**PHASE 3 REPORT EXPERIMENTAL EVALUATION**

**Group 20**

# **SPATIAL AGGREGATION QUERY**

**Function Name:** SpatialAggregationQuery

**Arguments:**

1. String InputLocation1: location of the input1 in HDFS
2. String InputLocation2: location of the input2 in HDFS
3. String OutputLocation: location of the output in HDFS
4. String input1Type: indicates input1 is point or rectangle.

**Requirement:** Load two sets of polygons, output the join query result of these sets.

**Input1 Dataset Schema:**

1. A-id, x1, y1, x2, y2: Every row is a pair of points which defines a rectangle. This set has bunch of rectangles. Both contained and overlapped are considered.

(or)

1. A-id, x1, y1: Every row defines a point. This set has bunch of points. Points on and inside the rectangle are considered.

**Input2 Dataset Schema**:

B-id, x1, y1, x2, y2: Every row is a pair of points which defines a rectangle. This set has bunch of rectangles.

**Output Dataset Schema:**

B-id, count: Every row is a pair which represents the Rectangle ID and count represents the number of points lie in that rectangle.

**Algorithm/ Pseudocode:**

1. Creates a spark configuration
2. Creates java spark context using spark configuration
3. Invoked SpatialJoinQuery which is explained in detail above
4. The output files generated from step 3 are read using *context.textFile().*
5. Each line in each file is converted to a Iterable JavaRDD<String> using the flatMap().
6. Step 5 JavaRDD Iterable is converted to JavaPairRDD using *maptToPair()* by split each line into key value pair where key is the rectangle id and the value is count of array obtained by splitting line by comma and subtracting 1 which gives total number of points in the rectangle.
7. The obtained key value pairs in JavaPairRDD are reduced by key using *reduceByKey()* which counts the total count for the same rectangle id.
8. The results are then sorted by key using *sortbyKey()*.
9. The fineal JavaPairRDD is then converted back to JavaRDD and key value are combined to string using a comma as separator.
10. Result is combined using coalesce function and output is saved to Hadoop.

**How to run:**

./spark-submit \

--class edu.asu.cse512.aggregation \

--master <Spark Master IP> \

--jars convexHull-0.1.jar,joinQuery-0.1.jar \

spatialAggregation-0.1.jar \

hdfs://<HDFS Master IP>/inputFile1 \

hdfs://<HDFS Master IP>/inputFile2 \

hdfs://<HDFS Master IP>/outputFile rectangle

# **EXPERIMENTAL SETUP**

Our experiments were run on a single laptop with an Intel Core I7 processor and 16GB of RAM as a host machine. The master and slave computer were all run on this host machine, in Ubuntu Linux virtual machines. Network communications was between these virtual machines but completely within the host machine.

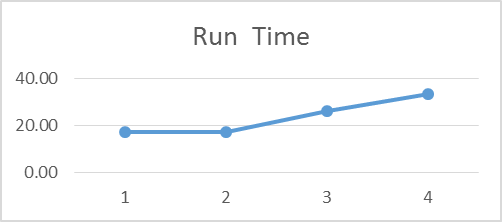
# **EXPERIMENTAL EVALUATION**

The following sections detail the experimental evaluation of the original six functions and the added Geospatial Aggregation function.

The displays of the Spark Status screens and the Ganglia screens are moved to Section 4 - Addendum – Run Time Figures as they required full width to display properly.

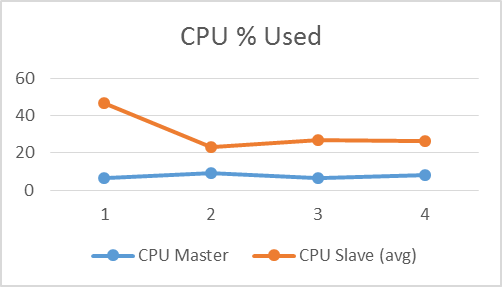
### ***Geometry union***

**Run Time:** As seen from Figure 1, the run-time of the functions increases as we increase the number of nodes in the cluster. This is an expected behavior for the cluster configuration, multiple low-resourced virtual machines on a single host. If the cluster comprised of multiple high-resourced physical machines, we would have seen a drop in run time, until the network overhead grows above the increase in computing power.



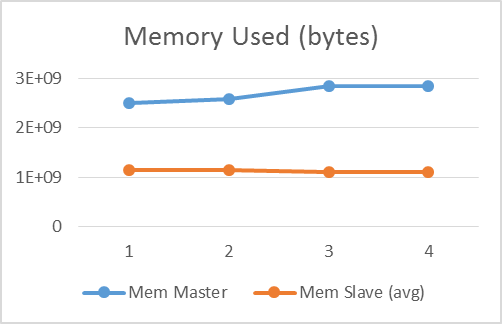
**Figure 1. Geometry union run time**

**CPU Used:** As seen from Figure 2, the CPU utilization of the master node remains constant when we increase the number of nodes in the cluster. However, the sharp drop in percentage of CPU cycles used in a slave (worker) node directly reflects that the total work was chunked into smaller work assignments for the slave nodes.



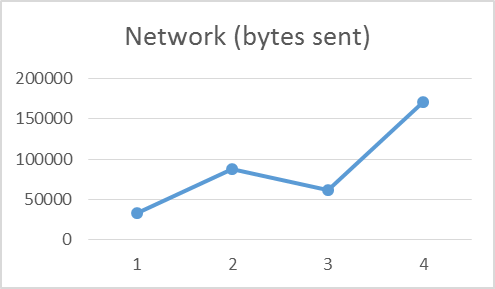
**Figure 2. Geometry union CPU used (% Cycles spent in user mode v/s # of nodes)**

**Memory Used:** As seen from Figure 3, it is quite evident that the memory utilized by the master increases (slightly) as it has to perform more overhead processes to keep track of distributed work. Another interesting point to note is that the master is utilizing its allocated memory completely. Thus suggesting that the master node has to be high-resourced when compared to the slave nodes. For the slave (worker) nodes, there is hardly any change in memory utilized, possibly due to the limited test data set.



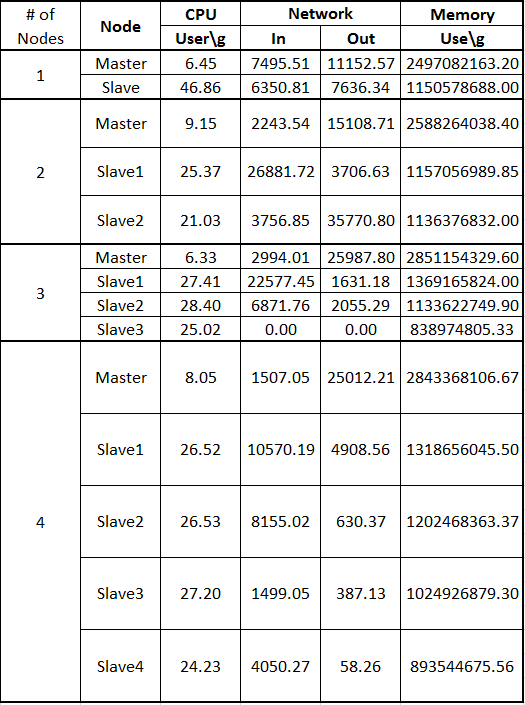
**Figure 3. Geometry union memory used**

**Network Traffic:** As the number of nodes in a cluster increases, we see an increase in the network traffic. This is an expected behavior, since the master has to send data to more number of slave (worker) nodes, and the slave nodes have to send the status and/or output back to the master node. If you notice the graph, there is an anomaly when number of nodes is 3. This is not because the network traffic was less for this configuration, but because the data monitored had a drop in data point. Figure 11 can explain this anomaly better.



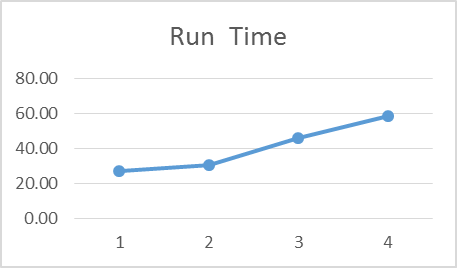
**Figure 4. Geometry union network traffic**

The data that is shown in Figure 5 is analyzed base on the readings from ganglia and by further analyzing this data above graphs are generated.



