CSE 512 Project Phase 4 Report

Audrey Wong, Boxin Du, Lei Chen, Rongyu Lin, Yihan Lu,

# INTRODUCTION

## Motivation

The goal of this project is to implement six geospatial operations--Geometry Union, Geometry Convex Hull, Geometry Farthest Pair, Geometry Closest Pair, Spatial Range Query, and Spatial Join Query--as well as a spatial aggregation (heat map) application. These operations are implemented using Apache Hadoop and Spark with Java. Additionally, experimental analyses are performed on the operations and application to measure and evaluate CPU utilization, memory utilization, and runtime costs.

## Problem Statement & Requirements

Requirements for the Geometry Union, Geometry Convex Hull, Geometry Farthest Pair, Geometry Closest Pair, Spatial Range Query, and Spatial Join Query operations, as well as the spatial aggregation application are as follows:

All operations load input from one or more files in HDFS, and they write output to a file in HDFS.

### Geometry Union

Geometry Union loads a set of polygons and outputs the union of the set. The polygons have the format “x1, y1, x2, y2” which is a pair of points (longitude, latitude), while the union has a bunch of points with the format “x, y” (longitude, latitude). The union consists of all the points that lie in the perimeter formed by the polygons.

### Geometry Convex Hull

Geometry Convex Hull loads a set of points and outputs the convex hull of the set. The points have the format “x, y” (longitude, latitude), while the convex hull has a bunch of points that use the same format. The convex hull consists of the points that make up the smallest convex polygon that contains all the input points; the convex hull points should be ordered in the clockwise direction.

### Geometry Farthest Pair

Geometry Farthest Pair loads a set of points and outputs the farthest pair of the set. The points have the format “x, y” (longitude, latitude), while the farthest pair has two points that use the same format. The farthest pair consists of the two points with the largest Euclidean distance between them.

### Geometry Closest Pair

Geometry Closest Pair loads a set of points and outputs the closest pair of the set. The points have the format “x, y” (longitude, latitude), while the closest pair has two points that use the same format. The closest pair consists of the two points with the smallest Euclidean distance between them; the points should lie on the convex hull.

### Spatial Range Query

Spatial Range Query loads a set of polygons and outputs the query result of the set. The polygons are loaded from two different inputs--the first input has the format “id, x1, y1, x2, y2” which contains a polygon defined by a pair of points (longitude, latitude) and an id, while the second input has the format “x1, y1, x2, y2” which is a polygon defined similarly (but without an id) and is the query window of the range query. The query result has a bunch of polygon ids with the format “id”. The spatial range consists of all the polygons that lie within the query window.

### Spatial Join Query

Spatial Join Query loads two sets of polygons and outputs the query result of the set. The polygons are loaded from two different inputs--the first input has the format “Aid, x1, y1, x2, y2” which contains a polygon defined by a pair of points (longitude, latitude) and an id, while the second input has the format “Bid, x1, y1, x2, y2” which is a polygon defined similarly. The query result has a bunch of rows with the format “Aid, Bid1, Bid2, Bid3, …” which contains polygon ids; each row has one Aid and zero or many Bids. The spatial join consists of all the polygons that intersect with another polygon.

### Spatial Aggregation Application

The spatial aggregation application takes the same input as the Spatial Join Query operation and outputs the number of points that lie in a polygon. The input consists of one target point dataset and one query rectangle set, while the output has the format “<Rectangle, Count>”. This application must be tested using two provided datasets: arealm (target point set) and zcta510 (query rectangle set).

### Experimental Analysis

Experimental analysis on the operations and application have three metrics: CPU utilization per node when running on the cluster, memory utilization per node when running on the cluster, and runtime with operation costs. Different cluster sizes must be used--1 node, 2 nodes, and 4 nodes--to demonstrate performance on a distributed system. CPU utilization and memory utilization should be measured using Ganglia Web UI, while runtime should be measured using Spark Web UI. Analyses should be supported by screenshots and graphs with explanations.

# SYSTEM ARCHITECTURE & IMPLEMENTATION

## System Architecture

From top down, the project is set up as a Maven project with modules for each of the six geospatial operations plus the spatial aggregation application. It runs on a number of virtual machines with Ubuntu 15.04 LTS using VMWare Player. The driver programs are implemented on the Spark cluster computing system by calling high-level Java API. Below the Spark system, the project applies HDFS (Hadoop Distributed File System) for storing inputs and outputs. The project system architecture can basically be divided into three components: the driver program, which implements the six geospatial functionalities, the Spark computing system that is used for cluster computations, and the Hadoop distributed file system for storing datasets. The detailed deployment of the Hadoop system is described in the Experimental Setup section.

## Implementation

The project uses various tools for its implementation. All the operations and the application are written in Java. Apache Hadoop allows input to be read from files in HDFS and output to be written to a file in HDFS. Apache Spark enables map-reduce functionality. Java Topology Suite (JTS) provides an API of spatial predicates and functions for processing geometry.

Each of the modules for the geospatial operations includes a helper class called “JTSUtils” with various functions to aid in the map-reduce process, such as converting input points into JTS Coordinate or Geometry objects or sorting coordinates into a particular order.

### Geometry Union

The Geometry Union operation was implemented using the following process:

Step 1: Read lines of input from HDFS.

Step 2: Map each line of input to a polygon.

Step 3: Reduce the polygons by unioning every two polygons.

Step 4: Get the coordinates from the final polygon and sort them.

Step 5: Write the coordinates to the output in HDFS.

Note that step 2 involves calling a function in JTSUtils which creates a polygon or rectangle (a JTS Geometry object) using the first point in the line as the top left corner of the rectangle and the second point as the bottom right corner of the rectangle. Step 4 involves calling another function in JTSUtils to get the coordinates from the final polygons and sort them.

### Geometry Convex Hull

The Geometry Convex Hull operation was implemented using the following process:

Step 1: Read lines of input from HDFS.

Step 2: Map each line of input to a convex hull

Step 3: Reduce the convex hulls by combining every two ones.

Step 4: Get the coordinates from the final convex hull and sort them.

Step 5: Write the coordinates to the output in HDFS.

Note that step 2 involves calling a function in JTSUtils which creates a convex hull using the single point. Step 4 calls the same function as step 4 in the Geometry Union operation to get the coordinates of the final convex hull and sort them.

### Geometry Farthest Pair

The Geometry Farthest Pair operation was implemented using the following process:

Step 1: Read lines of input from HDFS.

Step 2: Get the coordinates of the convex hull using the map-reduce process from the Geometry Convex Hull operation.

Step 3: Map pairs of coordinates to line segments.

Step 4: Reduce the line segments to the line segment with the greatest length.

Step 5: Get the coordinates from the final line segment and sort them.

Step 6: Write the coordinates to the output in HDFS.

### Geometry Closest Pair

The Geometry Closest Pair operation was implemented using the following process:

Step 1: Read lines of input from HDFS.

Step 2: Map each line of input to a coordinate.

Step 3: Get all pairs of coordinates.

Step 4: Filter pairs of the same coordinate.

Step 5: Map pairs of coordinates to line segments.

Step 6: Reduce line segments to the line segment with the shortest length.

Step 7: Get the coordinates from the final line segment and sort them.

Step 8: Write the coordinates to the output in HDFS.

Note that step 6 involves calling a JTSUtils function to create a JTS Coordinate object from the two points in the line.

### Spatial Range Query

The Spatial Range Query operation was implemented using the following process:

Step 1: Read lines of first input from HDFS.

Step 2: Map each line of first input to a pair consisting of an id and a point.

Step 3: Read lines of second input from HDFS.

Step 4: Read the lines of second input to get the query window.

Step 5: Filter the pairs of ids and points to the ones contained in the query window.

Step 6: Map the final pairs to their ids.

Step 7: Collect the ids and sort them.

Step 8: Write the ids to the output in HDFS.

