UNIVERSITY OF FRIBOURG

MASTER THESIS

Real Time Data Analysis for Water Distribution Network using Storm

Author:
Simpal KUMAR

Supervisors:
Prof.Philippe Cudré-Mauroux
Djellel E DIFALLAH

A thesis submitted in fulfilment of the requirements for the degree of MSc Computer Science

in the

XI—the eXascale Infolab

May 2014

"Information is the oil of the 21st century, and analytics is the combustion engine."

Peter Sondergaard

Abstract

Prof.Philippe Cudré-Mauroux
Djellel E Dfallah

MSc Computer Science

Real Time Data Analysis for Water Distribution Network using Storm

by Simpal Kumar

Thesis Purpose This thesis investigates, analyses, designs and provides a complete solution to find out the anomalies in a water distribution network (WDN) topology. Real time sensor values are used to compute Local Indicator Spatial Association (LISA) value and visualize to detect anomaly easily.

Theoretical Perspective The study is grounded on LISA statistics computation on real time sensor values e.g. pressure in water pipes in a WDN of defined topology. Theory is limited to compute LISA value on one parameter at one time and assumes that all the sensors write information at same time. Topology is predefined and Storm cluster is used to process information.

Thesis Methodology Storm is extensively used for processing but other technologies have also been used to setup cluster, visualization and simulation of sensors in the topology.

Analysis and Conclusion The application was tested with 16 nodes network with different scenarios: with actual neighbours, with various numbers of fake neighbours for each node (1,2,4,8,12,15). Also, different combinations of spouts and bolts were tested. Scalability of solution was tested with one test conducted upon topology of 1600 nodes. Speed of computing LISA and ability to detect anomaly were used for measuring the performance.

Results showed that while Twitter Storm is efficient and good tool for anomaly detection but did not appear that promising statistical significance.Results have proved that detecting an anomaly is quite fast even in densely connected network. Visualization is helpful not only in detecting anomaly but also to figure out and measure the impact up on neighbours too.

¡

Acknowledgements

I would specially like to thank Prof.Philippe Cudré-Mauroux and Djellel E. Dfallah under whom supervision I finished this thesis. I thank them for their continuous advice, feedback and patience throughout the project. Without them the project would never have lifted off the ground, not to mention come to a conclusion.

XI Scalable infolab team was really helpful to me. They gave me a lot of guidance to set up the cluster of machines used for the use cases. It's been an amazing time and has shaped me both intellectually and as a person. To them I would like to express my gratitude and encourage them to continue enabling other students to do likewise.

I would specially like to thank my parents, two younger sisters, and elder brother. They were always supporting me and encouraging me with their best wishes. My greatest appreciation goes to my friends for being always there and supporting me.

Most importantly I would like to thank my husband and my daughter, for they have always stood behind me to provide support and motivation throughout my studies.

Contents

Abstract ii

Acknowledgements iii

Contents iv

List of Figures vi

List of Tables vii

Abbreviations viii

1 Introduction 1
- 1.1 Motivation and Goals . . . 1
- 1.2 Report Structure . . . 3

2 LISA- Local Indicator Spatial Association 4
- 2.1 Spatial Cluster and WDN . . . 4
  - 2.1.1 Spatial Auto Correlation: . . . 5
- 2.2 Local Indicator Spatial Association . . . 6
  - 2.2.1 Moran's I . . . 6
    - 2.2.1.1 LISA from test case . . . 7
  - 2.2.2 Definition Global Getis-Ord G . . . 8

3 Background 9
- 3.1 Introduction to Storm . . . 9
- 3.2 Concepts . . . 10
  - 3.2.1 Topology . . . 10
  - 3.2.2 Storm Cluster . . . 10
    - 3.2.2.1 Nimbus . . . 10
    - 3.2.2.2 Supervisor . . . 11
    - 3.2.2.3 Worker . . . 12
    - 3.2.2.4 Zookeeper . . . 12
    - 3.2.2.5 User Interface . . . 13
  - 3.2.3 Task . . . 13
    - 3.2.3.1 Spout . . . 13

3.2.3.2 Bolt . . . 13
3.2.3.3 Stream Grouping . . . 14
3.2.4 Reliability . . . 14
3.2.5 Limitations . . . 15
3.3 Other Tools and Technologies . . . 15

4 Design and Implementation 16
4.1 Storm Topology Application . . . 18
4.2 Monitoring And Visualization Module . . . 22
4.2.1 Monitoring Network . . . 22
4.2.2 Visualization Module . . . 24
4.3 Network Simulator . . . 25

5 Evaluation 27
5.1 Test Set up . . . 27
5.2 Performance Measures . . . 28
5.2.1 Test Case 16 node Actual LISA . . . 29
5.2.2 Test Case 16 node with fake random neighbours 100 times . . . 31
5.2.3 Test Case 16 node Anomaly introducing real time and visualization 34
5.2.4 Test case with various spouts and bolts combinations . . . 37
5.2.5 Test Case 1600 node LISA for a node with 4 fake random neighbours 1000 times . . . 37
5.3 Discussions . . . 38
5.3.1 Problems encountered . . . 38
5.3.2 Limitations . . . 39

6 Findings and Conclusion 40
6.1 Findings . . . 40
6.2 Conclusion . . . 41
6.3 Future Work . . . 41

A Running Application on Cluster 42

B Sample of Files 44

C Storm ui screen shots 48

Bibliography 50

List of Figures

1.1 Example of real topology set up 1

4.1 Example of real topology set up 17
4.2 Components and interaction between them 18
4.3 Visualizing 16 nodes network 25
4.4 Visualizing 1600 node 25

5.1 Example of real topology set up 29
5.2 Test case LISA computation time with real neighbours 30
5.3 Scatter Plot of LISA with real neighbours 30
5.4 Speed of LISA computation for 100 permutations 31
5.5 Time taken to compute LISA values for each node for 100 permutations . 31
5.6 Time for LISA values with 1 neighbour for each node randomly 32
5.7 Time for LISA values with 2 neighbours for each node randomly 33
5.8 Time for LISA values with 4 neighbours for each node randomly 33
5.9 Time for LISA values with 8 neighbours for each node randomly 33
5.10 Time for LISA values with 12 neighbours for each node randomly 34
5.11 Time for LISA values with 15 neighbours for each node randomly 34
5.12 Network visualization without anomaly 35
5.13 Network visualization with anomaly 36
5.14 Network visualization with anomaly 37
5.15 1600 nodes network 1000 LISA Values with 4 random neighbours 38
5.16 1600 nodes network 1000 permutation LISA Values with 4 random neighbours compared to normal LISA speed computation 38

C.1 Storm UI Topology submitted 48
C.2 Storm UI Topology latency details 49

List of Tables

5.1 LISA values of nodes with actual neighbours 30
5.2 Speed of LISA computation for 100 permutations with random neighbours of each node 32
5.3 LISA values without anomaly 35
5.4 LISA values after introducing anomaly 36

Abbreviations

LISA	**L**ocal **I**ndicators of **S**patial **A**ssociation
WDN	**W**ater **D**istribution **N**etwork
D3	**D**ata **D**riven **D**ocuments

Chapter 1

Introduction

1.1 Motivation and Goals

Around the world cities are getting intelligent day by day. City administrations are focussing to take inputs from everyday life of its inhabitants and utilize those inputs to manage and channelize the resources. In 2050, 70 percent of people will live in urban areas around the world. Transportation, water supply, education, healthcare are essential entities of population centres in modern civilized systems. Many of these entities are already digitized and produce data streams. These data streams are growing at a fast pace and are getting difficult to analyse, compute and visualize in real time to make smarter decisions. e.g. in water distribution networks data management architecture suggested by [9], we have water pipes connected with sensors, these sensors are generating huge flow of data streams every second shown in fig 1.1. Aim of this thesis is to provide a solution for this huge real time stream processing.

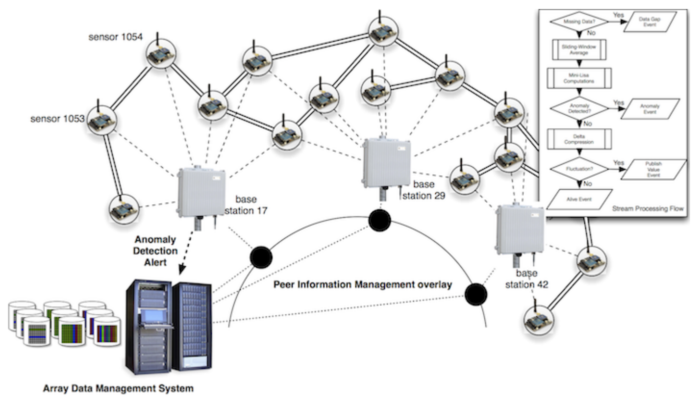


Figure 1.1: Example topology with 16 nodes with 4 base stations

Researchers and enterprises alike (e.g. IBM) have harnessed this massive amount of content generated. For instance,[1] has explored how sentiment analysis can be used to predict stock market movement. Tweets by authors e.g. ”I'm feeling” and ”makes me” states the mood which are passed to tools to predict the mood such as calm, alert, sure, vital, kind and happy. Machine learning techniques are then used to establish link between mood and stock market movement (e.g. ”calm” and the Dow Jones Industrial Average.

In another paper, Sakaki [2] has used social networks to detect earthquakes in real-time. Author has considered each user a sensor and each tweet from user as sensor information. These posts are then classified based on the terms ”earthquake” and ”shaking” appearing in posts using machine learning techniques.By reading user's GPS co ordinates, location is approximated, with detection rate of 96% for JMA intensity scale 3 earthquakes.

Smarter Cities where massive amount of data is generated every second, processing this data in real-time can only be achieved by distributing the workload across many computers. e.g. WDNs where sensors are attached to pipes are generating data (pressure, temperature etc.) and sending it to base stations. As cities and these networks are growing, information each second coming to these base stations are huge and ever increasing . In WDN information coming each second from sensors is not of relevance but observation of network based on geographic locations is point of interest when some anomaly is detected. How an anomaly can be defined or detected in WDN is defined by computing LISA values for each sensor considering it's neighbours' values. LISA (Local Indicator of Spatial Association) can be used to detect instability in the network. LISA statistics is discussed further in detail in chapter 2.

While software frameworks such as Apache Hadoop, Google's MapReduce exists, their scalability is limited by throughput of core-level switches based on studies. Hadoop is limited to process data by batch and is not good for processing latest version of data.Stream processing on the other hand process a constant influx of data, in real time. i.e. as the data arrived, system should react and take decisions quickly. In WDN e.g. if pressure goes beyond certain threshold value it should be processed and should pass this information fast to take decision on. In stream processing, the speed at which new data is created and needs to be processed can be extremely high and so more efficient approach or technique is required to process this huge velocity and volume of data. Storm is becoming popular day by day and seems promising to handle massive amount of information in real time and is used as primary tool for this thesis.

Therefore, goal of this thesis are:

- Study of LISA statistics, Stream processing and Analysis of WDN

- Storm configuring, cluster set up, prototype design and implementation
- Test Cases, Analysis and conclusion based on prototype designed

1.2 Report Structure

Chapter 1 gives an introduction to the relevant concepts and motivation behind this thesis work. Chapter 2 elaborates the LISA statistics, used extensively to detect instability in spatial regions. Chapter 3 gives an overview of Technologies used e.g Storm, Zookeeper, ZeroMQ, D3 etc. Chapter 4 describe the design of prototype and its logical and physical placement in the cluster. This chapter also provides an insight into decisions taken and discusses the alternatives. Chapter 5 provides graphs and numbers of the performance results achieved. These are analysed to give a sensible interpretation. Along with this project limitations are discussed. Furthermore this chapter provides insight into the challenges and problems faced during the design and implementation phases. Chapter 6 In the conclusive remarks goals achieved by this thesis and future work is discussed.

Chapter 2

LISA- Local Indicator Spatial Association

2.1 Spatial Cluster and WDN

Cities are getting smarter in their civil infrastructure - roads, pipes, rail lines, conduits, treatment, storage and disposal system which are basically dealing with daily activities of population e.g. movement of traffic, water and sewage , energy throughout the city. To urbanize these cities the existing infrastructure is extended or replaced. Because of emerging sensing technologies and sensor networks, this is possible without much additional cost. Sensors can be easily placed to gather information from existing infrastructure. Real time sensing exists everywhere within these systems, so new technologies are needed to integrate the large amount of data generated by these sensors and further can be analysed to extract useful information from these real time data streams.

