Team Toyota

Team Members:

Hyo Jeong, Ambarish Karole, AJ Ruiz, Grace Lee, Harikumar Kumar Sulochana

Faculty Advisors:

Patrick Tague, Jia Zhang

# 

**Table of Contents**

[Introduction](#h.2f0n7c4bvw8j)

[Motivation](#h.cfxl0mq5rl4l)

[Related work](#h.yrk9t3g63but)

[System design](#h.x6rtm323uemr)

[Distributed Querying API](#h.cmgh5nrei1gz)

[Distributed Processing](#h.7f1bznq1o2vz)

[System implementation](#h.nxcgry94a5ng)

[Distributed Querying](#h.msp1k7y6meib)

[Joining a cluster](#h.g883rrhhktwm)

[Querying the Cluster](#h.9fznsjurckpa)

[Informing the Cluster](#h.a47r5bk5wemx)

[Distributed Processing](#h.fsy8qh1cjgv2)

[Background Research](#h.qxvfevnch2up)

[Apache Storm](#h.r4tr4sjqbzlw)

[StormCV](#h.489nc1ebsdf8)

[StormCV Installation](#h.qzdn3kdws0w1)

[Compiling and executing StormCV](#h.v8ynj5g8qps8)

[Existing bugs:](#h.6mtkzkofrfr5)

[Distributed Querying](#h.gcch54laq1iy)

[Experiments and Analysis](#h.usgfu67lxz7f)

[Conclusions and future work](#h.fyhz8r9274yv)

[Appendix](#h.wsweyj30x88)

[Logistical Information](#h.msjfeslkmvty)

## Introduction

Given that more and more vehicles are laden with various sensors and that we are headed to a future where driverless smart cars will be a common sight, there is a pressing need for an infrastructure which will support such services. In order for a vehicle to be truly smart, the optimal use of sensor data (of both the sensors in the vehicle and of other vehicles in the vicinity) is the key. With the proliferation of smart mobile phones, it has already been demonstrated that cloud computing is the answer. But although present cloud solutions are excellent for all the things we hope to achieve for a smart vehicle, the fact of the matter remains that these services are not **real time**. And for a solution to be taken seriously with respect to smart vehicles, its is imperative that it be real time.

A promising approach is for a group of vehicles to create their own **mobile cloud**.

Our solution is driven by increasing storage and processing capacity of mobile devices (which will be present in vehicles as well) and the need to communicate and keep locally relevant content on the mobiles instead of uploading to cloud. Each vehicle will be a node in the mobile cloud.

Some of the other reasons why mobile cloud computing (MCC) is a better solution than traditional cloud computing are:

* Reduced communication delay, reduced spectrum costs, and amply expanded range of applications.
* Smart cars provide far greater storage and processing capabilities over smartphones, thereby network latency is reduced by offloading the cloud processing to local nodes.
* Mobile agents can be both users as well as service providers
* Mobile node can interact and collaborate to sense environment, process data, propagate results and more generally share resources.

## 

## Motivation

Toyota is interested in utilizing mobile cloud computing as it would greatly aid Toyota’s efforts in building semi or fully autonomous vehicles that can enhance driver safety. In addition, the use of mobile cloud computing will enhance collaborative services and in-vehicle entertainment.

Specifically, Toyota’s key use cases are the following:

* TUS 1 Driver Safety - Collaborative sensing (negotiating a blind turn or backing out of parking space)
* TUS 2 Driver Safety - Merge assistant
* TUS 3- Collaborative services - Navigation
* TUS 4- Collaborative services- Emergency assistance
* TUS 5 - Co-ordinated driving/platooning
* TUS 6 - In Vehicle entertainment - Shared content markets

In order to accomplish the use cases by utilizing mobile cloud computing, Toyota needs a middle tier application that provides an infrastructure for both distributed processing of computation needs, as well as a method of real-time distributed querying from multiple databases.

In order to meet the client needs and use cases, we developed a proof of concept for a middle tier application which is capable of providing two key features.

1. A vehicle should be able to request other vehicles in the vehicular cloud to process tasks when there is a need to multitask or when the task becomes very big for it to process.
2. A vehicle should be able to query other vehicles in the vehicular cloud for information when it requires information from others to arrive at a logical conclusion

## 

## Related work

While a lot of research and real world solutions are present for traditional cloud computing , mobile cloud computing (MCC) is relatively a new field. Being in nascent stage we came across several preliminary research publications which address the issues and challenges of MCC. The paper Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges, by Zohreh Sanaei et al, discusses the problem of heterogeneity among devices to implement MCC, while the paper Vehicular Cloud Computing by Mario Gerla discusses the research and design issues to implement MCC.

Therefore, while most papers discuss the challenges and the direction research should take to make MCC a reality, we have implemented MCC using open source projects. Therefore, demonstrating not only the feasibility of MCC but also that MCC can be implemented today using open source technology and research present today.

