NSL SPINDLE
MAP-REDUCE AT THE EDGE IN A V2V ENVIRONMENT

By

William Rory Kronmiller

A Dissertation Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Examining Committee:

_____

Dr. Stacy Patterson, Dissertation Adviser

Rensselaer Polytechnic Institute
Troy, New York

March 3
(For Graduation May 2017)

CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

ACKNOWLEDGMENT vi

ABSTRACT vii

1. INTRODUCTION 1

2. Background and Related Work 3

2.1 Connected Vehicles 3

2.1.1 Vehicle Clusters 3

2.2 Cloud Data Processing Technologies 3

2.2.1 Spark Streaming 3

2.2.2 Kafka 4

2.3 Other Software 4

2.3.1 Akka 4

2.3.2 Postgres 4

3. System Architecture 5

3.1 Theorized Architecture 5

3.2 Simulation Architecture 5

4. Experiments 9

4.1 Data Sets 9

4.1.1 Vehicle Traces 9

4.1.2 Test Regions 9

4.1.3 Window Sizes 11

4.1.4 Test Clusters 12

4.1.5 Clustering Algorithms 12

4.2 Test Software 12

4.3 Test Configurations 14

4.3.1 Example Map/Reduce Programs 14

4.4 Simulator Results 14

5. Conclusion 19
5.1 Future Work 19
LITERATURE CITED 19
APPENDICES
A. THIS IS AN APPENDIX 21
A.1 A Section Heading 21
B. THIS IS ANOTHER APPENDIX 22

LIST OF TABLES

LIST OF FIGURES

3.1 A diagram of the theorized system's major components. Vehicles send data to their Cluster Heads, which perform reduce operations over incoming mapped tuples. The resultant reduced tuples are then sent to the middleware to be distributed to the connected Spark jobs. 6

3.2 A sequence diagram showing how a query passes from a connected Spark job through the middleware and vehicle systems and how the reduced results then return to the Spark job. 7

3.3 A sequence diagram showing the high-level interactions of the Spindle simulator components. A Postgres database stores the simulation configurations and pre-computed velocity and connectivity data, as well as simulation results. 8

4.1 Illustration of VEINS Vehicles with MANET Clustering, Courtesy Xiaotian Duan 10

4.2 A Snapshot of Active Vehicles in VEINS Traces with Selected Regions Highlighted 11

4.3 The total number of bytes sent to the middleware, averaged over multiple trials for the speedSum map/reduce job. The x-axis contians different configurations of regions, clusterings, and reduce window sizes (10, 15, and 30 seconds). Smaller values in the same geographic region indicate better performance, where the best performance comes from using clustering with a 30 second window size. 15

4.4 shows the scaled number of bytes sent to the middleware in a given region and window size configuration where y=1 indicates the number of bytes sent without clustering for a given region and reduce window size. The figure demonstrates the data savings that occur as a result of using vehicle clusters. The figure also illustrates how data reduction is affected by regional vehicle density - areas of high vehicle density can take better advantage of clustering to get more savings. 16

4.5 shows the total number of bytes sent to the middleware, averaged over multiple trials for the geoMapped map/reduce job. The x-axis contians different configurations of regions, clusterings, and reduce window sizes (10, 15, and 30 seconds). Despite being a slightly different map/reduce query, the performance of the geoMapped tests is similar to that of the speedSum tests, where the best performing configuration is the clustered configuration with a 30-second reduce window size. 17

4.6 shows the scaled number of bytes sent to the middleware in a given region and window size configuration where y=1 indicates the number of bytes sent without clustering for a given region and reduce window size. The figure shows a similar reduction in data transfer by using clusters to that of the speedSum map/reduce job. 18

ACKNOWLEDGMENT

The acknowledgment text goes here. Unlike chapter headings, this heading is not numbered.

ABSTRACT

Write your abstract here. Again, the heading does receive a number.

1. INTRODUCTION

Spindle serves as a scalable hybrid vehicle-to-vehicle and vehicle-to-internet architecture for efficient near-real-time processing of streaming sensor data from network-connected vehicles. According to a 2015 Hitachi white-paper, some contemporary vehicles produce as much as 25 gigabytes of data every hour with prototype vehicles fitted with "cameras and additional sensors" producing up to 250 gigabytes per hour [8]. At the same time, work is underway to develop and test a variety of communications systems for connecting vehicles to one another and to road infrastructure and the internet [12]. Spindle addresses the problem of managing the vast quantities of available vehicle data by applying the map/reduce [9] paradigm to logical clusters of interconnected vehicles. In particular, Spindle exploits edge-computation by exchanging data over vehicle-to-vehicle communications systems in order to reduce the amount of data that must be sent over the internet to the cloud. Spindle is designed to support the use case of a developer or analyst writing a streaming map/reduce program that can then be deployed to the cloud and distributed to clusters of vehicles; vehicles in each cluster can apply the map operation to their incoming data streams and send the map outputs to a single leader (Clusterhead); the Clusterhead then applies the reduce operation to a buffer of incoming mapper outputs and sends the result of the reduce operation to the cloud, which forwards the data to the user's client program where the data can be displayed or further analyzed.

Our contribution is to demonstrate the bandwidth savings of performing map/reduce at the edge, in vehicle networks, by implementing a subset of the Spindle architecture inside a simulation framework. We implement the data Spindle processing pipeline for vehicles and Clusterheads, which we then integrate with a system that provides realistic mobility and connectivity data, such that we can simulate vehicles moving through space, forming clusters, processing map/reduce queries, and producing outputs to send to the cloud. We instrument the simulator to measure the number of bytes outputby simulated Clusterheads, destined for the cloud. We work

in collaboration with Mike Wittie of Montana State University to obtain realistic position and connectivity information for vehicles in the simulation. We also work in collaboration with Xiaotian Duan, from whose work we get vehicle-to-cluster assignments over time for our simulation. In evaluating our experiments, we focus in particular on the effect of different reducer window sizes on the amount of data sent to the cloud; in so doing, we explore the trade-off between resolution and latency versus bandwidth consumption. Having a smaller reducer window means that reduced messages are sent more frequently and client programs are given more data points to analyze, while having a larger reducer window size means more messages can be batched together for reduction before data is transmitted to the cloud. We also explore the effects of regions with differing vehicle densities on bandwidth savings; regions of space that have a higher density of vehicles offer the opportunity to form larger clusters and to reduce the total number of clusters relative to the total number of vehicles present; a smaller number of clusters equates to a smaller number of Clusterheads transmitting data at any given time window, which has an effect on overall bandwidth usage.

2. Background and Related Work

2.1 Connected Vehicles

2.1.1 Vehicle Clusters

2.2 Cloud Data Processing Technologies

The past few years have seen a number of critical developments in IT infrastructure and data processing software that, combined, provide a formidable platform for extracting useful information from vast quantities of data at scale.