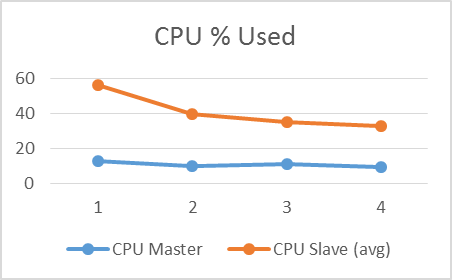
**Figure 5. Geometry union raw data**

### ***Geometry convex hull***



**Figure 6. Convex hull run time**

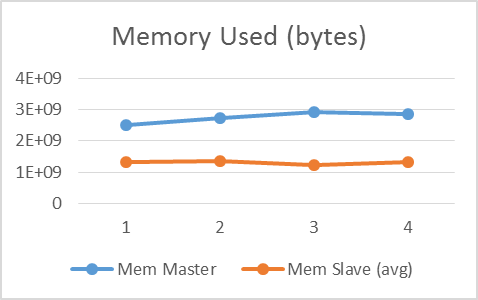
As expected, the run-time of the function increases as the number of nodes in the cluster increases as seen in Figure 6. This is due to the nature of the experimental system which consists of multiple virtual machines running on a single host system. If the experiment were run in a cluster of multiple physical machines, then a decrease of elapsed time would be expected as the cluster size increases until the communications overhead overwhelms the advantage of increasing processing power.



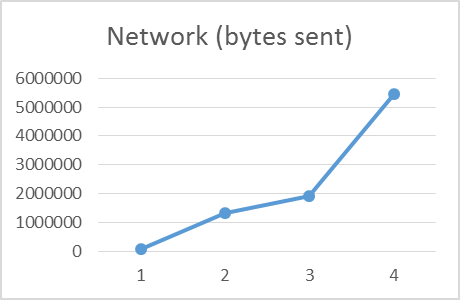
**Figure 7. Convex hull CPU Used**

As shown in Figure 7, the master node uses about the same amount of CPU as the number of nodes increases. There is a very small drop-off. This is likely due to the increased run-time. The master node will perform the same tasks over a longer period, lowering the average CPU load.

Figure 8 shows the memory used on the master and the average of the slave nodes. The master shows a small increase as the number of slaves increases. This is likely due to the need to keep track of the work from these slave nodes. The slaves show almost no change in memory usage as the number of nodes increases. One possible explanation is the limited size of the experimental data set.



**Figure 8. Convex hull memory used**

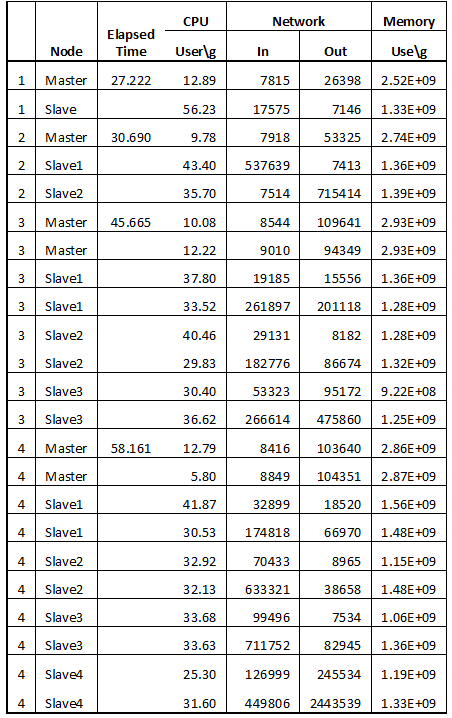


**Figure 9. Convex Hull Network Bytes Sent**

The increase in network traffic shown in Figure 9 as more nodes are added to the cluster is expected, as more coordination between the master and slave nodes to perform the algorithm is required as the cluster gets more complicated.

Figure 10 shows the raw results of running the Geometry Convex Hull algorithm on the class provided point data set. One apparent anomaly in the raw data is the last line, showing more than three times the network traffic of any other item, including five times as much output as input. This somewhat skews the remaining results.

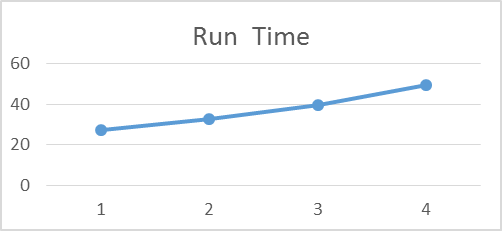
This Figure 10 also illustrates a possible anomaly seen in Figure 9. The network bytes sent shows a consistent increase as the number of nodes increases. The increase with four slave nodes is dramatically more than the rest. Looking at the raw results in Figure 10, the last node in each cluster shows the highest network output values of any node.



**Figure 10. Convex hull raw data**

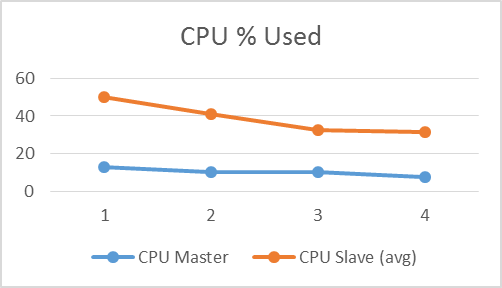
### ***Geometry farthest pair***

As farthest pair internally uses convex hull to compute farthest pairs the implementation analysis to convex hull should able applicable here and same reasoning hold for each of them.



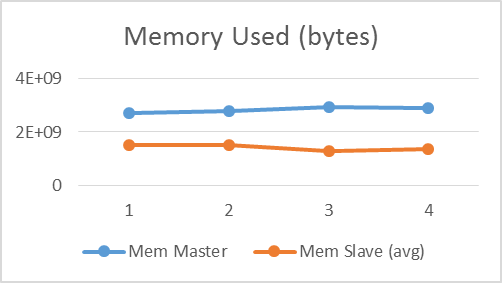
**Figure 11. Farthest pair run time**

As expected, the run-time of the function increases as the number of nodes in the cluster increases as seen in Figure 1. This is due to the nature of the experimental system which consists of multiple virtual machines running on a single host system. If the experiment were run in a cluster of multiple physical machines, then a decrease of elapsed time would be expected as the cluster size increases until the communications overhead overwhelms the advantage of increasing processing power.



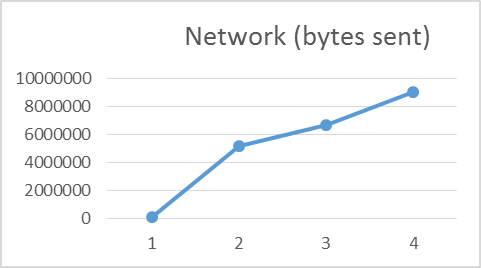
**Figure 1. Farthest pair CPU used**

As shown in Figure 12, the master node uses about the same amount of CPU as the number of nodes increases. There is a very small drop-off. This is likely due to the increased run-time. The master node will perform the same tasks over a longer period, lowering the average CPU load.



**Figure 13. Farthest pair memory used**

Figure 13 shows the memory used on the master and the average of the slave nodes. The master shows a small increase as the number of slaves increases. This is likely due to the need to keep track of the work from these slave nodes. The slaves show almost no change in memory usage as the number of nodes increases. One possible explanation is the limited size of the experimental data set.

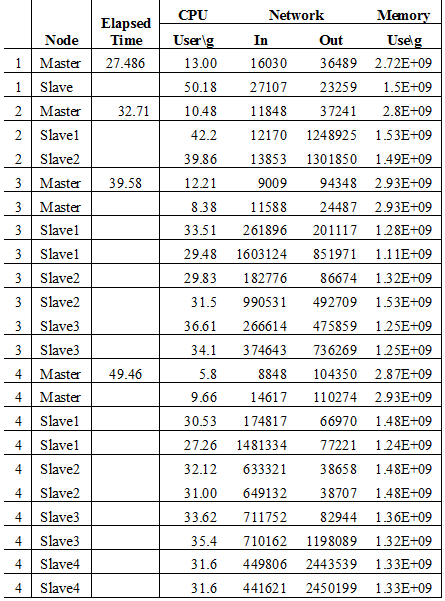


**Figure 14. Farthest pair network bytes sent**

The increase in network traffic shown in Figure 14 as more nodes are added to the cluster is expected, as more coordination between the master and slave nodes to perform the algorithm is required as the cluster gets more complicated.