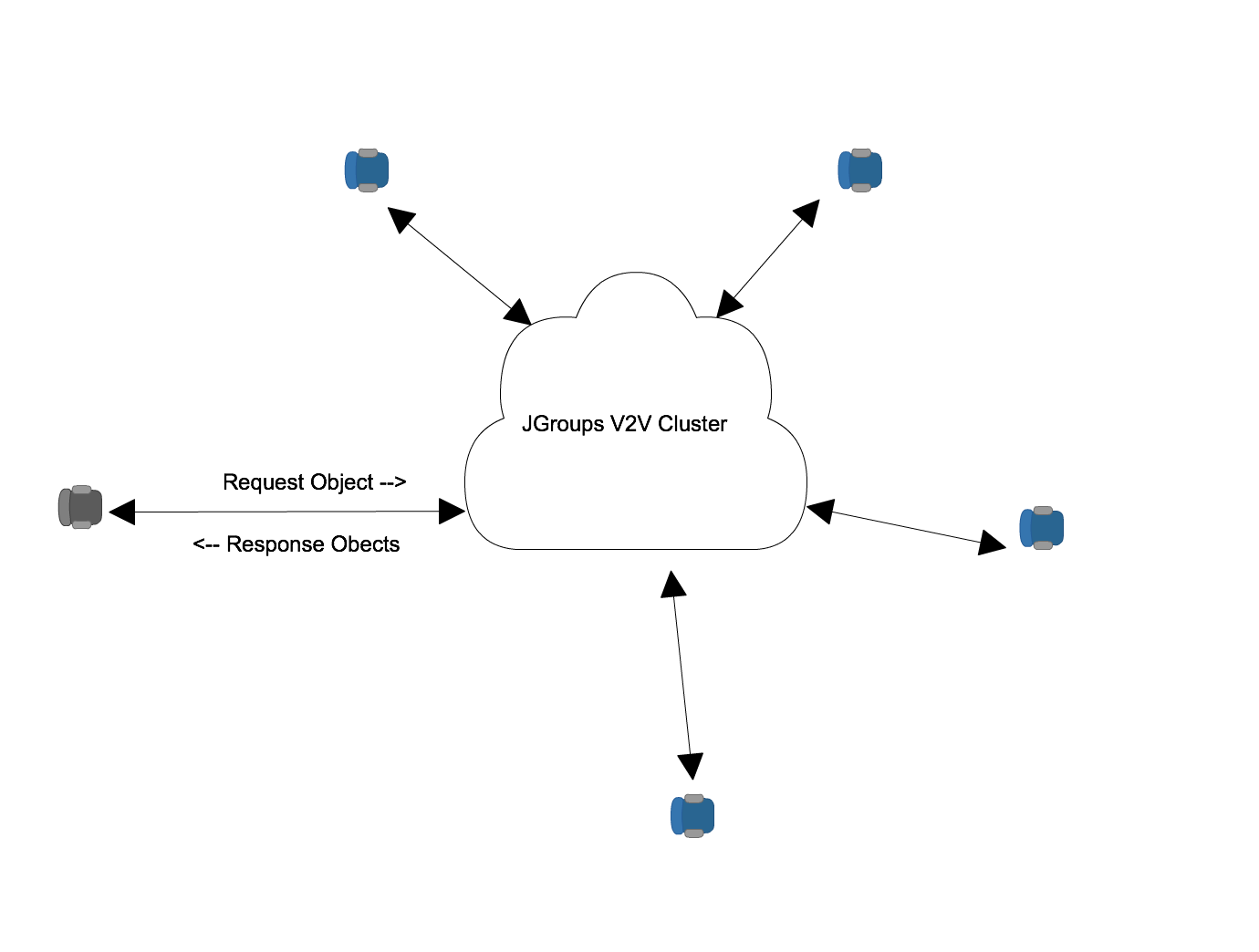
## System design

Two high level requirements evolved from the requirement analysis.

1. A distributed querying API: which could be used to query vehicles in the cloud for specific information
2. A distributed processing API : which could be used to leverage the computing power of multiple vehicles to process information faster by distributing load

### Distributed Querying API

The distributed querying API should allow querying of vehicles for information. This will benefit use cases such as when a vehicle needs information from other vehicles. In a pure distributed querying use-case, a vehicle would query the other vehicles for information and process the information sent back by other vehicles. This would be fine for use cases such as in-vehicle entertainment. But for emergency response that needs to be processed in real time, it would be better to have an API which will inform all vehicles in the cloud of an event thereby enabling other vehicles to take necessary actions. In order to increase performance, it was decided to design the distributed querying API to provide the additional functionality of informing the other nodes in the system about an event. Figure 1 mentioned below shows the way a distributed querying API is supposed to work.