Amazon Web Services Amazon Web Services (AWS) is a set of "Cloud Computing" offerings from Amazon.com that provide access to elastic computation and storage resources on an "on-demand" basis [10]. One particularly valuable service offered is EC2, which provides access to collections of virtual machines running on Amazon's cloud. These EC2 virtual machines (instances) can be used to perform resource-intensive cluster computing operations without the need for fixed infrastructure on the part of the end-user. One way in which a user can purchase access to EC2 instances is through Spot Requests, wherein the user places a bid for the maximum amount s/he is willing to pay per hour for a given EC2 instance configuration. If the user is not outbid, then the user is given access to an instance of the specified configuration. As soon as the user is outbid (within approximately 2 minutes), the user's EC2 instance is shut down and its local data is lost. Spot Requests provide very low cost access to computing resources at the cost of reliability.Finally of note, Amazon provides an object storage layer - one can associate a key string with some binary blob that is stored in a replicated file system - called S3.

2.2.1 Spark Streaming

One framework that is capable of processing large quantities of data in a distributed fashion, such as on a cluster of EC2 instances, is Apache Spark. Spark offers a micro-batch package for processing streaming data called Spark Streaming. Spark Streaming allows a developer (or other analyst) to manipulate time-bounded

batches of streaming data tuples by writing a program that performs immutable transformations such as map, filter, and reduce on a Spark Streaming abstraction called a DStream. Spark is able to take the client program, break it up into "stages" in which all processing across tuples can occur in parallel, then partition each stream and stage across nodes to be processed in parallel. In sum, Spark Streaming provides a computing framework and a set of abstractions which allow a developer to write map/reduce-style programs for streaming data with relative ease [11].

2.2.2 Kafka

2.3 Other Software

2.3.1 Akka

2.3.2 Postgres

3. System Architecture

Spindle provides a novel streaming data processing platform by building on existing cutting edge and industry standard distributed systems, primarily from the Apache ecosystem.

The Spindle system consists of three major components: a set of Apache Spark streaming programs managed by one or more clients, a custom data ingestion and query management Middleware, and edge computing software running on network connected vehicles.

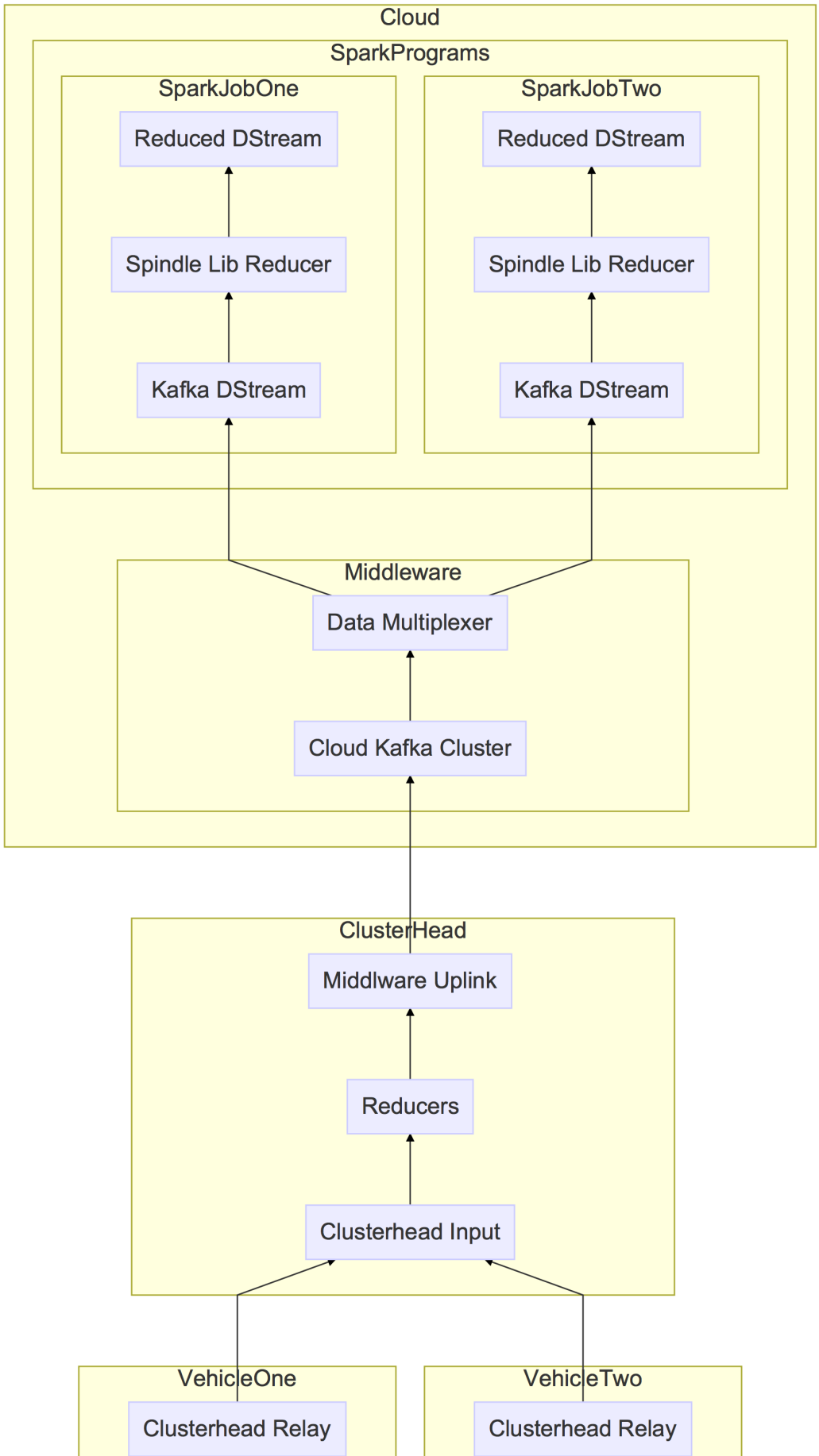
Spindle operates on the vehicle level as a Scala [4] program running on the Akka [5] framework; this program is responsible for handling cluster formation, data collection, and edge computation.

Data passes through components of the Vehicle software in the form of messages sent to Kafka [6] topics. Similarly, data is transferred at the cloud layer using Kafka.

Clients receive data from the middleware as Kafka messages that a custom Spark [7] library processes into DStream tuples which can be further processed using any of the available Spark streaming operations written for DStreams and RDDs.

