data-Centric Publish-Subscribe (DCPS) model based on topics, which makes the complex programming of distributed communication protocols transparent to the developer. Topics are typed structures that connect Publishers to Subscribers and is where it is located the information that will be exchanged on the network. Publishers and Subscribers of a DDS Domain (the collection of nodes pertaining to a single application) are containers for Data Writers and Data Readers, respectively, which exchange typed data through a common Topic.

Specifically, the DCPS automatically manages delivery of all DDS messages in a totally decoupled and asynchronous way, i.e. without requiring the application to explicitly determine which will be the sender and receiver of each message, or handle message acknowledgements and retransmissions. 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. 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) [3] [4].

Unlike traditional Publish-Subscribe middleware, DDS can explicitly control the latency and efficient use of network resources through fine-tuning of its Network Services, that are critical for implementing real-time and soft real-time systems that use QoS policies such as Deadline, Latency Budget, Transport Priority, etc.

Among the several existing implementations, we chose CoreDX DDS [5] as the basic communication substrate. The reason for choosing DDS as the communication layer is the fact that it is focused on high-performance and low-overhead, with great latency and message throughput numbers. CoreDX DDS includes fundamental design principles aimed to meet the requirements of real-time and near-real time systems, including minimal data copies, compact encoding on the wire, light-weight notification mechanisms, pre-allocation of resources and pre-compilation of type-specific code blocs.

B. MAPE-K: a Reference Model for Autonomic Systems

The MAPE-K [6] (*Monitoring, Analysis, Planning, Executing and Knowledge*) model, illustrated in Figure 1, is a general architecture for the development of autonomic software components, as proposed by IBM [7]. This model is being increasingly used to interrelate the architectural components of autonomic systems. It is divided into two main components: autonomic manager and managed resource.

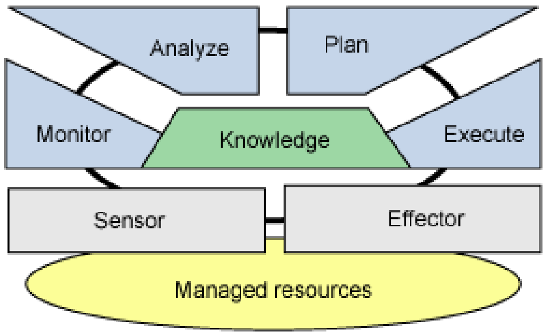

Figure 1. MAPE-K Autonomic Architecture

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. This model requires two types of touch points within and outside the managed resource: sensors and effectors. Only they have direct access to the managed resource. Sensors are responsible for collecting information from the managed resource, which can be, for instance, the customers requests response time, if the managed resource is a Web server. The information collected by the sensors are reach the monitors, where they are interpreted, classifieds and placed in a higher level of abstraction. They are then sent to the next step of the cycle, the analysis and planning phase. This stage produces an action plan, which consists of a set of adaptation actions to be performed by the executor. The effectors are the components that allow the autonomic managers to perform adjustments in the managed resources. The decision of which adaptation actions must be applied in a given situation requires a knowledge representation of the computing system 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 services of the autonomic manager.

III. The SDDL Middleware and its Extensions

A. *Overview of the Scalable Data Distribution Layer*

The Scalable Data Distribution Layer (SDDL) [4] [8] 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 2. 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) 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 2 shows other nodes in SDDL that are Load Balancer, PoA-Manager and Processing Nodes. All nodes showed in Figure 2 will be explained throughout this work.

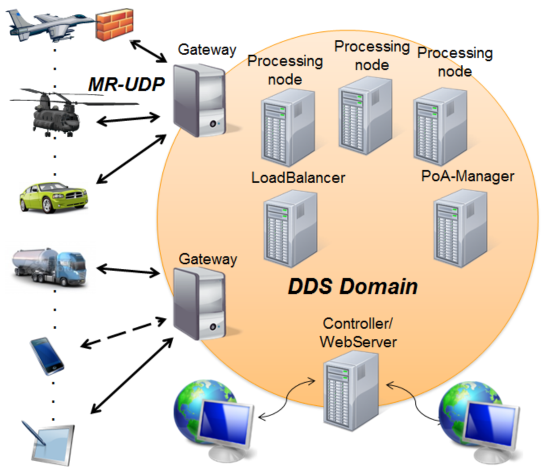

Figure 2. SDDL Architecture

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: an Autonomic Extension for the 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. It comprises four services: Monitoring Service (MS), Local Event Service (LES), Analysis and Planning Service (APS), and Control and Executing Service (CES).

