3
Data Processing Slice Load Balancing

This thesis proposes a load balancing mechanism 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 (and atomic) unit of data processing load to be distributed among server nodes in a DDS Domain. These server nodes will be called *Processing Nodes* throughout the remainder of this thesis. The general idea is that each *Processing Node* (PN) has some DPS assigned to it, and that each DPS covers an equal range within the space of all possible values of a chosen data identifier or attribute. Thus, in DPSLB, load balancing is equivalent to a redistribution of the sets of assigned DPS among the *Processing Nodes* according to their current load. The Processing Node’s current load is indicated by several metrics, such as its CPU and memory utilization rates.

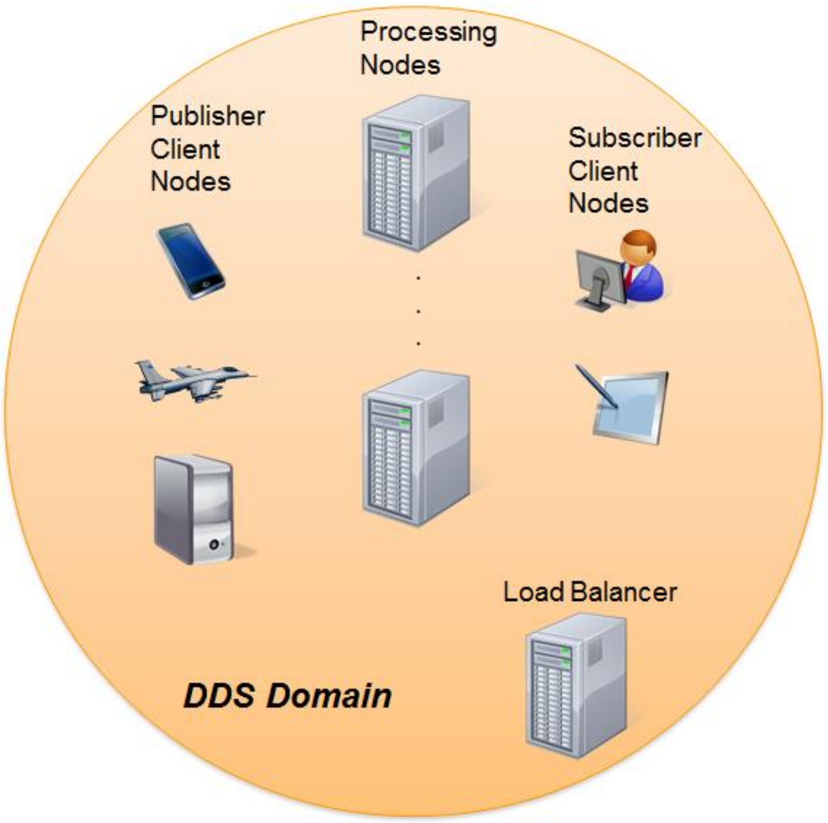


Figure 9 – DPS Load Balancing Architecture

Figure 9 shows the two main nodes that compose *the DPS Load Balancing* Solution within a DDS domain that has several Client Publisher and Subscriber Nodes: a set of homogeneous *Processing Nodes* and a *Load Balancer*. The Processing Node (PN), as suggested by its name, is in charge of receiving and processing data produced by Publisher Client Nodes. The specific processing task (on each data item produced by a Publisher) is entirely application dependent, and the DPSLB is totally agnostic to it. It may range from a simple translation/codification of data, to a complex analysis based on the content or attributes of the data items. The *Load Balancer* (LB) is responsible for monitoring the current load metrics of all Processing Nodes, defining the actions that accomplish the redistribution of the system's workload (i.e. reassignment of DPS) when an unbalance is detected and synchronizing the actions executed by PNs to move DPSs between them. It is expected that there is at least one client node publishing data on the DDS Domain and that this data needs to be processed by Processing Nodes.

The DPSLB solution was designed to Pub/Sub systems that supports content-based subscriptions and are *broker-less*, i.e. Pub/Sub systems that do not have brokers and employ a fully decentralized P2P architecture such as the DDS standard, explained in section 2.1. 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.