3.1 Theorized Architecture

3.2 Simulation Architecture


Cloud
SparkPrograms
SparkJobOne
Reduced DStream
Spindle Lib Reducer
Kafka DStream
SparkJobTwo
Reduced DStream
Spindle Lib Reducer
Kafka DStream
Middleware
Data Multiplexer
Cloud Kafka Cluster
ClusterHead
Middlware Uplink
Reducers
Clusterhead Input
VehicleOne
Clusterhead Relay
VehicleTwo
Clusterhead Relay

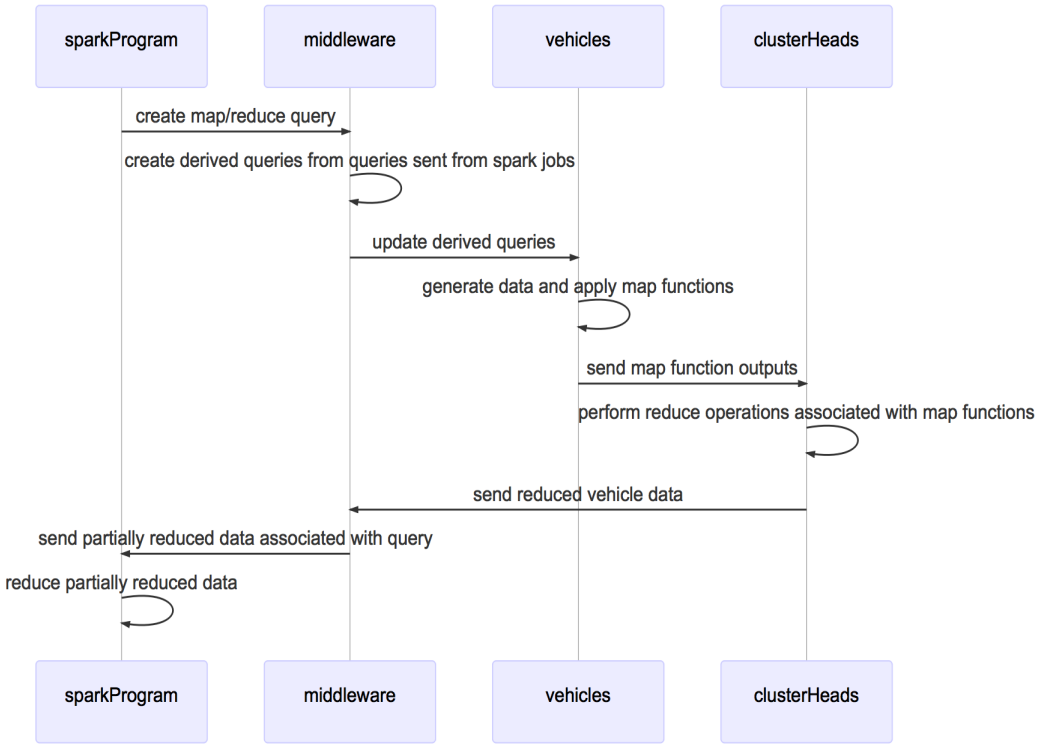


Figure 3.2: A sequence diagram showing how a query passes from a connected Spark job through the middleware and vehicle systems and how the reduced results then return to the Spark job.

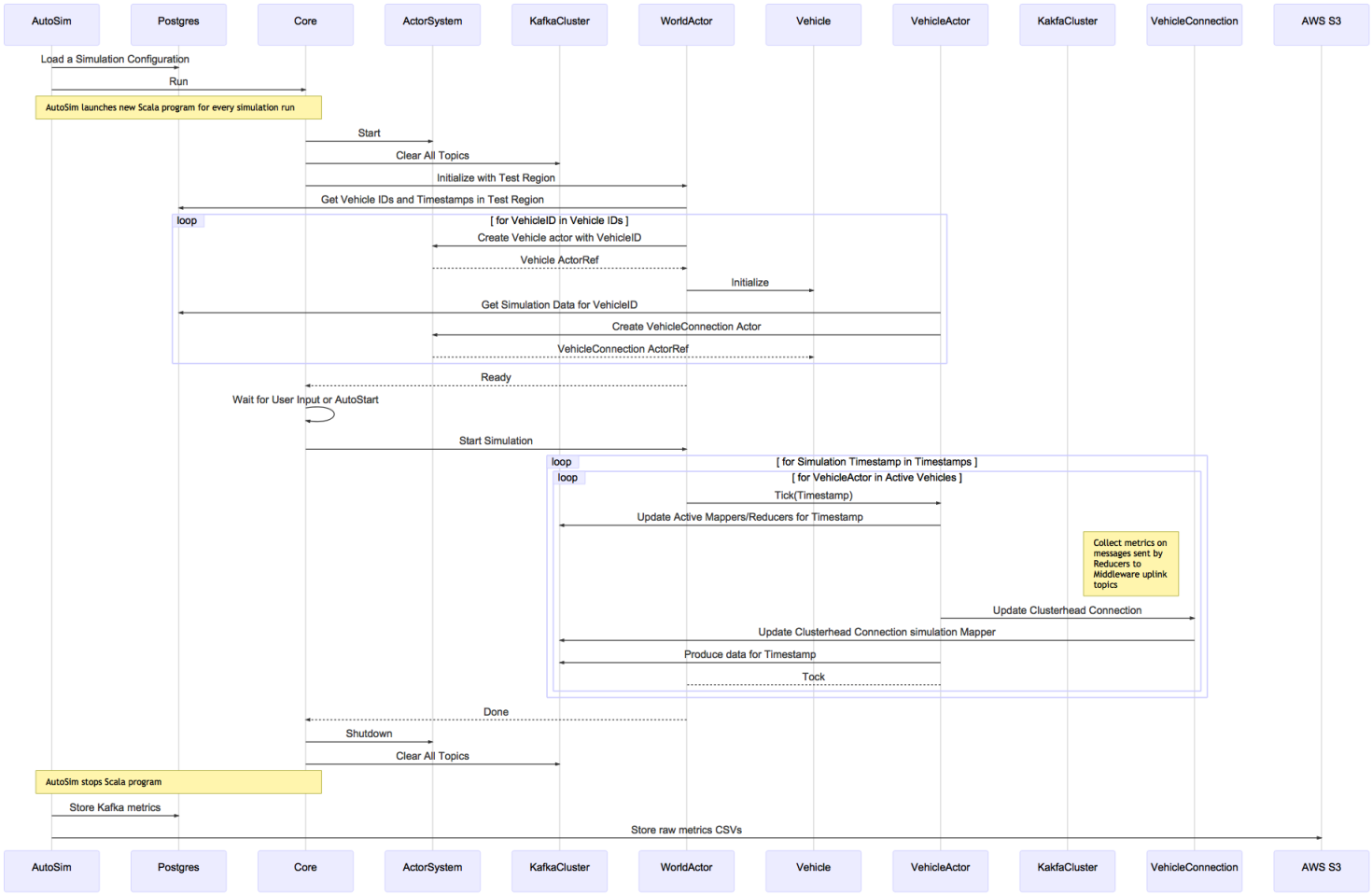


Figure 3.3: A sequence diagram showing the high-level interactions of the Spindle simulator components. A Postgres database stores the simulation configurations and pre-computed velocity and connectivity data, as well as simulation results.

