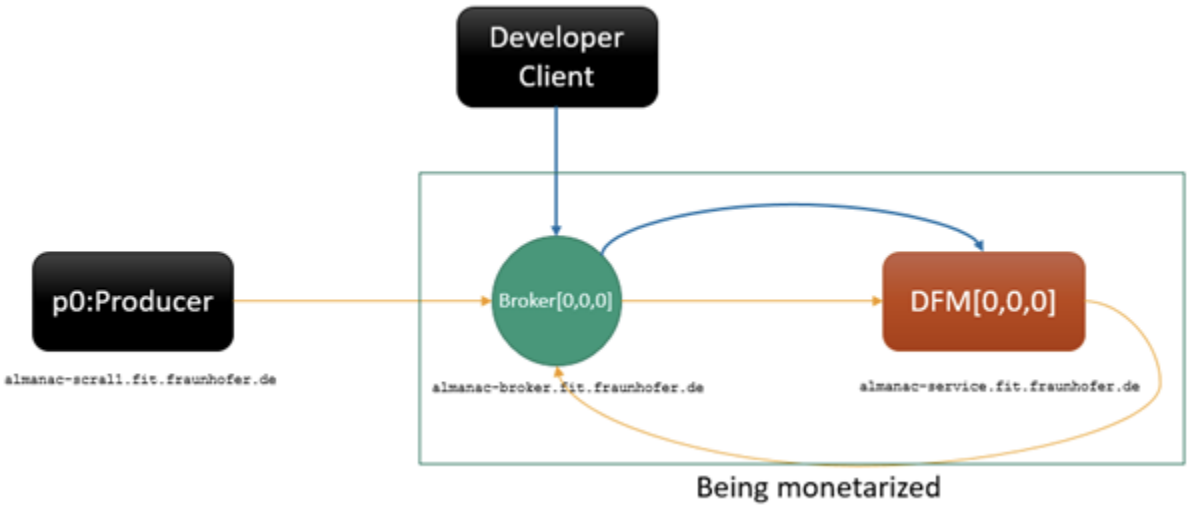
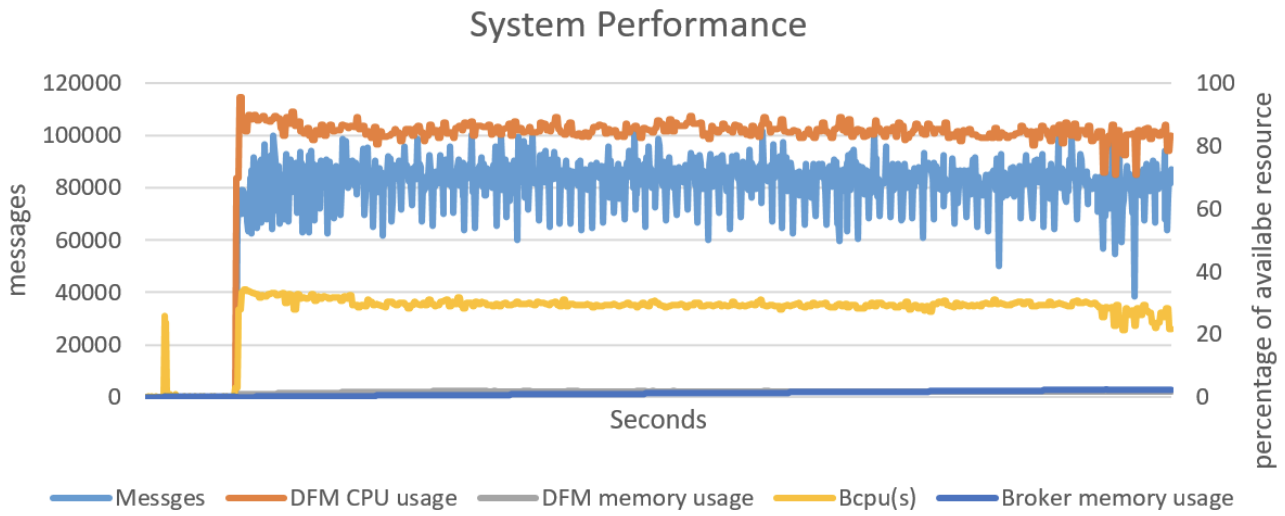
IoT Data-Processing Agent Results

