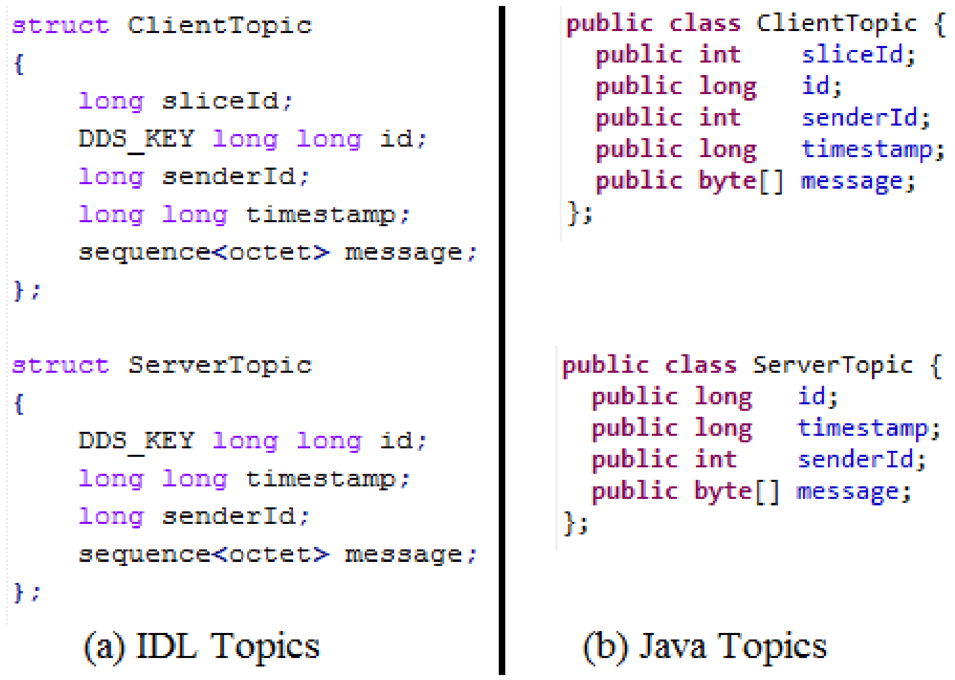
5
Evaluation

In order to evaluate the Data Processing Slice Load Balancing solution proposed and implemented in this thesis, it was developed a prototype application that utilizes the DPSLB prototype for balancing of data processing load. This prototype application consists of clients that publish into the DDS domain color images and servers – behaving as PNs – that receive the images, convert them to grayscale and, after processing each image, inform the corresponding client about completion of the image processing. This communication happens trough two DDS Topics, as follows.



```
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;
};
```

```
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;
};
```

Figure 23 – Evaluation application topics

Figure 23 (a) shows the *ClientTopic* and *ServerTopic* defined in Interface Description Language and Figure 23 (b) shows the corresponding Java classes generated from the IDL Topics. Clients send images through the *ClientTopic* and servers send completion notifications using the *ServerTopic*. The *ClientTopic* has fields: *sliceId* (required by DPSLB); *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* and *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.