For this thesis Water Distribution Networks are primary focus. WDN can be viewed as a directed graph, where edges are pipes, and nodes in graph are pipe junctions and end points. Information is produce at sensors and stored over the base station nodes and forwarded further in topology. The rate of transmission for water is in order of cm's/sec. To operate these networks efficiently there is need to monitor the parameters e.g. hydraulic(pressure, flow etc) and water quality(chlorine, pH, specific conductance etc.). Currently the hydraulic parameters are monitored at a small fraction- 0.01 to 0.001 approximately. But the majority of water quality monitoring is still performed with non-continuous samples taken at discrete times and location within the network. Real time networks were deployed in past decade with continuous water quality monitoring According to [3] physical constraints and economic costs limited the number of monitoring stations being installed. Most of networks are relying on citizens by reporting situation e,g breaks in water mains or odour in water to inform about poor water quality.

Because of advancement in technologies, now low cost with less power consumption, easy to mount with improved network communication sensors are available. Large scale data analytic has revolutionized a way that now municipal infrastructures can be monitored in better way. The data streams generated by these sensors can be used to operate on these networks more efficiently. With the invention of smart meters, it is possible to gather real time data of consumption at service connections throughout the distribution network. This data can be transmitted over internet to make it available to service provider. Sampling intervals of the sensors are between 15 to 60 minutes, but can be as small as every few seconds. Analytical solutions applied on this data can detect theft or leakage of resources, improve the load/ demand forecasts. The biggest challenge in these networks are: First to install the sensors on each node, which should be cost economic and easy to install in existing infrastructure. Second is for theft or leakage, data acquisition delay should be minimal, current sensor deployments delays are in of tens or minutes to several hours, which needs to be reduced. This is big problem in case of very large area WDN's with thousands or millions of nodes. Purpose of this thesis is to deal with second problem and provide a solution for the same. The details of architecture and solution for WDN's is discussed in detail in Chapter 4. Real time information gathered from sensors needs to be analysed and provide the useful information for better operations. To analyse this data in this thesis LISA statistics is used to measure LISA at each local node and then can be used to compute Global LISA which is directly proportional to local LISA.

2.1.1 Spatial Auto Correlation:

Spatial Autocorrelation summarize the spatial structure in some property. Cliff and Ord[6] discuss some key issues in analysis of spatial autocorrelation e.g. elevation and precipitation tend to vary smoothly and are usually positively spatially autocorrelated. Areal data was considered for this discussion. It was concluded that values at locations close together tend to be similar. In contrast, grey scala in a remotely sensed image may be negatively autocorrelated if, e.g. there are neighbouring fields in an agricultural image that have very different characteristics.

The capabilities for visualization, rapid data retrieval, and manipulation in geographic information systems (GIS) have created the need for new techniques of exploratory data analysis that focus on the "spatial" aspects of the data. The identification of local patterns of spatial association is an important concern in this respect. Although many methods are available in the toolbox of the geographical analyst, only few of those are appropriate to deal explicitly with the "spatial" aspects in these large data sets (Anselin 1993b)[4]. In the analysis of spatial association, it has long been recognized that assumption of stationarity or structural stability over space may be highly unrealistic, especially when a large number of spatial observations are used. Spatial structural instability or spatial drift has been incorporated in a number of modelling approaches. A focus on local patterns of association (hot spots) and an allowance for local instabilities in overall spatial association has only recently been suggested as a more appropriate perspective, for example, in Getis and Ord (1992), Openshaw (1993), and Anselin (1993b).

2.2 Local Indicator Spatial Association

Local spatial clusters, sometimes referred to as hot spots, may be identified as those locations or sets of contiguous locations for which the LISA is significant. These indicators allow for the decomposition of global indicators, such as Moran's I, into the contribution of each individual observation.

Anselin defined the local form of Moran's I and Geary's C. LISA statistics allow for decomposition of global indicators. Where I is positive this indicates clustering of similar values, whilst where I is negative this indicates clustering of dissimilar values; a value of zero indicates zero spatial autocorrelation.

2.2.1 Moran's I

As an operational definition, I suggest that a local indicator of spatial association (LISA) is any statistic that satisfies the following two requirements:

- the LISA for each observation gives an indication of the extent of significant spatial clustering of similar values around that observation;
- the sum of LISAs for all observations is proportional to a global indicator of spatial association.

A LISA is given as a statistic Li for a variable yi observed at the location i:

$$L_i = f(y_i, y_J i)$$

where f is a function and yJi are the observed value (or deviations from the mean) in neighbourhood Ji. Local Moran's I for observation i is given as :

$$I_i = (z_i/m_2)\sum_{j=1}^{n} w_i j z_j, j \neq i$$

the observations zi are deviations from the mean

$$z_i = (y_i - \bar{y})$$

and m2 is the variance. wij represents the weight which may be in row standardized form (sum equal to one) and so

$$w_i j = 1/n$$

where n is number of neighbours, which means summation includes only the neighbouring zones. y values can be taken from raw observations or some standardized form too.

2.2.1.1 LISA from test case

It would be more clear how LISA can be computed and used for statistical significance by considering an example. In this thesis we are using normal distribution to generate the values from sensors so from that sample here neighbours values for a node i are given as:

Total number of nodes are 1600. Mean of 1600 nodes is : -0.014266266590053426 and variance is : 1.005881796332947

Node in consideration : N-H82EQ (-0.5373494464575331)

4 Neighbour nodes are:

N-1EZEK (1.143572306582178) N-GBL9J (-0.7293499055446114) N-RMRWR (-0.055567582400571164) N-LH237 (0.27619204641674594)

Using above formula LISA value for N-H82EQ

I = ((-0.5373494464575331-0.014266266590053426)/1.005881796332947) * ((0.25 *(1.143572306582178-(-0.014266266590053426)))+ ((0.25 *(-0.7293499055446114-(-0.014266266590053426))) + ((0.25 *(-0.055567582400571164-(-0.014266266590053426)))+ ((0.25 *(0.27619204641674594-(-0.014266266590053426))) = -0.0899527893316361

As there are four neighbours so weight is 0.25. This weight is multiplied by deviation of value at node from mean. After calculating the LISA it can be used to identify anomalies

/ nodes where values are significantly way high or low. Moran Scatter plot can be drawn to visualize where each quadrant corresponds to one of the four different types of spatial association (SA).

Two observations can be made here about spatial association are: 1. locations of positive spatial association (which means "I'm similar to my neighbours") which corresponds to high-high or low-low 2. locations of negative spatial association(which means "I'm different from my neighbours"). which corresponds to Low-High or High-Low

2.2.2 Definition Global Getis-Ord G

As mentioned before that global Moran is directly proportional to local Moran's so G will be high where high values cluster and G will be low where low values cluster

$$G(d) = \sum_i \sum_j w_i j(d) x_i x_j / \sum_i \sum_j x_i x_j$$

Until now LISA statistics has been used for analysing areal data for time stationary problems in geographic domain(e.g. identification of hotspots of criminal activity or cancer mortality). LISA is recently been applied to network topologies which still do not include WDN's. This thesis emphasizes on providing solution for WDNs by defining LISA networks using topology of WDN's and identify anomalies within the sensor data. Details how WDN's are considered as LISA topology are discussed in Chapter 4.

Chapter 3

Background

3.1 Introduction to Storm

Twitter Storm is a distributed real-time computation system that aims to fill the void left by the Hadoop and MapReduce systems. Storm was created at Backtype, a company acquired by Twitter in 2011. It is a free and open source project licensed under the Eclipse Public License.Storm is written in Clojure and Java. Hadoop and MapReduce provide general frameworks for batch processing, while Storm provides a general framework for real-time processing. e.g., Twitter processes tweets in real-time with Storm for their publisher analytic product, Storm can be used to analyze, filter and normalize the information and many others regular-expression filters on logs in real-time. Important feature of Storm is fault tolerance and guaranteed data processing. Storm can be used for:- Stream processing, Continuous computation and distributed RPC.

- It can be used to process a stream of new data and update databases in real time
- It can do continuous query and stream the results to clients in real time
- It can put in parallel an intense query on the fly

For this thesis, Storm is used for real time stream processing. The main concepts of Storm are discussed in detail in the next section.

3.2 Concepts

3.2.1 Topology

A topology is a set of tasks. A task can be defined as a bolt or a spout, connected by streams. Topology can be viewed as elements of a directed graph, where tasks (bolts and spouts) represent vertices and streams represent directed edges. Because a topology is assumed to process a real-time stream, it processes messages forever as they arrive. The only way to terminate a running topology is to send the “kill” command. Though it’s the only way but for development purposes a time out with subsequent termination can be defined.

3.2.2 Storm Cluster

A storm cluster can be viewed as Hadoop cluster. There are two kinds of nodes ”master” and ”workers”.

3.2.2.1 Nimbus

A daemon called ”Nimbus” run on master node. On all worker nodes, location of node where nimbus is running, should be configured. Nimbus implements a set of Apache Thrift RPC functions which read and adjust the cluster state. RPC function can be called by command line client using ./storm nimbus command. Alternatively, the RPC functions can be called directly from any language supported by Thrift. The most important functionality of Nimbus is:

- Defining a topology: A topology can be defined by passing Nimbus a Java archive (“jar”) containing a class which builds a topology using Storm’s TopologyBuilder class. The class is a wrapper around a subset of the remote functions Nimbus implements. Thrift RPC can also be used for setting up a topology from outside of Storm.
- **Starting a topology:** The Nimbus scheduler keeps track of how many assignments each worker has.When a topology is started tasks are scheduled. If there are empty slots available on supervisors, tasks are assigned to those slots. Otherwise tasks are assigned to running workers, the most lightly loaded workers are given preferrence. The files belonging to the task are uploaded to the supervisors by Nimbus before starting the topology.

- **Killing a topology:** Nimbus first stops all spouts, waits a defined timeout for the bolts to finish processing in-flight tuples and then stops the bolts.It is responsbile for distributing code around the cluster, assigning tasks to worker machines and monitoring failure.
- **Registering supervisors:** When a new supervisor is started it registers itself with Nimbus. Most importantly, Nimbus stores the number of available workers on the supervisor and its IP address or domain name.
- **Monitoring supervisors and tasks:** Both supervisors and tasks send heartbeats to Nimbus in regular intervals. If a supervisor goes down the tasks of its workers are reassigned to another supervisor. If a task goes down it is reassigned to another worker.
- **Rebalancing tasks:** When the available worker pool changes, e.g. new supervisors start or go down, the workload may become unevenly distributed. An explicit call for rebalancing re-distributes tasks to workers.

3.2.2.2 Supervisor

Storm workers are managed by a supervisor local to each node. A supervisor has one or more slots, each slot specifying the port a worker listens on.The supervisor functions as follows:

- A new worker is launched when a task is assigned to a slot which is not yet filled. The worker is assigned a generated ID, a slot (port) and its supervisor by the supervisor.
- On launch of a topology, all files necessary for launching a task are downloaded from Nimbus and copied to all directories of workers assigned to the topology.
- Running workers are monitored. If a worker heartbeat times out too often, the worker is cleaned up, i.e. the process killed if it still exists and files removed.
- ZooKeeper is monitored for changes. If there are open slots workers are started on-demand, worker configuration changes are written to file.
- All communication with workers is performed via file I/O.Worker heartbeats are written in a heartbeat directory as files with increasing sequence numbers as names. Worker configuration is passed in system specified files.

3.2.2.3 Worker

tasks are assigned to and run by worker processes. Workers have an ID, although workers are often identified by their node and port.Every task is started in its own thread and the worker does the following:

- In a thread the worker starts a “virtual port”. The virtual port binds the worker’s assigned port with a ZeroMQ pull socket and multiplexes tuples from all incoming Storm streams to the respective tasks based on the task ID found in the tuple. ZeroMQ pair sockets are used for passing tuples to tasks.
- Tasks write outgoing tuples to a Java LinkedBlockingQueue, which is thread-safe. The worker sets up ZeroMQ push connections to the superset of destinations of all tasks. A thread blocks on the queue and multiplexes tuples to their destination nodes.
- The worker monitors its configuration file for changes. Tasks and outgoing ZeroMQ connections are added and removed on demand.
- A thread writes heartbeat files, as described previously.

Each worker machine run ”supervisor” using ./storm supervisor. Communication between nimbus and supervisors are done through Zookeeper. It is a centralized service for maintaining configuration information and providing distributed synchronization and group services.

3.2.2.4 Zookeeper

Apache ZooKeeper stores the cluster configuration and state. For example:

- Topology, supervisor, and task information written by Nimbus.
- Supervisors and workers read their configuration from ZooKeeper.
- Task statistics on processed, in-flight and failed tuples.
- Heartbeats of supervisors and tasks.