Data Processing Slice Load Balancing suits any application that demands high performance and scalable processing of large data streams, produced continuously by many sources (Publishers). Example applications are onboard car survivability assistance, fleet tracking and management, video processing, participatory (collaborative) sensing, Smart Cities, etc. A concrete example are the applications that have to re-encode videos or process images. Some fleet tracking and management systems, for instance, should generate alerts when vehicles make an unexpected stop or deviate from their planned routes. Such applications can process the data items in any order and each data item do not have any kind of dependency or relationship with the others.

3.1 Data Processing Slice

As mentioned, the proposed solution relies on the concept of *Data Processing Slice (DPS)*, or simply *Slice*. A *Slice* represents a percentage of the total data stream volume to be processed by all PNs. Every data item of the data stream (e.g. produced by a Publisher Client Node) must be assigned to a single DPS in order to be processed by some Processing Node. If a data item is not assigned to any *Slice* it will not be delivered to a PN for processing.

Each *Slice* is identified by a unique logical numeric ID (identifier), ranging from zero to the total number of defined *Slices*, minus one. Thus, the DDS Topic carrying application data has a specific numeric field, *Slice ID*, assigned to each data item. The total number of *Slices* is a constant defined in the PN software layer that must be equal to all PNs. Figure 10 shows an example of data item objects published by temperature sensors (with their sensor ID and the measured temperature). The field *Slice ID* is required for the DPS Solution, as aforesaid. The same temperature sensor may produce data items assigned to distinct *Slice IDs*, as illustrated by the data item objects published by Sensor ID 1 in Figure 10, and different sensors can publish data item objects into the same *Slice ID*.

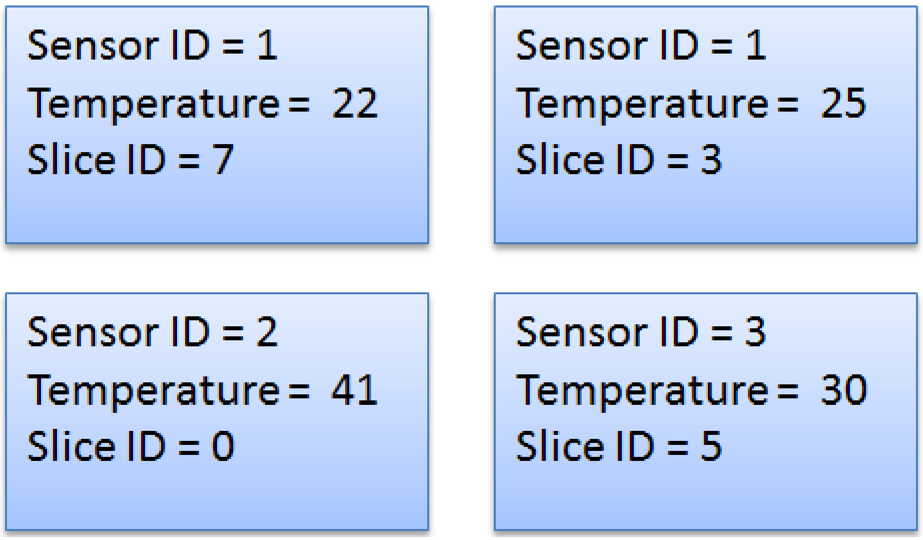


Figure 10 – Example of data item objects

Unlike *Virtual Servers* [36] [37] [38], Data Processing Slices do not behave as new Processing Nodes – 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 [50]. The number of DPS is also a upper bound for the maximum quantity of Processing Nodes, since each PN needs at least one DPS to get involved into the DPSLB.