Note that Step 2 involves calling a JTSUtils function to get the id and point from the line, while step 4 involves calling another JTSUtils function to create the query window rectangle by using the first point in the line as the top left corner and the second point in the line as the bottom right corner.

### Spatial Join Query

The Spatial Join Query operation was implemented using the following process:

Step 1: Read lines of first input from HDFS.

Step 2: Read lines of second input from HDFS.

Step 3: Map each line of first input to a pair consisting of an id and polygon.

Step 4: Map each line of second input to a pair consisting of an id and polygon.

Step 5: Get pairs of pairs consisting of an id and polygon.

Step 6: Filter the pairs to get the pairs that intersect.

Step 7: Map the pairs to their ids.

Step 8: Sort the output.

Step 9: Write the output to HDFS.

Note that steps 3 and 4 both involve calling the JTSUtils function to get the id and geometry from the line.

### Spatial Aggregation Application

The spacial aggregation application was implemented using the following process:

Step 1: Read lines of first input from HDFS.

Step 2: Read lines of second input from HDFS.

Step 3: Get the range of x and y that covers all the coordinates.

Step 4: Divide the range of x and y into a grid of N rows and columns, and each cell has an ID in the format of row number and column number connected by an underscore.

Step 5: Map each line of input to a pair that has the cell ID as the key. A rectangle may be mapped to more than one pair if it intersects several cells.

Step 6: Join the pairs from input1 and input2 by the same key.

Step 7: Filter the joined pairs if both geometry intersects.

Step 8: Map each pair to a new pair where the value is number one, and reduce by key.

Step 9: Write the output to HDFS.

# EXPERIMENTAL SETUP

The experiment used VMWare Player to form clusters with various numbers (1, 2, and 4) of virtual machines with Ubuntu 15.04 LTS. We had 5 virtual machines running on 3 laptops, the following was the setup:

Laptop1: Hadoop Datanode, Spark master, GangliaWebUI

Laptop2: Hadoop Datanode, Spark worker2

Laptop3: Hadoop Datanode, Spark worker3, Spark worker4, Spark worker5.

We were using a NetGear R6200 Wireless Router to connect the laptops. One thing to notice was that the Laptop3 was relatively more powerful; therefore we had three instances of VMs running on it.

Hadoop was set up for the experiments. Three data nodes are deployed as one master and two workers. Master can SSH to both workers and workers can SSH to master, but workers cannot SSH to each other.

Spark was set up with up to four worker nodes for the experiments.

The performance metrics used were the following: CPU utilization, memory utilization, and runtime.

CPU utilization and memory utilization were measured using Ganglia Web UI, while runtime was measured using Spark Web UI.

The following experiments were performed on all geospatial operations and the spatial aggregation application: (1) Measure the communication costs (in terms of sent bytes), (2) Measure the memory utilization on each machine, (3) Measure the runtime for different datasets on a single machine, and (4) Measure the runtime while increasing the number of machines.

For each function, we used test data sets with increasing size. Each run of the test was named by the format “Group2-<FunctionName>-Size”. For example, “Group2-Union-5” meant the test data was of size of 10000(ten to the power of 5) and was applied to the Union function.

# EXPERIMENTAL EVALUATION

## Geometry Union

We ran the tests using test data of different sizes and with different number of worker nodes. The running time was in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
|  | One node | Two nodes | Four nodes |
| 10 | 14 sec | 12 sec | ?? sec |
| 100 | 15 sec | 11 sec | ?? sec |
| 1000 | 16 sec | 13 sec | ?? sec |
| 10000 | 32 sec | 20 sec | ?? sec |
| 100000 | 46 sec | 32 sec | ?? sec |
| 1000000 | 204 sec | 108 sec | 66 |

Table 4.1-1 Running time of Geometry Union

Since launching a spark application needed some time which was unneglectable in our test cases, we could approximately use the smallest running time as the application launching overhead and subtract it from all other times. After the subtraction, we have the following table of actual function running times.

|  |  |  |  |
| --- | --- | --- | --- |
|  | One node | Two nodes | Four nodes |
| 10 | 3 sec | 1 sec | ?? sec |
| 100 | 4 sec | 0 sec | ?? sec |
| 1000 | 5 sec | 2 sec | ?? sec |
| 10000 | 21 sec | 9 sec | ?? sec |
| 100000 | 35 sec | 21 sec | ?? sec |
| 1000000 | 193 sec | 97 sec | 55 sec |

Table 4.1-2 Running time without the launching overhead

Using a bar chart as following, we can see that by adding more worker nodes, we were able to reduce the running time near proportionally.

[###################ADD\_CHART#################]

### Ganglia Web UI Statistics

We use ganglia to monitor the metric for the worker nodes as well as the master node. Since the number of nodes had little affect over the metric pattern and the pattern was very similar across each worker node, we will use the one worker setting to illustrate the metric.

The following were metrics obtained from the Ganglia Web UI for the master node:

Figure 4.1.1-2 shows its CPU utilization.

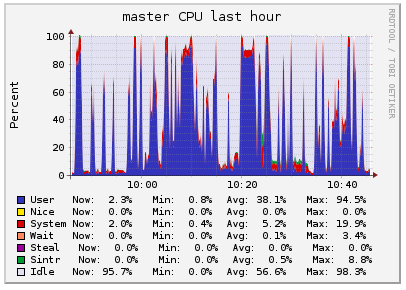


Figure 4.1.1-2. Geometry Union CPU utilization for master node

Figure 4.1.1-3 shows its memory utilization.

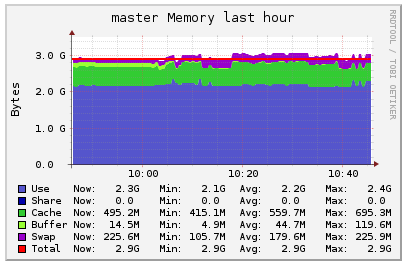


Figure 4.1.1-3. Geometry Union memory utilization for master node

The following are statistics obtained from the Ganglia Web UI for the worker node:

Figure 4.1.1-5 shows its CPU utilization. Around the 10:40 mark, there are four short and thin peaks followed by one tall and wide peak. Each peak corresponds to a different job using an increasingly larger dataset size. The peaks are shorter and thinner in the beginning because a smaller dataset requires less CPU and time to finish. As the dataset becomes very large at the end, CPU utilization appears to reach 100 percent.

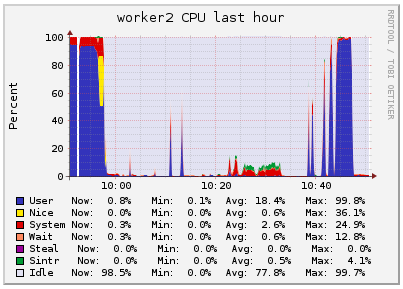


Figure 4.1.1-5. Geometry Union CPU utilization for worker node

Figure 4.1.1-6 shows its memory utilization. Similar to the CPU utilization graph, there are smaller, thinner peaks followed by a larger, wider peak around the 10:40 mark. These peaks also correspond to the different dataset sizes. Larger dataset sizes require larger amounts of memory than smaller dataset sizes.

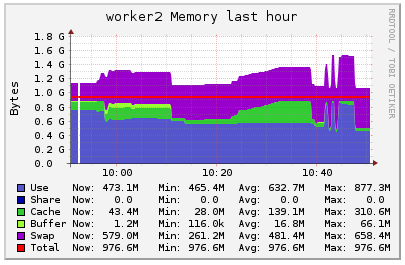


Figure 4.1.1-6. Geometry Union memory utilization for worker node

Figure 4.1.1-7 shows the communication costs of applications using different sizes of datasets. From 10:00 to 10:20, 12 processes are created at most and whenever there is a Spark process going on, there is a small communication cost of less than 0.1 M between master and workers. Figure 4.1.1-8 shows the curve is of the same shape, but the color is green. That means in the process, master calls workers for most of the time. The master should call the workers and tell the tasktracker how to allocate tasks and tell the datanodes which data to retrieve. From 10:20 to 10:38, 4 big peaks can be observed from the graphs. When the dataset is small, the communication cost is small (less than 0.1 M), and when dataset becomes larger, the communication cost will grow to more than 1 M.