3.2.2.5 User Interface

The user interface accesses statistics and state information stored on ZooKeeper and displays them via HTTP. Controlling the cluster from the UI is not possible. This can be run by command line using ./storm ui and can be seen at browser at 8888 port by default.

In a simple set up, a spout emits a stream of generated tuples to a bolt, which does some processing and sends the intermediate result to a second bolt. The second bolt completes the processing and sends the final result to a third bolt, and so on. Final bolt aggregates all final results sent to it in a distributed relational database or where you want to store this result. A topology only describes an abstract relationship between tasks. The physical layout of an active topology is controlled by the Storm scheduler.

3.2.3 Task

A Storm task implements either a bolt or a spout interface. The worker is designed as an implicit loop, hence functions of the interface are called in a loop by the worker. The task registers the types of fields in outgoing streams with Storm.

Stream is defined as an unbounded sequences of tuples and a tuple is defined as a named list of values. These values may have different types. A stream can be splitting and its tuple can be directed to multiple bolts for parallel processing. Every stream and tuple is assigned an identifier for selecting a stream and to acknowledge a tuple for reliability.

3.2.3.1 Spout

It is defined as a source of one or more streams, emitting tuples. Stream can be used to generate tuples from anywhere, e.g. files, databases, a non-storm stream etc. For this thesis pllain text files are used.

3.2.3.2 Bolt

A bolt is defined as a processing element absorbing one or more streams and emitting none, one or more streams. It can do any kind of processing on incoming tuples e.g. aggregation, filtering, transformation etc.

All tasks(spouts and bolts) are assigned a component ID, which is used to identify its position in the topology, and a task ID, which is used for locating a task instance in the cluster and routing its streams to it.

3.2.3.3 Stream Grouping

A stream grouping is the method of distributing a stream's tuples among the parallel instances of a bolt. A bolt absorbs multiple streams and can define a different stream grouping for each stream. The stream groupings provided by Storm are:

- **Shuffle grouping:** A uniform distribution of tuples to bolts. Fields grouping: A field of a stream is defined as the key. Tuples with equal keys are mapped to the same bolt instance.
- **All grouping:** A broadcast. The stream is replicated on all bolt instances.
- **None grouping:** Distribution of tuples is undefined. The current Storm implementation does shuffle grouping.
- **Direct grouping:** The emitting task specifies the absorbing bolt for each emitted tuple.
- **Local or shuffle grouping:** Bolts located on the same worker as the emitting task are preferred. Equivalent to shuffle grouping for remote bolts.

For this thesis All grouping, field grouping and shuffle grouping are used for various bolts. This is discussed in detail in implementation chapter.

3.2.4 Reliability

Storm provides a reliability framework for guaranteed tuple processing: Every tuple is given an ID. Every tuple emitted by a spout is seen as the root of a tree. When a tuple is processed by a bolt, the new result tuples are added to the tree as children. Adding children to the tree must be done explicitly by anchoring the tuple being emitted to another tuple. At each step traversing down the tree an acknowledgment or negative acknowledgment (fail) message is sent to a tracking task (acker). If all children up until the leafs are acknowledged successfully, the whole tree is considered fully processed. Otherwise, if either a fail is sent or a tuple times out, the whole tree is considered failed and the root tuple ID is passed to a failure handler in the spout.

3.2.5 Limitations

In the current Storm implementation the task scheduler does not consider locality, which could potentially reduce network traffic.

- The scheduler does not consider the processor load, memory usage, available bandwidth and other system information when allocating tasks. The sole consideration is the number of tasks running on a worker. This could lead to uneven load distribution among nodes.
- Setting up a topology from a language other than Java is cumbersome. In Java a native wrapper API with convenience features is provided, all other languages must interface the Thrift API.

3.3 Other Tools and Technologies

This project is mainly implemented in Java. Following are other tools tools used for whole set up:

- **Java** : A network simulator is implemented in JAVA which is responsible to write sensor information in files as per the topology definition, Also it is responsible to clean the files when data is too large.
- **JSON**: Java thread is responsible for reading and converting to JSON the computed values of LISA from file where storm application write information real time, It will continuously rewrite the information in JSON file every minute
- **D3**: D3 is used for visualizing the sensor nodes and showing the nodes clustered based on LISA value. e.g. all nodes with +ve moran I are displayed in different color than with -ve Moran I. nodes for which information was missing is shown in different color, which could be very helpful for monitoring purpose.
- **github**: Source code for both storm application and network simulator are maintained on github at https://github.com/simpalK/TwitterStormSensorSimulation https://github.com/simpalK/SensorSimulations
- **Documentation**: Tools like LateX, excel are used for graphs, calculations.

Chapter 4

Design and Implementation

WDN consists of pipes with sensors, base stations and connection between these stations could be via internet. Simple overview is shown in Fig 1. Where the sensors are connected to Base Stations and sending information continuously to BS. Each Base station has number of sensors attached to it. Information coming from sensors are real time and needs to be processed real time taking in accordance the information coming from neighbours. Neighbours could be in the same base station or could be from the whole network. These base stations are communicating to each other via internet. To compute LISA we need mean and standard deviation of whole network and then for Moran's I only neighbours values are considered. After LISA value for each node is computed it is used for finding out anomalies in network i.e. which node/sensor is generating high value as compared to it's neighbours etc.

Figure 4.1 shows the example topology set up for testing in this thesis.

There assumptions considered in this solution are:

- Synchronization: each sensor will write the values at same time
- No Hardware failure: Assume that hardware is all correct
- Time: Each sensor writes information every minute
- Network: Network will be fixed but neighbours' topology can be changed
- Storm is up and running and no internet or hardware failure regarding storm is considered

After analysing the whole problem solution is given as follows:

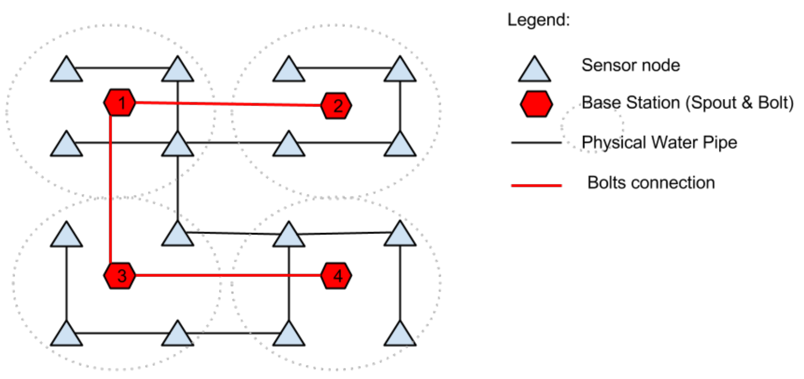


FIGURE 4.1: Example topology with 16 nodes with 4 base stations

- **Storm Topology Application** Using Twitter Storm storm cluster is set up and application using java as language will run on this cluster. Main class is Topology where spouts and bolts instances are defined and other configuration is done for Topology. This application is run as jar (created using mvn) on storm using ./storm command. This application is responsible for reading information real time and process it, compute LISA at each node and store it in file (It can be stored in database also). Details are discussed in next section.
- **Monitoring and Visualizing** Information stored in file is read every minute by monitoring module written in Java. It's responsibility is to parse information and write in JSON file. Which is then used to represent the network topology using D3 library in browser. Lisa value is shown by node colors. if its +ve or -ve or no value available at that time stamp.
- **Network Simulator** Network simulator is implemented in Java which is responsible for generating sensor values and write them in text file to provide as input to Storm application. This file will keep the last values only at last time stamp and periodically will be deleted. For thesis test case 1600 nodes were considered, and information was stored in 4 files. Basically Spout will read the information which is installed on each Base Station. This module is for testing purpose which will be further replaced with real network and base stations.

Figure 4.2 shows the three components interacting with each other.

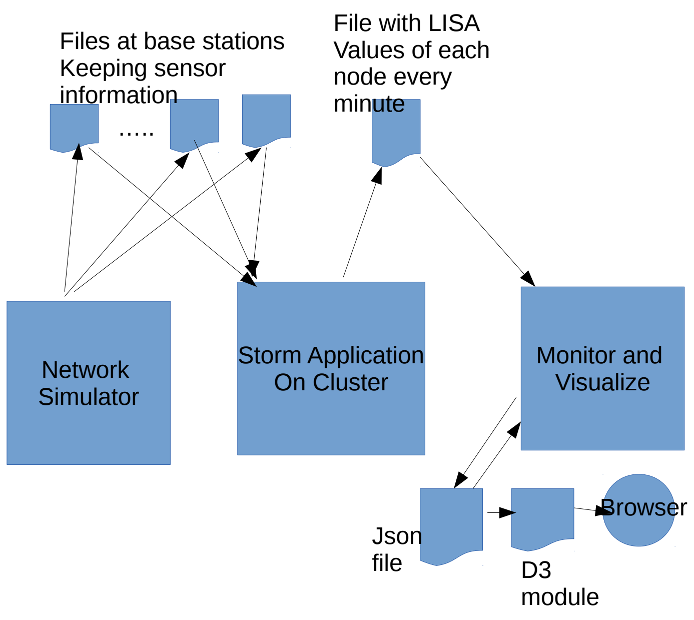


Fig: Modules and integration between them

FIGURE 4.2: Components and Interaction between them

4.1 Storm Topology Application

Storm Topology application is responsible for reading the real time data generated by sensors from base stations and process them to compute LISA values for each node and store them in text file. As discussed in chapter 3 storm application is basically a set of spouts and bolts connected via defining topology to process information. e.g. listing below shows us topology implemented in application. Three kinds of groupings are used. allGrouping(so that all the information should flow in network), fieldgrouping(only neighbors values are streamed based on information from actual topology stored) shuffleGrouping(compute LISA for a node and write LISA in JSON). Storm is written in java and closure for thesis java is used for storm application.

```
TopologyBuilder builder = new TopologyBuilder();
                builder.setSpout("SensorEmitter",new SensorEmitter(),4);
                builder.setBolt("SensorGetter", new SensorRealTimeGetter(),4)
                        .allGrouping("SensorEmitter");
                builder.setBolt("SensorBolt2", new SensorRealTimeLevel2Bolt(),4)
                .fieldsGrouping("SensorGetter", new Fields("groupIds"));
                builder.setBolt("SensorLisaBolt", new LISABoltOutputJSon(),4)
                .shuffleGrouping("SensorBolt2");
```

SensorEmitter is spout which is reading data from file every second. These files are stored at base stations and sensors are writing information in these files every minute. Every fifteen minutes these files contents will be deleted by network simulator as storing information is not required when data is getting processed real time. SensorEmitter spout reads and forward the information to bolts using allGrouping as topology can be changed on run time so information should flow in whole network. Following code shows that last N lines are read from file as only values with latest timestamp will be passed.

```
//spout reads the last N lines and emit to bolt
String fromfileData1 = lastNlines(fileSensor1,N);
this.collector.emit(new Values(fromfileData1),fromfileData1);
```

SensorGetter is Bolt which consumes the tuples generated by spouts, compute the mean and average of all sensors values, read topology information written in a file and direct the tuples only with neighbors information to next bolt for computing LISA.This information is available in a file generated by csvParser and stored in a 2D array.

```
//Read the information from file
1.open file at specified path /../topologyInformation.txt"));
                 ....
2.read each line to get information of each node neighbours
         String line = inputStream.nextLine();
         String[] lineValues = line.split(",");
                       ...
     topoFromFile[row][column] = Integer.parseInt(lineValues[column]);
                ....
```

Code below is for parsing information available in csv file where node information e.g. ids are stored, and edges information is used to define topology information i.e. neighbours information.

```
//Function to parse the csv file and retrieve information in an array
public String[] parseCsv(Reader reader, String separator, boolean hasHeader)
1. Define an empty list for column names: List<String> columnNames
2. String array to store information for each node: String[] nodesInfo
3. Counter for counting nodes int nodeCount;
4. Define buffer reader :BufferedReader br and initialize it
5. while not end of file i.e no line left to read
    if !line.startsWith("#") then
      String[] tokens = line.split(separator);
```

```
        if tokens != null then
          if numLines == 0 then
             for (int i = 0; i < tokens.length; ++i)
                 columnNames.add(hasHeader ? tokens[i] :("row_"+i));
           else
              nodesInfo[nodeCount++]=  tokens[0];
           end if
             increment numoflines
         end if
       end while
6. return nodesInfo;
```

Mean and Variance is calculated for all nodes and now for each node a string id is generated to route only the neighbours information for LISA compute, and make it faster to compute. This string id is used for field grouping so that only required neighbours values are used.

```
1. Get the information emitted from bolt input.getString(0);
2. take each line in array String[] tokens= sentence.split("\n");
3. compute how many values are retrived
                for(String senseVal: tokens)
                        counterSensorVal++;
4. take values of only last time stamp,
which is basically is taken from last value written in sensor files.
String[] lastTimeStamp = tokens[counterSensorVal-1].split(",");
5. Compute the variance using values only from last time stamp
        for(String senseVal: tokens)
         String[] vKValues= senseVal.split(",");
                if(vKValues[2].contentEquals(lastTimeStamp[2])) then
                  findTimeStampVal[counterVK++]  = Double.parseDouble(vKValues[1]);
                  sumOfAllSensors += Double.parseDouble(vKValues[1]);
                end if
                Double meanOfAllSensors = sumOfAllSensors/(counterVK-1);
     end for
        for(int i=0; i< counterVK; i++)
          varianceSumOfAllSensors += (findTimeStampVal[i] -
          meanOfAllSensors)*(findTimeStampVal[i] - meanOfAllSensors);

        varianceOfAllSensors = varianceSumOfAllSensors/(counterVK-1);
6. Compute group Ids for tuples which is used to couple neighbours information
based on topology information and emit all neighbours information along with
mean and variance to next bolt
collector.emit(new Values(groupIds,str,meanOfAllSensors,
 varianceOfAllSensors,lastTimeStamp[2]));
7. print the information in logs to track
System.out.print("groupIds" + groupIds + "word" + str + "mean" +meanOfAllSensors+
"variance"+varianceOfAllSensors +"\n");
        collector.ack(input);
```

SensorLISABolt is bolt which consumes the tuples generated from SensorGetter bolt and use this information for computing LISA values. Computed LISA values along with node information and time stamp are forwarded to next bolt for storing this information.