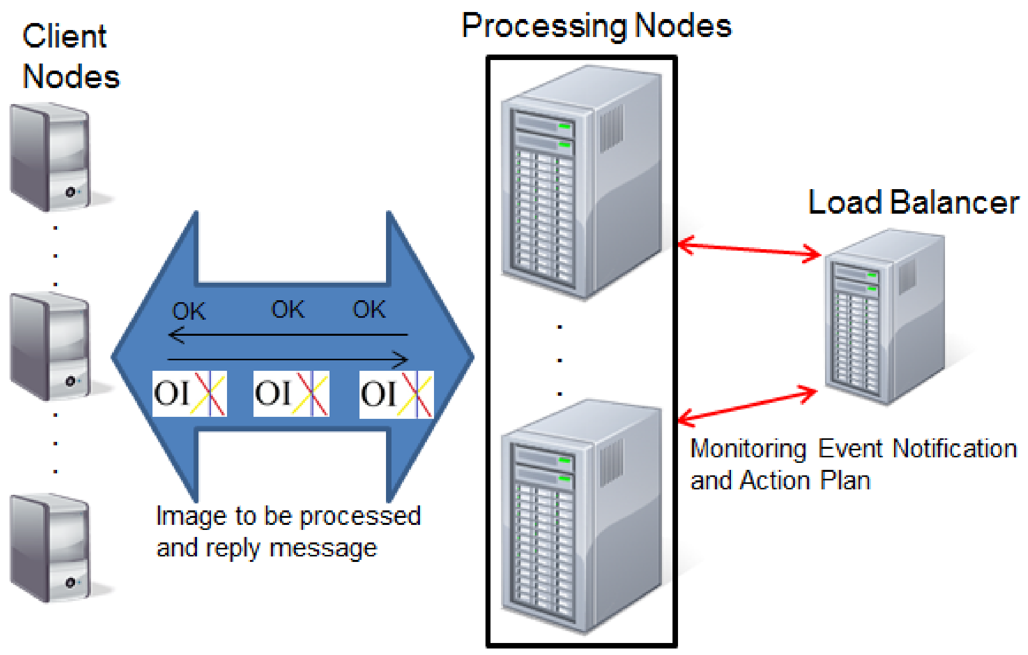

Client Nodes
Processing Nodes
Load Balancer
OK
OK
OK
OI
OI
OI
Monitoring Event Notification and Action Plan
Image to be processed and reply message

Figure 24 – Deployment of the evaluation application

Figure 24 illustrates the deployment of the prototype application used for evaluation. Client Nodes send images through DDS Domain, each message is processed by a single server (PN), that replies to the client informing that the image was successfully processed. 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.

The application running at the PNs uses one thread for each CPU core available on the machine. Therefore, the application is able to process images in parallel so as to maximize the image processing rate. The prototype application applies the equation 1 to calculate the new pixel´s color value.

$$grayscalePixel = redComponent * 0.3 + greenComponent * 0.59 + blueComponent * 0.11 \tag{1}$$

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.

5.1 Experimental Setup

The goal was to evaluate the DPSLB prototype´s performance in terms of throughput, CPU usage, Round-trip Delay (RTD) of the client-server interaction, and the overhead incurred by the monitoring and load distribution mechanism. To do so, the DPSLB prototype was used for a processing task demanding high CPU utilization, and was tested with data/image publication rate of 160 up to 1.365 data items per second. The *Assignment Function* of choice was the modulo operator applied on the *id* field and the number of available *Slice* was chosen to be ten since the number of *Slices* has minor impact in the DPSLB performance, as shown in this chapter.