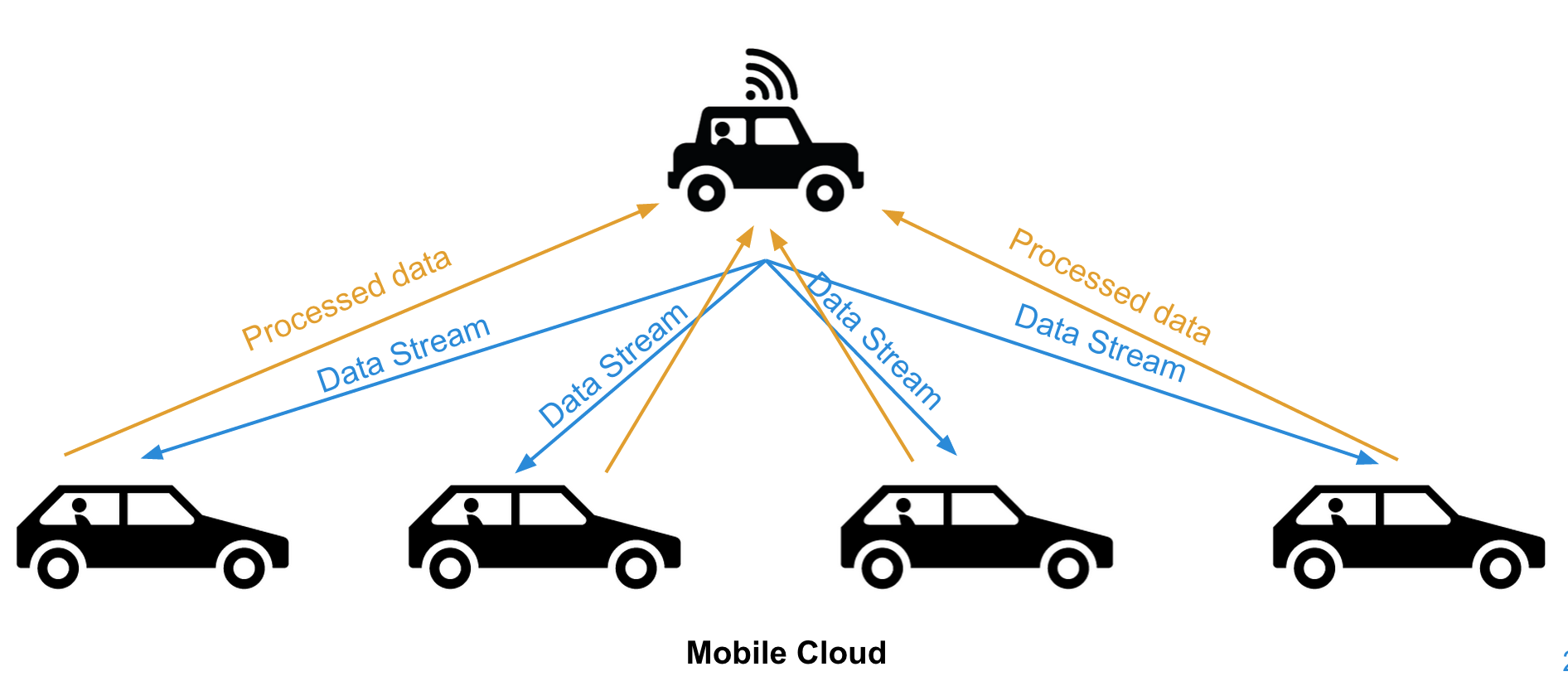
**Figure 1**

In Figure 1 above, the Request and Response Object belong to the same Type. The Request Object can be used to send information to the cluster and the Response Objects received as responses from different vehicles in the cluster can contain the necessary information packaged in the attributes of the object. This is how the API functions in a distributed querying mode. As an offshoo,t the API can also be used to just send information to other vehicles in the cluster. This is possible by providing necessary data in the attribute of the Request object which will be received by other vehicles.

### Distributed Processing

There are many use cases in the self-driving car domain that would benefit from distributed processing. One use case is in-vehicle entertainment. When the passenger wants to download a video from a distribution channel, say Netflix, the car can request nearby cars to download segments of the video in parallel and send them over. This would cut down the download time by n, n being the number of cars. Another use case that benefits from distributed processing is vehicle safety. The LIDAR sensor on the car will collect data that can be used to build a real-time 3D map of the environment to improve situational awareness of the car. The passenger’s car can distribute these LIDAR data to nearby vehicles, and these fleet of vehicles can process the LIDAR data and return parts of the 3D map, which can then be pieced together to create a more complete map of the environment for navigation.

Below is a simple diagram of the system design for distributed processing [Figure 2]. The self driving car identifies the nearby nodes, divides up the data that needs to be processed, and sends them to the nodes. Each node can then process the input data and send the output back to the car that requested the data to be processed.



**Figure 2**

## System implementation

During the initial phase of research and analysis of various open source software for supporting the APIs, it became increasingly clear that it will be difficult or impractical to get an open source software which can provide support for both the Distributed Querying and Distributed Processing APIs. So it was decided to use different stacks for these APIs which also allow them to be deployed independently if required. Provided below are the implementation details of both of the APIs.

### Distributed Querying

In-order to satisfy the API requirements it was decided to use jgroups as the messaging framework. As referred in [www.jgroups.org](http://www.jgroups.org) JGroups is a reliable toolkit for messaging. It allows nodes to easily form a cluster which can be used for messaging over UDP( IP Multicasting) or TCP. It allows nodes to join and leave the cluster seamlessly. It supports point to point as well as point to multipoint messaging which fits our API requirements perfectly. Moreover JGroups is distributed under Apache License 2.0.

**Compiling and Executing**

The development environment requires the following libraries to compile and build the source code.

* Maven 3+
* Java SDK 1.7

Recommended IDE for use is Eclipse

### Joining a cluster

In the JGroups based implementation of the distributed querying API, vehicles need to join the JGroups cluster before they can query information. The cluster can be any name and it can also ideally get auto generated names for cluster name , fetched from a cloud server. For demo purposes, a hard-coded name is used for the cluster and one can join the cluster using the following code ( using the cmu.practicum. JgroupsRpc class)

*JgroupsRpc jgroupsRpc = JgroupsRpc.getInstance()*

*jgroupsRpc.start();*

### Querying the Cluster

Vehicles can query the cluster for information from the cluster. In order to query information, a java class needs to be created - representing the application. This class holding information for queried data needs to be deployed across all the nodes in the cluster. This is nothing but a simple Java class with some getters for the attributes and an execute method which will populate the attributes at the processing node. The class also needs to extend from the cmu.practicum.CommonAPI class. For example, please refer to the class cmu.practicum.app.VehicleDistance class.

Example on how to query can be seen in the cmu.practicum.app.SampleApp class.

The following API is used to query the information

*public <T> org.jgroups.util.RspList<T> dispatch(org.jgroups.blocks.ResponseMode responseMode, int timeout T val,* [*Class*](http://docs.oracle.com/javase/8/docs/api/java/lang/Class.html?is-external=true)*<T> valType)*

Parameters:

responseMode - ResponseMode.GET\_ALL/ GET\_FIRST/GET\_MAJORITY

timeout : time to wait

val: the instance of the object

valType: the classname of the object

Returns: Returns a RspList with objects from different vehicles. See example from SampleApp class for how to parse RspList

For example:

*RspList<VehicleDistance> rsp\_list=jrpc.dispatch(ResponseMode.****GET\_ALL****, 5000,* ***new*** *VehicleDistance(), VehicleDistance.****class****);*

### Informing the Cluster

In distributed information passing mode, the request object can contain some information. So for example to use the above VehicleDistance object in a distributed information passing mode, one can do as follows

1. Construct a VehicleDistance object with some information

VehicleDistance vh = new VehicleDIstance();

2. Add some information to it for passing

vh.setDistance(50);

3. Use the same object instead of a using a new object as in query mode.

*RspList<VehicleDistance> rsp\_list=jrpc.dispatch(ResponseMode.****GET\_ALL****, 5000,* ***vh,*** *VehicleDistance.****class****);*

4. Vehicles receiving the information can take necessary action as described in the execute() method.

### Distributed Processing

#### Background Research

During our first phase, our team researched different technology stacks for distributed processing on the mobile cloud and narrowed down our choices to the following top three stacks:

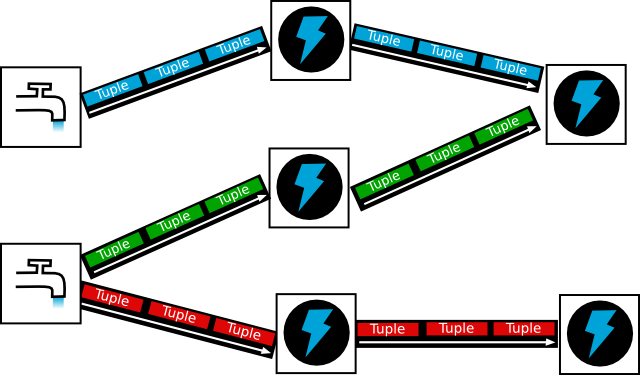
|  |  |  |
| --- | --- | --- |
|  | **Pros** | **Cons** |
| **OpenStack** | * well-documented and widely used cloud computing platform * Scalable, reliable, fault-tolerant | * No real-time distributed processing capability |
| **Apache Storm** | * Real-time distributed processing capability * Scalable, reliable, fault-tolerant, and fast | * Not as widely used as OpenStack (doesn’t have MarketPlace for plug-ins,etc) |
| **Hyrax** | * Specifically developed to enable mobile cloud computing. * Already works on Android * CMU product | * Lack of documentation and support group. * Difficult to get hold of the source code * In beta stage |

**Figure 3**

Each technology stack had its own strengths and weaknesses. The only technology stack that included the crucial capability to perform real-time distributed processing was Apache Storm. This was the most important factor in our decision to use Apache Storm for distributed processing.

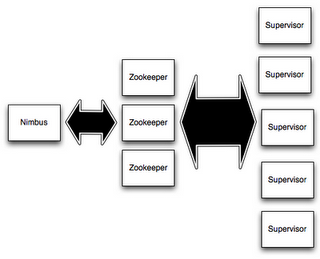
#### Apache Storm

To do real-time, distributed computation on Apache Storm, you create what are called “topologies”. A topology is a graph of computation consisting of bolts and spouts. Bolt is a core function of a streaming computation. It receives tuples (Apache Storm’s preferred data type) and performs computation on them. Spouts are sources of the data stream and are responsible for data input. Streams are unbounded sequence of tuples that connects bolts and spouts together. Storm runs these bolts and spouts on separate nodes in parallel, thus achieving the distributed computing capability.



**Figure 4: Apache Storm topology**

Figure 5 shows an example Storm Cluster. Nimbus is a daemon on the master node that handles scheduling of tasks in the cluster. Supervisor supervises the process that actually executes your code. The Zookeeper handles the cluster management.



**Figure 5: Storm Cluster**

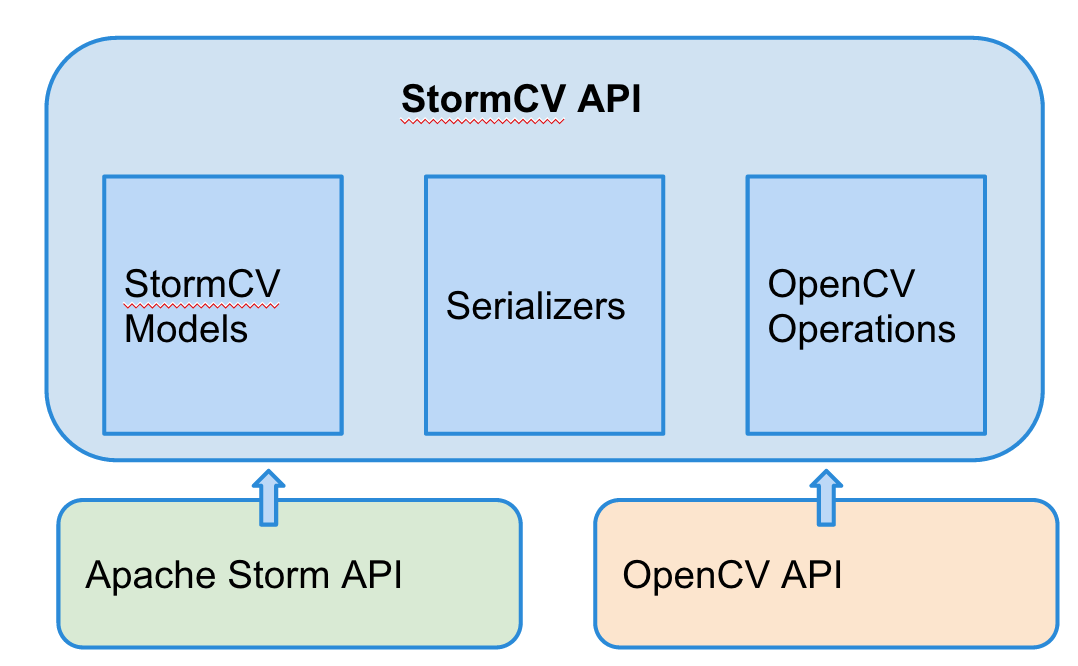
To summarize briefly, here is a pipeline of how the topology is run on the Storm Cluster:

1. Topology submitter uploads topology (jar file to the Nimbus).
2. Nimbus calculates assignments and sends to Zookeeper.
3. Supervisor nodes receive assignment info from Zookeeper.
4. Supervisor nodes download topology from Nimbus.
5. Supervisor spawn workers (JVM processes) to start the topology.

#### StormCV

Many of the Toyota uses cases call for image processing and computer processing capabilities. And often these image-related operations need to be done in real-time to meet the safety requirements, as well as customer satisfaction for in-vehicle entertainment. To enable this real-time distributed image processing and computer vision capability, we have modified StormCV, an existing library that uses Apache Storm to run OpenCV, an open source computer vision and machine learning software that is widely used in the robotics and computer vision-related industry. Through StormCV, one can run the OpenCV operations in real-time in a distributed fashion, thereby achieving significant performance enhancements.

Figure 6 shows a high-level architecture diagram for StormCV.