4. Experiments

4.1 Data Sets

4.1.1 Vehicle Traces

The Spindle experiments depend on a set of vehicle traces generated using "the open source vehicular network simulation framework" VEINS, which simulates vehicle movement and radio connectivity in a "realistic" manner [3]. The VEINS traces were generated by Mike Wittie of the Montana State University as part of a larger overall research project. The traces come in the form of sharded CSV files containing time-series mappings of three different types: speed, x-position, y-position, and connectivity to other vehicles. Each time-series mapping includes a timestamp, a vehicle uid, and a value (speed, position, reachable-vehicle-id). These shards have been parsed into a Postgres database with a table for each mapping type.

4.1.2 Test Regions

Original VEINS Traces The original VEINS traces span a region from $(41735, 85178.4)$ to $(63071.5, 114426)$ and includes 3340 distinct vehicles. This dataset includes more vehicles than the current simulator implementation can handle due to the large number of threads, network connections, and Kakfa topics that would be required (the thread/topic count scales with the number of vehicles multiplied by a constant multiple of the average number of map/reduce queries running on each node). A real-world deployment would distribute the threads/topics over the set of vehicles in the system such that each vehicle and kafka cluster would have threads and topics proportional to the number of active map/reduce queries. In essence, this is a temporary limitation of the simulation architecture, not the real-world architecture.

To overcome the simulator's limitations, we defined two test regions, one dense region and one sparse region being transited by approximately the same number of vehicles. The sparse and dense regions were selected based on the number of vehicles transiting them and based on the layout of the underlying road networks.

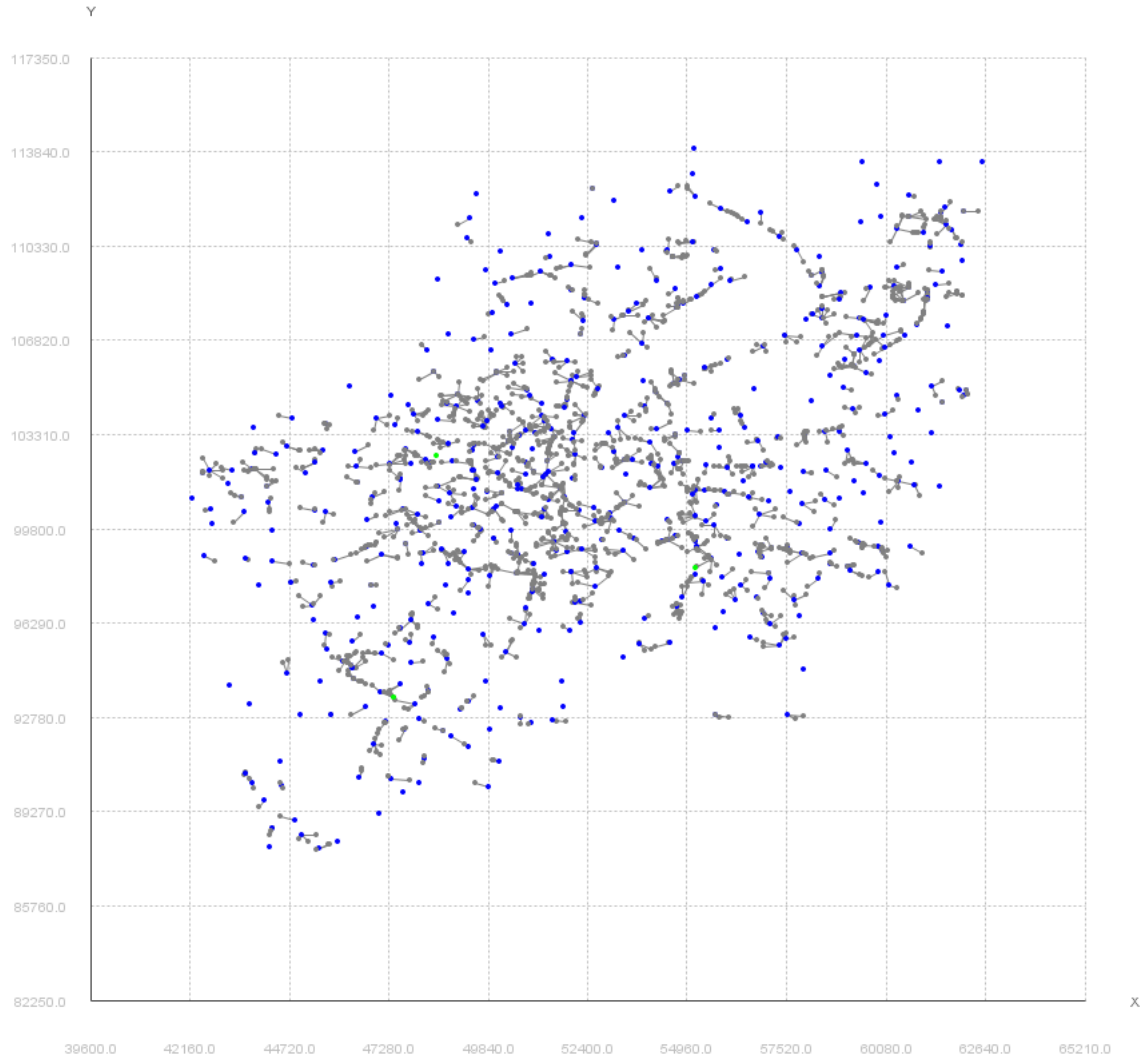


Figure 4.1: Illustration of VEINS Vehicles with MANET Clustering, Courtesy Xiaotian Duan

Sparse Region The sparse region is transited by 148 distinct vehicles over the course of the simulation and defines a bounding box from $(40000, 106800)$ to $(53000, 114000)$. The sparse region covers an area on the outskirts of the city, with a relatively low density of vehicles. The sparse region is illustrated by a purple box in 4.2.