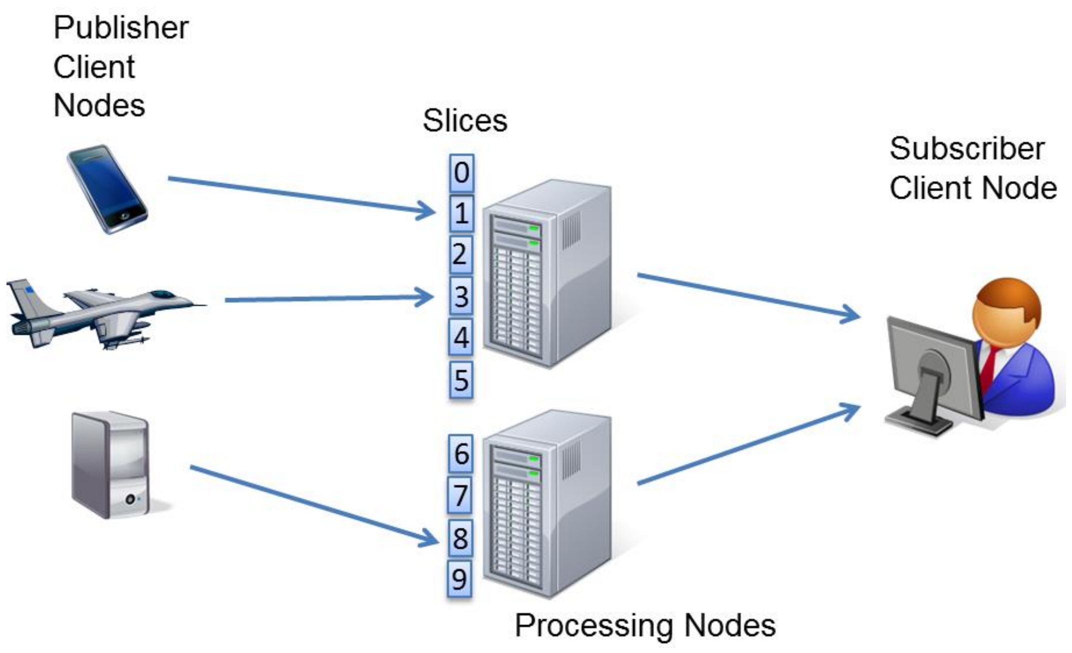


Figure 11 – A possible DPS flow and distribution

Figure 11 illustrates a possible DPS flow and distribution where three Publisher Client Nodes write data items that are processed by two Processing Nodes and where the results are consumed by a Subscriber Client Node. Data items produced from the smartphone (*Slice* 1) and aircraft (*Slice 3*) are processed by the first PN. Data items from the desktop (*Slice* 8) are processed by the second PN, which is in charge of processing data assigned to *Slices* 6 to 9. Considering that other Publisher Client Nodes would generate data items, each DPS would represent about 10% of the system workload. Hence, the first PN would process roughly 60% of all data items published while the second PN would process roughly 40% of it.

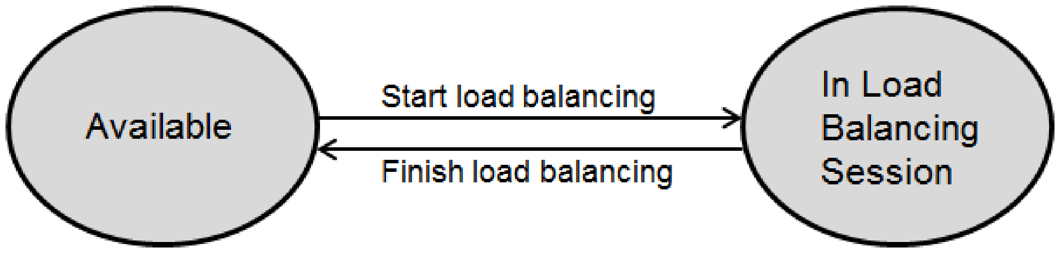


Figure 12 – DPS state transitions

As shown in Figure 12, each *Slice* has two possible states: (i) Available or (ii) In Load Balancing Session. The available state informs that a *Slice* is operational and data items assigned to it can be delivered to PNs for processing. While a *Slice* is *In Load Balancing Session* the DPSLB layer must cache data items assigned to this *Slice*, for later delivery to a PN. This caching is necessary for ensuring that all data items are delivered (and processed only once) by a single PN during the *Load Balancing Process*, as will be explained in detail in section 3.4.