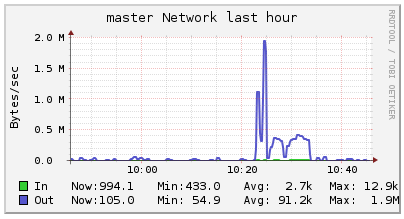


Figure 4.1.1-7. Geometry Union network utilization for master node

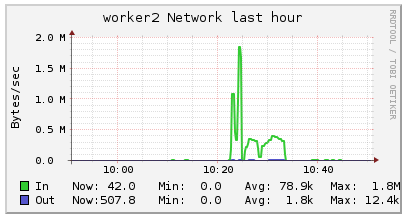


Figure 4.1.1-8. Geometry Union network utilization for worker node

.

## Geometry Convex Hull

### Ganglia Web UI Statistics

The following are statistics obtained from the Ganglia Web UI for the master node:

Figure 4.2.1-2 shows its CPU utilization.

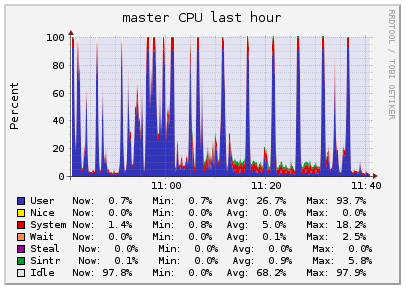


Figure 4.2.1-2. Geometry Convex Hull CPU utilization for master node

Figure 4.2.1-3 shows its memory utilization.

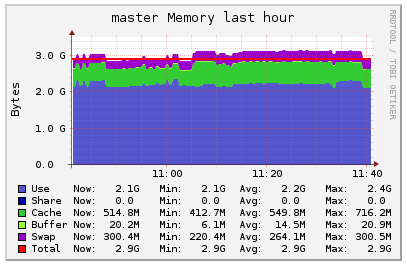


Figure 4.2.1-3. Geometry Convex Hull memory utilization for master node

The following are statistics obtained from the Ganglia Web UI for the worker node:

Figure 4.2.1-5 shows its CPU utilization. Around the 11:00 mark, there are five thin peaks; the first three are relatively short, while the last two are tall. These peaks correspond to different dataset sizes that were tested. As the dataset size increased, the CPU utilization also increased. All five peaks are thin because the durations for which that level of CPU utilization was required are short. However, towards the right side of the graph before the 11:40 mark, is a single tall and wide peak reaching 100 percent. This peak corresponds to an extremely large dataset size that required more CPU for a longer amount of time.

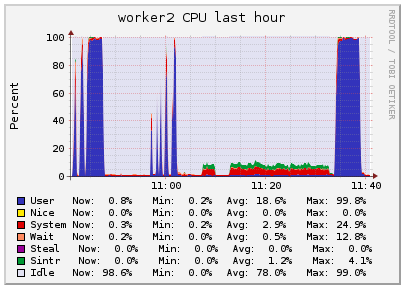


Figure 4.2.1-5. Geometry Convex Hull CPU utilization for worker node

Figure 4.2.1-6 shows its memory utilization. Around the 11:00 mark are a few thin peaks, while around the 11:35 mark is a single wide peak. Memory utilization was small for the smaller dataset sizes used in the beginning. It became large for the larger dataset sizes used later, plus longer for the largest dataset size used at the end. Overall, memory utilization increases as the dataset size increases.

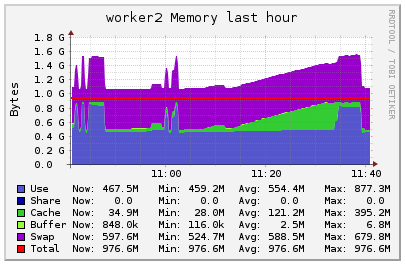


Figure 4.2.1-6. Geometry Convex Hull memory utilization for worker node

Communication costs can be observed in Figures 4.2.1-7 and 4.2.1-8. Whenever there is an Spark application going on, there is a communication cost of less than 100 k for each ongoing application. There is little difference among the communication costs when the dataset is relatively small, but when the dataset is large, a noticeable peak is observed. That means in the Geometry Convex Hull operation, communication costs will rise up sharply when the dataset is large. At 11:30 when the dataset is very large, the communication costs rose up to more than 400 k.

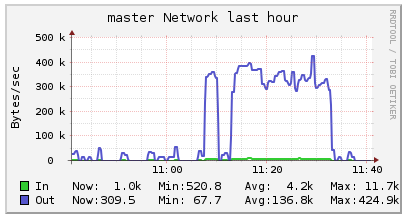


Figure 4.2.1-7. Geometry Convex Hull network utilization for master node

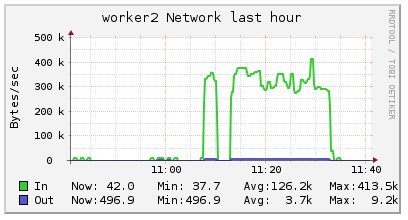


Figure 4.2.1-8. Geometry Convex Hull network utilization for worker node

### Spark Web UI Statistics for 1 Machine

The Spark Web UI has statistics for showing the runtimes for different dataset sizes on 1 machine. The list of completed applications has six different dataset sizes tested for a single machine: Group2-ConvexHull-2, Group2-ConvexHull-3, Group2-ConvexHull-4, Group2-ConvexHull-5, and Group2-ConvexHull-6, and Group2-ConvexHull-7. The number at the end of the application name, such as the “2” in “Group2-ConvexHull-2” indicates the dataset used contained 10^2 rows.

Figure 4.2.2-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Similar to the Geometry Union operation, the runtime increases as the dataset size increases overall. There is no increase in runtime between the 10^2 and 10^3 datasets, while the increase is very small between the 10^3 and 10^4 datasets. The runtime gradually increases from the 10^4 to the 10^6 datasets before dramatically increasing between the 10^6 and 10^7 datasets. Based on these observations, the program appears to easily handle dataset sizes below 10^4 rows. 10^6 rows is its limit before the Geometry Convex Hull operation starts to significantly slow down.

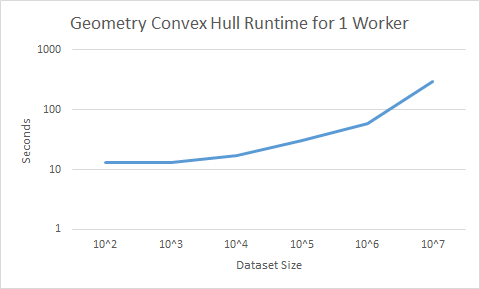


Figure 4.2.2-2. Geometry Convex Hull runtime for different dataset sizes on 1 machine

### Spark Web UI Statistics for 2 Machines

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 2 machines. The list of completed applications has six different dataset sizes tested for a single machine: Group2-ConvexHull-2, Group2-ConvexHull-3, Group2-ConvexHull-4, Group2-ConvexHull-5, and Group2-ConvexHull-6, and Group2-ConvexHull-7. The number at the end of the application name, such as the “2” in “Group2-ConvexHull-2” indicates the dataset used contained 10^2 rows.

Figure 4.2.3-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Just as with the Geometry Union operation, the shape of the graph for 2 machines appears very similar to the graph for 1 machine. There is little difference in runtime between the 10^2 and 10^3 datasets, with a gradually rising runtime between the 10^3 and 10^6 datasets. Then there is a shaper increase from the 10^6 and 10^7 datasets. The same trend is evident here: as the dataset size increases, the runtime also increases. However, it can be observed that with 2 machines, the overall runtime is slightly lower than the overall runtime with 1 machine because the line in the graph is also slightly lower.