Figure 15 shows the raw results of running the Geometry Convex Hull algorithm on the class provided point data set. One apparent anomaly in the raw data is the last line, showing more than three times the network traffic of any other item, including five times as much output as input. This somewhat skews the remaining results.

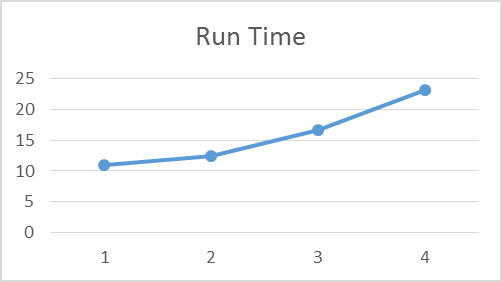
This table also illustrates a possible anomaly seen in Figure 14. The network bytes sent shows a consistent increase as the number of nodes increases. The increase with four slave nodes is dramatically more than the rest. Looking at the raw results in Figure 15, the last node in each cluster shows the highest network output values of any node. This may just be the



**Figure 15. Farthest pair raw data**

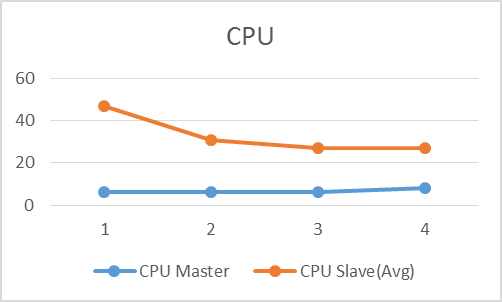
### ***Geometry closest pair***

**Run Time:** As expected, the run-time of the operation increased when the number of nodes increased. The reason that we expected this behavior was that our configuration had virtual machines running on a single host machine. Secondly, the machines are low-resourced. We expect the run-time to decrease if the system configuration was with high-resourced physical machines, until the network overheads exceed the gained computing power.



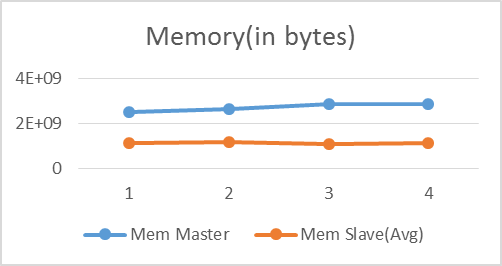
**Figure 16. Geometry closest pair run time**

**CPU Used:** As seen from figure below, the CPU utilization of the master increases (slightly) when the system setup included 4 worker nodes. We expect this behavior due to the overhead costs of maintaining the 4 worker nodes. Also, it is evident that there is a sharp drop in the worker node’s CPU utilization, because the initial data set has been distributed. Now, the worker nodes have less data to process and run the algorithm on.



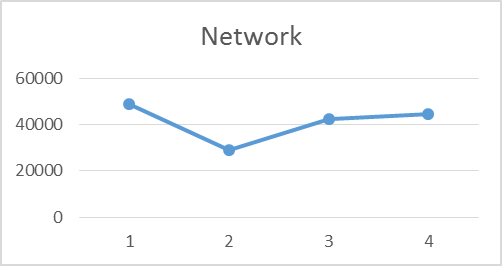
**Figure 17. Geometry closest pair CPU used (% Cycles spent in user mode v/s # of nodes)**

**Memory Used:** As seen from figure below, the master is almost running at its memory capacity. This is because the master has to perform overhead processes for the cluster to maintain its integrity. Also, the workers are running at their memory capacity since the operation utilized an in-memory divide-and-conquer algorithm.

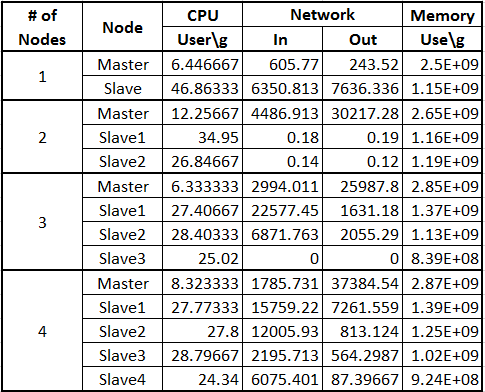


**Figure 182. Geometry closest pair memory used**

**Network Traffic:** As the number of nodes in a cluster increases, we should see an increase in the network traffic. However, looking at the first two data points in the figure below, we see that the network traffic has dropped. This anomaly is caused by missing data points from Ganglia, especially for the second data point. The reason the network traffic should increase is that the master has to communicate more as the worker nodes increase. Table can explain this anomaly better.

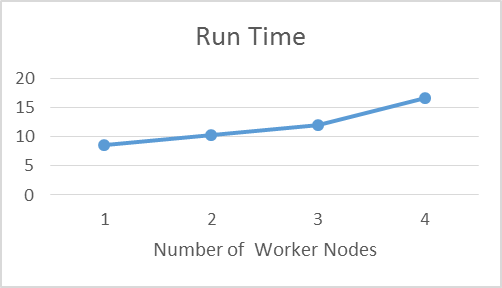


**Figure 19. Geometry closest pair network traffic**



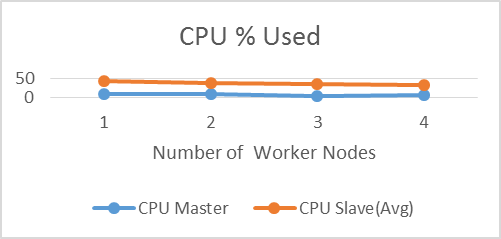
**Figure 20. Geometry closest pair raw data (Averaged for data points)**

### ***Spatial range query***



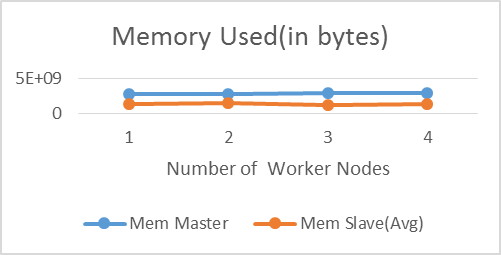
**Figure 21. Spatial range query run time**

As expected, the run-time of the function increases as the number of nodes in the cluster increases as seen in Figure 21. This is due to the nature of the experimental system which consists of multiple virtual machines running on a single host system. If the experiment were run in a cluster of multiple physical machines, then a decrease of elapsed time would be expected as the cluster size increases until the communications overhead overwhelms the advantage of increasing processing power.



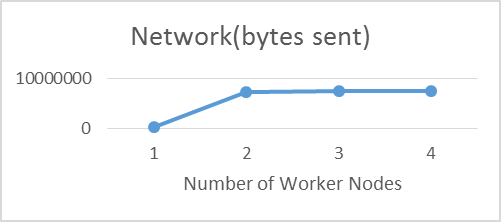
**Figure 22. Spatial range query CPU used**

As shown in Figure2, the master node uses about the same amount of CPU as the number of nodes increases. There is a very small drop-off. This is likely due to the increased run-time. The master node will perform the same tasks over a longer period, lowering the average CPU load.



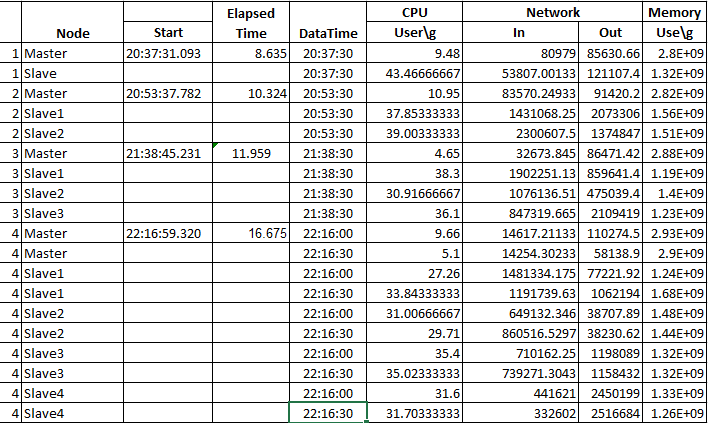
**Figure 23. Spatial range query memory used**

Figure 23 shows the memory used on the master and the average of the slave nodes. The master shows a small increase as the number of slaves increases. This is likely due to the need to keep track of the work from these slave nodes. The slaves show a tweaked change in memory usage, it increased when there are two nodes and then almost remained constant. One possible explanation is the limited size of the experimental data set and also because of some other issue with the memory utilization.