3.2 Assignment Function

As previously mentioned, each produced data item must be assigned to a single DPS. This is done by the *Assignment Function*, which is responsible for determining a valid *Data Processing Slice* for each produced data items. The DPSLB solution requires the Assignment Function to be a very fast and low cost function, since it has to choose a DPS for each produced data item, and such data items will probably be produced at a very high rate. It is also desirable that *Assignment Function* produces a more or less uniform distribution of data items onto the set of Slices. That is, all possible output values (Slice IDs) should have the same probability of occurrence. But Assignment Functions that do not produce uniform distributions – i.e. where some *Slices* receive more data items than others – can also be used, since the workload can be balanced by re-arranging the number of Slices assigned to each Processing Node.

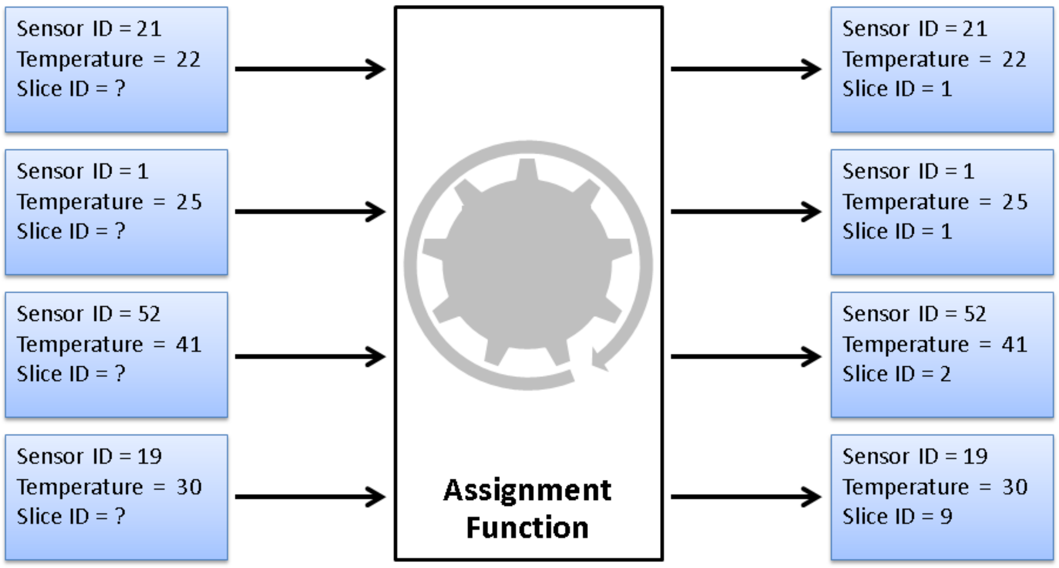


Figure 13 – An example of Assignment Function applied upon data item

The Assignment 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 the modulo operator (remainder of integer division). Figure 13 illustrates an Assignment Function that consists of a hash function which applies the modulo operator to the *Sensor ID* field, in a configuration with ten *Slices*. Hence, each data item computed by the Assignment Function is assigned to one DPS. As shown, Sensor IDs 21, 1, 52 and 19 are mapped to Slice IDs 1, 1, 2 and 9, respectively. The Assignment Function must be called before the data item is published in the DDS Domain.

As the Assignment Function has to choose a DPS for each data item, it is possible to implement an Assignment Function specific to an application. In a hypothetical application that the data item has fields that hold the latitude and longitude, the Assignment Function can choose a *Slice* based on the data item geographical position. For instance, if latitude is greater than zero the Assignment Function chooses the *Slice ID* 0, otherwise it chooses the *Slice ID* 1.

3.3 DPS Solution

At the Processing Node there are two distinct software layers, the DPSLB software layer and the application layer that uses DPSLB as a base software layer. In order to receive data items, Processing Nodes create a *DDS Content Filtered Topic* for each *Topic* of the application and associate it to a DDS *Data Reader*. If the application wants to subscribe to *Topic A*, for instance, the DPSLB software layer at Processing Node creates a *DDS* Content Filtered Topic of type *Topic A* and next associates it to a *DDS Data Reader*. In the scenario of Figure 11, the first PN would have a filter expression such "*sliceID >= 0 and sliceID <= 5*" and the second PN a filter expression "*sliceID >= 6 and sliceID <= 9*". When a data item is received by the PN, it verifies whether the data item came from an DPS in state *Available*. If so, PN forwards the item to the application, and otherwise it stores the item on its local cache, as will be explained in section 3.4.