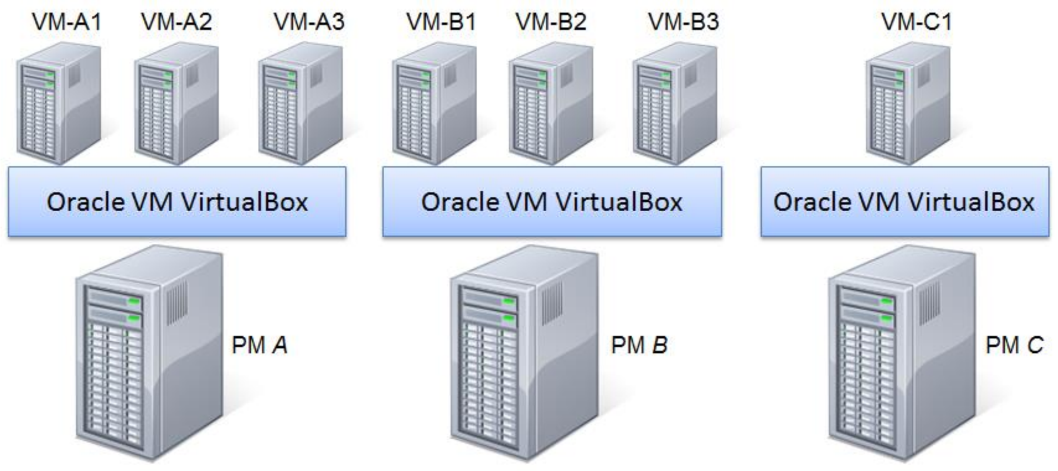


Figure 25 – Deployment of the Virtual Machines

The setup used for the experiment involves five PNs, one Load Balancer and a Client simulator deployed on three Physical Machines (PMs), as illustrated in Figure 25. Each PN runs on a dedicated Virtual Machine (VM), which executes

Ubuntu [56] 12.04 32-bit Operating System (OS). A VM labeled VM-A2 means that it is the second VM deployed on Physical Machine *A*. The chosen virtualization product was Oracle VM VirtualBox [57] since it is free, has cross-platform support.

The Virtual Machines VM-A1, VM-A2, VM-B1, VM-B2 and VM-B3 have a single PN running on them. Virtual Machine VM-A3 executes the Load Balancer and VM-C1 executes a Client Node. The resource capacities of each PM are shown in Table 1. CPU speed is measured in gigahertz (GHz), Memory size in MB (megabytes), Memory speed in megahertz (MHz) and bandwidth in Mbps (megabits per second). The Virtual Machines were configured to use one CPU core and 512 MB, except the Client Node on VM-C1 was configured to use three CPU cores.

Table 1 – Physical Machine specifications used in the experiment

Physical Machine	CPU	Memory	OS	Bandwidth
A	Intel i5 4 x 2.66 GHz	8 GB DDR3 1333 MHz	Windows 7 64-bit	100 Mbps
B	Intel i5 4 x 3.1 GHz	8 GB DDR3 1333 MHz	Fedora 15 64-bit	100 Mbps
C	Intel Dual-Core 4 x 2.66	8 GB DDR2 667 MHz	Mac OS X 10.7.5	100 Mbps

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.

5.2 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 receive at the specified instant of time from all PNs.

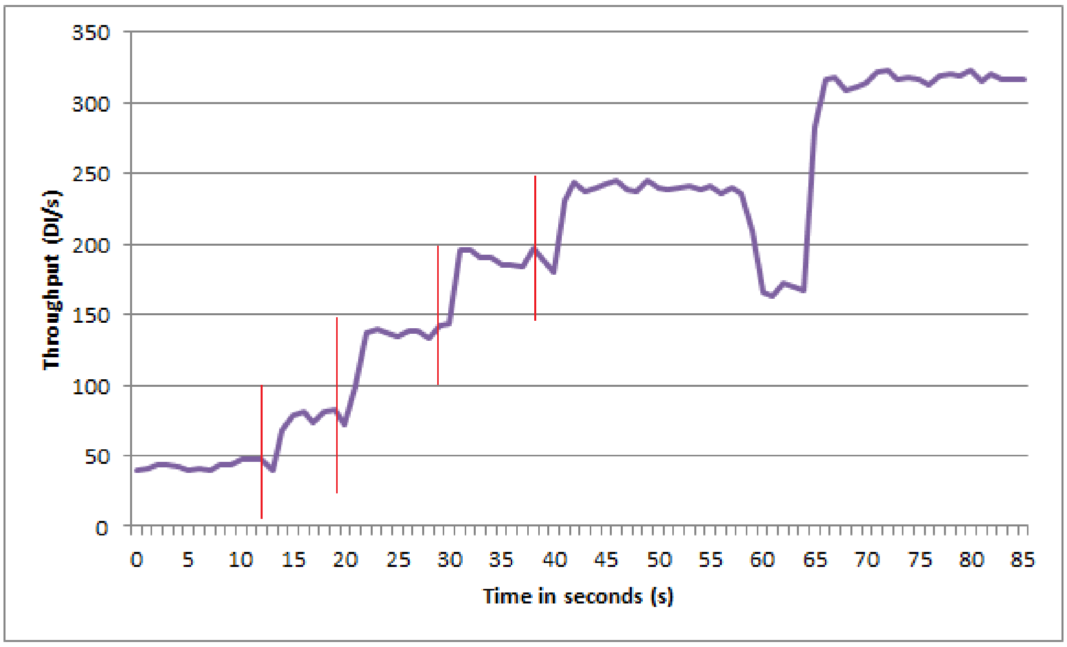


Figure 26 – Throughput over the time of the experiment (DI/s X seconds)