Dense Region The dense region defines a bounding box from $(50000, 100000)$ to $(51000, 102000)$ and is transitied by 131 vehicles over the course of the simulation. The dense region covers a small area near the city center where the density of vehicles

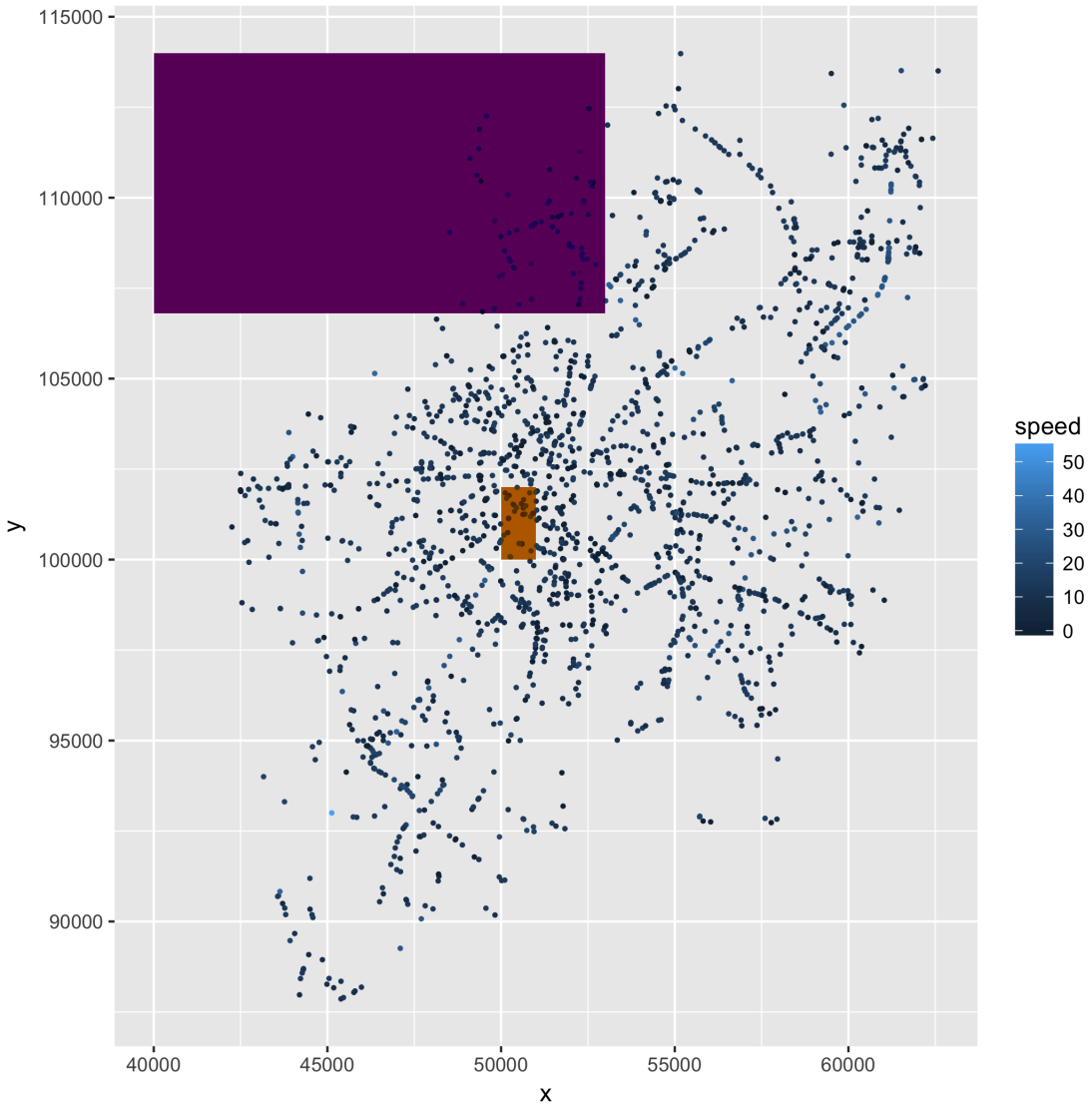


Figure 4.2: A Snapshot of Active Vehicles in VEINS Traces with Selected Regions Highlighted

is higher. The dense region is illustrated by an orange box in 4.2.

4.1.3 Window Sizes

Kafka Streams can perform reduce operations over messages received over the course of a user-defined time-window and will write the results to a KTable, a logical structure whose API approximates that of a key-value store. Spindle's Kafka Streams reducer adds a `Batcher`, which transmits the final value of a time-window's KTable entry to some destination Kafka topic (in this case the middleware). As such,

Spindle supports micro-batch operations on Kafka streams with user-configurable window sizes. The sizes tested were 10, 15, and 30 seconds.

4.1.4 Test Clusters

The cluster head assignments for each node are pre-computed using either Xiaotian's clustering algorithm or some base-line `single_clusterhead` (all vehicles in simulation share a single clusterhead) or `self_clusters` (all vehicles act as their own cluster heads).

4.1.5 Clustering Algorithms

The spindle architecture takes advantage of work done by fellow RPI student Xiaotian Duan on generating clusters of vehicles based on vehicle connectivity and lane position.

4.2 Test Software

SpindleSim The Spindle simulator (SpindleSim) is implemented in several thousand lines of Scala and is implemented on top of the Akka actor model framework. The Scala/Akka portions of the simulator implement as realistically as is practical the software that would run on a real-world deployment of Spindle. This is accomplished by creating a separate Akka actor for each simulated vehicle, where each actor is given a cache of time-series simulation events and data and a connection to a "World" actor responsible for keeping track of global simulation time; this global time-keeper can be thought of as a stand-in for a GPS clock running on each vehicle. Each vehicle actor is then responsible for managing its publish-subscribe streams and messages, as well as its simulated connections to other vehicles in its cluster. Messages are exchanged between vehicles and send from vehicles to "the cloud" by way of special "Stream Relay" Kakfa Streams programs responsible for filtering expired and "Canary" messages and responsible for keeping a running sum of the number of bytes sent on a per-relay basis and logging these sums to separate CSV files at run-time.

AutoSim The Scala/Akka simulator (SpindleSim) currently has a number of pre-configured map/reduce jobs written that can be turned on or off using the program's `application.conf` file. A secondary program, dubbed AutoSim, runs the simulator iteratively, writing to the conf file, launching SpindleSim, then gathering the results and uploading them to a database and permanent storage. In order to iterate through the desired test configurations, AutoSim reads from a table `sim_configs_vx` in a Postgres 9.6.1 database hosted on an AWS EC2 `m4.large` reserved instance. AutoSim will choose among the least-tested configurations in the configs table, write the configuration to the SpindleSim `application.conf`, and launch an instance of SpindleSim. While SpindleSim is running, AutoSim greps the console logs for exception messages. If an exception is detected, the current SpindleSim instance is killed and a notification is sent via AWS SNS topic reporting the error message in order to facilitate debugging. If the SpindleSim operation completes, AutoSim uploads the CSV logs to an AWS S3 bucket and parses the final message byte sums from the CSV files then writes the results to a `sim_results_vx` table in Postgres. AutoSim is a simple Node.JS ES6 application that is run from the Babel-Node transpiler.