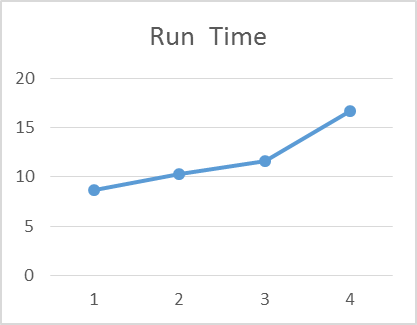
**Figure 24. Spatial range query network bytes sent**

The increase in network traffic shown in Figure 24 as from going to more than one node due to the increase in coordination between the master and slave nodes to perform the algorithm is required as the cluster gets more complicated. But after a certain point we don’t see much difference. One possible explanation could be the limited size of the experimental data.



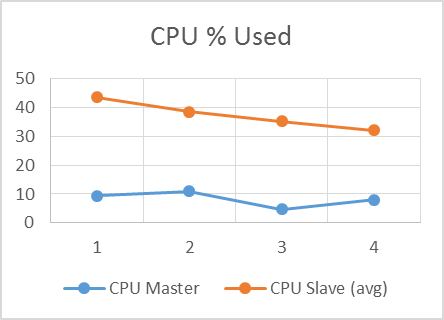
**Figure 25. Spatial range query raw data**

### ***Spatial join query***



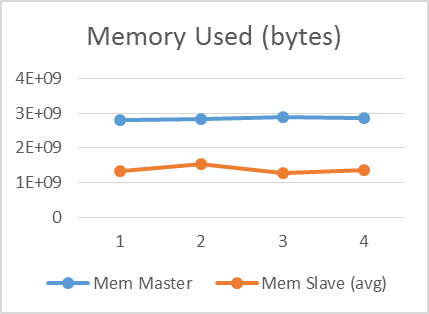
**Figure 26. Spatial join query run time**

As expected, the run-time of the function increases as the number of nodes in the cluster increases as seen in Figure 26. This is due to the nature of the experimental system which consists of multiple virtual machines running on a single host system. If the experiment were run in a cluster of multiple physical machines, then a decrease of elapsed time would be expected as the cluster size increases until the communications overhead overwhelms the advantage of increasing processing power.



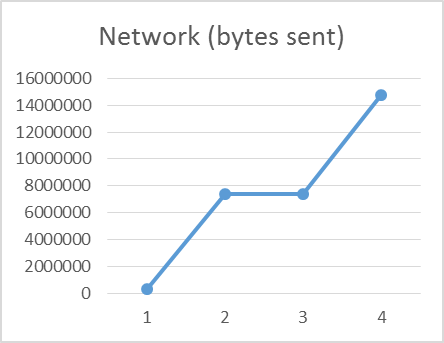
**Figure 27. Spatial join query CPU used**

As shown in Figure 27, the master node uses about the same amount of CPU as the number of nodes increases. This is likely due to the increased run-time. The master node will perform the same tasks over a longer period, lowering the average CPU load.



**Figure 28. Spatial join query memory used**

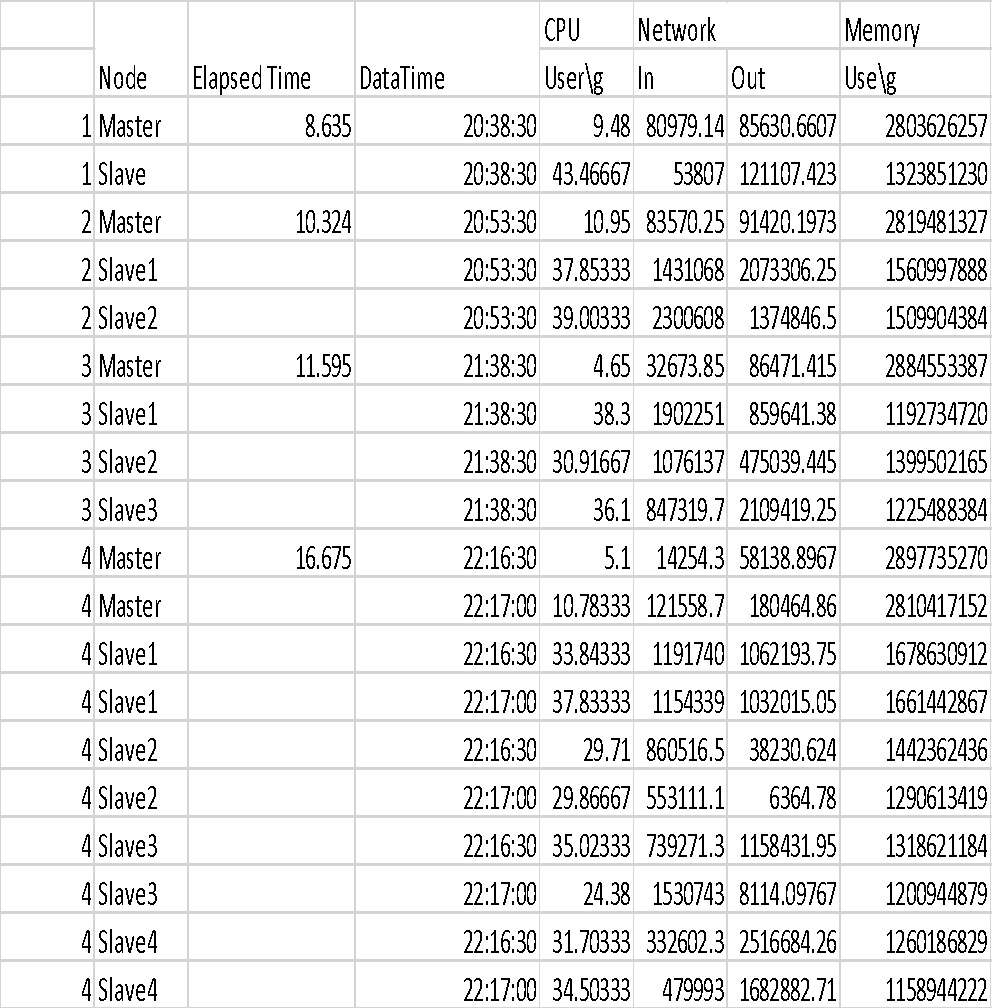
Figure 28 shows the memory used on the master and the average of the slave nodes. The master shows a small increase as the number of slaves increases. This is likely due to the need to keep track of the work from these slave nodes. The slaves show a tweaked change in memory usage, it increased when there are 2 nodes and then almost remained constant. One possible explanation is the limited size of the experimental data set and also because of some other issue with the memory utilization.



**Figure 3 Spatial join query network bytes sent**

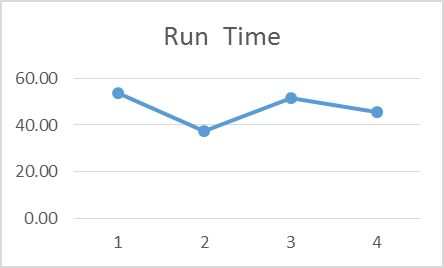
The increase in network traffic shown in Figure 29 as more nodes are added to the cluster is expected, as more coordination between the master and slave nodes to perform the algorithm is required as the cluster gets more complicated. In this for 2 and 3 mode cluster network traffic remained same because of limited data. But as expected for 4 node cluster it increased.

Figure 30 shows the raw results of running the Spatial join algorithm on the class provided point data set. This is analyzed from files taken from Ganglia and later processed to produce the above explained charts.