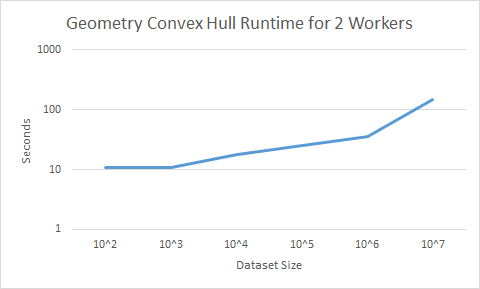


Figure 4.2.3-2. Geometry Convex Hull runtime for different dataset sizes on 2 machines

## Geometry Farthest Pair

### Ganglia Web UI Statistics

The following are statistics obtained from the Ganglia Web UI for the master node:

As can be seen in Figure 4.3.1-2, between 14:20 and 14:27, there is a peak of 5 Loads/Procs, which means 5 processes are running on this period of time on peak for the Geometry Farthest Pair operation.

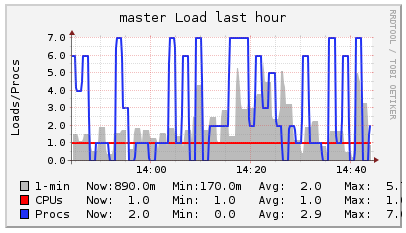


Figure 4.3.1-2. Geometry Farthest Pair load utilization for master node

Figure 4.3.1-3 shows its CPU utilization.

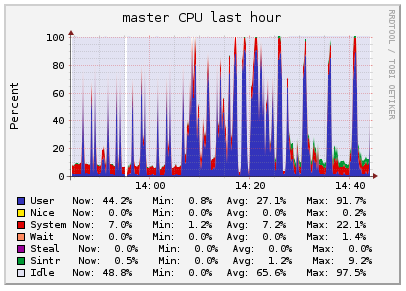


Figure 4.3.1-3. Geometry Farthest Pair CPU utilization for master node

Figure 4.3.1-4 shows its memory utilization.

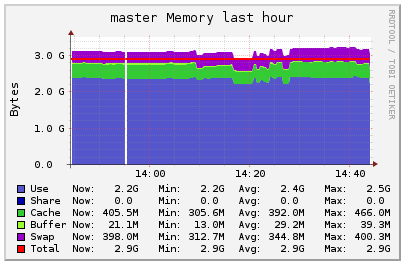


Figure 4.3.1-4. Geometry Farthest Pair memory utilization for master node

The following are statistics obtained from the Ganglia Web UI for the worker node:

Figure 4.3.1-6 shows its CPU utilization. Applications were submitted to test different dataset sizes between 14:20 and 14:27. Because of the way the peaks grow taller and slightly wider across this interval, there is a gradual increase in CPU utilization as the dataset size increases. The largest dataset size tested did not reach 100 percent CPU utilization unlike the Geometry Union and Geometry Convex Hull operations.

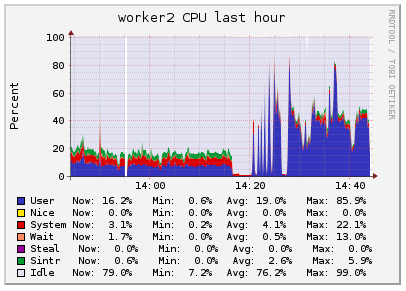


Figure 4.3.1-6. Geometry Farthest Pair CPU utilization for worker node

Figure 4.3.1-7 shows its memory utilization. Between 14:20 and 14:27 are a few peaks corresponding to the different dataset sizes that were tested. The peaks gradually grow taller as the dataset size increased, which indicates that memory utilization also increases as dataset size increases.

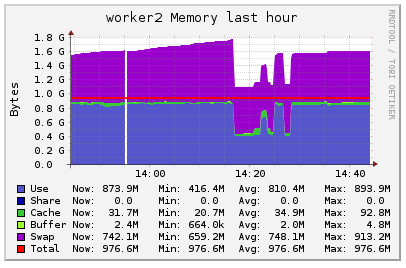


Figure 4.3.1-7. Geometry Farthest Pair memory utilization for worker node

In Figures 4.3.1-8 and 4.3.1-9, there is an ascending part when the applications were tested, which shows the rising communication costs between master and workers. The maximum communication cost appeared when the time was 14:26. and the cost was about 550k.

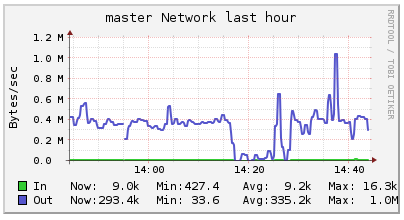


Figure 4.3.1-8. Geometry Farthest Pair network utilization for master node

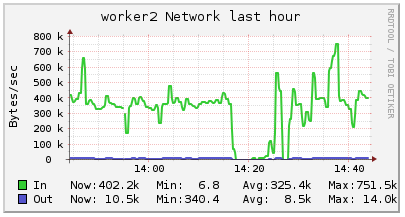


Figure 4.3.1-9. Geometry Farthest Pair network utilization for worker node

### Spark Web UI Statistics for 1 Machine

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 1 machine. The list of completed applications has seven different dataset sizes tested for a single machine: Group2-FarthestPair-1, Group2-FarthestPair-2, Group2-FarthestPair-3, Group2-FarthestPair-4, Group2-FarthestPair-5, and Group2-FarthestPair-6, and Group2-FarthestPair-7. The number at the end of the application name, such as the “1” in “Group2-FarthestPair-1” indicates the dataset used contained 10^1 rows.

Figure 4.3.2-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Similar to the previous operations, the runtime increases as the dataset size increases overall. The increase is subtle between the 10^2 and 10^5 datasets, less subtle between the 10^5 and 10^6 datasets, and dramatic between 10^6 and 10^7. The Geometry Farthest Pair operation likely performs the fastest for dataset sizes below 10^5 rows; it reaches its limit at 10^6 rows before a major slowdown is experienced.

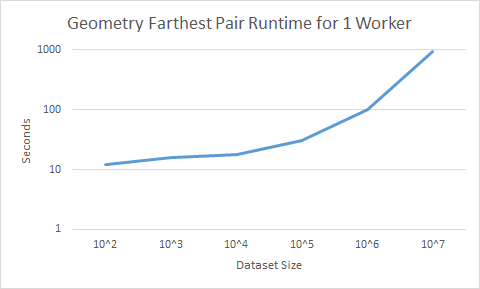


Figure 4.3.2-2. Geometry Farthest Pair runtime for different dataset sizes on 1 machine

### Spark Web UI Statistics for 2 Machines

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 2 machines. The list of completed applications has seven different dataset sizes tested for a single machine: Group2-FarthestPair-1, Group2-FarthestPair-2, Group2-FarthestPair-3, Group2-FarthestPair-4, Group2-FarthestPair-5, and Group2-FarthestPair-6, and Group2-FarthestPair-7. The number at the end of the application name, such as the “1” in “Group2-FarthestPair-1” indicates the dataset used contained 10^1 rows.

Figure 4.3.3-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Overall, the shape of the graph appears to be slightly more linear than the graph for 1 machine. The line is also slightly lower in height. The runtime has significantly decreased for the 10^6 and 10^7 datasets, indicating that 2 machines perform the Geometry Farthest Pair operation much faster than 1 machine. Additionally, 2 machines would probably handle even larger datasets faster than 1 machine can.

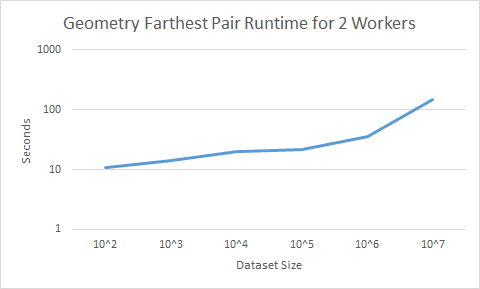


Figure 4.3.3-2. Geometry Farthest Pair runtime for different dataset sizes on 2 machines

## Geometry Closest Pair

### Ganglia Web UI Statistics

The following are statistics obtained from the Ganglia Web UI for the master node:

Figure 4.4.1-2 shows its CPU utilization.