Environment AutoSim and SpindleSim are packaged inside a docker image, `wkronmiller/nsl-spindle-simulator` that can be run from a laptop, desktop, or EC2 instance. The test framework and container are designed to survive the total loss of local storage and/or the running simulator container by storing the results of all completed simiulation operations on S3 and a remote database. This design decision allows the test framework to be run on an AWS EC2 Spot Fleet which offers discounts of roughly 70-90% in exchange for the requirement that any software running on a Spot instance be designed to killed at any time with little or no warning. The architecture also enables trivial transfer of the simulator across different machines, easing debugging and mitigating problems related to transient EC2 network problems.

4.3 Test Configurations

4.3.1 Example Map/Reduce Programs

speedSum The most simple map/reduce operation tested is the "speedSum" job, which simply maps each vehicle's available data to a tuple containing only the vehicle's speed. The reduce operation run at the clusterheads takes the sum of the speeds of each of the cluster member vehicles.

geoFiltered This map/reduce job extends speedSum by performing a word-count-style map operation that maps each vehicle's sensors to the vehicle count and the vehicle speed: `(1, [speed])`. The reduce operation again simply sums the count and speed of each member vehicle `([numVehiclesInCluster], [sumOfSpeeds])`. The geoFiltered query also is selective, in that for a given test region the geoFiltered query operates only on a subset of the test region containing approximately half of the vehicles being tested.

geoMapped The geoMapped query performs a similar map/reduce operation to geoFiltered, but instead of filtering out half the test region, the geoMapped query maps half the test region to one region ID and maps the other half of the test region to a second region ID, then performs a reducebykey. The map operation produces the following: `(regionId) -> (1, [speed])` and the reduce produces `(regionId) -> ([numVehiclesInCluster], [sumOfSpeeds])`.

4.4 Simulator Results

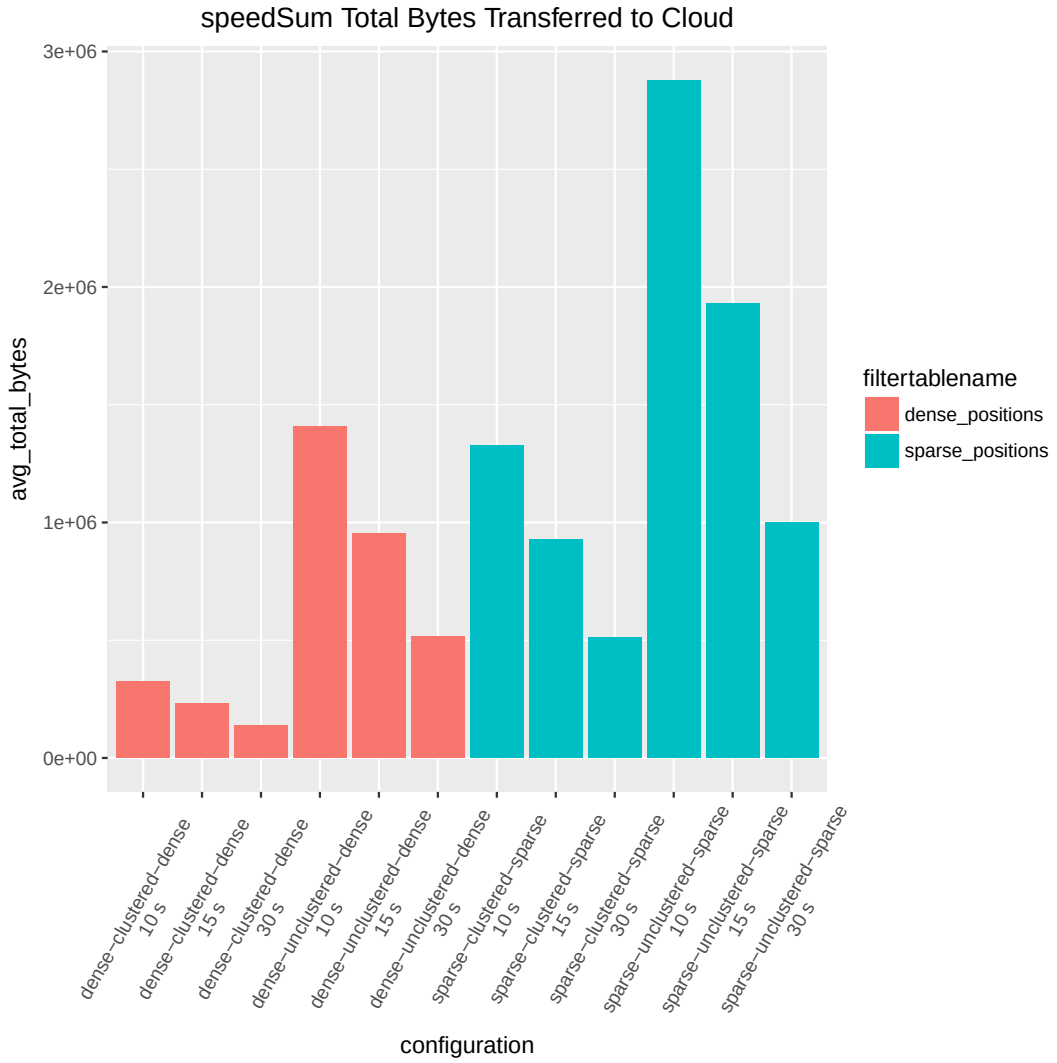


Figure 4.3: The total number of bytes sent to the middleware, averaged over multiple trials for the speedSum map/reduce job. The x-axis contians different configurations of regions, clusterings, and reduce window sizes (10, 15, and 30 seconds). Smaller values in the same geographic region indicate better performance, where the best performance comes from using clustering with a 30 second window size.