Figure 26 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. The throughput started from 40 DI/s and reached up to 323 DI/s at 72s. With one PN, the system was able process about 43 DI/s. After the second PN arrives, at 13s, the throughput grew up to 78 DI/s. The third PN, at 20s, collaborated to increase the throughput from 78 DI/s to 133 DI/s. The throughput promptly increased to 190 DI/s after the fourth PN joins the system, at 30s. 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 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.

5.3 CPU Usage

The CPU usage shows that, using the modulo operator as Assignment 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.

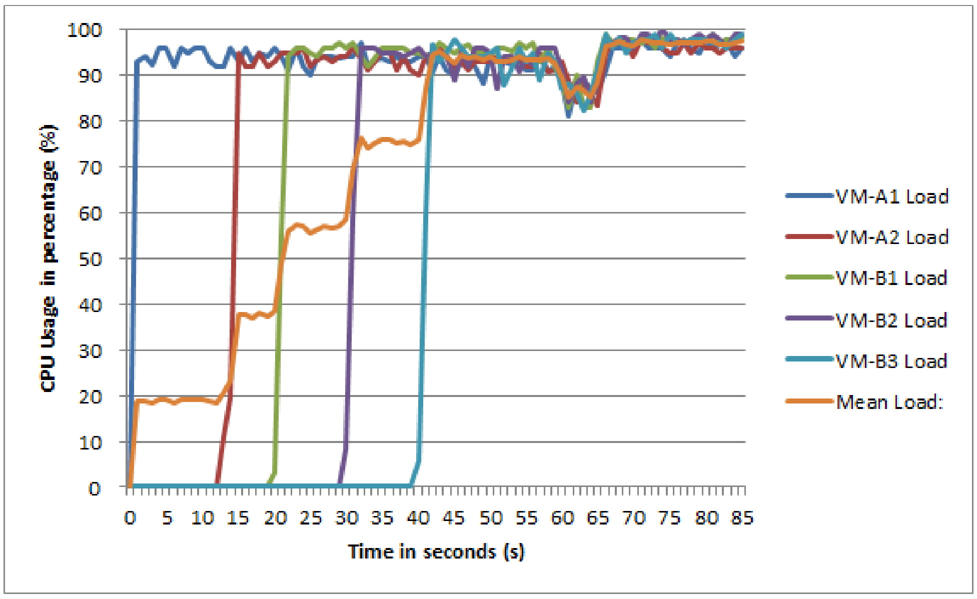


Figure 27 – CPU usage over the time of the experiment (% X seconds)

Analyzing Figure 27 is possible 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 suggests that it was caused by a bottleneck at the network and DDS communication layer.

5.4 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 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.