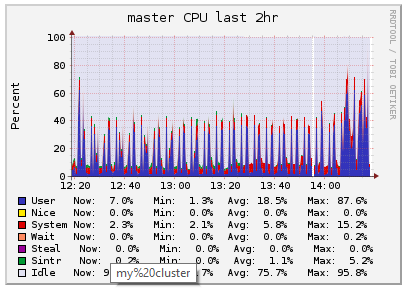


Figure 4.4.1-2. Geometry Closest Pair CPU utilization for master node

Figure 4.4.1-3 shows its memory utilization.

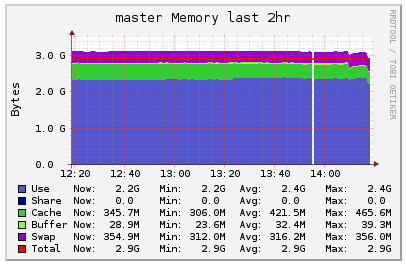


Figure 4.4.1-3. Geometry Closest Pair memory utilization for master node

The following are statistics obtained from the Ganglia Web UI for the worker node:

Figure 4.4.1-5 shows its CPU utilization.

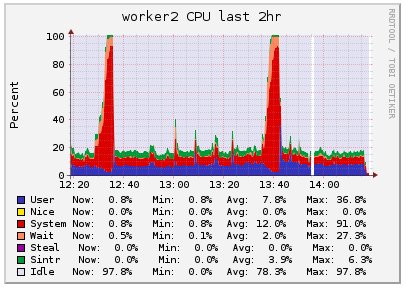


Figure 4.4.1-5. Geometry Closest Pair CPU utilization for worker node

Figure 4.4.1-6 shows its memory utilization.

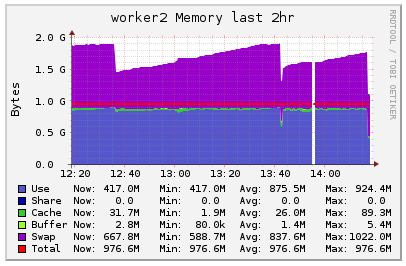


Figure 4.4.1-6. Geometry Closest Pair memory utilization for worker node

Figures 4.4.1-7 and 4.4.1-8 show that the communication cost is relatively stable except for 4 peaks. The average communication cost is 364.1k and the peak communication cost is more than 750k which appeared at time 13:24. Compared to the other operations, the communication cost is not as large.

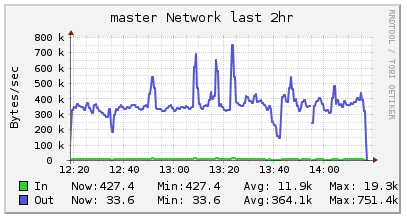


Figure 4.4.1-7. Geometry Closest Pair network utilization for master node

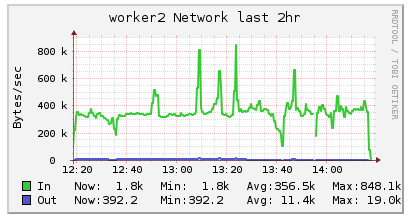


Figure 4.4.1-8. Geometry Closest Pair network utilization for worker node

### Spark Web UI Statistics for 1 Machine

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 1 machine. The list of completed applications has three different dataset sizes tested for a single machine: Group2-ClosestPair-2, Group2-ClosestPair-3, and Group2-ClosestPair-4. The number at the end of the application name, such as the “2” in “Group2-ClosestPair-2” indicates the dataset used contained 10^2 rows.

Figure 4.4.2-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Although this operation was not tested with as many datasets as the other operations, the general trend appears to be the same: as the size of the dataset size increases, the runtime increases as well. The increase was little between the 10^2 and 10^3 datasets compared to the increase between the 10^3 and 10^4 datasets. Unlike the other operations, it would appear that the Geometry Closest Pair operation does not perform as well for datasets larger than 10^3 rows. The runtime becomes much slower more easily.

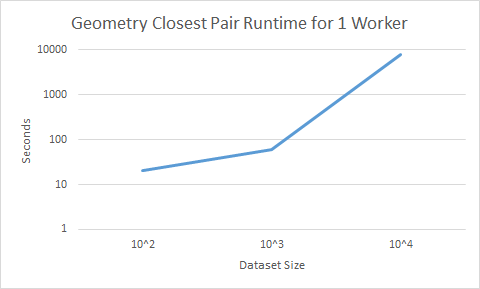


Figure 4.4.2-2. Geometry Closest Pair runtime for different dataset sizes on 1 machine

### Spark Web UI Statistics for 2 Machines

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 2 machines. The list of completed applications has three different dataset sizes tested for a single machine: Group2-ClosestPair-1, Group2-ClosestPair-2, and Group2-ClosestPair-3. The number at the end of the application name, such as the “1” in “Group2-ClosestPair-1” indicates the dataset used contained 10^1 rows.

Figure 4.4.3-2 is a graph of the runtime for different dataset sizes on a 2 machines. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. The same observation can be made about this operation as was made with the previous operations: the overall shape appears to be more linear, and the line is lower, meaning that the runtime for the Geometry Closest Pair operation is faster on 2 machines than it is on 1 machine. The runtimes of all three datasets are significantly lower on 2 machines--for example, the 10^3 dataset took 7920 seconds to finish on 1 machine; it only took 34 seconds to finish on 2 machines.

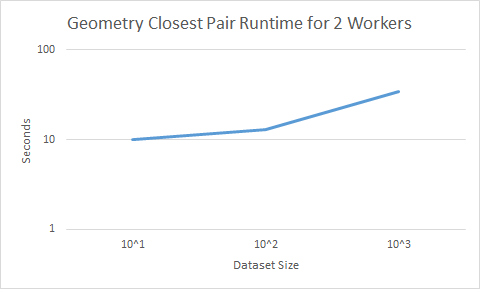


Figure 4.4.3-2. Geometry Closest Pair runtime for different dataset sizes on 2 machines

## Spatial Range Query

### Ganglia Web UI Statistics

The following are statistics obtained from the Ganglia Web UI for the master node:

Figure 4.5.1-2 shows its CPU utilization.

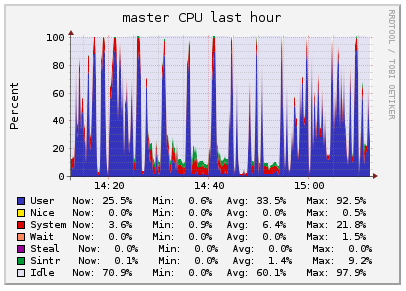


Figure 4.5.1-2. Spatial Range Query CPU utilization for master node

Figure 4.5.1-3 shows its memory utilization.

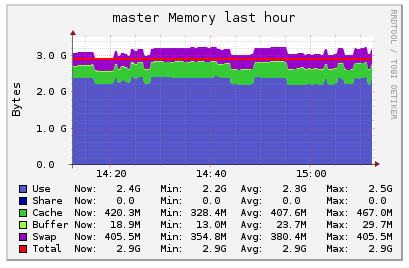


Figure 4.5.1-3. Spatial Range Query memory utilization for master node

The following are statistics obtained from the Ganglia Web UI for the worker node:

Figure 4.5.1-5 shows its CPU utilization. There are several thin peaks between 15:00 and 15:09 that correspond to the different sizes of datasets that were tested. As the datasets increased in size, the peaks grew slightly taller, indicating that the CPU utilization also increased. Overall, the Spatial Range Query operation does not require as much CPU as the other operations; the max CPU utilization reached around 60 percent.