Test: Load Single Query

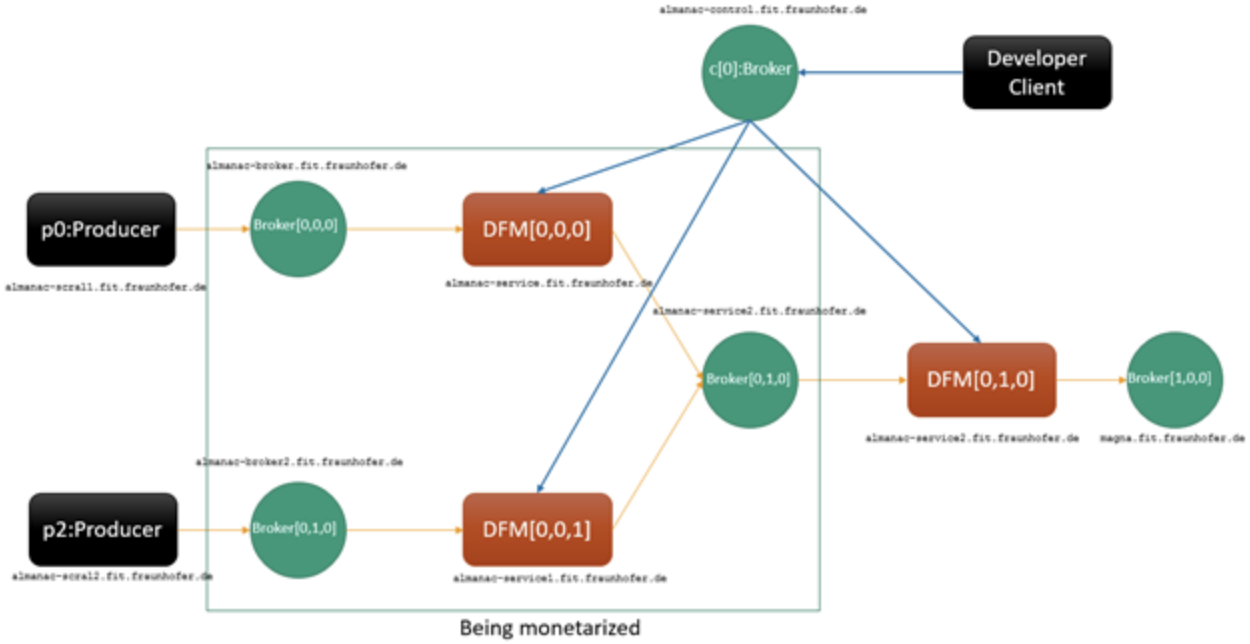


Scenario	**Burst Test 1**: The test is design to test and discover the behaviour of the Data-Fusion Manger(DFM) in case of a burst of incoming events in a reasonable period of time. **Deployment Description**: 3 Virtual machines (VM): **Producer VM**: CPU: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz cache 61440 KB RAM: 2.0 GB **Broker VM**: CPU: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz cache 61440 KB RAM: 4.0 GB **DFM VM**: CPU: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz cache 61440 KB RAM: 2.0 GB The data producer and consumer have the same hardware characteristic to test the performance in symmetric conditions. In this manner, the performance of capturing, handling, processing, and forwarding messages vs just generating and sending events is tested in equal hardware conditions.
Test implementation	In the Producer VM resides 10 clients producing as much load they can produce. The Broker VM is resides the MQTT Mosquitto Broker. Finally, in the DFM VM the DFM service is deployed. The test started when one DFM statement is loaded in the DFM service. The statement is a simple one (i. e. just arithmetical operations using the events payload). Afterwards, the producer clients generates the load for 15 minutes.