**Figure 6: StormCV Architecture**

StormCV API contains its own models, serializers, and OpenCV operations. The StormCV models include relevant classes for computer vision operations, such as the feature class, image frame classes, etc. The data serializer takes the tuple object, which is the data structure used in Apache Storm, and serializes/deserializes them for OpenCV operations.

Using StormCV to perform vision and machine related processing is simple. All you need to do is write a new topology that calls one of the OpenCV operations available in the StormCV’s operations package. There are also many examples available inside the StormCV repository including operations such as gray scaling the image, face detection, tiling, background subtraction, and contrast enhancement. We have found that it is easy to model a new topology following one of these example topologies.

#### StormCV Installation

Our version of StormCV is hosted on the public github account. It can be forked or cloned using this url: <https://github.com/yglee/StormCV> The github page includes a detailed README that explains how to install and execute the topology using the StormCV library.

#### Compiling and executing StormCV

To compile StormCV, cd into the StormCV/stormcv directory and run

"mvn clean install -DskipTests=true -X"

To execute a topology on Storm on a local cluster, follow the steps below:

* From the StormCV/stormcv directory, run "mvn compile exec:java -Dstorm.topology=nl.tno.stormcv.[topology name]"

For example:

"mvn compile exec:java -Dstorm.topology=nl.tno.stormcv.E9\_ContrastEnhancementTopology" will run the Contrast Enhancement Topology.

Available examples of topologies include:

* E1\_GrayScaledTopology
* E4\_SequentialFeaturesTopology
* E7\_FetchOperateCombiTopology
* E2\_FacedetectionTopology
* E5\_TilingTopology
* E8\_BackgroundSubtractionTopology
* E3\_MultipleFeaturesTopology
* E6\_GroupOfFramesTopology
* E9\_ContrastEnhancementTopology

\*\* Note: The Topologies are set to run on the LocalCluster and will timeout after 2 minutes. To modify this behavior, tweak the code in main function of each topology class.

StormCV creates its own simple web service used to view resulting MJPEG streams. Once the topology is running, execute one of the following calls by typing in the url into the browser:

* http://IP:PORT/streaming/streams : lists the available JPG and MJPEG urls
* http://IP:PORT/streaming/picture/{streamid}.jpg : url to grab jpg pictures
* http://IP:PORT/streaming/tiles : provides a visual overview of all the streams available at this service. Clicking an image will open the mjpeg stream
* http://IP:PORT/streaming/mjpeg/{streamid}.mjpeg : provides a possibly never ending mjpeg formatted stream

Make sure to replace the "IP" with your network IP and use "8558" for Port. For example, my computer has IP of 10.0.0.9, so I type in "<http://10.0.0.9:8558/streaming/tiles>" to view the output streams.

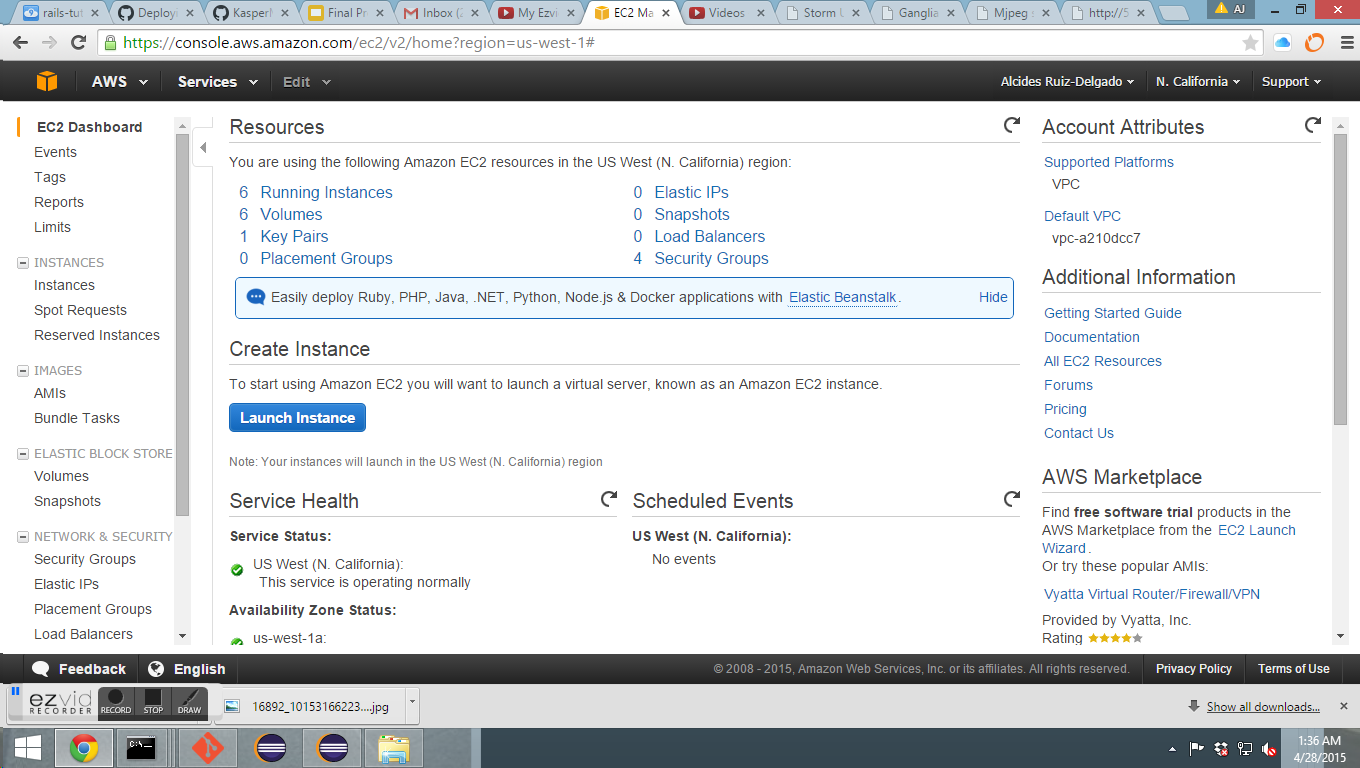
***How-to Demo***  
Please see the following link for a how-to video for setting up StormCV and executing topologies:

StormCV: <https://www.youtube.com/watch?v=4PlpfhPExww>

## Experiments and Analysis

**Deploying to a Remote Cluster**

We were blocked in terms of implementing a testing framework on a mobile cloud of Toyota vehicles because we didn’t have access to any Toyota smart car hardware. We decided that the best way to simulate a real world environment was to leverage remote servers where each server can act as a vehicle or node in our mobile cloud. We created an Amazon EC2 account and created our remote cluster. Please see Figure 7.