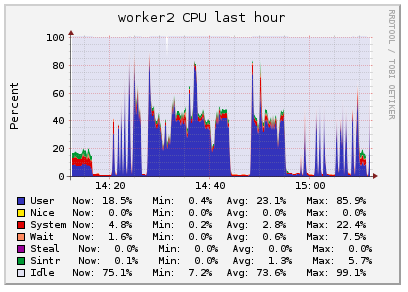


Figure 4.5.1-5. Spatial Range Query CPU utilization for worker node

Figure 4.5.1-6 shows its memory utilization. Between 15:00 and 15:09, the memory utilization appears relatively stagnant until a sudden peak towards the end. The Spatial Range Query operation seems to use roughly the same amount of memory for smaller datasets but requires more memory for larger datasets.

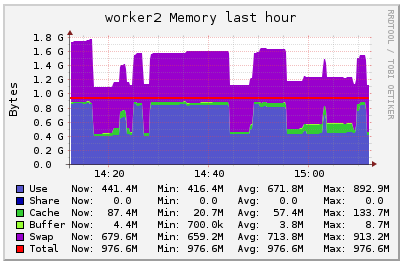


Figure 4.5.1-6. Spatial Range Query memory utilization for worker node

In Figures 4.5.1-7 and 4.5.1-8, the communication cost of the Spatial Range Query operation is quite small (less than 0.1M). There are only 3 small peaks on the timeline between 15:01 to 15:09.

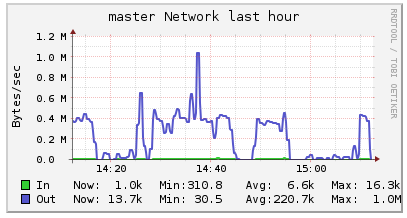


Figure 4.5.1-7. Spatial Range Query network utilization for master node

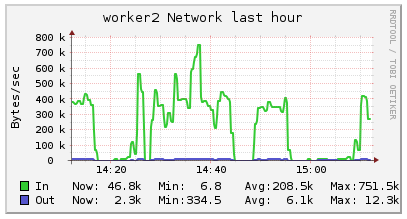


Figure 4.5.1-8. Spatial Range Query network utilization for worker node

### Spark Web UI Statistics for 1 Machine

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 1 machine. The list of completed applications has six different dataset sizes tested for a single machine: Group2-RangeQuery-1, Group2-RangeQuery-2, Group2-RangeQuery-3, Group2-RangeQuery-4, Group2-RangeQuery-5, and Group2-RangeQuery-6. The number at the end of the application name, such as the “1” in “Group2-RangeQuery-1” indicates the dataset used contained 10^1 rows.

Figure 4.5.2-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. This graph also illustrates the same trend that has been seen in the previous operations: as the dataset size increased, the runtime also increased. The increase is small between the 10^2 and 10^3 datasets and gradually grows larger between the 10^3 and 10^5 datasets. There is a sharper increase between the 10^5 and 10^6 datasets. This means the Spatial Range Query operation most likely runs the fastest for datasets with fewer than 10^3 rows, and its limit is reached at 10^5 rows before experiencing a significant slowdown.

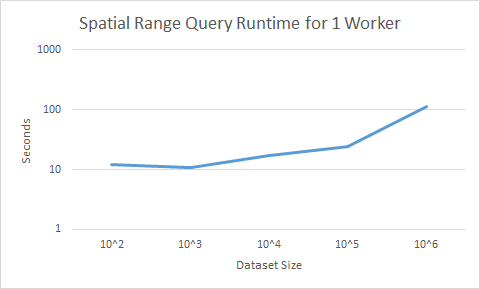


Figure 4.5.2-2. Spatial Range Query runtime for different dataset sizes on 1 machine

### Spark Web UI Statistics for 2 Machines

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 2 machines. The list of completed applications has six different dataset sizes tested for a single machine: Group2-RangeQuery-1, Group2-RangeQuery-2, Group2-RangeQuery-3, Group2-RangeQuery-4, Group2-RangeQuery-5, and Group2-RangeQuery-6. The number at the end of the application name, such as the “1” in “Group2-RangeQuery-1” indicates the dataset used contained 10^1 rows.

Figure 4.5.3-2 is a graph of the runtime for different dataset sizes on 2 machines. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Overall, the graph is similar to the one for 1 machine, except now it is flatter and lower, indicating that the runtime for the Spatial Range Query operation is also faster on 2 machines than it is on 1 machine. The flatter shape might indicate this operation is more scaleable. Increasing the dataset size does not significantly impact its runtime on 2 machines, but increasing the number of machines does significantly reduce the runtime.

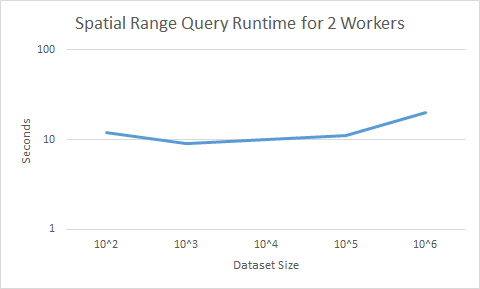


Figure 4.5.3-2. Spatial Range Query runtime for different dataset sizes on 2 machines

## Spatial Join Query

### Ganglia Web UI Statistics

The following are statistics obtained from the Ganglia Web UI for the master node:

As it shows in Figure 4.6.1-2, after 17:25, 15 processes were created for the operation.

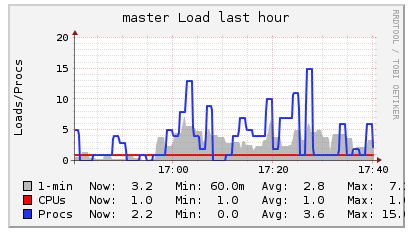


Figure 4.6.1-2. Spatial Join Query load utilization for master node

Figure 4.6.1-3 shows its CPU utilization.

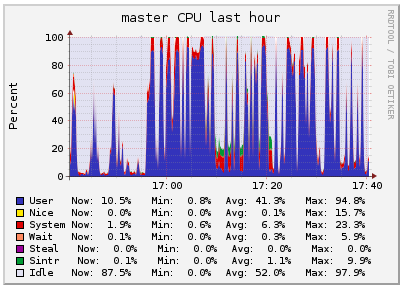


Figure 4.6.1-3. Spatial Join Query CPU utilization for master node

Figure 4.6.1-4 shows its memory utilization.

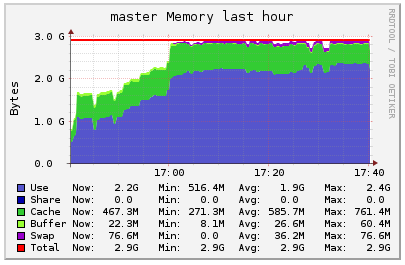


Figure 4.6.1-4. Spatial Join Query memory utilization for master node

The following are statistics obtained from the Ganglia Web UI for the worker node:

Figure 4.6.1-6 shows its CPU utilization. Between 17:26 and 17:31, there are several peaks corresponding to the different dataset sizes that were tested. Overall, the trend appears to be the same--as the dataset size increased, the CPU utilization also increased. The Spatial Join Query operation seems to require more CPU than the Spatial Range Query operation because the maximum CPU utilization is around 80 percent.

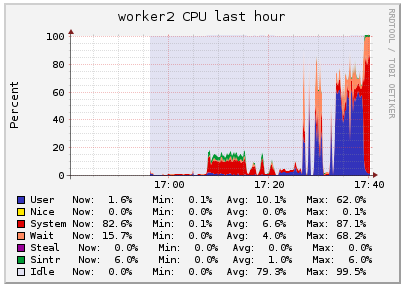


Figure 4.6.1-6. Spatial Join Query CPU utilization for worker node

Figure 4.6.1-7 shows its memory utilization. There is one small peak and one larger peak between 17:26 and 17:31. The smaller peak corresponds to smaller dataset sizes, which required less memory, while the larger peak corresponds to larger dataset sizes, which required more memory.