Following code shows LISA computation, pass this LISA value to bolt responsible for writing information in file which is further used by monitoring tool.

```
1. Find and filter Neighbours values at same time stamp
if more than one time stamp values reached at bolt for a node and print the
neighbours information
 System.out.print("Neighbors: " +findNeighborsVal[counterVK-1] + "\n");
2. Compute LISA Algorithm
LISA equation contains two parts and so its computed in two steps
        i . compute part 1 and print it
            Double lisaEqPart1 = (vA - mean)/variance;
                System.out.print("Lisa Part1: " + lisaEqPart1+ "\n");
        ii. compute part 2 using neighbors and spatial matrix.
        iii. Print this values in logs
         Double spatialRowNormal =  (1.0/(double)counterVK);
        System.out.print("Lisa Part2 computation values neighbour num: " +
        spatialRowNormal + "mean" + mean +"value"+
        findNeighborsVal[i] + "\n");
        lisaPart2 += spatialRowNormal * (findNeighborsVal[i]- mean);
        System.out.print("Lisa Part2 computation values at step " + i +
        "value"+ spatialRowNormal * (findNeighborsVal[i]- mean) + "\n");
        System.out.print("Lisa Part2: " + lisaPart2+ "\n");

        iv multiply part1 and part2
            computeLisa = lisaEqPart1 * lisaPart2;
3. log the information for each node with LISA value
   System.out.print("Bolt Information for node: " + vAValues[0] +"," +
  vAValues[1] + "," + vAValues[2] + "Va: "+vA+
  "LISA Value:" + computeLisa + "\n");
4. send this information to next bolt
   collector.emit(new Values(sensorValues[0],computeLisa));
   collector.ack(input);
```

JsonSensorBolt is bolt which consumes the information from SensorLISABolt and write this information into JsonSensorFile which is a text file. It can by anything, database also. In this thesis text file is used as this information is used for monitoring purpose. Which is discussed in detail further in this chapter.

```
1. Take the information emitted by previous bolt and store it in text file from
where this information is read and converted to json form for visualization
     i take information input.getString(0);
         ii split this information dat.split(",");
         iii open the file and write information in the same
                File file = new File(jsonFilePath);
                jsonFileWriter = new FileWriter(file.getAbsoluteFile(),true);
                BufferedWriter bw = new BufferedWriter(jsonFileWriter);
                bw.write(value);
                bw.newLine();
                bw.close();
```

4.2 Monitoring And Visualization Module

The module is written in Java and responsible for use the information generated by storm application, consume it and visualize the network. For monitoring java threads are launch which every minute read the information from file, and write in json file. Also as file gets longer with time so a separate thread is responsible for deleting the information 15 minutes. For visualization d3 and javascript is used. It's a simple web module which will show the network in forms of nodes and edges. Node color represent the LISA values. different colors are used for monitoring purpose for 3 different cases: positive lisa, negative and no value.

4.2.1 Monitoring Network

A java thread is running to get the latest information from file and keep updated json file to be vizualized by d3 application. Following snippet shows the thread is running continuously after 100 sec sleep time.

```
1. thread is run continuously to read information from text file and convert it into
JSON every 60 seconds
while(true)
   SensorInfoJson sensorJsonInfo = new SensorInfoJson(sensorsIds);
   sensorJsonInfo.start ();
   TimeUnit.SECONDS.sleep(60);
```

The links are defined based on information from topology file. as JsonObject. Following snippet shows how this is implemented.

```
 1 define links based on information stored in csv file
   i. Read the information from file and store
   topology information in 2d
   array like before from file File("/../topologyInformation.txt"));
   ii if(topoFromFile[i][j]==1) then define link
        JsonObject jsonObjectLink = Json.createObjectBuilder()
                                                .add("source", i)
                                                .add("target", j)
                                                .add("value",1)
                                                .build();
                                        jsonArrayBuildLinks.add(jsonObjectLink);
```

Next step is to gather LISA information value for each node at last time stamp and create JsonObject for the same. If LISA Value is not available for a node, it will be storing as "no data" and will be visualized as color change of node on browser. The following listing is for fetching information from file.

```
1. Read the information from file /../jsonSensorFile.txt
where information was written
```

```
from storm application and keep it in array
BufferedReader reader = new BufferedReader(fileReader);
                while((str = reader.readLine()) != null)
                        tokens[countLines++] =str;
2. take the value from last time stamp and look for
node with no value for that time
stamp and store "no value for the same"
if(tokens[0]!=null){
  String[] senseVal = tokens[countLines-1].split(",");
  String lastTimestamp = senseVal[2];
  for(int i=countLines-1;i>=0;i--)
     String word = tokens[i];
     String[] sensorVal = word.split(",");
     word = word.trim();
     if(!word.isEmpty() && sensorVal[2].contentEquals(lastTimestamp)){
       Boolean foundItem = false;
                for(int j=0;j<countFilterLines;j++){
                String[] sensorCurrentVal = filteredTokens[j].split(",");
                        if(sensorCurrentVal[0].contains(sensorVal[0])){
                                foundItem = true;
                        break;

3. if item found save as it is else if not then
       finalAddedTokens[k] = sensorsIds[k] + ",no data,notimestamp,noValue";
```

And the listing below is creating JsonObject along with co ordinates of node. Fixed parameter is set to true as nodes are fixed in network. Neighbors can change for computing LISA.

```
//JSON links were defined before based on topology file information now json nodes
are defined and lisa value computed is assigned.
JsonArrayBuilder jsonArrayBuild= Json.createArrayBuilder();

1. for each value read from file for node check if LISA is less than 0 or greater
than 0, if less than then set orange color else blue color
     int sensorLisaValue;
     if(Double.parseDouble(senseVal[3])<0.0)
     sensorLisaValue = (int) (225 * (Math.pow(Double.parseDouble(senseVal[3]),20)));
     else
       sensorLisaValue = 123;
2. define the json node to save it in file
     JsonObject jsonObject1 = Json.createObjectBuilder()
                .add("sensorId", senseVal[0])
                .add("group", sensorLisaValue)
                .add("x", coOrdinates[i][0] )
                .add("y", coOrdinates[i][1])
                .add("fixed", true)
                .build();
                .build();
                jsonArrayBuild.add(jsonObject1);
3. build the nodes and links
          jsonArrayNode = jsonArrayBuild.build();
          jsonArrayLinks = jsonArrayBuildLinks.build();
```

These JSON objects are written in file which is used by visualizer.

4.2.2 Visualization Module

This module is implemented in d3, html, javascript. Nodes and links defined in json file are used and node color is defined by lisa value.

```
d3.json("jsonDataD3.json", function(error, graph) {
  force
      .nodes(graph.nodes)
      .links(graph.links)
      .start();

  var link = svg.selectAll(".link")
      .data(graph.links)
    .enter().append("line")
      .attr("class", "link")
      .style("stroke", function(d) { return color(d.value); });

  var node = svg.selectAll(".node")
      .data(graph.nodes)
    .enter().append("circle")
      .attr("class", "node")
      .attr("r", 5)
      .style("fill", function(d) { return color(d.group); })
      .call(force.drag);

  node.append("title")
      .text(function(d) { return d.name; });

  force.on("tick", function() {
    link.attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    node.attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; });
  });
});
```

Figures 4.3, 4.4 shows the visualization of 16 nodes and 1600 nodes in d3 respectively.

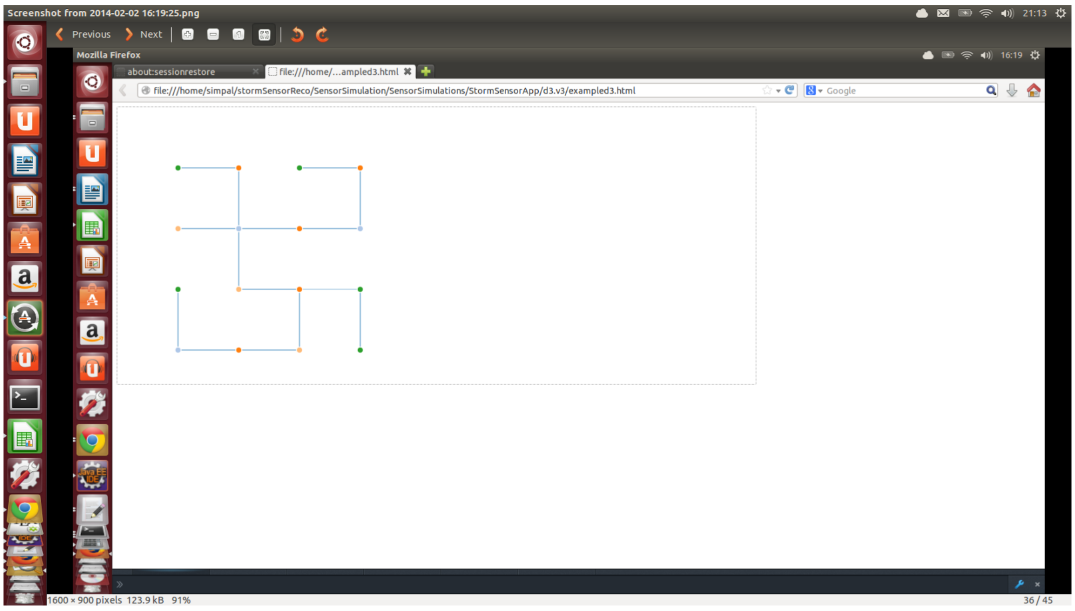

Screenshot from 2014-02-02 16:19:25.png
Previous
Next
Mozilla Firefox
about:sessionrestore
file:///home/...ampled3.html
file:///home/simpal/stormSensorReco/SensorSimulation/SensorSimulations/StormSensorApp/d3.v3/exampled3.html
Google
1600 × 900 pixels 123.9 kB 91%
36 / 45

FIGURE 4.3: Visualizing 16 nodes network

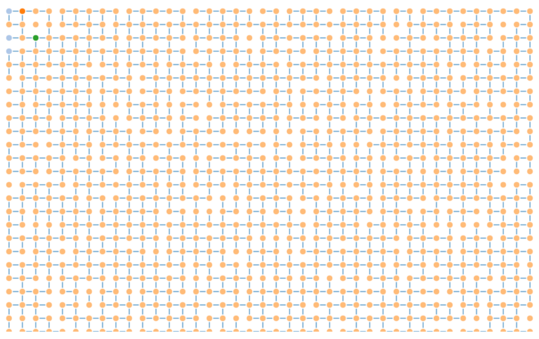


FIGURE 4.4: 1600 nodes network visualization

4.3 Network Simulator

Network simulator module is java application which is responsible for simulating the sensor node values for testing our application. Network simulator basically prepare the information to be processed by storm application. In real world scenario this either could be replaced by another application or could be used to read information from sensors installed and keep it at base station where spout is installed. Number of Java threads are running depending on number of spouts so that each thread will generate information gathered from sensors attached to a base station. For testing purpose gaussian distribution is used for generating random sensor values. Another responsibility of this module is to delete the information which is older than some timestamp e.g. 15 minutes as the data is processed by storm application in real time, it's not required to store it in db or files.

```
1. Read the information for nodes from csv file and get sensor ids
FileReader fr = new FileReader((args.length > 0) ? args[0] : "nodes_test.csv");
   String[] sensorsIds = parseCsv(fr, ",", true);

2.Start the sensor threads to write sensor values for simulation used by
storm application
SensorDataSet sensorStart = new SensorDataSet (sensorsIds[i], fileSensor);
    sensorStart.start ();
             }
```