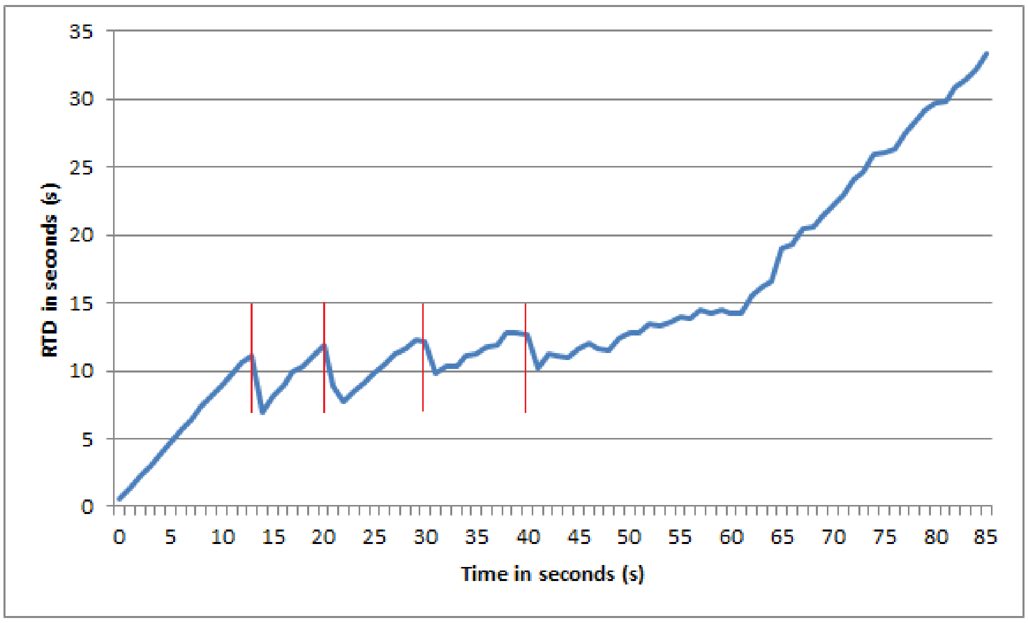


Figure 28 – Round-trip Delay over the time of the experiment (RTD in seconds X time in seconds)

The RTD during the experiment is shown in Figure 28, where the vertical red lines indicate when a new PN joined the system, as those in Figure 26. 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 28 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.

5.5 Overhead

In order to estimate the overhead incurred by both the DPSLB prototype and the Load Balancing Process, some experiments were realized.

5.5.1 DPSLB Overhead

To assess the Load Balancing Overhead, it was compared the throughput and the mean RTDs of the same image processing application using DDS in two configurations: using the DPSLB solution and without Load Balancing support. The overhead of the DPSLB solution is calculated as 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 DI/s.

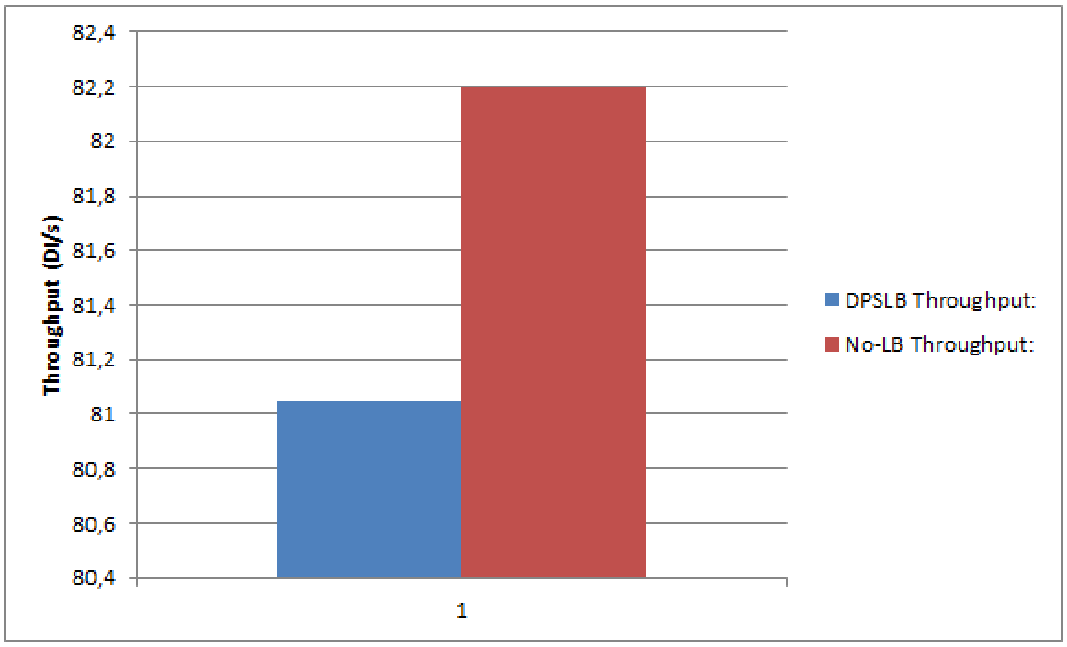


Figure 29 – Mean throughput (DI/s) comparison among DPSLB solution and another without Load Balancing