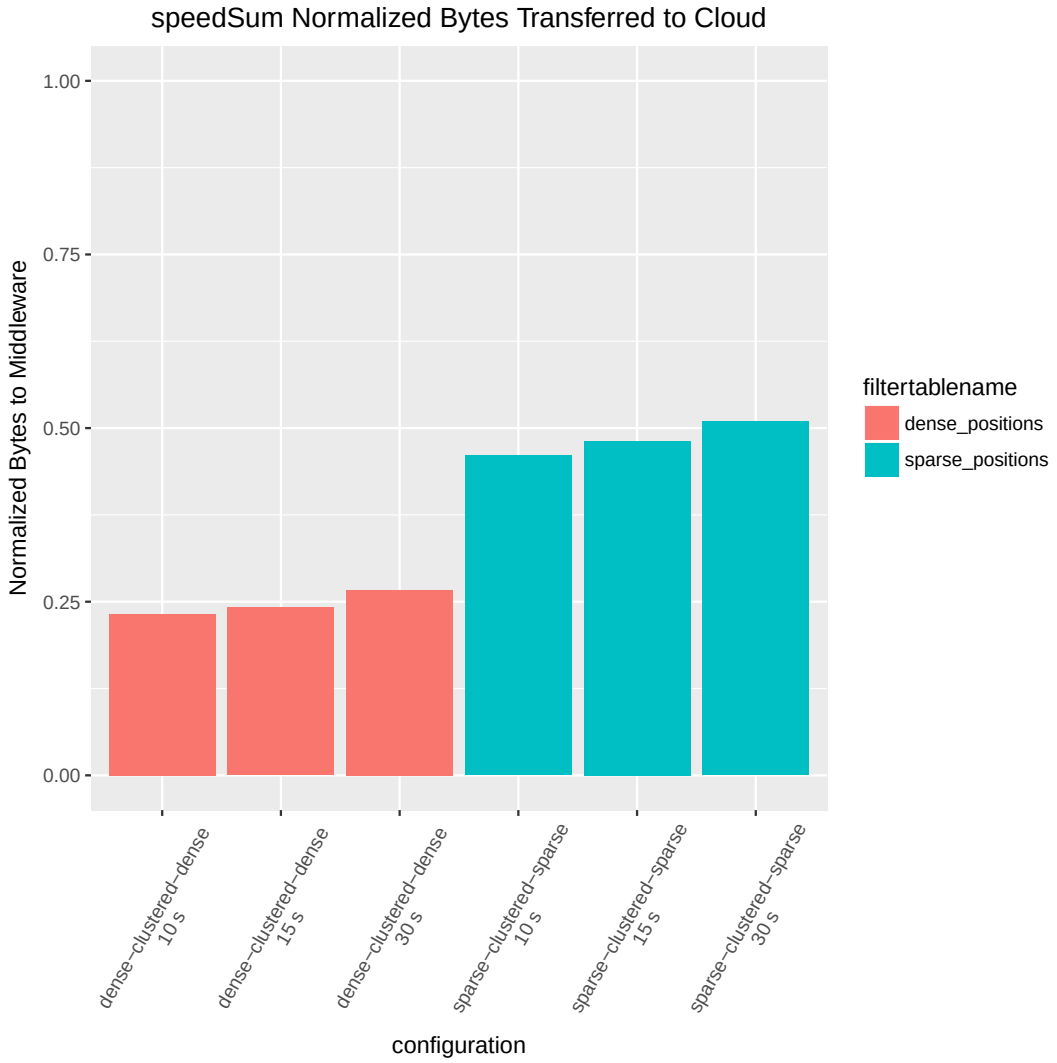


Figure 4.4: shows the scaled number of bytes sent to the middleware in a given region and window size configuration where y=1 indicates the number of bytes sent without clustering for a given region and reduce window size. The figure demonstrates the data savings that occur as a result of using vehicle clusters. The figure also illustrates how data reduction is affected by regional vehicle density - areas of high vehicle density can take better advantage of clustering to get more savings.

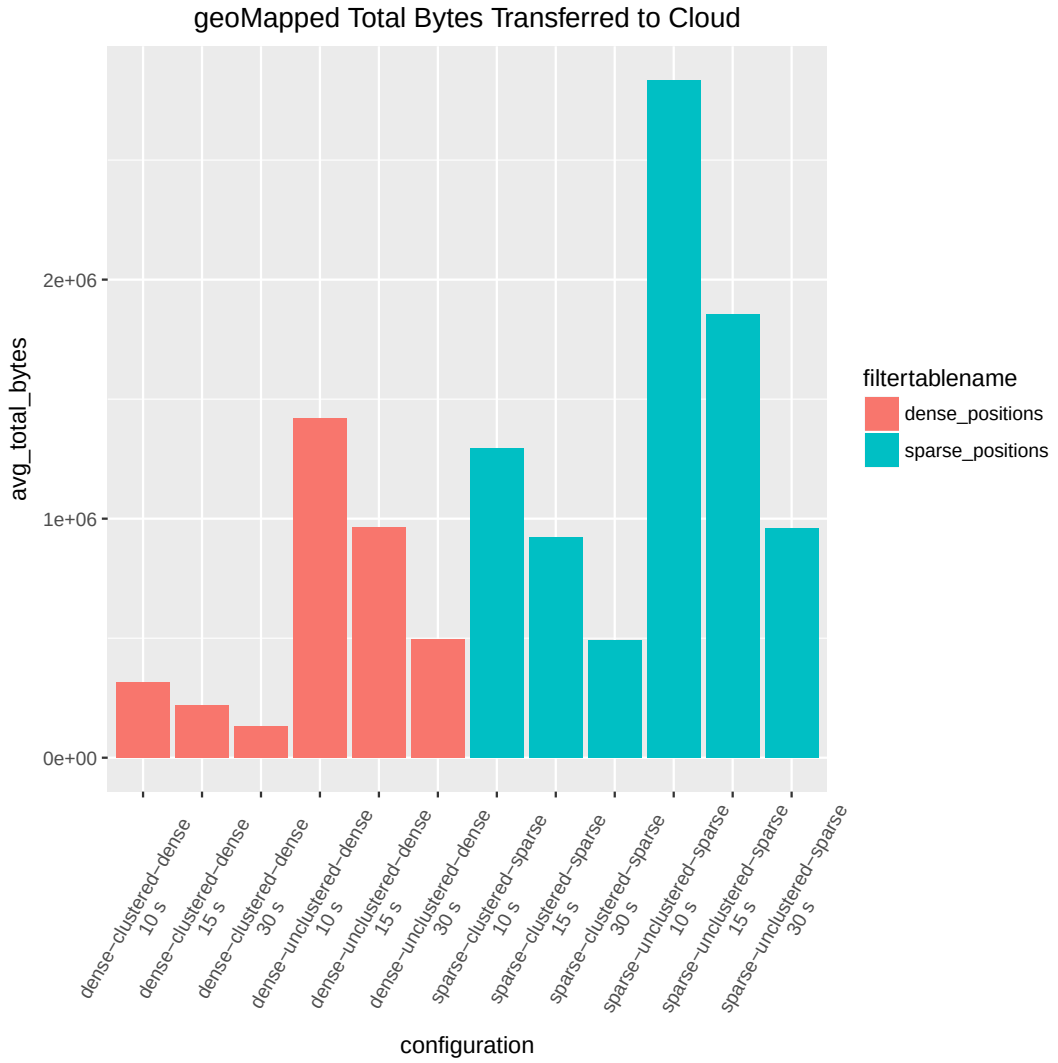


Figure 4.5: shows the total number of bytes sent to the middleware, averaged over multiple trials for the geoMapped map/reduce job. The x-axis contians different configurations of regions, clusterings, and reduce window sizes (10, 15, and 30 seconds). Despite being a slightly different map/reduce query, the performance of the geoMapped tests is similar to that of the speedSum tests, where the best performing configuration is the clustered configuration with a 30-second reduce window size.