**Figure 7: Amazon EC2 Remote Cluster Dashboard**

***Unit Test Infrastructure***

We chose against using Storm Deploy which was recommended by storm.apache.org because it required the use of deprecated Java Runtime Environment version 1.7.0\_u51 as well as several configuration headaches, such as multiple bindings of the same dependency located in the Storm jar file as well as in the StormCV jar file, SSH key issues and incompatibility problems with newer versions of Storm.

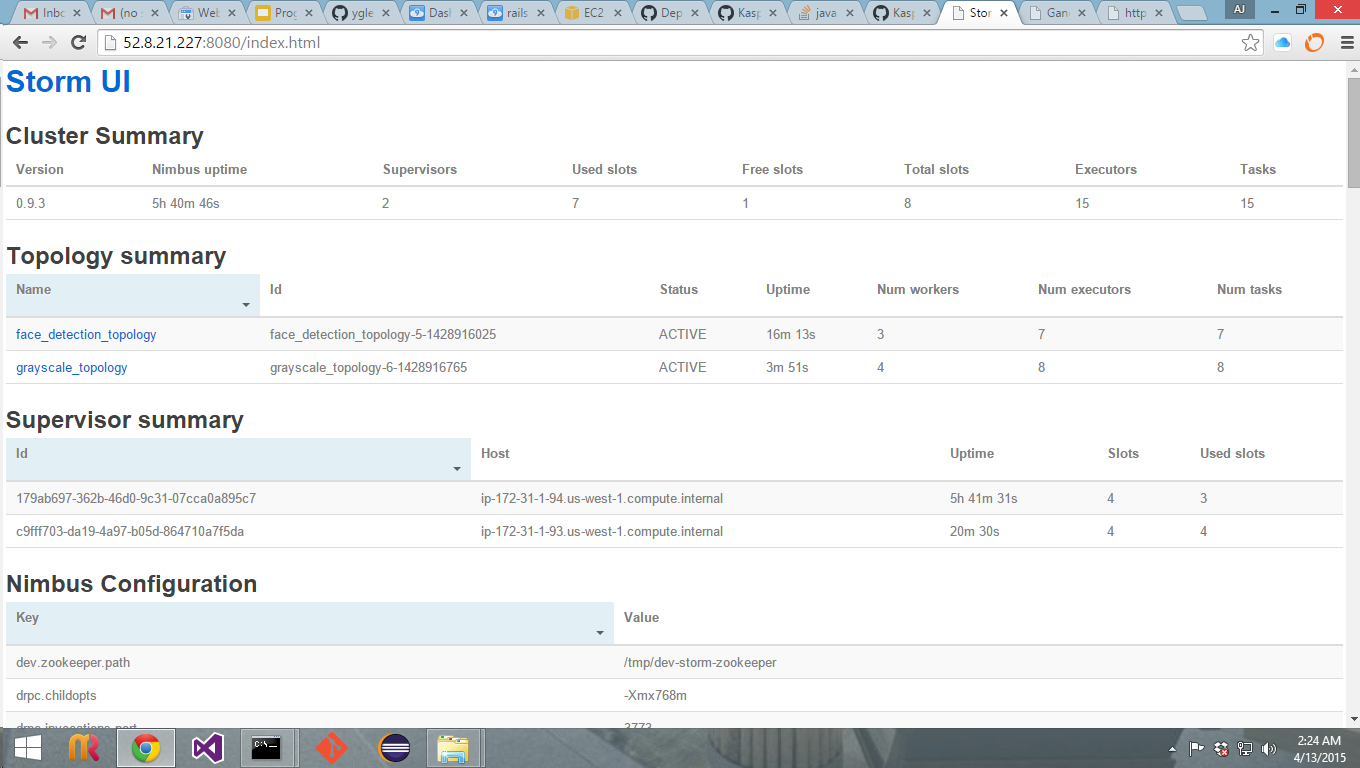
We found another tool to deploy StormCV topologies on Amazon EC2 instances called Storm Deploy Alternative at the following GitHub link. This tool was relatively easy to use compared to other tools. It required editing a configuration file for describing the remote cluster setup and a credentials file for authorizing access to our Amazon EC2 user account. The tool also wrote the public IP addresses of all servers in the remote cluster to a .storm directory in our Home folder.

<https://github.com/KasperMadsen/storm-deploy-alternative>

We were able to execute concurrent topologies on a remote cluster as well as scale our cluster without the need to reboot it. By adding more nodes as well as allowing nodes to depart, it simulates the real world environment of a dynamic vehicular mobile cloud. We were able to ssh into any of the remote servers in our cluster.

An extremely important feature is that it enables a unit test infrastructure to be executed as part of the deployment process. We can test for proper configuration and performance baselines in addition to our functional testing.