In the followig code snippet thread check if the number of lines in file exceeds n*5 (last 5 minutes values generated by sensors) then delete the content and then generate new values and write in empty file. The information stored for each sensor is nodeId, timestamp and value.

```
synchronized void sense () throws InterruptedException {
1. if file doesnt exists, then create it
         if (!file.exists())
           file.createNewFile();
         else
2. Read the lines from file and see if it exceeds N*5 then delete the previous
content as sensor values before 5 minutes are not required to be processed.
InputStream is = new BufferedInputStream(new FileInputStream("logDataSensor.txt"));
        while ((readChars = is.read(c)) != -1)
            empty = false;
            for (int i = 0; i < readChars; ++i)
                if (c[i] == '\n')
                    ++count;
//deleting data from file if it exceeds N*5 lines for N sensors
       if(count>N*5){
                PrintWriter writer = new PrintWriter(file);
                writer.print("");
                writer.close();
3. write the new random value generated in file for sensors
   DateFormat dateFormat = new SimpleDateFormat("yyyyMMddHHmm");
  //get current date time with Date()
   Date date = new Date();
   //System.out.println(dateFormat.format(date));
   //get current date time with Calendar()
   Calendar cal = Calendar.getInstance();
  // System.out.println(dateFormat.format(cal.getTime()));
  String value = sensId + "," + (randomParameterVal.nextGaussian())
  + "," + dateFormat.format(cal.getTime());
  FileWriter fw = new FileWriter(file.getAbsoluteFile(),true);
  BufferedWriter bw = new BufferedWriter(fw);
  bw.write(value);
  bw.newLine();
```

Chapter 5

Evaluation

In this chapter the performance measurements and interpretation of these measures are discussed. The problems encountered are reviewed while designing and implementing our project, the limitations project has and how these limitations could be resolved.

5.1 Test Set up

The software configurations used are Ubuntu 12.04, Java 1.7.0, zookeeper-3.4.5, Storm-0.8.2, d3, eclipse kepler. Storm Cluster with 3 nodes was set up. Following listings show the config files set up for the same

```
########### These MUST be filled in for a storm configuration
storm.zookeeper.servers:
     - "134.21.245.67"
     - "134.21.73.201"
     - "134.21.73.210"

storm.local.dir: "/tmp/"
java.library.path: "/usr/local/lib/"
nimbus.host: localhost
nimbus.task.launch.secs: 240
supervisor.worker.start.timeout.secs: 240
supervisor.worker.timeout.secs: 240
supervisor.slots.ports:
      - 6700
      - 6701
      - 6702
      - 6703
```