The application using the DPSLB prototype was able to process 81,044 DI/s and the application that used directly DDS but without any load balancing support, was able to process 82,194 DI/s, as shown in Figure 29. These numbers account for an overhead of 1,4% introduced by the DPSLB prototype.

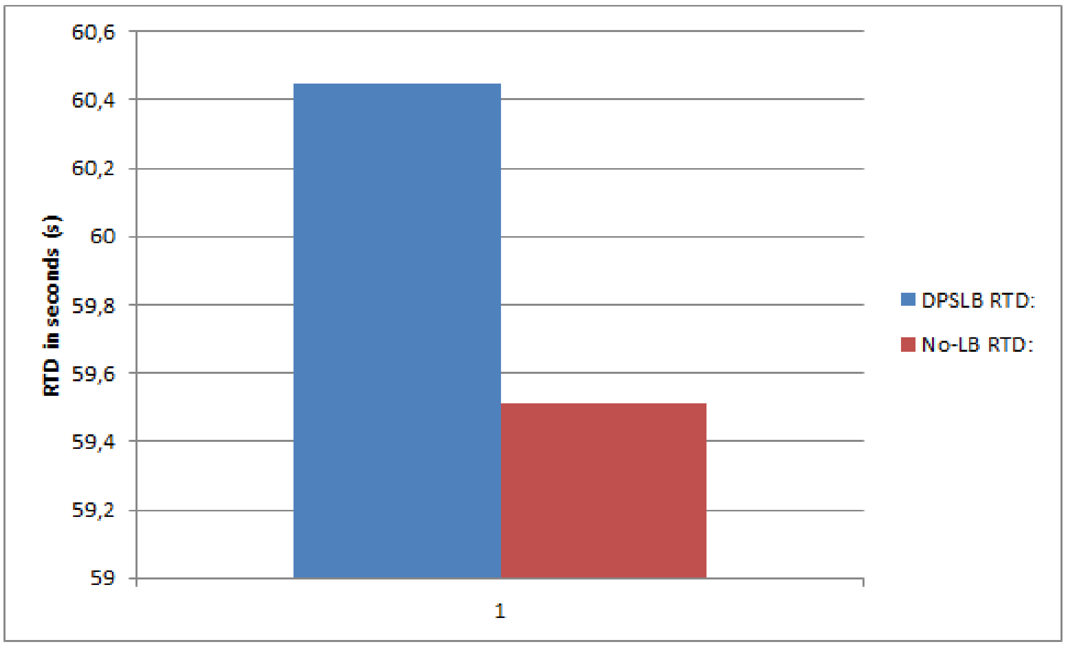


Figure 30 – Mean Round-trip Delay comparison among DPSLB solution and another without Load Balancing

Regarding RTD, shown in Figure 30, the application version using the DPSLB prototype had a mean RTD of 60,45 seconds. On the other hand, the application version not using the DPSLB prototype, had a mean RTD of 59,51. This difference represents an increase of 1,58% on the RTD.

It is believed 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.

5.5.2 Load Balancing Process Overhead

The Load Balancing Process Overhead tries to capture the impact of the Load Balancing Process on the system´s throughput and mean RTD.

Table 2 – 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

While in Load Balancing Process, the DPSLB prototype resulted in a mean CPU overhead of 1,4% on the *Slice-Giving* PN when analyzing the CPU usage in section 5.3. The RTD and throughput before, during and after the Load Balancing Process are shown in Table 2. 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.

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.

Table 3 – Load Balancing Process overhead for different the numbers of *Slice*s and data item production rates

Data Production Rate	**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

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 3 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*.