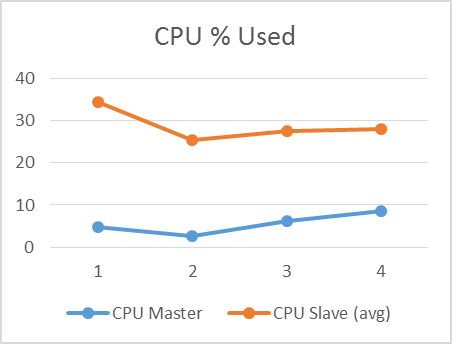
**Figure 30. Spatial join query raw data**

### ***Geospatial Aggregation***



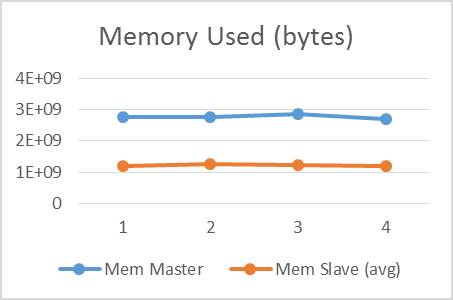
**Figure 31. Geospatial aggregation run time**

Figure 31 shows the runtime for spatial aggregation drops dramatically with two slaves in the cluster. Then it increases with three, and then drops again with four. Looking at the raw data in Figure 42**,** it appears the algorithm is not fully utilizing the slaves in with two or four, like it is with one and three.



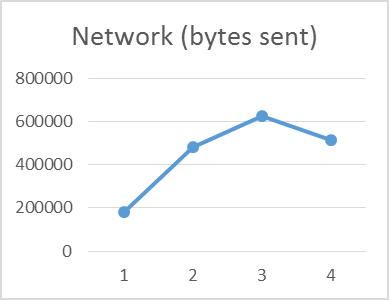
**Figure 32. Geospatial aggregation CPU used**

In Figure 32, we see the average slave CPU values drop with two, then slightly increase with three and four. This is likely due to the increased communications overhead as more slaves are added to the cluster without a real increase in CPU power. We also see the master CPU usage drop with two slaves, but then increase with three or four. This is probably due to the increased amount of communications with the slaves required as the cluster grows.



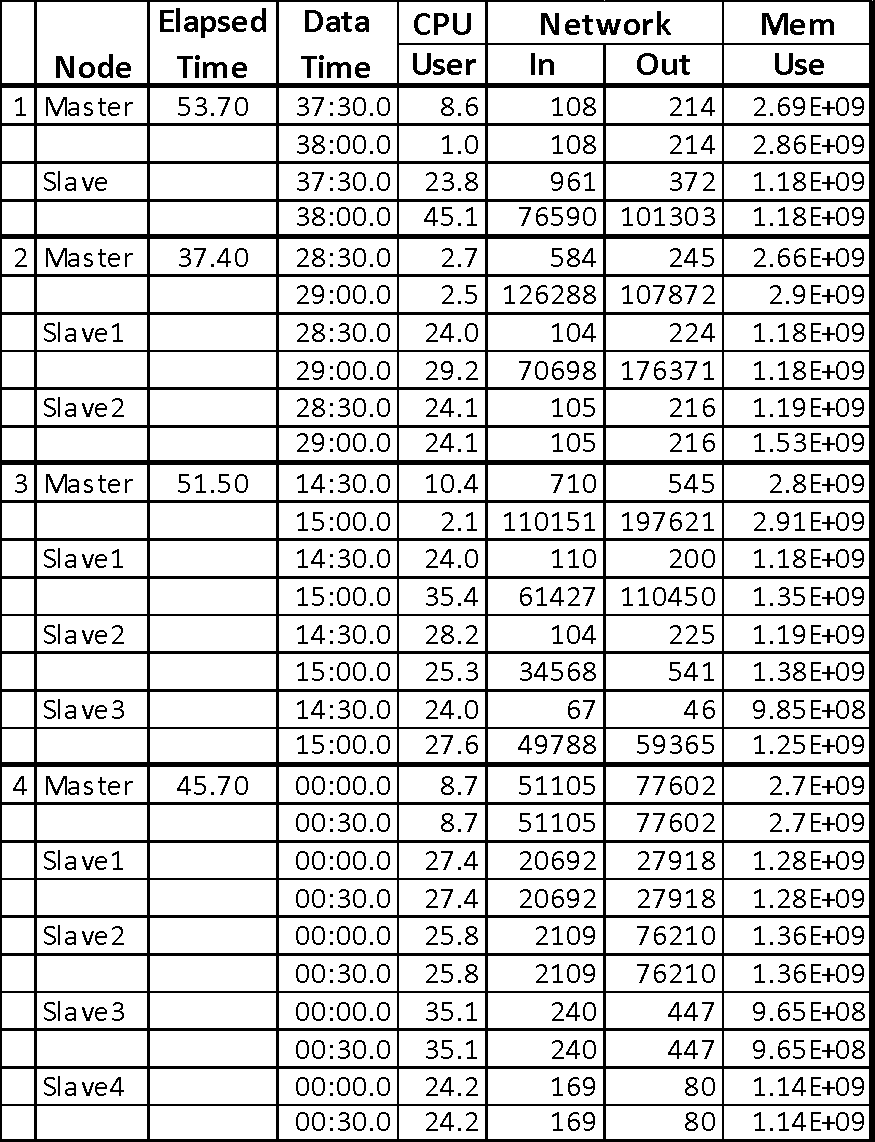
**Figure 33. Geospatial aggregation Memory used**

Figure 33 shows the memory used on the master and average memory across the slaves for each cluster size. The data in this chart doesn’t appear to give us any differences between the runs with different cluster sizes. Figure 34 show the total network bytes sent/received by all nodes in the cluster. The increasing network communications is expected as the master must coordinate the activities of more nodes. For a cluster with four slaves the amount of communication drops, which is not expected. This is explained by looking at Figure 35, which seems to show clusters three and four less active in this run than with smaller clusters. The raw data used to generate the graphs above is shown in Figure 35 The CPU and Memory usage is averaged across all readings for the master and all slaves. The network usage is totalled of all readings for master and slaves for the cluster.



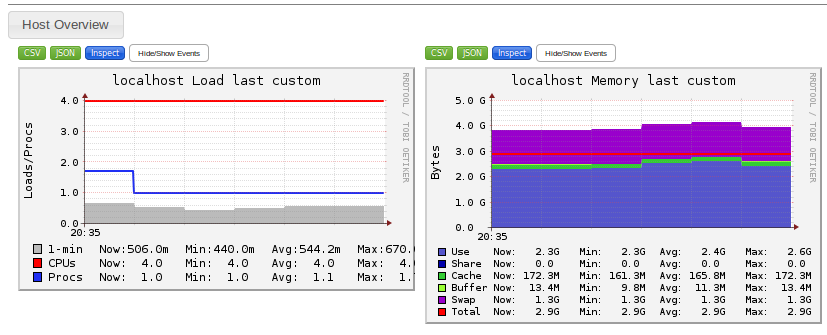
**Figure 34. Geospatial Aggregation Network Used**

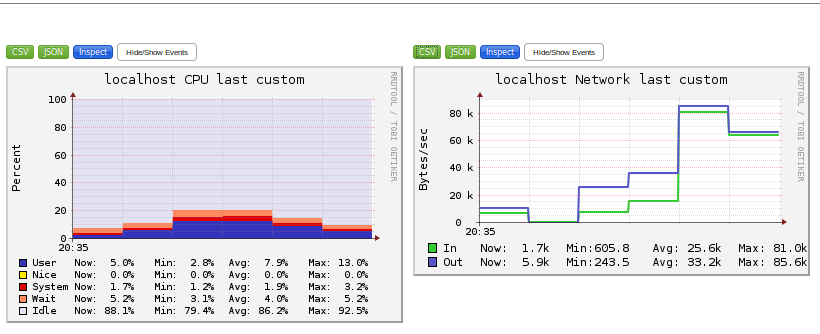
As discussed above, the Slave2 in the cluster with two slaves, and Slave3 and Slave4 in the cluster with four slaves do not appear to be taking their share of the load. This may be due to the way this geospatial aggregation function was designed and implemented.