The MS collects data from any SDDL 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 MNs (by each Gateway) or number of DPSs assigned to each Processing Node (see DPS concept in Section IV-A). 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 (located at the same node) whenever the monitored property switches its operation range, which might indicate a significant change on resource usage.

The LES receives these range change events from the MS and publishes event notifications to subscribed components. Events are occurrences which 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 events when short-lived situations occur (e.g., a CPU load peak on a Processing Node during a few seconds).

The APS analyzes the received event notifications and makes a diagnosis of the problems to be solved. Mobile connection overload on the Gateways, and unbalanced load between Processing Nodes are examples of problems that are already diagnosed by the APS of MAPE-SDDL. After diagnosis, the APS will seek the dynamic reconfiguration actions to resolve the problem, and then build an appropriate action plan. The decision-making for building the plan is defined by the user through the use of rules and a rule processing engine. Each type of reconfiguration action supported by CES receives a UID. This identifier is included in the action plan in order to allow the CES instances to know what reconfiguration actions must be performed. The action plan for mobile connectivity management 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. In the case of the load balancing process, the action plan is a sequence of actions to the Processing Nodes, as will be explained in Section IV-A

Finally, the CES is the adaptation engine that applies the corresponding reconfiguration actions at the resources in response to their availability/load changes. Among the types of dynamic reconfiguration actions supported is the ability of moving DPSs from a Processing Node to another (cf. Section IV-A) The ability to migrate to some set of MNs form one Gateway to another (cf. Section III-A) is also implemented by CES, which in this case, resides in the mobile Client Lib, which performs the disconnection from one Gateway and the reconnection to the new Gateway. In the load balancing process the CES is implemented as part of the Cache manager at each Processing Node.

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). 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).

The types of DDS nodes that compose the DPSLB approach, showed in Figure 2, are *PNs*, which execute the MS, LES and CES (only Execution Service) services, and the *Load Balancer*, which executes the APS and CES (only Control Service) services of the MAPE-SDDL. 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.

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 have assigned 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.

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). The Attribution Function does not have to ensure that the data items are uniformly distributed over the total set of *Slices*.

Load Balancing Process 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 load balance. 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 actions executed in the *Load Balancing Process*, which are effectively executed by the overloaded and the underloaded PNs. Figure 3 illustrates the redirection of the data stream when *Slice* 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

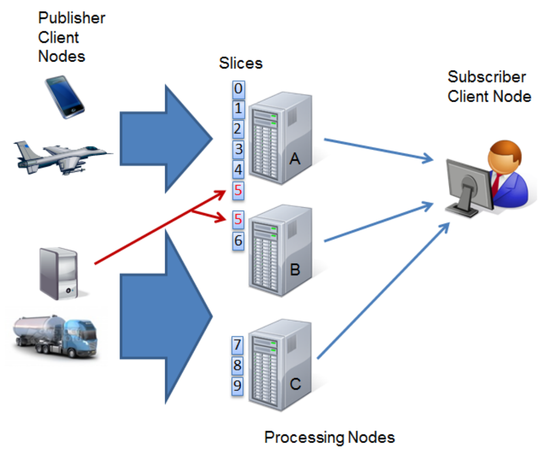

Figure 3. Data flow during Load Balancing Process

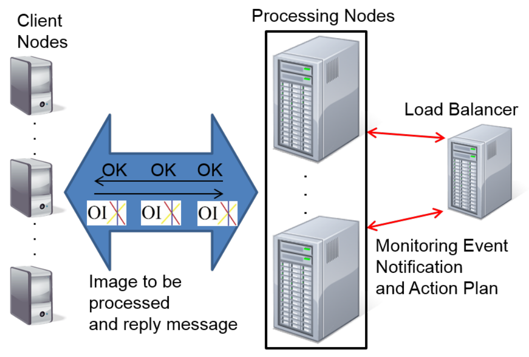

Figure 4. Deployment of the evaluation application

duplicates. Thus, the specific sequence of actions sent by the Load Balancer to move a DPS between two PNs are: (i) Update DPS's state at *A* to *Cache Data Items*, (ii) Add DPS at *B* with *Cache Data Items* state, (iii) Remove DPS at *A*, (iv) Update DPS's state at *B* to *Process 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.