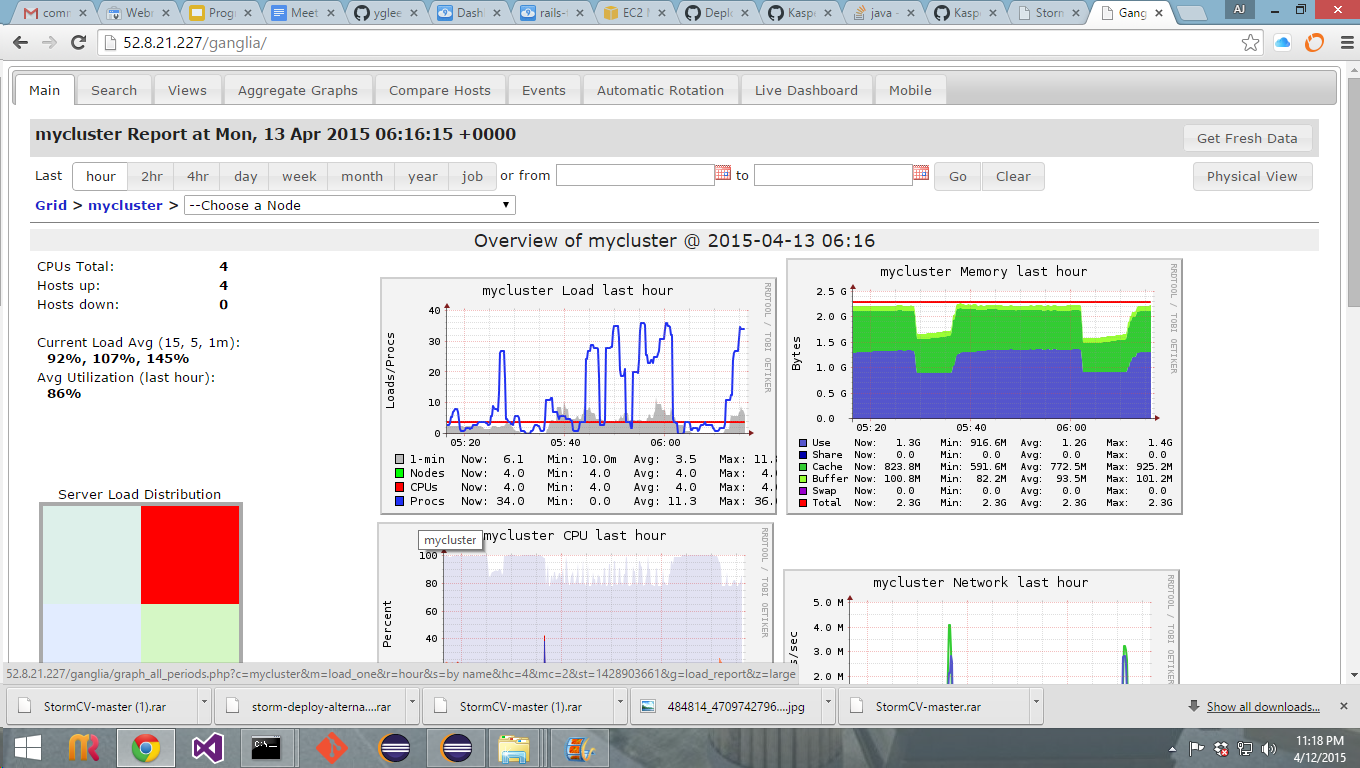
The Storm UI allows us to see the cluster summary, the topology summary, the supervisor summary, and the nimbus configuration. In addition, if we click on a particular topology, we can execute topology actions such as activate, deactivate, rebalance, and kill. We can also view individual topology stats, the spouts, bolts, and topology configuration. Please see Figure 8 below.



**Figure 8: Storm UI**

***Performance Metrics***

In the real world, performance is essential since life and death situations occur constantly in a mobile vehicle environment. For example, notifying other vehicles that a lane change is about to occur and receiving the message back in real time that a lane change is not safe at that moment would preempt a vehicle collision and possibly save lives. We monitor the performance of our topologies by accessing the Ganglia UI. It provides metrics on the overall cluster load, memory usage, CPU utilization, and network speed. In addition, we can access the same metrics on any individual node in the cluster. We can also download the metrics as a CSV file, we can export it in JSON format, and we can inspect the performance graphs as illustrated in Figure 9 below.