Results	Total load generated: 104871868 messages (msg) Total load processed: 79425813 messages Total average processed: : 82821.49426 msg/s Processed rate: 75.74 % AVG, MAX CPU* usage DFM VM: 84.80%, 95.35% AVG, MAX CPU* usage DFM VM: 169.60%, 190.7% AVG, MAX RAM usage Broker VM: 52.16MB, 95 MB AVG, MAX RAM usage DFM VM: 35.96MB, 37.9MB *The CPU is measured regards the theoretical maximum can be reached by the process. i.e. the Brokers can just take a single core, while the DFMs can take all of them. Therefore, the brokers % is measure using 1 core as base, while DFM used 2 cores as base.
Interpretation	The test was performed until one of the three sides reached its limit. In this test the producer used an average of 97.7% (both cores) and was unable to generate more load. This means in equal conditions, the DFM could outperform the producers. The processed rate shows how many messages were processed at the end of the experiment. The remaining events were at the queue (either at the broker or DFM). This means that the system was stacking events faster than they could be processed. This happened with a rate of 2.4 MB/min. With this rate, the DFM would overflow its memory in about 13.64 hours of sustained burst. The mosquito broker increased the memory in a faster rate 3.48 MB/min, but the broker VM had the double amount of memory (4 instead of 2 GB). This means first, the broker VM needed 18.71 hours before its collapse. Secondly, the broker queue was increasing faster than the DFM. This indicates that in such deployments, the broker will demand more memory. The broker shows better performance than the DFM at the first glance. However, Mosquitto is a mono-thread service, this means, that the service can just leverage form one CPU. Therefore, the performance measured in all CPU is unimportant. Even more, the increase of cores will not improve performance. In contrast, the DFM is a multi-thread service. Hence, the DFM will probably scale better as the Mosquitto, due to it is easier to increase cores as the clock speed-rate on a CPU. This had been already observed in other experiments, where the load increased over the load used in this experiment.
Scalability perspective	The test shows that the effects of a heavy load in the system affect different on other components. This point out that the first bottleneck in the processing chain is the broker. Furtherer tests confirm that. The broker is the first component to be overloaded. This can be addressed by deploying more brokers which balance load between them. It is important that each broker is deployed in a single core machine so there is no waste of CPUs. On the other hand, the DFM cannot hold more load over the one presented in this experiment. Nevertheless, the DFM did not crash when the CPU reached its limits in contrast to Mosquitto. But it will when the DFM reached its physical memory limit. Increasing the number of cores will further increase the performance of the DFM. Additionally, deploying multiple DFM on different machines will also increase the performance of the system, similar to the deployment of multiple brokers.



Test: Load Single Query Advance deployment



Scenario	**Burst Test 1**: The test is design to show the horizontal data processing capability by increasing the amount of DFM in the deployment **Deployment Description**: 10 hosts, 9 VM and one a physical machine. There are Three kind of machines: **VM Type A**: CPU: 2 cores: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz cache 61440 KB RAM: 2.0 GB **VM Type B**: CPU: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz cache 61440 KB RAM: 4.0 GB **Host Type M**: CPU: 4 cores: Intel(R) Xeon(R) CPU E5-2603 0 @ 1.80GHz RAM: 4.0 GB There are hosts used as broker: Broker[0,0,0]:TypeB, Broker[0,0,1]:TypeB, Broker[0,1,0]:TypeA, Broker[1,0,0]:TypeM, BrokerControl:TypeA. 2 producers: p1:TypeA, p2:TypeA. 3 Data-Fusion Managers: DFM[0,0]:TypeA, DFM[0,1]:TypeA, DFM[1,0]:TypeA. The deployment is shown in figure 1.
Testimplementation	In this test we have two parallel data channels. The first, started by the producer one (p1) VM and the second started by producer (p2) VM. Both channels are merged by DFM[0,1,0] VM, for thereafter follow a single data channel. In each Producer VMs resides 10 clients producing up to 100 Kmsg/s (msg = messages). The p1 (producer one) publish the load on Broker[0,0,0], while p2 publish on Broker[0,0,1]. Each broker machine has a MQTT Mosquitto Broker. The broker machines forward the messages to the corresponding DFM machines, Broker[0,0,0] to DFM[0,0,0] and Broker [0,0,1] to DFM[0,0,1]. Both DFM[0,0,*i*] aggregates the data and publish their results on Broker[0,1,0]. The DFM[0,1,0] aggregates the results of both (DFM[0,0,1], DFM[0,0,0]) and publish its results on Broker[1,0,0]. On the other hand, BrokerControl is there for deploying the DFM statements and monitoring all DFMs simultaneously. This means, that the statement is publish once and multi-deployed in parallel in all DFMs. Finally, for the scope of this test, we will concentrate in the first aggregation layer hosts (Broker[0,0,0],Broker[0,0,1],DFM[0,0,0], DFM[0,0,1]), while the second is just to add the data in real-time without interfere with the test.