The Load Balancer plays the role of coordinator of the actions executed in the Load Balancing Process, but the actions are effectively executed by the overloaded and the underloaded PNs. The LB has a module that contains the *Load Balancing Algorithm*. This algorithm analyzes the load of the PNs and decides if the system is unbalanced. In this case, the algorithm 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 such as [38]. This module is notified about new Processing

Nodes that are able to join the DPSLB solution and called when the Load Balancer needs to analyze the system workload. The algorithm [38] classifies nodes into overloaded, normal and underloaded. The main idea is transfer load from overloaded nodes to underloaded ones. After being called, the algorithm has to classify the Processing Nodes 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 Processing Nodes.

Figure 14 illustrates the interations 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 14).

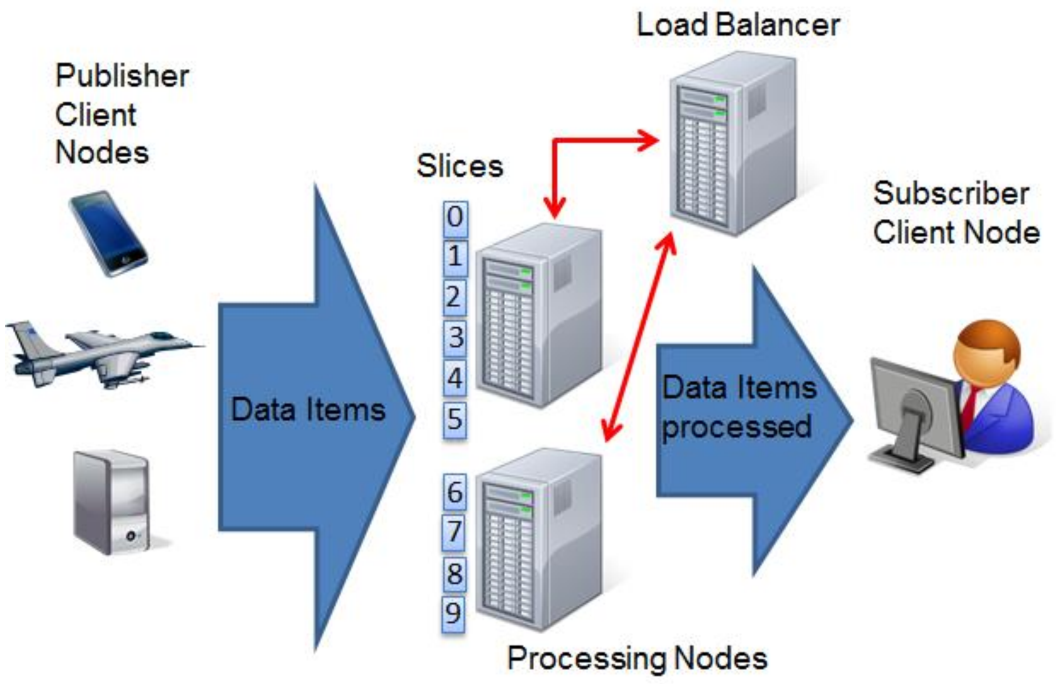
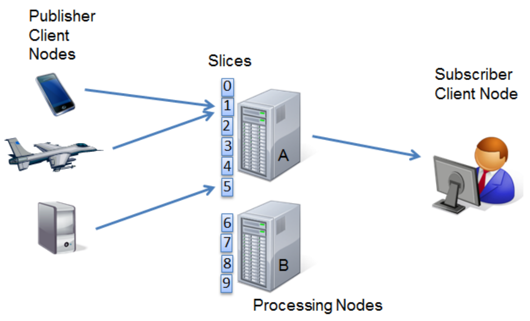


Figure 14 – Interactions between clients, Processing Nodes and Load Balancer

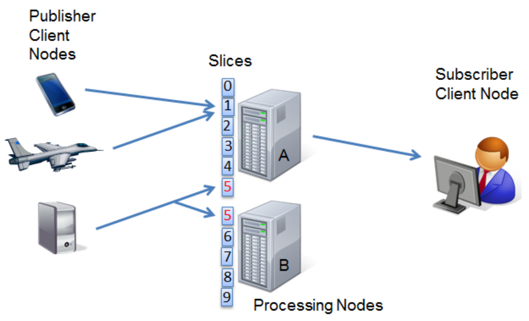
With the aim of uniquely identifying each data item and generating the Merged Cache, explained in section 3.4, it is mandatory that all data items hold a numeric ID field.