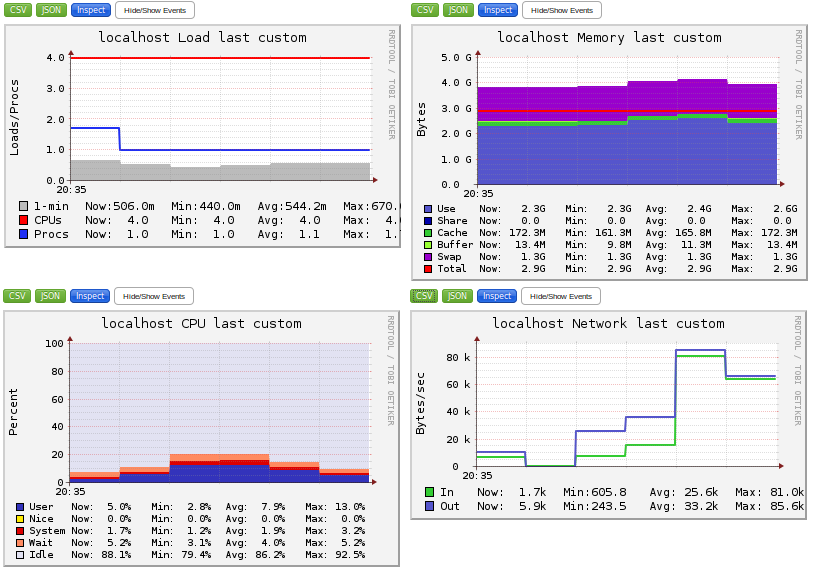
**Figure 35. Geospatial aggregation raw data**

# **Addendum – Run Time Figures**

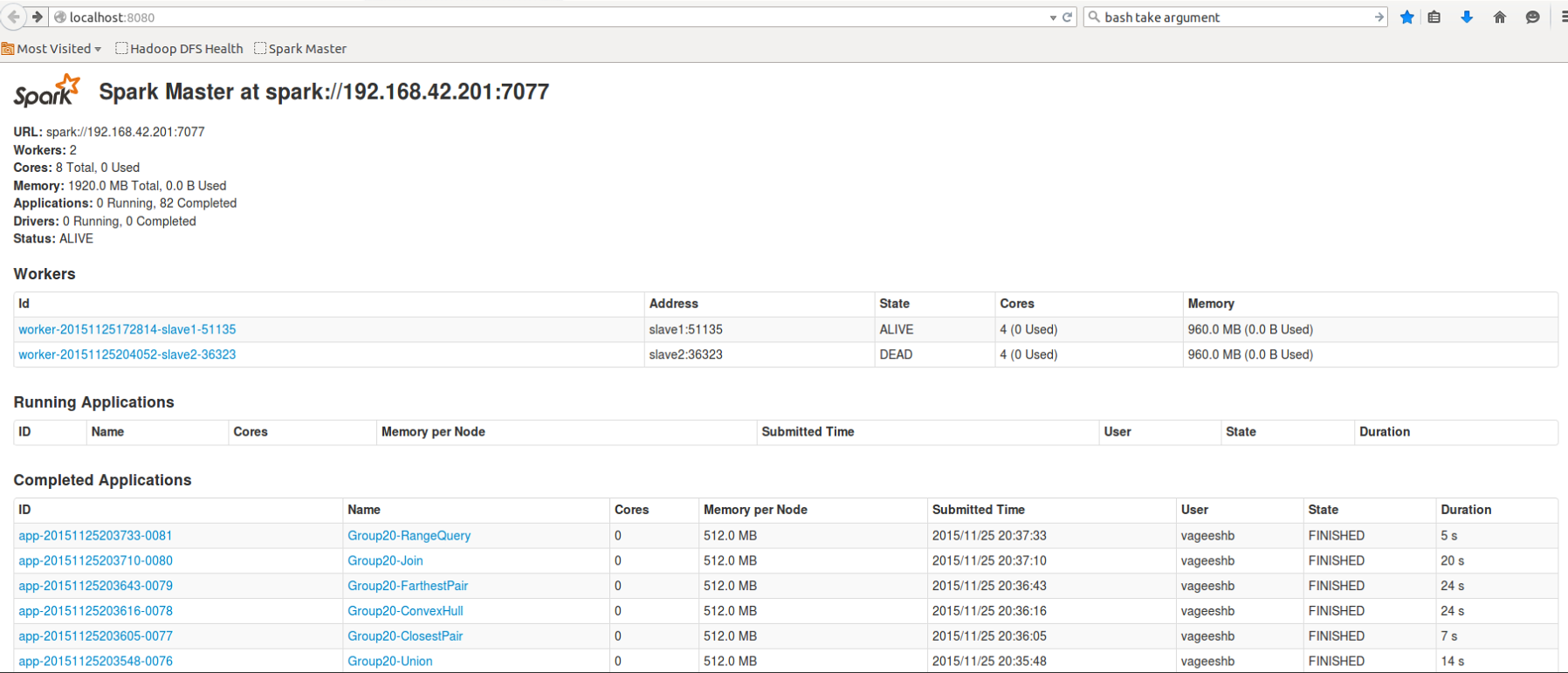
****

****

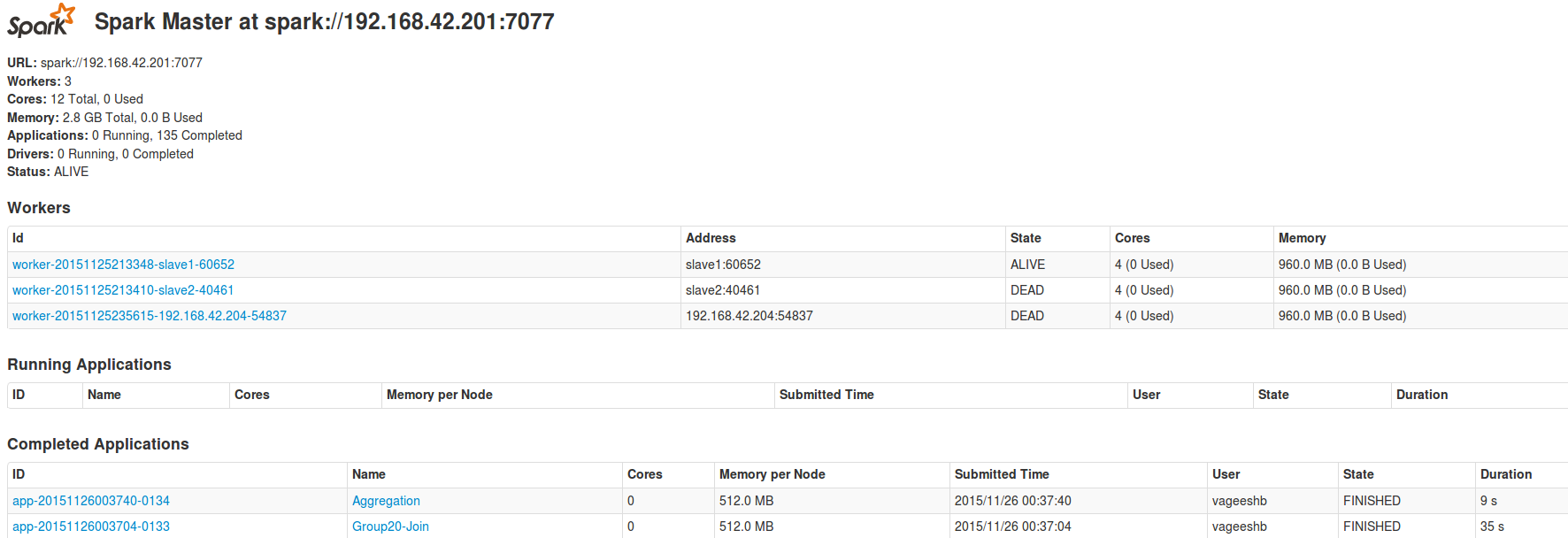
**Figure 36. Original six functions – Master ganglia reports**