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
```

```
initLimit=5
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=2
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/home/simpal/simpal/stormtemp
# the port at which the clients will connect
clientPort=2181
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
server.1=134.21.245.67:5181:5888
server.2=134.21.73.201:5181:5888
server.3=134.21.73.210:5181:5888
```

Two categories of tests were conducted:

- 16 nodes network (most of the tests were conducted with this network shown in chapter4.
- 1600 nodes network test for scalability test

5.2 Performance Measures

To measure the performance of 16 nodes network on storm cluster multiple tests were done with different scenarios.

- Run the topology with real time network simulator data and store the information in file. It was verified from logs that neighbours are correct and LISA value is getting calculated with correctly
- For statistical significance the experiment was repeated 100 times considering different numbers. Following scenarios were considered:
  - Compute LISA for each node considering random neighbours i.e. 1, 2, 4, 8, 12 neighbours for each node.

  - Compute LISA using different spouts and bolts combinations i.e. (2 spouts, 2 bolts), (4 spouts, 4 bolts), (1 spout, 4 bolts)
  - Introducing Anomaly in the 16 node network real time and visualization

- The speed of computation for above experiments was considered to compare performance
- For scalability one experiment was done by taking 4 random neighbours for a node in 1600 nodes network

These test cases, their findings and results are compared now in detail in next section. Figure 5.1 the example topology set up for testing in this thesis.

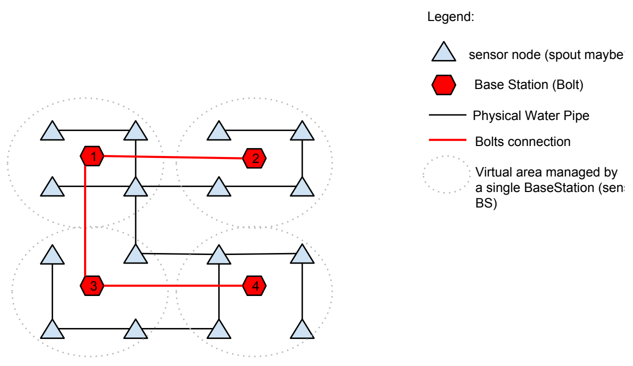


FIGURE 5.1: Example topology with 16 nodes with 4 base stations

5.2.1 Test Case 16 node Actual LISA

16 nodes network was run for computing LISA with actual(real) neighbours. Figure 5.2 shows the time taken to compute LISA with real neighbours.

Table 5.1 shows the LISA values of each node with actual neighbours for 16 node network. Figure shows the ScatterPlot where it is visible the negative LISA values lies in region of Low-High and High-Low. LISA with positive value means similarity and LISA with -ve value means dissimilarity.

Figure 5.3 shows the ScatterPlot of LISA with real neighbours.

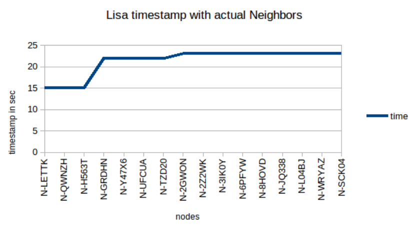


FIGURE 5.2: Test case LISA computation time with real neighbours for each node in 16 nodes network

Node	Value	LISA
N-2GWON	0.47	0.14
N-2Z2WK	-1.73	0.14
N-3IK0Y	2.08	1.87
N-6PFYW	-1.13	0.22
N-8HOVD	-0.04	0.01
N-GRDHN	-0.61	-0.82
N-H563T	-0.28	0.03
N-JQ338	1.22	1.79
N-L04BJ	1.70	0.40
N-LETTK	0.69	-0.13
N-QWNZH	-0.16	-0.03
N-SCK04	-0.18	0.17
N-TZD20	-1.40	-1.21
N-UFCUA	-0.29	-3.13E-004
N-WRYAZ	1.10	-0.13
N-Y47X6	-1.29	1.79

TABLE 5.1: LISA values of nodes with actual neighbours

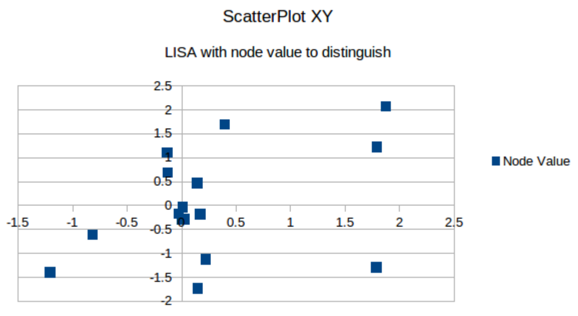


FIGURE 5.3: Scatter Plot of LISA with real neighbours for each node in 16 nodes network

5.2.2 Test Case 16 node with fake random neighbours 100 times

Second test case was to run the 16 node network and compute LISA with various fake neighbours for each node. These fake neighbours were chosen randomly by storm application and this was repeated for 100 permutations for each node. Figure 5.4 shows the result of LISA computation for 100 permutations with fake neighbours.

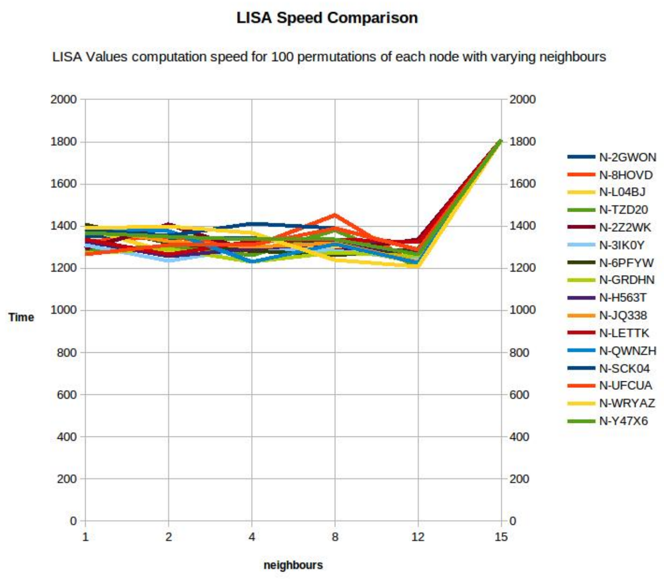


FIGURE 5.4: Speed of LISA computation for 100 permutations of each node

Table 5.2 shows the speed for computing LISA values with various neighbours for 100 permutations of each node.

Figure 5.5 shows the result in graphical form, how much time it took to compute LISA for each node in taking more than 100 permutations of random neighbours.

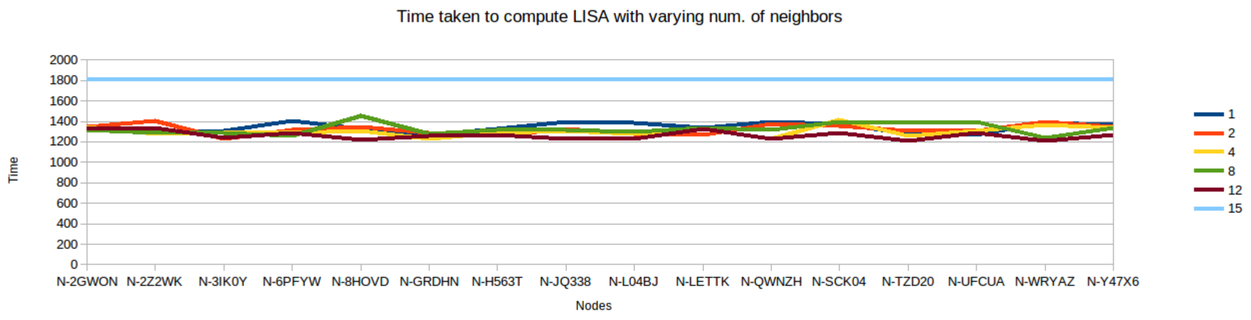


FIGURE 5.5: Time taken to compute LISA for each node 100 times with various random neighbours

Node	1(in secs)	2(secs)	4(secs)	8(secs)	12(secs)	15(in secs)
N-2GWON	1353.38	1342.53	1344.654	1321.05	1327.81	1811.11
N-2Z2WK	1291.10	1408.50	1280.24	1284.65	1337.67	1811.11
N-3IK0Y	1310.90	1233.43	1294.85	1284.65	1243.08	1811.11
N-6PFYW	1406.69	1317.83	1284.45	1261.81	1290.20	1811.11
N-8HOVD	1326.81	1347.88	1302.74	1450.87	1214.55	1811.11
N-GRDHN	1268.01	1292.19	1231.41	1274.11	1257.05	1811.11
N-H563T	1327.76	1258.66	1291.84	1320.26	1269.94	1811.11
N-JQ338	1391.72	1327.24	1302.74	1319.15	1230.98	1811.11
N-L04BJ	1389.11	1274.15	1280.24	1301.33	1227.30	1811.11
N-LETTK	1335.87	1263.47	1327.02	1335.90	1324.42	1811.11
N-QWNZH	1394.61	1374.75	1231.41	1314.14	1226.30	1811.11
N-SCK04	1379.67	1353.27	1411.22	1390.58	1286.22	1811.11
N-TZD20	1281.40	1305.36	1260.47	1381.84	1208.73	1811.11
N-UFCUA	1264.94	1310.44	1309.82	1391.41	1286.54	1811.11
N-WRYAZ	1388.12	1396.96	1366.13	1238.77	1207.83	1811.11
N-Y47X6	1368.73	1348.86	1336.25	1335.11	1264.98	1811.11

Table 5.2: Speed of LISA computation for 100 permutations with random neighbours of each node

It was observed from LISA values that time taken to compute LISA for nodes lies between 1200-1400 except when number of neighbours chosen were 15. Following figures show the time taken with various neighbours.

Figure 5.6 shows the results when only 1 neighbour was selected per node randomly by application.

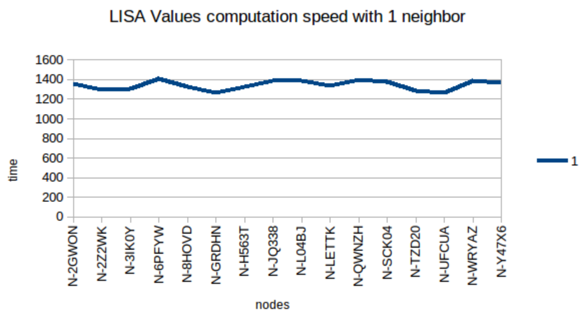


Figure 5.6: time for LISA values with 1 neighbour for each node randomly

Figure 5.6 shows the results when 2 neighbour were chosen randomly per node for more than 100 permutations.

Figure 5.6 shows the results when 4 fake neighbours selected for each node randomly.

For dense network Figure 5.8 shows the results when 8 neighbours were chosen per node randomly by application.

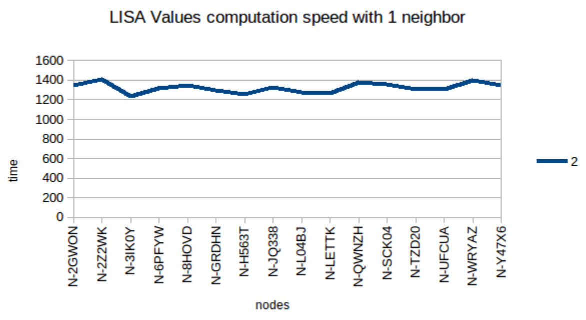


FIGURE 5.7: time for LISA values with 2 neighbours for each node randomly

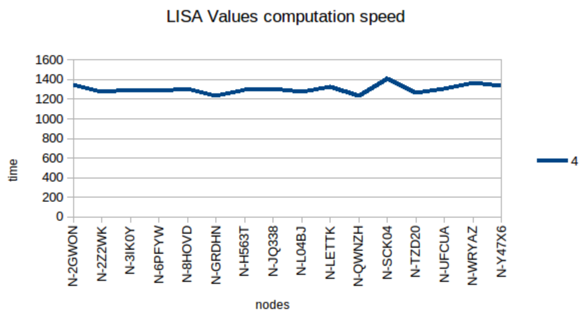


FIGURE 5.8: time for LISA values with 4 neighbours for each node randomly

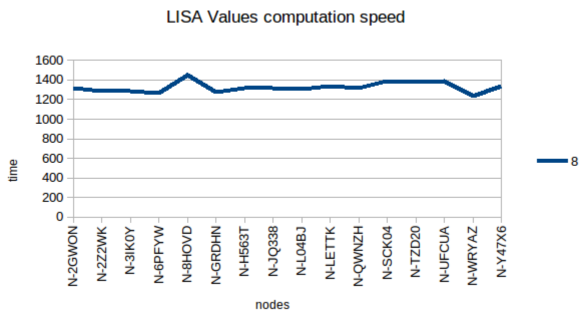


FIGURE 5.9: time for LISA values with 8 neighbours for each node randomly

Figure 5.9 and 5.10 shows the results when very dense network is selected as neighbours per node randomly by application 12 and 15.

Another observation made was that with more neighbours the LISA value is almost same for each node e.g. in following figures it is clear that with 12 neighbours LISA value is almost close to 0. While with 1 neighbour or 2 neighbours the spikes are more, which clearly indicates that LISA value varies depending on neighbouring nodes.

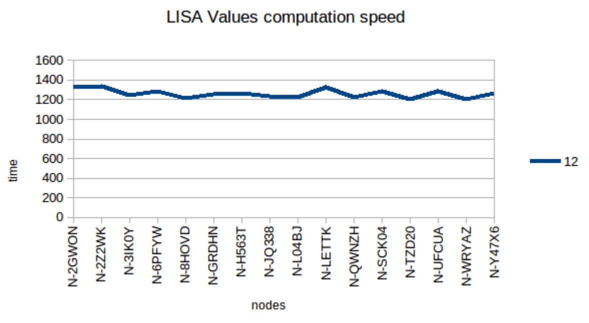


FIGURE 5.10: time for LISA values with 12 neighbours for each node randomly

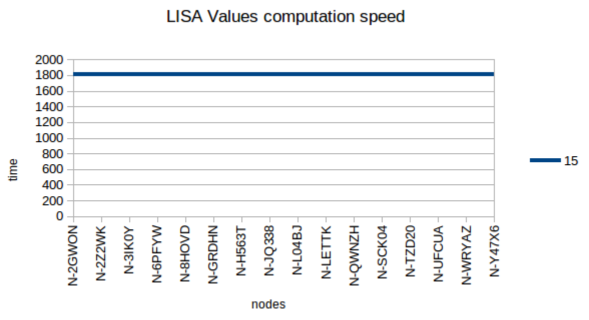


FIGURE 5.11: time for LISA values with 15 neighbours for each node randomly

5.2.3 Test Case 16 node Anomaly introducing real time and visualization

Initially the network has no anomaly. All nodes have same value shown in table 5.3

Figure 5.11 shows the network without anomaly.

Then an anomaly was introduced node N-UFCUA. Changing value at this node results in change is LISA for nodes which have N-UFCUA as neighbour. The table 5.4 shows the result with LISA values after anomaly introduction.

Figure 5.12 and 5.13 shows the network after introducing the anomaly N-UFCUA by changing vale to 3.6. This is shown in picture with light red color. As other nodes where this node is a neighbour node also get affected. These are shown with dark orange color in figure 5.13.

LISA value for the node where anomaly was introduced computed immediately but it's neighbour nodes took time to get affected.Difference was 3 seconds between anomaly node LISA impact and neighbour nodes.

Node	Value	LISA
N-UFCUA	0.6	0.94
N-GRDHN	0.6	0.94
N-2GWON	0.6	0.94
N-8HOVD	0.6	0.94
N-L04BJ	0.6	0.94
N-2Z2WK	0.6	0.94
N-3IK0Y	0.6	0.94
N-Y47X6	0.6	0.94
N-JQ338	0.6	0.94
N-SCK04	0.6	0.94
N-QWNZH	0.6	0.94
N-LETTK	0.6	0.94
N-H563T	0.6	0.94
N-WRYAZ	0.6	0.94
N-TZD20	0.6	0.94
N-6PFYW	0.6	0.94

TABLE 5.3: LISA values without anomaly

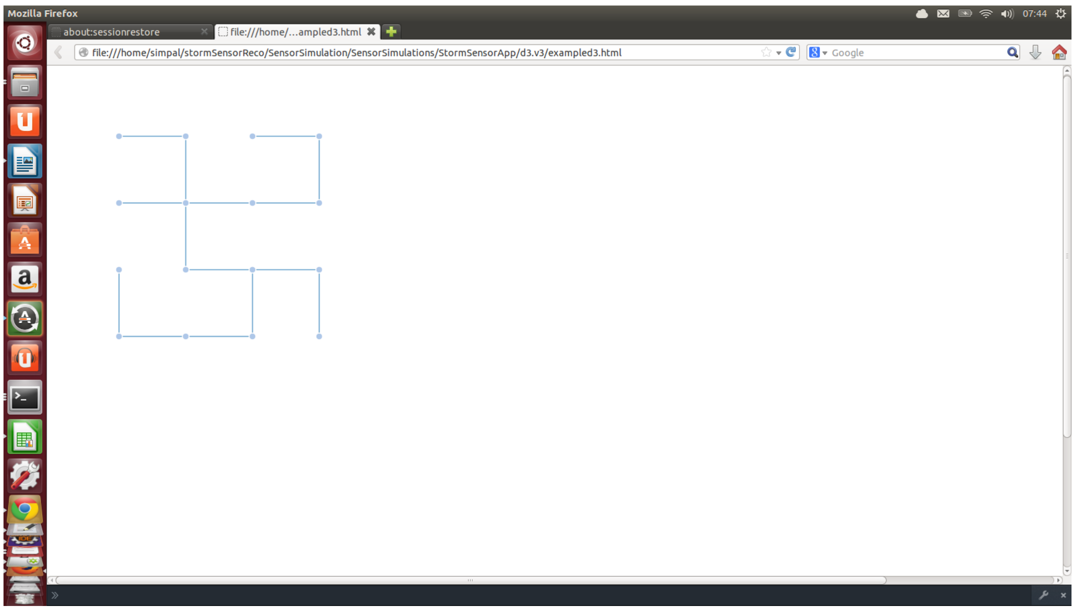

Mozilla Firefox
about:sessionrestore
file:///home/...ampled3.html
file:///home/simpal/stormSensorReco/SensorSimulation/SensorSimulations/StormSensorApp/d3.v3/exampled3.html
Google
07:44

FIGURE 5.12: Network without anomaly

Node	Value	LISA
N-UFCUA	3.6	-1.17
N-GRDHN	0.6	0.94
N-2GWON	0.6	-0.54
N-8HOVD	0.6	0.94
N-L04BJ	0.6	0.94
N-2Z2WK	0.6	0.94
N-3IK0Y	0.6	0.94
N-Y47X6	0.6	0.94
N-JQ338	0.6	0.94
N-SCK04	0.6	0.94
N-QWNZH	0.6	0.94
N-LETTK	0.6	-0.53
N-H563T	0.6	0.94
N-WRYAZ	0.6	0.94
N-TZD20	0.6	0.94
N-6PFYW	0.6	-0.54

TABLE 5.4: LISA values after introducing anomaly

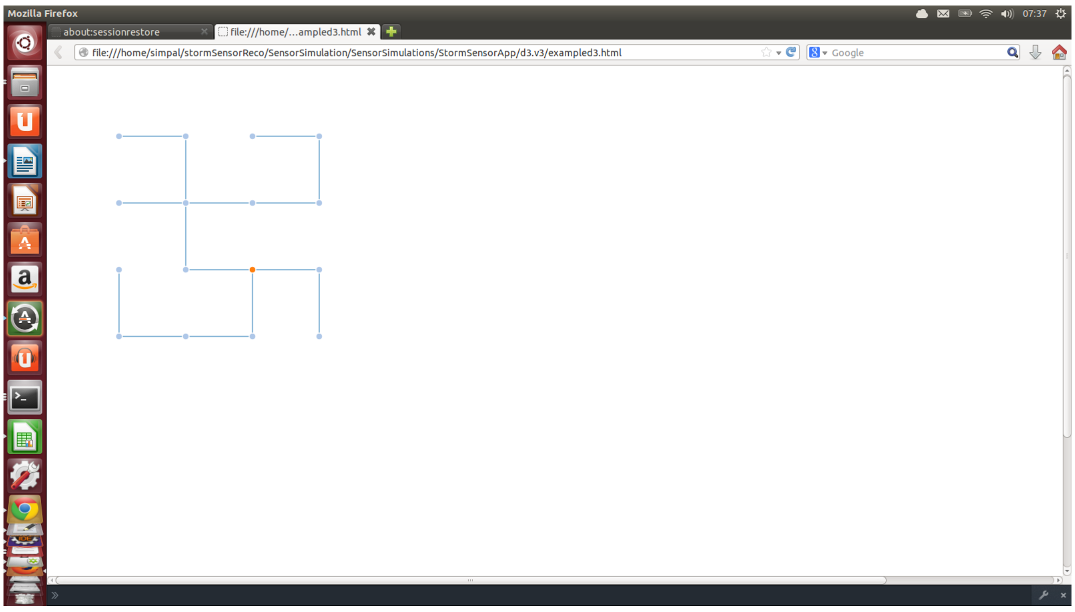

Mozilla Firefox
about:sessionrestore
file:///home/...ampled3.html
file:///home/simpal/stormSensorReco/SensorSimulation/SensorSimulations/StormSensorApp/d3.v3/exampled3.html
Google
07:37

FIGURE 5.13: Network with anomaly

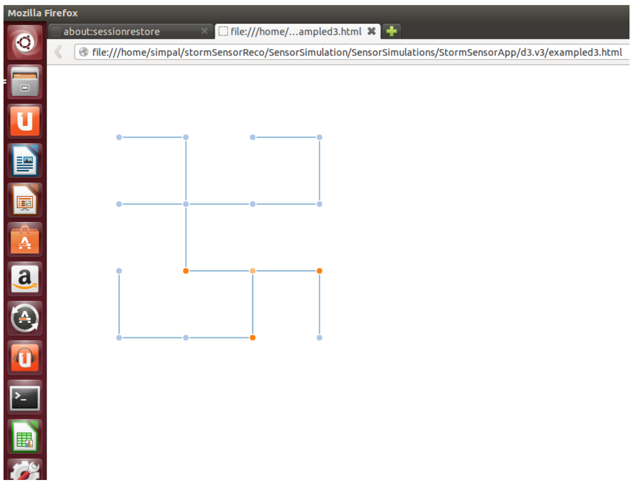

Mozilla Firefox
about:sessionrestore
file:///home/...ampled3.html
file:///home/simpal/stormSensorReco/SensorSimulation/SensorSimulations/StormSensorApp/d3.v3/exampled3.html

FIGURE 5.14: Network with anomaly and neighbours

5.2.4 Test case with various spouts and bolts combinations

This test was done to check if increasing and decreasing the spouts and bolts does impact the performance of LISA computation or not? In this test case for 16 nodes network various combinations tested were: (4 spouts, 4 bolts),(2 spouts, 4 bolts), (1 spout, bolts). (4 spouts, 4 bolts) gave the better performance. (1 spout, 4 bolts) was quite slow and (2 spouts, 4 bolts) is also quite slower. Increasing the spouts and bolts in proportional to each other gives better performance.

5.2.5 Test Case 1600 node LISA for a node with 4 fake random neighbours 1000 times

To test scalability test with very large network was done with 1600 nodes, 1000 times for one node. With one supervisor and 3 workers it was very slow. Takes almost 1 minute to write value for 1 node in file. Figure 5.14 shows the result of this experiment.

The reason for slowness was number of workers and supervisors. As it was tested with 16 nodes network also with less number of spouts it gets slower. So for 1600 nodes network 400 spouts with 400 bolts will give better performance. The avg time for computing LISA normally is 13.03 while for statistical testing for 1600 node it's very slow. Even for one node it takes 42.75 seconds. Figure 5.16 shows this comparison.

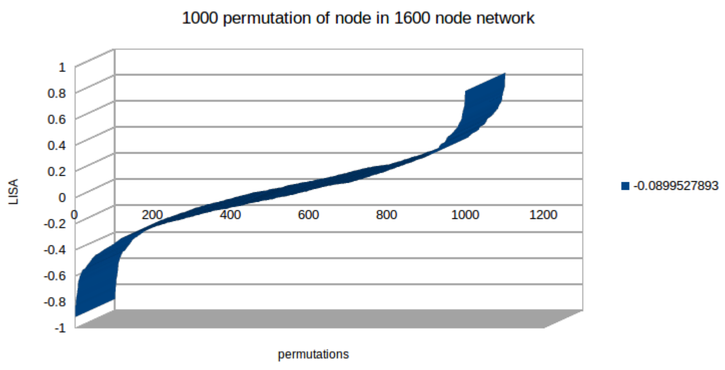


FIGURE 5.15: 1600 nodes network 1000 LISA Values with 4 random neighbours

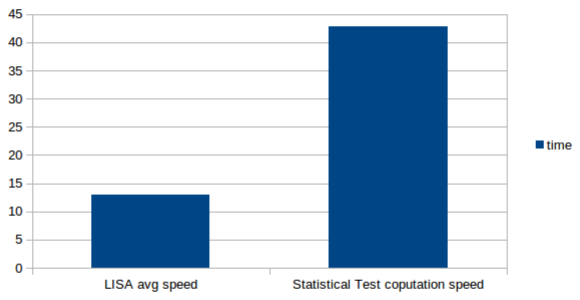


FIGURE 5.16: 1600 nodes network 1000 LISA Values with 4 random neighbours comparison to normal LISA Computation

5.3 Discussions

5.3.1 Problems encountered

- In the beginning while setting up storm cluster faced some problem with version 0.8.1 so switched to 0.8.2. Otherwise setting up storm cluster was easy.Sometimes supervisor was getting killed automatically which was sorted out by running it with sudo.
- When some tuples fail, there is no way to detect which bolt exactly failed. Only at spout it was seen that tuples failed. collector.ack was used but still not enough information is given by storm logs that clearly states where information processing is failing. A custom logging was used for clearly identifying the point of failure.
- Storm is known for scalability but speed of computation depends on hardware of machine and computation performed. e.g. processing with strings or integer value is faster but as in this thesis mathematical computations values of type Double were used,that makes the processing slower. Also if application need more

information to be read from disks then hardware of machine where spout or bolt is running limits the speed of computation.

5.3.2 Limitations

- All information is processed using text files which includes couple of disk read write operations which makes processing slower
- Information for each node is flowing in whole network, it could be good if values can be directly forwarded to bolt which requires the information
- Values are shown and calculated with same time stamp. If there is gap of 1 second it will not be considered.

Chapter 6

Findings and Conclusion

6.1 Findings

The topology with 16 nodes was run for 100 permutations of random neighbours of each node. The logs were analysed and following major findings were made from these tests.

- Anomaly detection takes place simultaneously to LISA computation. Twitter storm application is efficient tool to detect anomalies.
- Scatter Plot clearly depicts the anomalies i.e -ve LISA value regions and +ve LISA value regions are easily separable. LISA statistics is very effective way to detect anomalies and experimental outcomes can be shown with ease.
- Sparse and Dense network impact: Computing time for LISA value increases when its very dense network e.g when all nodes are connected to each other it takes equal time for each node. With 15 neighbours , time taken was 1811 seconds per 100 values(average time on each node) while with four neighbours it's 1303 secs per 100 values(average time on each node). The variation in time for other neighbours shows us that LISA computation speed depends on location of neighbours chosen.
- While Twitter storm proved and showed good performance in anomaly detection, it was not that promising for statistical significance. For smaller network like 16 nodes it's performance was good, but for larger network e.g 1600 nodes it was significantly slow. Avg time for LISA computation/anomaly detection was 13 secs while for statistical testing(taking 1000 permutations of 4 randomly selected neighbours) it took 42 secs for one node per value. So for running topology of 1600nodes network with 1000 permutations taking 4 randomly selected neighbours of each node it might take 2-3 full days. It concludes that instead of using twitter

storm for these tests, these values can be stored in some database e.g SciDb and analysed by some other statistical tool.

6.2 Conclusion

Main goal of this thesis was to find solution for processing massive real time data coming from sensors and detect anomaly in WDN. Solution based on LISA algorithm using storm technology, sensor simulator and visualizer was proposed and tested on 16 nodes network with different scenarios (using random fake neighbours with 100 permutations, introducing anomaly manually, tried with 1600 node network etc.) It was observed that performance doesn't get much impacted with sparse or dense network but in which region neighbours lie affects the speed of computation. This was also observed that less number of spouts decrease the performance. Increasing the number of nodes will increase more workers and will improve performance for larger networks.

Processed information was visualized in browser which is refreshed every 5 seconds to show if any node has LISA less than 0. Value less than 0 meant that there is some node with very high or low value as compared to neighbours. The anomaly and neighbours impacted are shown with different color nodes. This information is really helpful to detect and fix the problem in real WDN.

From the findings it is clear that while Twitter storm performed very good in detecting anomalies, it miserably failed to perform statistical calculations in real time. So it can be concluded that for efficient statistical analysis data should be stored in a database like SciDB and can be processed further with some statistical tool.

6.3 Future Work

For future, it would be good to analyse more parameters than only one parameter. Also it will be useful to store the important information is some database than text file for future use and historical analysis. Also testing on large cluster with thousands of node would be good test for testing scalability.

Appendix A

Running Application on Cluster