3.4 Load Balancing Process

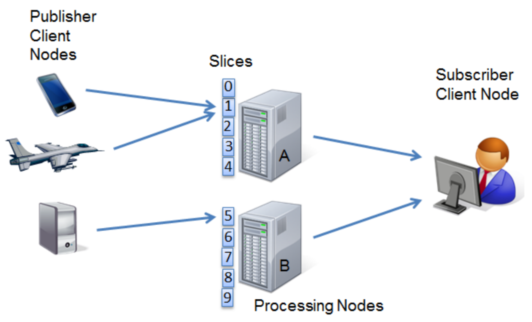
Load Balancing Process is the process of moving *Slices* from a PN to another. The process is started when the *Load Balancer* detects a load unbalance in the system and decides that some DPS should be moved to a different Processing Node for load balancing. During this process both *Processing Nodes* involved, i.e. the *Slice-giving* and *Slice-taking*, must work in a coordinated manner to guarantee that all data items of the moved *Slice* are received and processed exactly once by any PN.



(a) Before Load Balancing Process



(b) During Load Balancing Process



(c) After Load Balancing Process

Figure 15 – Data flow before, during and after the Load Balancing Process

Figure 15 illustrates the redirection of the data flow when *Slice* 5 is moved from Processing Node *A* to *B*. During the *Load Balancing Process* (Figure 15 b) both Processing Nodes receive data items assigned to DPS 5, although initially none of them will process data items from this DPS. Instead they store these received data items 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. Thus, the specific actions sent by the Load Balancer to move a DPS between two PNs are:

- **Update DPS´s state:** *A* updates *Slice*´s state to *In Load Balancing Session*;

- **Add DPS:** *B* adds the *Slice* with *In Load Balancing Session* state;
- **Remove DPS:** *A* removes the *Slice*;
- **Update DPS´s state:** *B* updates *Slice*´s state to *Available*;
- **Send cache:** *A* sends its data item cache to *B* that merges its own cache and *A*´s cache to generate de *Merged Cache*. After this, *B* notifies the application about the data items.

After these actions the system completes the *Load Balancing Process* and changes its data flow as shown in Figure 15 c. The *Add* and *Remove* actions determine if the corresponding data items of the Slice are delivered or not, respectively, to a node on the DDS Domain. This is possible by a dynamic adjustment of the Content Filtered Topic´s filter expression in DDS.

After receiving data items from PN *A's* cache, the node *B* has to identify the items that appear in both caches and then generate a *Merged Cache*, which contains all data items of either caches, but without occurrence of duplicated items. 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. 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. In the scenario illustrated by Figure 15, whether PN *B* is detected as an idle PN, all its *Slices* (only the *Slice 5* is assigned to PN *B*) should be moved to PN *A*. Thus, a new Load Balancing Process is started to move the *Slice 5* from PN *B* to *A*.

3.5 Discussion

While analyzing the Load Balancer classification, as explained in section 2.3.1, it is classified in centralized location, event-driven initiation and global approach. Despite being a centralized approach, data items need not pass through LB on their way from client Publisher nodes to Processing Nodes. Instead, LB only monitors the load of Processing Nodes and eventually becomes active to coordinate the load redistribution process. Therefore, LB is not a bottleneck.

Due to DDS limitations with *Content Filtered Topics*, the DPSLB solution does not support changing the expression applied on the DDS Content Filtered Topic. To do so, the DPSLB needs to create a new DDS Data Reader with the new Content Filtered Topic, destroy the old Data Reader and finally verify which data items were received by both Data Readers, similarly in the Load Balancing Process when the *Slice-taking* should generate its Merged Cache.