****

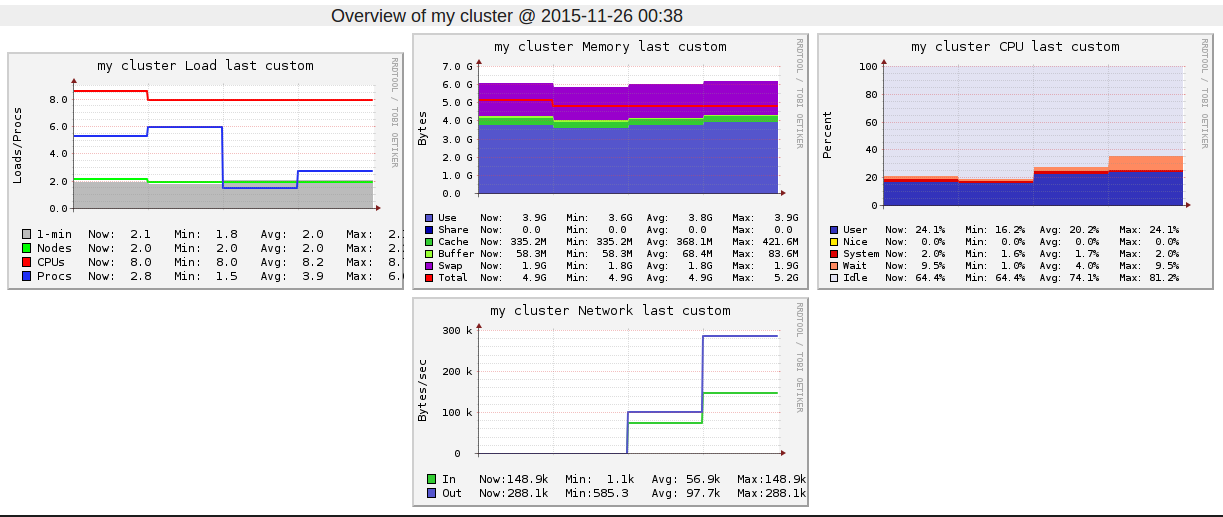
**Figure 37. Original six functions - Slave ganglia reports**



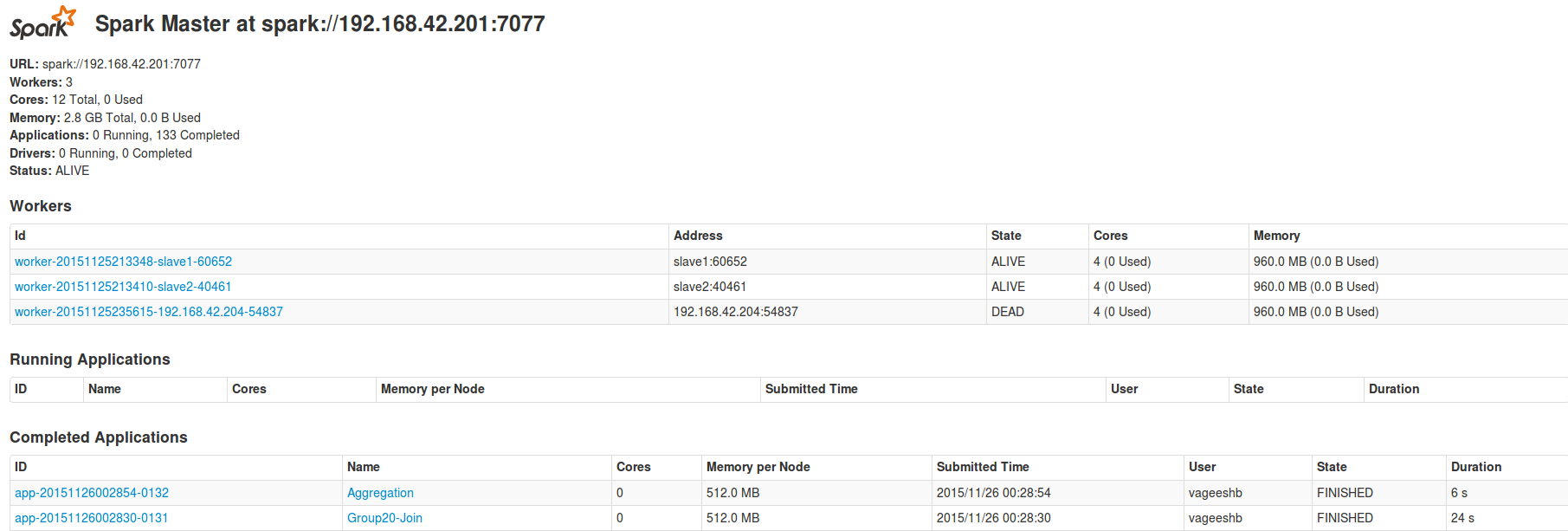
**Figure 38. Original six functions - Spark status**



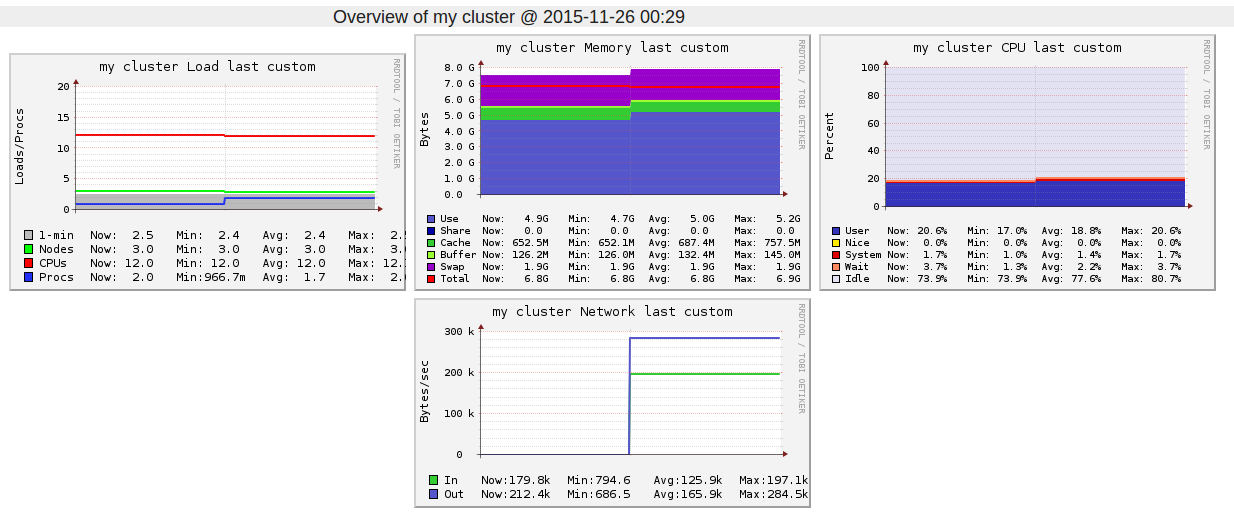
**Figure 39. Geospatial aggregation - Cluster 1 - Spark status**



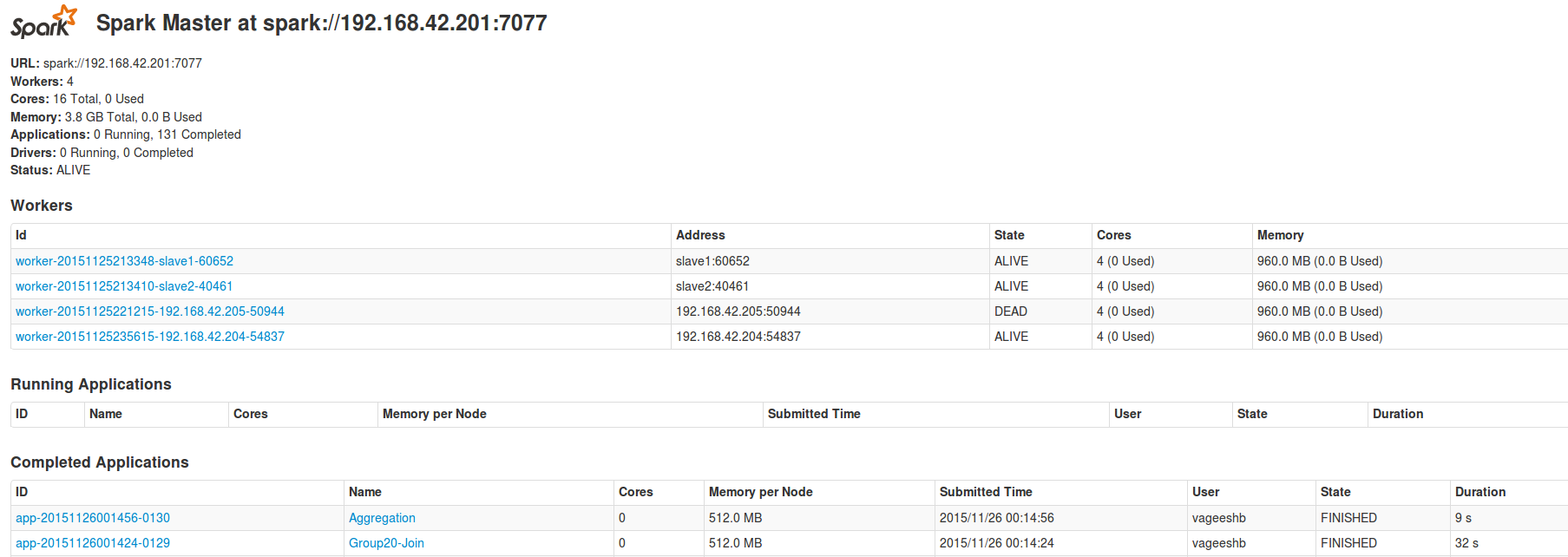
**Figure 40. Geospatial aggregation - Cluster 1 - RRDTool**