Results	Total Load generated: 143972396 msg (p1+p2), 62520137 msg (p1), 81452259 msg (p2). Average Load generated: 69551.88213 msg/s (p1+p2), 60405.92947 msg/s (p1), 78697.83478 msg/s (p2). Total load processed: 129091627 msg (Channel 1&2), 55986276 msg (Channel 1), 73863449 msg (Channel 2). Average load processed: 124726.2097 msg/s (Channel 1&2), 54093.02029 msg/s (Channel 1), 71365.65121 msg/s (Channel 2). Processed rate: 90.19%, 89.55% (channel 1), 90.68% (channel 2) AVG, MAX CPU usage* Broker[0,0,0] VM: 58.65%, 100% AVG, MAX CPU usage* Broker[0,0,1] VM: 48.33%, 58.3% AVG, MAX CPU usage* Broker[0,1,0] VM: 1.42%, 1.6% AVG, MAX CPU usage* Broker[1,0,0] VM: 0.01%, 0.3% AVG, MAX CPU usage* DFM[0,0,0] VM: 77.27%, 98.85% AVG, MAX CPU usage* DFM[0,0,1] VM: 57.22%, 98.7% AVG, MAX CPU usage* DFM[0,1,0] VM: 0.57%, 59.35% AVG, MAX CPU usage Channel 1: 67.96%, 99.43% AVG, MAX CPU usage Channel 2: 52.78%, 78.5% AVG, MAX Memory usage Broker[0,0,0] VM: 77.00%, 95.2% AVG, MAX Memory usage Broker[0,0,1] VM: 1.42%, 1.6% AVG, MAX Memory usage Broker[0,1,0] VM: 0.2%, 0.2% AVG, MAX Memory usage Broker[1,0,0] host: <0.01%, <0.01% AVG, MAX Memory usage DFM[0,0,0] VM: 28.73%, 30.2% AVG, MAX Memory usage DFM[0,0,1] VM: 28.89%, 30.4% AVG, MAX Memory usage DFM[0,1,0] VM: 5.75%, 5.9% AVG, MAX Memory usage Channel 1: 52.5%, 62.7% AVG, MAX Memory usage Channel 2: 15.16%, 16% *The CPU is measured regards the theoretical maximum can be reached by the process. i.e. the Brokers can just take a single core, while the DFMs can take all of them. Therefore, the brokers % is measure using 1 core as base, while DFM used 2 cores as base.
Interpretation	Before analysing the results, there is a need of take some times describe an non-symmetrical results obtained between both data channels. While both parallel data channels are symmetric in virtual hardware, they produce different results. While p2 had problems to go over the 80K msg/s, p2 was constantly close to the 100K msg/s threshold. This produced uneven load in the broker. The data channel 1 (started by p1), loaded till the limit the CPU limit of the Broker[0,0,0]. This produced that the Broker[0,0,0] needed to queue much more msg till was close to depleted all memory. On the channel 2, the data was more balanced, the load did not overload the Broker[0,0,1] allowing better results. This effect produced a difference of 18M msg in total between both channels. This difference it is provably produced due to the whole deployment is made in virtualized hardware running in the same cluster. Theoretically, each channel can be seen as independent simple load test. Nevertheless, the performance of both channels were not equal and both less than the first test. This probably was also problem made by the virtualized hardware running in the same machine. In case of channel 2 (Broker[0,0,1] and DFM[0,0,1]) the resources where used in a moderated manner achieving a higher processing rate than channel 1 with less usage of CPU and memory resources (52.78 CPU% and 15.16 MEM%). Additionally, the components in channel 2 do not show signs of being overloaded. On the other hand channel 1, reached the maximum capacity by reaching the maximum usage of the CPU of Broker [0,0,0] almost constantly. This triggered the overused of memory, for finally the reduction of the performance. This shows again that the Mosquitto broker is the weakest component regarding performance. However, even than the capacity of the broker of channel 1 reached its maximum capacity after 6 min the test started, the system was able to provide stable results for another 11 more minutes. This shows the robustness of the system in a heavy load situation.

Scalability perspective	Therefore, the merging of both channels provide a geometrical increment of aggregation computational capacity to the overall system. In other word, the system scale in a horizontal manner. In the first test the system was able to process 82K msg/s with maximum burst of 102 K msg/s, close to channel 2 with 71K msg/s and with maximum burst of 99K msg/s in channel 1. Furthermore, the combine deployment got an average 124K msg/s with maximum burst of 194K msg/s. This is an increment of 152% on average and 190% for the maximum burst. Moreover, there was an improvement of 15% in the processing rate.