B. Preliminary Test Results

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 developed a prototype application (in Java and using CoreDX DDS) 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 4 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*. The ClientTopic has 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.

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 slice was chosen to be ten. The setup used for the experiment was the following: 5 PNs, one Load Balancer and a Client simulator deployed on three Physical Machines. executing in a LAN with bandwidth 100 Mbps. Each PN executed on a dedicated Virtual Machine running the Ubuntu [64] 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.

To assess the Load Balancing overhead, we compared the throughput and the mean RoundTripDelays (RTD) of the same image processing application using CoreDX DDS 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). 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. These numbers show an overhead of 1,4% introduced by the DPSLB implementation. Regarding to the RTD, 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.

V. Related Works

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. However, [9] [10] [11] propose load balancing mechanisms for distributed systems, either for the routing layer or the data processing layer.

The work by Cheung et al. [9] has developed a load balancing mechanism to balance the subscription load among Brokers on the Padres Pub/Sub system, where publishers or subscribers may freely migrate among Brokers. While [9] 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 *Processing Nodes*.

REVENGE [10] is a DDS-compliant infrastructure for news dispatching among mobile nodes and which is capable of transparently balancing the data distribution load within the DDS network. In the same way, [10] only load balances the routing substrate, while MAPE-SDDL is able to load balance the mobile connections via PoA-Manager and processing nodes via Load Balancer in the DPSLB.

In [11], 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 Processing Node to another. This has the advantage of a more efficient and reliable load balancing, but the drawback of the dependability of the Load Balancer.

VI. 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 Processing Nodes 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 in 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 Processing Nodes with low overhead and is inherently scalable, i.e. it supports large numbers of nodes and large-volume data streams. 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.

Also, since Processing Nodes can dynamically join or leave the system during operation, DPSLB supports seamless variations of computational resources, i.e. scalable systems.

Finally, the DPSLB also supports customization since several load balancing algorithms (i.e. implemented in APS of MAPE-SDDL) can be applied at the Load Balancer.

VII. 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 Processing Nodes. 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.

Preliminary tests have yielded encouraging performance result, which motivate us to proceed with the development of SDDL's adaptivity and load balancing capabilities. For a data stream production rate of 10 MB/s, DPSLB is able to complete the Load Balancing Process of 5/10 DPS in 454 ms. And regarding load balancing of mobile node (MN) connections, MAPE-SDDL is able to migrate 300/600 MNs from one Gateway to another Gateway in less than 750 ms.

As future work, we are planning the design, implementation and evaluation of other 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 (Processing Nodes or Gateways) during Load Balancing Process. 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 Processing Nodes).

Acknowledgment

"This work is partly supported by project Mobile InfoPAE and CNPq scholarship n^{o}. 310253/2011-0"

References

[1] S. Karnouskos and A. Colombo, "Architecting the next generation of service-based scada/dcs system of systems," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, November 2011, pp. 359 –364.

[2] G. Pardo-Castellote, "OMG Data Distribution Service: Architectural Overview," in *Proc. of the IEEE Military Communications Conference (MILCOM '03)*, vol. 1, Octuber 2003, pp. 242 – 247.

[3] 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," 2010.

[4] L. D. Silva, R. Vasconcelos, R. A. Lucas Alves, G. Baptista, and M. Endler, "A communication middleware for scalable real-time mobile collaboration," in *Proc. of the IEEE 21st International WETICE, Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architectures (AROSA)*, June 2012, pp. 54–59.

[5] I. Twin Oaks Computing, "Twin oaks computing, inc." April 2012. [Online]. Available: http://www.twinoakscomputing.com/

[6] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, January 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1160055

[7] IBM, "An architectural blueprint for autonomic computing," *IBM White Paper*, 2006. [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf

[8] "Scalable data distribution layer - overview, use instructions and download," 2012. [Online]. Available: http://www.lac-rio.com/sddl/

[9] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=1880020http://portal.acm.org/citation.cfm?doid=1880018.1880020

[10] 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. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5546756http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5546756

[11] 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.