```
Here are the instructions for running applications on cluster.
A) For Storm Application
1. Storm-0.8.2, Zookeeper-3.4.5 version were used.
Config files are there in chapter 4 in thesis for
setting nimbus and other nodes ip addresses on storm cluster.
2. Once cluster is set up and zookeeper is running
on all machines then run the nimbus, supervisor and ui with following commands
sudo ./storm nimbus
sudo ./storm supervisor
sudo ./storm ui

if other machines needed to be added in cluster then
run storm supervisor command on other machines too
and it will be added in cluster.
4. git clone the code from repo. for two
applications (storm and network simulator)
https://github.com/simpalK/TwitterStormSensorSimulation
https://github.com/simpalK/SensorSimulations
5. To run storm application
i. executable jar can be used with following command
and providing the path for file where sensor nodes
are writing values along with number of nodes.

./storm jar /home/simpal/stormproject/storm-book-
examples-ch02-getting_started-8e42636/target/Getting-
Started-0.0.1-SNAPSHOT.jar TopologyMain
"/home/simpal/stormproject/storm-book-examples-ch02-
getting_started-8e42636/AllFilesRequiredForApplications"
 "16"

the mentioned jar file can be found on git repo.

ii. To make this executable jar go to application
directory and run the following command:
~/stormproject/storm-book-examples-ch02-
getting_started-8e42636$ mvn -f pom.xml clean install

iii Along with sensor node file there are other files
```

```
where nodes and topology information is stored. These
all files should be kept in same folder with name
kept in folder in repo.

6. Once topology is submitted it can be seen running
on ui interface on browser
localhost:8080 if running locally or else on ip address of nimbus machine.

B) Network Simulator and visualizer
1. git checkout the application
2. open in eclipse
3. run StartSensors.java as java application
4. For Json parser, conversion from text file into
form for d3 visualization we need to provide path for
multiple files which are set in thread. This should
be same as where we kept sensors data file for storm.
Also provide number of nodes as second argument.
5. d3.v3 folder contains file example.html which can
be run in browser to view the example topology, this
is automatically refreshed every 5 seconds.
```

Appendix B

Sample of Files

```
Sample Json File produced by application used for visualization:

{
    "nodes":[
        {
            "sensorId":"N-H563T",
            "group":123,
            "x":100,
            "y":400,
            "fixed":true
        },
        {
            "sensorId":"N-QWNZH",
            "group":123,
            "x":200,
            "y":400,
            "fixed":true
        },
        ....
                {
            "sensorId":"N-UFCUA",
            "group":5331,
            "x":300,
            "y":300,
            "fixed":true
        },
        {
            "sensorId":"N-6PFYW",
            "group":123,
            "x":400,
            "y":300,
            "fixed":true
        }
        .....
        ,
    "links":[
        {
            "source":0,
            "target":0,
            "value":1
        },
        {
            "source":0,
            "target":1,
            "value":1
        },
        {
            "source":0,
            "target":4,
            "value":1
        }
        ...
```

```
Input file for sensor application where network
simulator is writing sensor data in real time:

N-QWNZH,-0.9009381318652095,201404300815
N-LETTK,-1.263020203328888,201404300815
N-8HOVD,-1.7482555933371529,201404300815
N-2GWON,-1.3717426141897777,201404300815
N-SCK04,-1.0614616613219752,201404300815
N-6PFYW,-0.03128790080169101,201404300815
N-TZD20,-0.8203954222043047,201404300815
N-UFCUA,0.2536580467640717,201404300815
N-WRYAZ,-1.6295444386583953,201404300815
N-3IKOY,-1.3955691722242054,201404300815
N-JQ338,1.170798865371803,201404300815
N-Y47X6,-0.5128660555156468,201404300815
N-GRDHN,-0.35127349118192885,201404300815
N-2Z2WK,-2.651040225057969,201404300815
null,-1.7855808729358749,201404300815
N-L04BJ,-1.1691622922460019,201404300815
N-Y47X6,0.4992334078184451,201404302038
```

```
Sample of topology information file produced by
network simulator which reads values from csv file,
parse it and write the information in this file which
is used by storm application to detect neighbours and
for monitoring purpose also:

1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0
1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0
0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0
0,0,1,0,0,1,1,1,0,0,0,0,0,0,0,0
0,0,0,1,0,0,1,1,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0
0,0,0,0,0,1,0,0,1,1,1,0,0,1,0,0
0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0
0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1
0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0
```