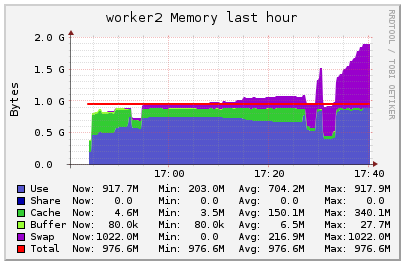


Figure 4.6.1-7. Spatial Join Query memory utilization for worker node

From Figures 4.6.1-8 and 4.6.1-9, the communication cost is more than 0.1M but less than 0.2M. The communication is also relatively small compared to other operations.

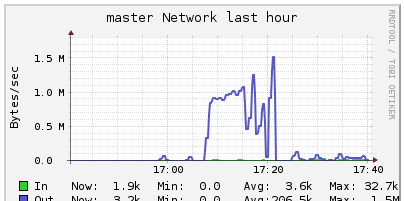


Figure 4.6.1-8. Spatial Join Query network utilization for master node

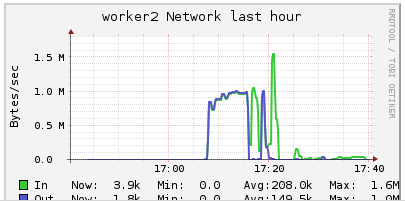


Figure 4.6.1-9. Spatial Join Query network utilization for worker node

### Spark Web UI Statistics for 1 Machine

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 1 machine. The list of completed applications has four different dataset sizes tested for a single machine: Group2-JoinQuery-1, Group2-JoinQuery-2, Group2-JoinQuery-3, and Group2-JoinQuery-4. The number at the end of the application name, such as the “1” in “Group2-JoinQuery-1” indicates the dataset used contained 10^1 rows.

Figure 4.6.2-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Like the other operations, as the dataset size increased, the runtime increased as well. However, the Spatial Join Query operation appears to increase in runtime quicker than the Spatial Range Query operation.

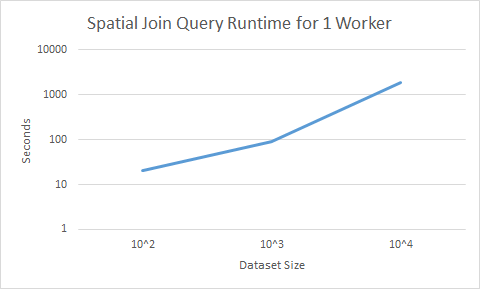


Figure 4.6.2-2. Spatial Join Query runtime for different dataset sizes on 1 machine

### Spark Web UI Statistics for 2 Machines

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 2 machines. The list of completed applications has four different dataset sizes tested for a single machine: Group2-JoinQuery-1, Group2-JoinQuery-2, Group2-JoinQuery-3, and Group2-JoinQuery-4. The number at the end of the application name, such as the “1” in “Group2-JoinQuery-1” indicates the dataset used contained 10^1 rows.

Figure 4.6.3-2 is a graph of the runtime for different dataset sizes on a 2 machines. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. Compared to the graph for 1 machine, the shape of this graph for 2 machines looks very similar. However, the overall height is much lower. This shows that the Spatial Join Query operation runs much faster on 2 machines than 1 machine. For instance, the 10^4 dataset took 1860 seconds to complete on 1 machine, but it only took 660 seconds on 2 machines--2 machines reduced the runtime to almost 30 percent.

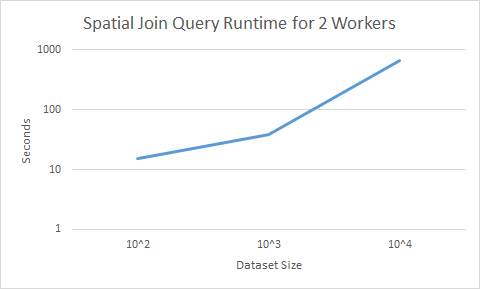


Figure 4.6.3-2. Spatial Join Query runtime for different dataset sizes on 2 machines

## Spatial Aggregation Application

### Ganglia Web UI Statistics

The following are statistics obtained from the Ganglia Web UI for the master node:

In Figure 4.7.1-2, processes are created after each job is submitted. At time point 18:03, 18 processes are created and it reaches the peak.

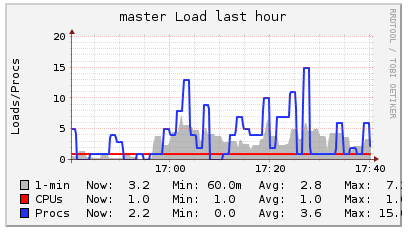


Figure 4.7.1-2. Spatial Aggregation Application load utilization for master node

Figure 4.7.1-3 shows its CPU utilization.

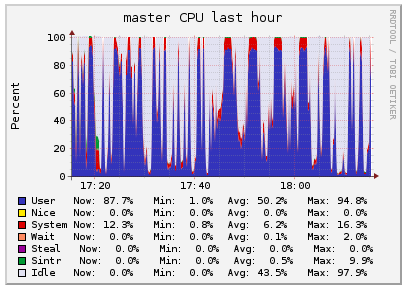


Figure 4.7.1-3. Spatial Aggregation Application CPU utilization for master node

Figure 4.7.1-4 shows its memory utilization.

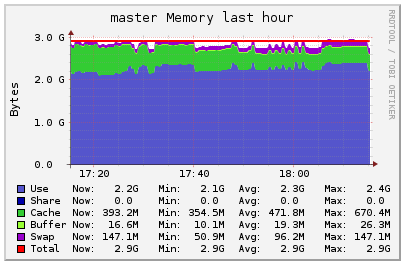


Figure 4.7.1-4. Spatial Aggregation Application memory utilization for master node

The following are statistics obtained from the Ganglia Web UI for the worker node:

Figure 4.7.1-6 shows its CPU utilization. From 17:48 to 18:05, there are several peaks which are mostly the same height until it spikes higher at the end. This indicates that the Spatial Aggregation Application requires roughly the same amount of CPU for datasets with a size in a certain range, but it requires a lot more CPU for datasets with a size greater than the upper bound of that range.

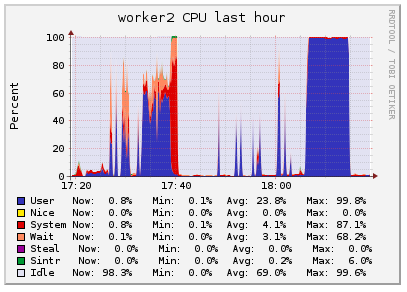


Figure 4.7.1-6. Spatial Aggregation Application CPU utilization for worker node

Figure 4.7.1-7 shows its memory utilization. There are not many peaks between 17:48 and 18:05, which seems to indicate that memory usage is roughly the same for smaller datasets and slowly increases for larger datasets.

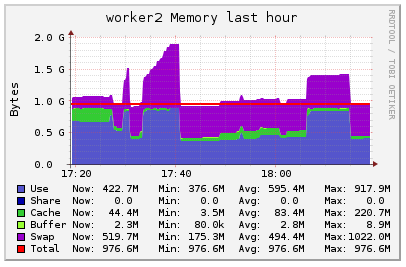


Figure 4.7.1-7. Spatial Aggregation Application memory utilization for worker node

From Figures 4.7.1-8 and 4.7.1-9, the communication cost of different sizes of datasets is also clear. As the dataset increases in size, the communication cost also increases. At 18:07, the communication cost reaches its peak of 0.2 M, which corresponds to the largest dataset tested.