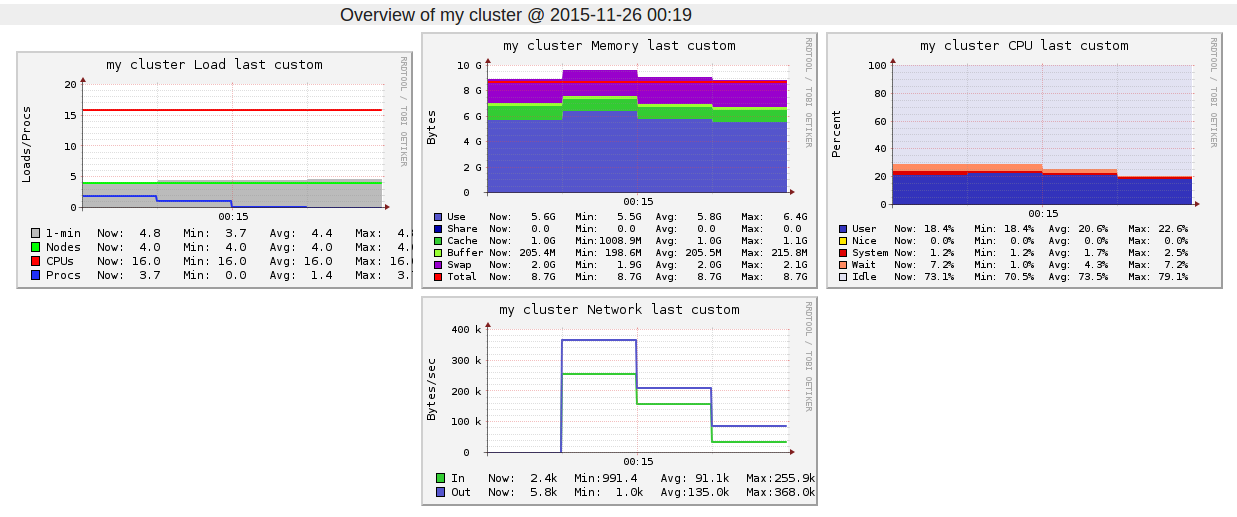
**Figure 41. Geospatial Aggregation - Cluster 2 - Spark Status**



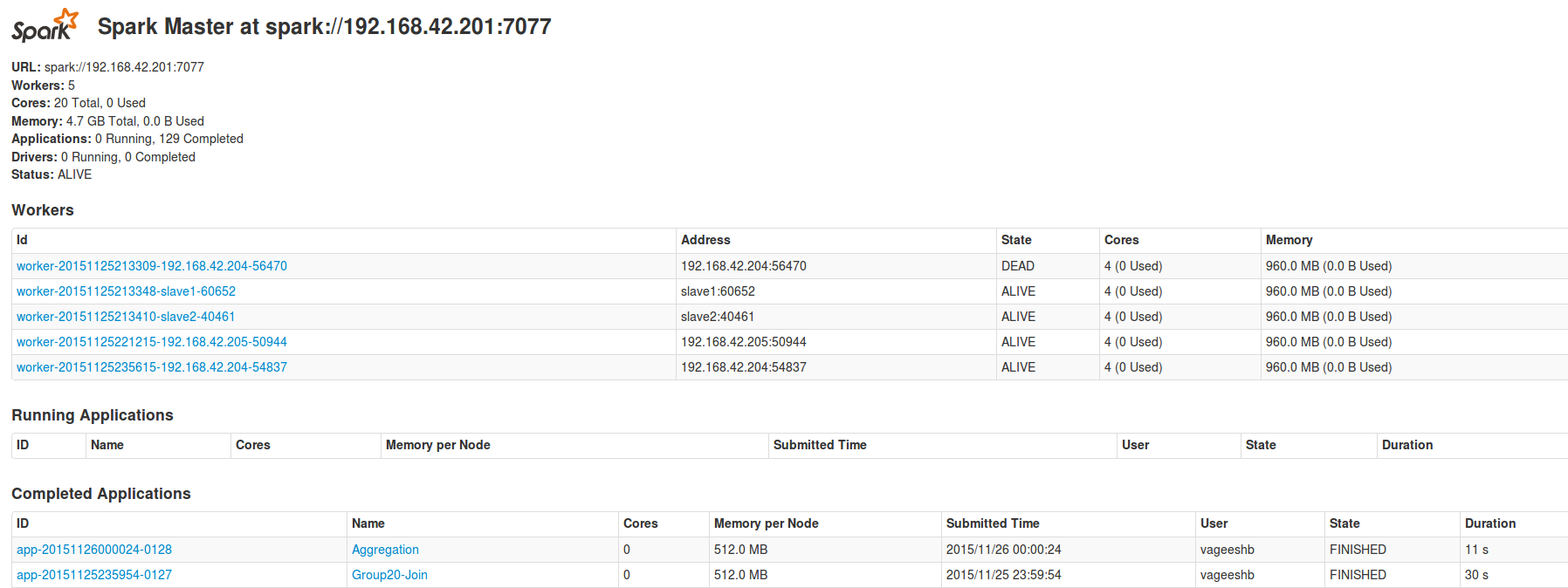
**Figure 42. Geospatial Aggregation - Cluster 2 – RRDTool**



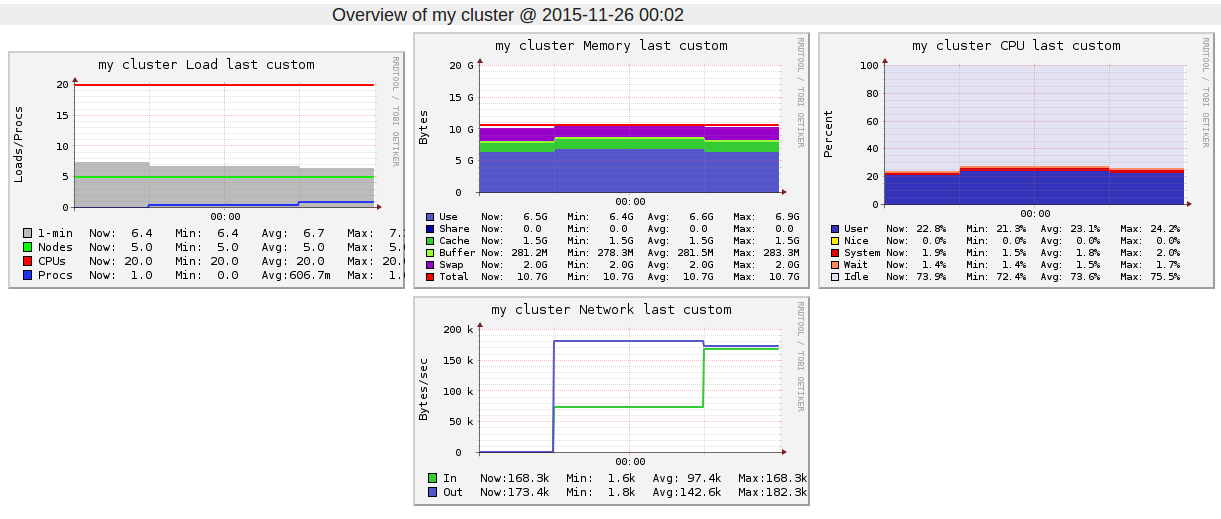
**Figure 43. Geospatial aggregation – Cluster 3 - Spark Status**



**Figure 44. Geospatial aggregation - Cluster 3 – RRDTool**

****

**Figure 45. Geospatial Aggregation – Cluster 4 – Spark Status**

****

**Figure 46. Geospatial aggregation - Cluster 4 – RRDTool**