```
0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1
```

```
Storm application writes the LISA value of nodes in
jsonSensorFile as below
N-GRDHN,0.6,201403220638,0.9374999999999999
N-UFCUA,0.6,201403220638,0.9374999999999999
N-JQ338,0.6,201403220638,0.9374999999999999
N-8HOVD,0.6,201403220638,0.9374999999999999
N-QWNZH,0.6,201403220638,0.9374999999999999
N-8HOVD,0.6,201403220638,0.9374999999999999
N-L04BJ,0.6,201403220638,0.9374999999999999
N-3IKOY,0.6,201403220638,0.9374999999999999
N-UFCUA,0.6,201403220638,0.9374999999999999
N-GRDHN,0.6,201403220638,0.9374999999999999
N-2Z2WK,0.6,201403220638,0.9374999999999999
.....
```

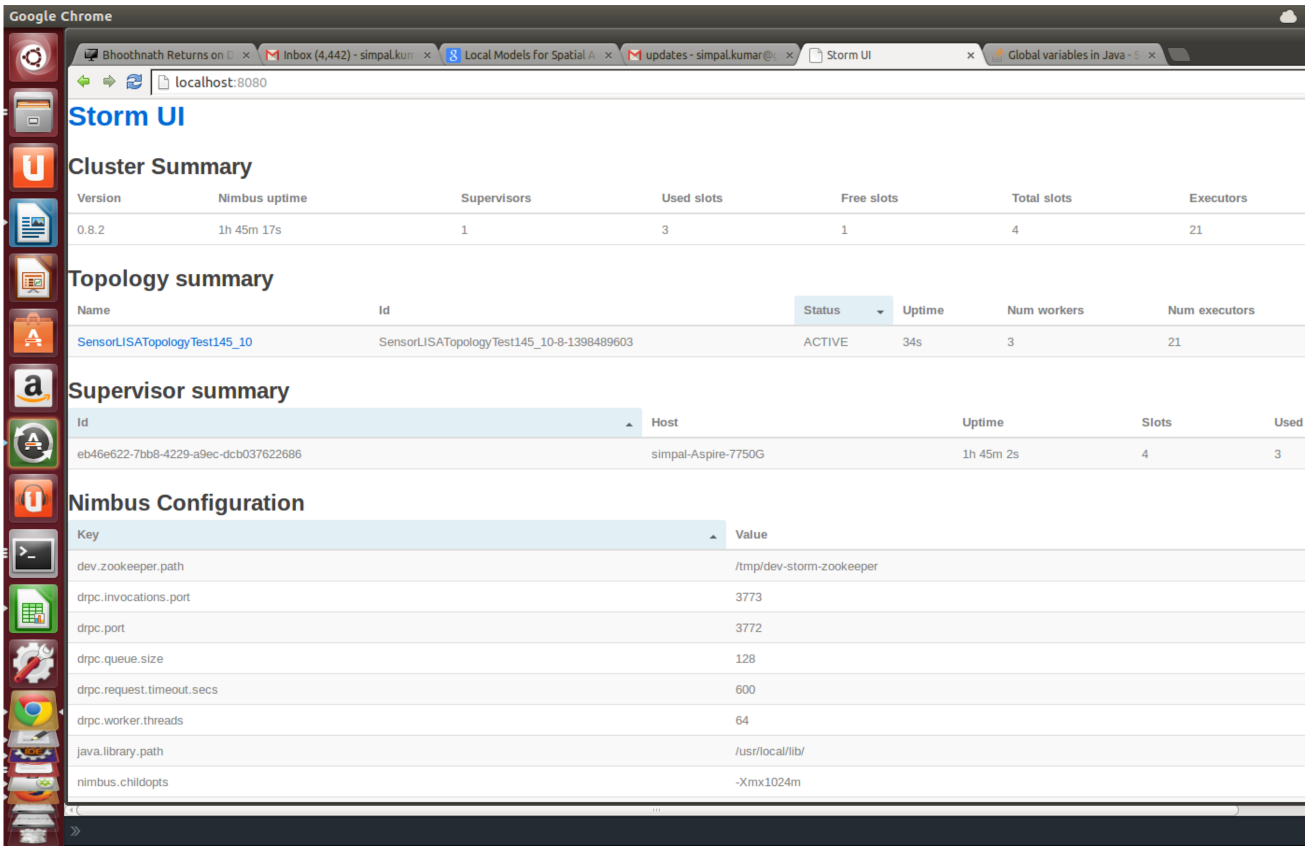
```
Logs of storm application for LISA computation step

2014-03-23 05:13:18 task [INFO] Emitting:
SensorGetter default [0001001100000000,
N-6PFYW,-1.1260429155929481,201403220638:N-
UFCUA,-0.29114990842834665,201403220638:N-
SCK04,-0.17732269034052123,201403220638:,
0.010557984123472272, 1.2726941243199128,
201403220638]
2014-03-23 05:13:18 STDIO [INFO] Find Neighbors: -0.1642343179055442
2014-03-23 05:13:18 STDIO [INFO] Mean: 0.010557984123472272
2014-03-23 05:13:18 STDIO [INFO] Variance: 1.2726941243199128
2014-03-23 05:13:18 STDIO [INFO] Number of Neighbors: 2
2014-03-23 05:13:18 STDIO [INFO] Bolt Information for
node: N-H563T,-0.28234801313940555,201403220638Va:
-0.28234801313940555LISA Value:0.02561153948028165
2014-03-23 05:13:18 task [INFO] Emitting:
SensorGetter __ack_ack [-1876144461165842219
-4257977569096191399]
2014-03-23 05:13:18 task [INFO] Emitting: SensorBolt2
default [N-H563T,-0.28234801313940555,201403220638,
0.02561153948028165]
2014-03-23 05:13:18 task

......
2014-03-23 05:14:24 task [INFO] Emitting: SensorGetter default
[0110001000000000, N-LETTK,
0.6937018920545661,201403220638:
N-UFCUA,-0.29114990842834665,201403220638:
N-QWNZH,-0.1642343179055442,201403220638:,
0.010557984123472272, 1.2726941243199128,
201403220638]
2014-03-23 05:14:24 STDIO [INFO] Mean: 0.010557984123472272
2014-03-23 05:14:24 STDIO [INFO] Variance: 1.2726941243199128
2014-03-23 05:14:24 STDIO [INFO] Number of Neighbors: 2
2014-03-23 05:14:24 STDIO [INFO] Bolt Information for
```

```
node: N-QWNZH,-0.1642343179055442,201403220638Va: -0.1642343179055442
LISA Value:-0.02679771260087365
2014-03-23 05:14:24 task [INFO] Emitting: SensorBolt2
default [N-QWNZH,-0.1642343179055442,201403220638,
-0.02679771260087365]
2014-03-23 05:14:24 executor [INFO] Proc
```

Appendix C

Storm ui screen shots

The following screen shots are taken while running a topology on cluster.



Storm UI

Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors
0.8.2	1h 45m 17s	1	3	1	4	21

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors
SensorLISATopologyTest145_10	SensorLISATopologyTest145_10-8-1398489603	ACTIVE	34s	3	21

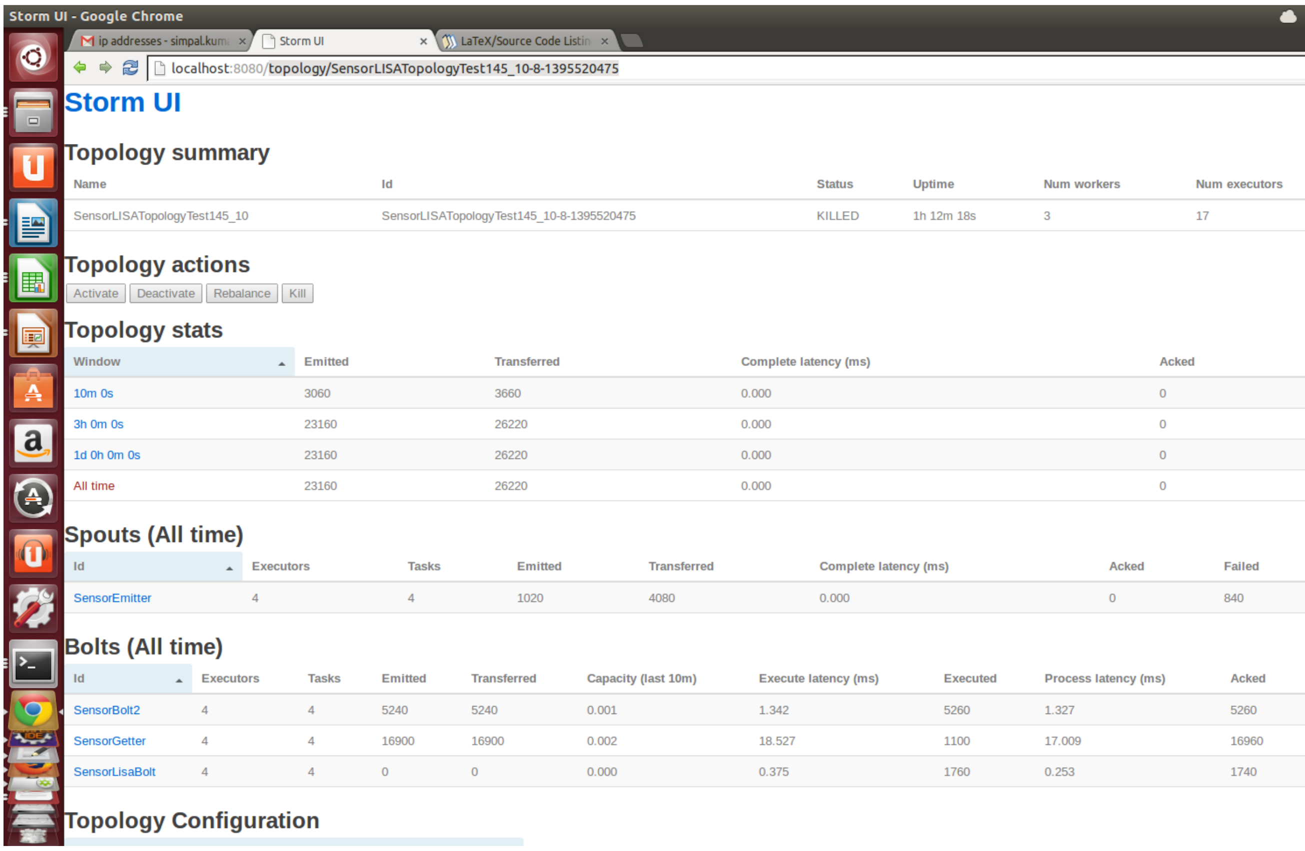
Supervisor summary

Id	Host	Uptime	Slots	Used
eb46e622-7bb8-4229-a9ec-dcb037622686	simpal-Aspire-7750G	1h 45m 2s	4	3

Nimbus Configuration

Key	Value
dev.zookeeper.path	/tmp/dev-storm-zookeeper
drpc.invocations.port	3773
drpc.port	3772
drpc.queue.size	128
drpc.request.timeout.secs	600
drpc.worker.threads	64
java.library.path	/usr/local/lib/
nimbus.childopts	-Xmx1024m

Figure C.1: Example topology with 16 nodes in storm cluster on UI


Storm UI - Google Chrome
ip addresses - simpal.kum
Storm UI
LaTeX/Source Code Listin
localhost:8080/topology/SensorLISATopologyTest145_10-8-1395520475

Storm UI

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors
SensorLISATopologyTest145_10	SensorLISATopologyTest145_10-8-1395520475	KILLED	1h 12m 18s	3	17

Topology actions

Activate Deactivate Rebalance Kill

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked
10m 0s	3060	3660	0.000	0
3h 0m 0s	23160	26220	0.000	0
1d 0h 0m 0s	23160	26220	0.000	0
All time	23160	26220	0.000	0

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed
SensorEmitter	4	4	1020	4080	0.000	0	840

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked
SensorBolt2	4	4	5240	5240	0.001	1.342	5260	1.327	5260
SensorGetter	4	4	16900	16900	0.002	18.527	1100	17.009	16960
SensorLisaBolt	4	4	0	0	0.000	0.375	1760	0.253	1740

Topology Configuration

FIGURE C.2: Storm UI showing details of Topology with latency

Bibliography

[1] J. Bollen, H. Mao, and X. Zeng, “Twitter mood predicts the stock market,” Journal of Computational Science, 2011.

[2] T. Sakaki, M. Okazaki, and Y. Matsuo, “Earthquake shakes twitter users: real-time event detection by social sensors,” in Proceedings of the 19th international conference on World wide web, p. 851–860, 2010.

[3] Thomas M Walski, James G Uber, William Eugene Hart, Cynthia Ann Phillips, and Jonathan W Berry. Water quality sensor placement in water networks with budget constraints. Technical report, Sandia National Laboratories, 2005.

[4] Luc Anselin. Local indicators of spatial associationlisa. Geographical analysis, 27(2):93–115, 1995.

[5] Jonathan Leibiusky, Gabriel Eisbruch, and Dario Simonassi. *Getting Started with Storm, Oreilly* 2012.

[6] Christopher D. Lloyd *Local Models of Spatial Analysis* 2007

[7] Nathanmarz *https://github.com/nathanmarz/storm Storm Wiki* 2012

[8] Data Driven Documents *http://d3js.org/*

[9] Scalable Anomaly Detection for Smart City Infrastructure Networks. Djellel Eddine Difallah , Prof.Philippe Cudré-Mauroux and Sean A. McKenna