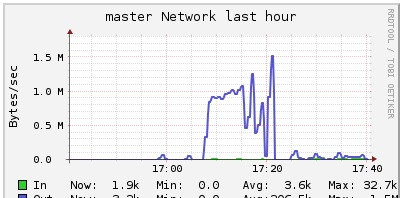


Figure 4.7.1-8. Spatial Aggregation Application network utilization for master node

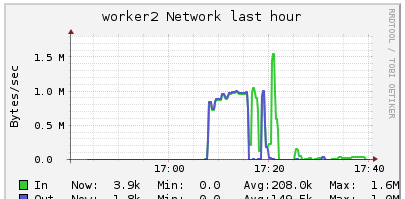


Figure 4.7.1-9. Spatial Aggregation Application network utilization for worker node

### Spark Web UI Statistics for 1 Machine

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 1 machine. The list of completed applications has five different dataset sizes tested for a single machine: Group2-Aggregation-1, Group2-Aggregation-2, Group2-Aggregation-3, Group2-Aggregation-4, and Group2-Aggregation-5. The number at the end of the application name, such as the “1” in “Group2-Aggregation-1” indicates the dataset used contained 10^1 rows.

Figure 4.7.2-2 is a graph of the runtime for different dataset sizes on a single machine. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. There is very little change in runtime between the 10^1 and 10^3 datasets; then there is a very small change between the 10^3 and 10^4 datasets followed by a more drastic increase between the 10^4 and 10^5 datasets. This seems to indicate that the Spatial Aggregation Application runs the fastest for datasets with fewer than 10^4 rows; for datasets with more than 10^4 rows, it starts to slow down significantly.

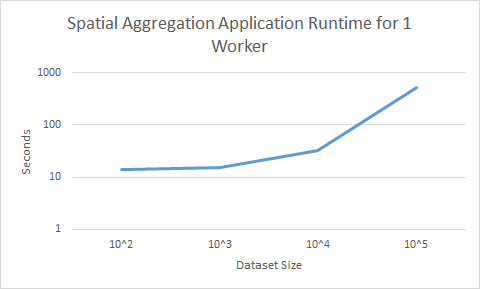


Figure 4.7.2-2. Spatial Aggregation Application runtime for different dataset sizes on 1 machine

### Spark Web UI Statistics for 2 Machines

The Spark Web UI has statistics showing the runtimes for different dataset sizes on 2 machines. The list of completed applications has five different dataset sizes tested for a single machine: Group2-Aggregation-1, Group2-Aggregation-2, Group2-Aggregation-3, Group2-Aggregation-4, and Group2-Aggregation-5. The number at the end of the application name, such as the “1” in “Group2-Aggregation-1” indicates the dataset used contained 10^1 rows.

Figure 4.7.3-2 is a graph of the runtime for different dataset sizes on a 2 machines. The runtime is measured in seconds, while the dataset sizes represent the number of rows in the input. For the Spatial Aggregation Application, there does not appear to be as much of a difference in runtime for datasets below 10^4 rows. However, 2 machines were able to finish a dataset with a size of 10^5 rows in 258 seconds compared to the 522 seconds taken for 1 machine to finish. This observation might indicate that adding more machines does not have a significant impact on the application runtime for small datasets, but it does have a significant impact on the application for large datasets.

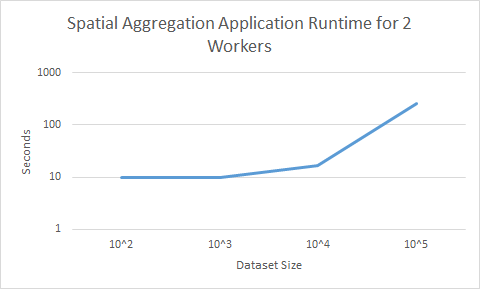


Figure 4.7.3-2. Spatial Aggregation Application runtime for different dataset sizes on 2 machines

## Spark Web UI Statistics for 4 Machines

The Spark Web UI has statistics showing the runtimes for all six geospatial operations and the spatial aggregation application on 4 machines. The largest dataset sizes from the previous experiments were used in this experiment. Thus, Geometry Union was tested with a dataset size of 10^6 rows. Geometry Convex Hull was tested with a dataset size of 10^7 rows. Geometry Farthest Pair was tested with a dataset size of 10^7 rows. Geometry Closest Pair was tested with a dataset size of 10^3 rows. Spatial Range Query was tested with a dataset size of 10^6 rows. Spatial Join Query was tested with a dataset size of 10^4 rows. And the Spatial Aggregation Application was tested with the provided datasets.

Figure 4.8-2 is a graph comparing the runtimes between all the geospatial operations and spatial aggregation application on 4 machines using the datasets described previously. Compared to the runtimes on 2 machines, almost all the operations appear to run much faster on 4 machines. Many of the runtimes have been reduced to more than half. For example, the dataset with 10^7 rows for the Geometry Union operation took 2 machines 150 seconds to complete, but it took 4 machines 66 seconds to complete; this is a difference of 84 seconds. However, this difference is not as drastic as the difference in runtimes between 1 machine and 2 machine--1 machine took 300 seconds to complete the same dataset size, resulting in a difference of 150 seconds. This observation seems to indicate that while adding more machines does decrease runtime, the decrease is not proportional to the number of machines.

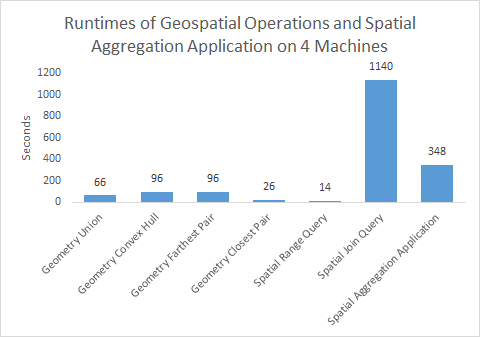


Figure 4.8-2. Geospatial operations and spatial aggregation application runtimes for largest datasets on 4 machines

# CONCLUSION

Six geospatial operations--Geometry Union, Geometry Convex Hull, Geometry Farthest Pair, Geometry Closest Pair, Spatial Range Query, and Spatial Join Query--plus a spatial aggregation application were implemented in this project. These operations and application were implemented using Java and Apache Hadoop and Spark and were tested on clusters that consists of 4 worker nodes.

Experimental analyses were performed to primarily measure CPU utilization, memory utilization, network traffic and overall runtime, while increasing the size of the dataset and number of machines. Several observations were shared among all the operations and the application. As the dataset size increased, the CPU utilization, memory utilization, communication costs, and runtime also increased. Additionally, as the number of machines increased, the runtime decreased. We thought there were possibly two reasons. The first one was because of the overhead of communication between worker nodes. The second one was that sometimes the faster nodes will wait for the slower nodes to finish.

An interesting observation made throughout the experiments was that if one worker node is far more powerful than the other worker nodes, it will sometimes process more tasks and therefore reduce the total runtime by more than half. For example, when using a single worker node to process the Geometry Farthest Pair operation on a test data of size 10^7, the overall runtime was 16 minutes. After adding a node running on a powerful machine, the runtime reduced to 2.5 minutes. Standing in the perspective of the interaction between the Hadoop job tracker, task tracker and scheduler, we thought that this is because of the heartbeats that task trackers send to the job tracker on master node at intervals. In the heartbeat, it indicates if a slave (worker) node can take new tasks for execution. Then the job tracker consults the scheduler to assign tasks to the task tracker and sends the list of tasks as part of the heartbeat response to the task tracker. So when one worker node is far more powerful than the other worker nodes, it finishes its task assigned by job tracker much earlier than other nodes and therefore the job tracker will assign more tasks for it to do. Since the average speed for executing all tasks is increased, the total runtime is reduced. This phenomenon is more obvious when the dataset has a relatively large size.

From the test, we had a conclusion that our functions were implemented with good scalability. When the size of the input increased, with fixed number of worker nodes, the running time increased almost proportionally. For the same amount of test data, when the number of worker nodes increased, the running time decreased proportionally.

One improvement that can be done in the future is that we can apply the gridding techniques we used in the spatial aggregation application to other operations such as Geometry Closest Pair and Spatial Join Query. Some future work includes code optimization to achieve better local performance, as well as some tuning of the Spark cluster to better fit the hardware and the test data.