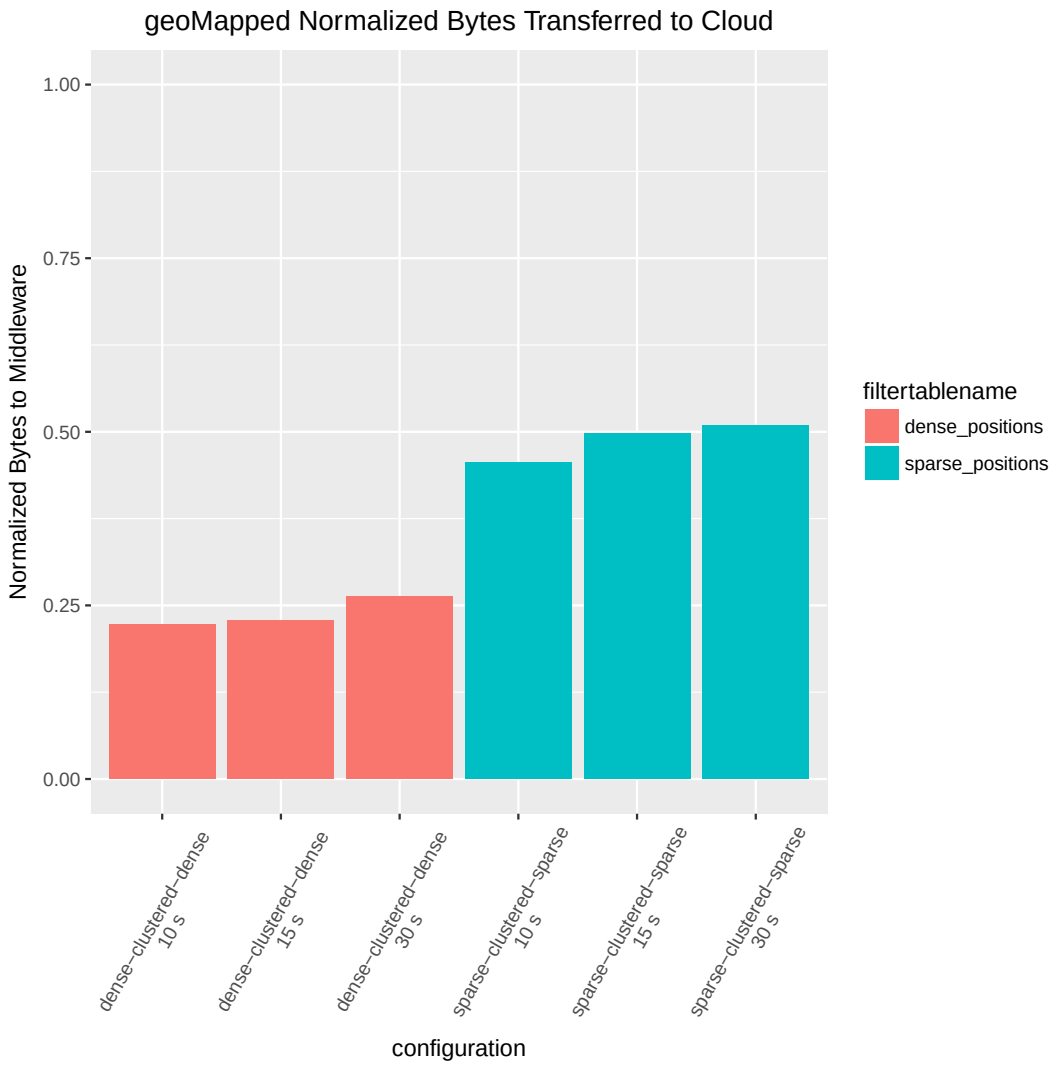


Figure 4.6: shows the scaled number of bytes sent to the middleware in a given region and window size configuration where y=1 indicates the number of bytes sent without clustering for a given region and reduce window size. The figure shows a similar reduction in data transfer by using clusters to that of the speedSum map/reduce job.

5. Conclusion

5.1 Future Work

LITERATURE CITED

[1] This is the first item in the Bibliography. Let's make it very long so it takes more than one line. Let's make it very long so it takes more than one line.

[2] The second item in the Bibliography.

[3] http://veins.car2x.org

[4] http://www.scala-lang.org

[5] http://akka.io

[6] https://kafka.apache.org

[7] http://spark.apache.org/streaming/

[8] https://www.hds.com/en-ous/pdf/white-paper/hitachi-white-paper-internet-on-wheels.pdf

[9] https://research.google.com/archive/mapreduce.html

[10] http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/awsgsg-intro.pdf

[11] http://people.csail.mit.edu/matei/papers/2013/sosp_spark_streaming.pdf

[12] @ARTICLE6823640, author=N. Lu and N. Cheng and N. Zhang and X. Shen and J. W. Mark, journal=IEEE Internet of Things Journal, title=Connected Vehicles: Solutions and Challenges, year=2014, volume=1, number=4, pages=289-299, keywords=intelligent transportation systems;next generation networks;road vehicles;vehicular ad hoc networks;Internet of Vehicles;IoV;automotive revolution;connected vehicles;next generation intelligent transportation systems;vehicle communication;vehicle-to-Internet infrastructure connectivity;vehicle-to-road infrastructure connectivity;vehicle-to-sensor infrastructure connectivity;vehicle-to-vehicle infrastructure connectivity;vehicle-to-x connectivity;wireless connectivities;Communication system security;Reliability;Safety;Vehicles;Wireless communication;Wireless sensor networks;Zigbee;Connected vehicles;Internet of Vehicles (IoV);intelligent transportation systems (ITSs);intra-vehicle wireless sensor networks;vehicular networks, doi=10.1109/JIOT.2014.2327587, ISSN=2327-4662, month=Aug,

APPENDIX A
THIS IS AN APPENDIX

Note the numbering of the chapter heading is changed. This is a sentence to take up space and look like text.

A.1 A Section Heading

This is how equations are numbered in an appendix:

$$x^2 + y^2 = z^2 \tag{A.1}$$

APPENDIX B
THIS IS ANOTHER APPENDIX

This is a sentence to take up space and look like text.