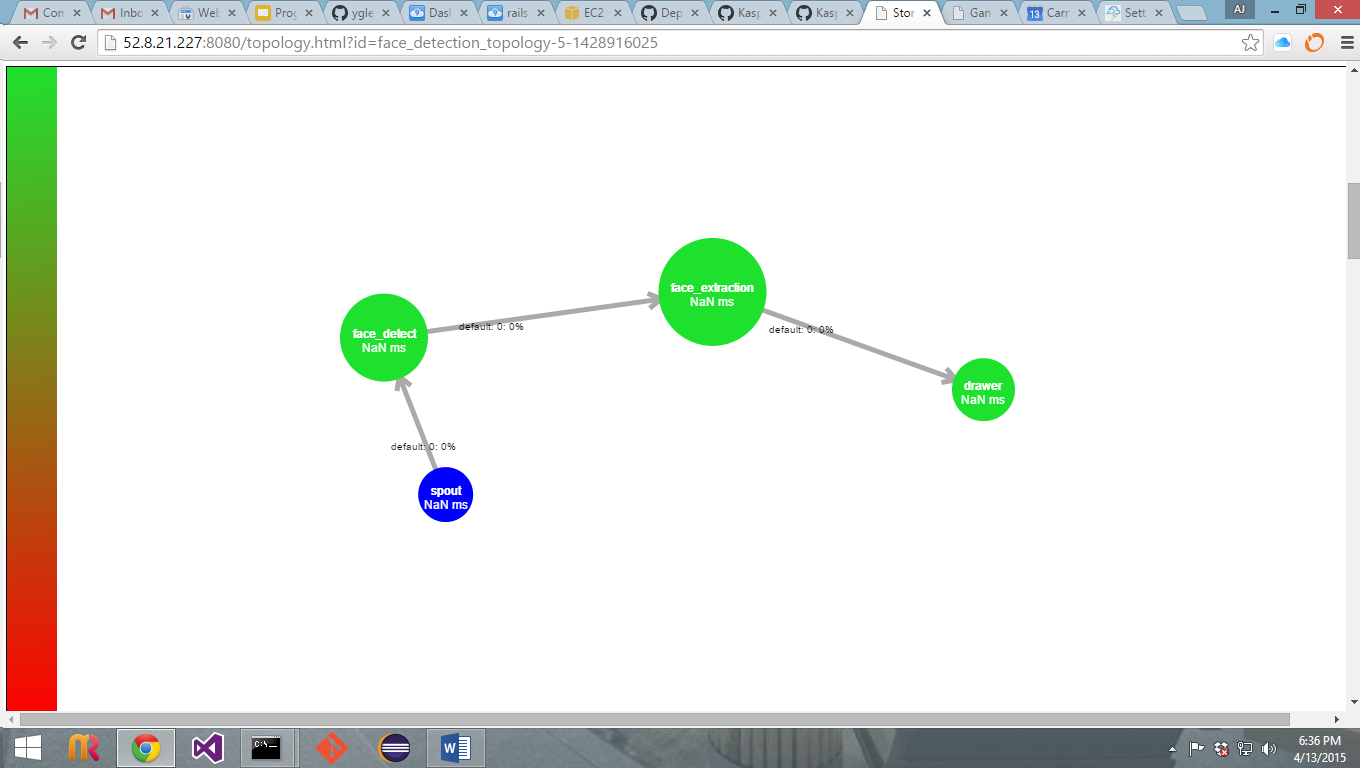


**Figure 9: Ganglia UI**

***Topology Visualization***

We can also analyze our software using qualitative data. The Face detection topology creates a list with files to be processed by the spout. It creates one spout which reads images from this directory and sends it to the face\_detect bolt. It creates the face\_detect bolt with a HaarCascade classifier leveraging the lbpcascade\_frontalface model which it uses for detecting faces. It outputs a frame ,including the features with the detected faces, to the face\_extraction bolt. The face\_extraction bolt extracts the bounding boxes of the face feature from the frame and emits them as separate frames to the drawer bolt. Finally, the drawer bolt draws the features into the frame and writes the frame to the local file system at ../output/facedetections

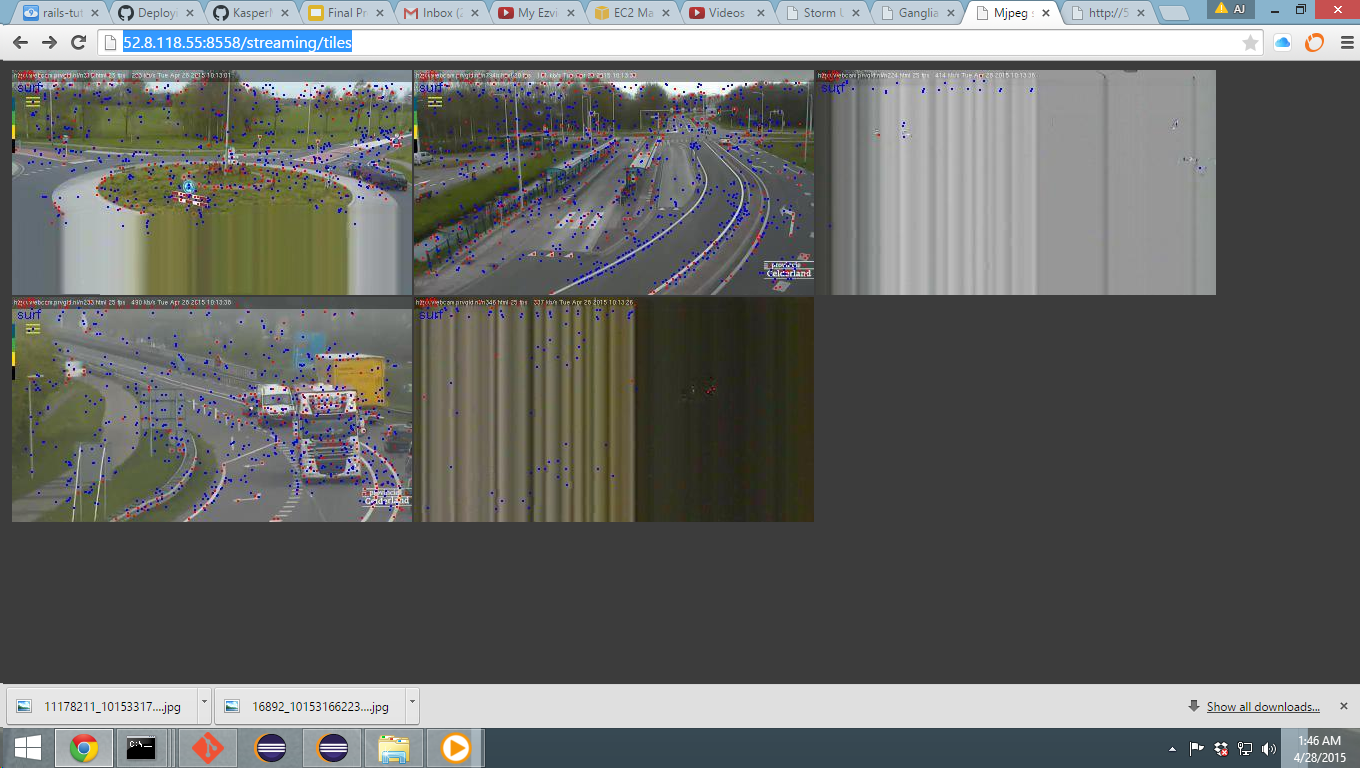
Without a topology visualization, the only way to know this is would be by reading the topology code or the comments if it is well documented. However, with a topology visualization, we can get a quick overview of the data stream flow. Please see Figure 10 below.



**Figure 10: Topology Visualization**

***Web Service***

StormCV enables the use of [Apache Storm](https://storm.apache.org/) for video processing by adding computer vision (CV) specific operations and a data model. The platform enables the development of distributed video processing pipelines which can be deployed on Storm clusters. Figure 11 below is a screenshot taken from a distributed pipeline that calculates SIFT and SURF features in real time on five live streams. We can create a web service on a supervisor port enabling us to see this video stream result.



***Installation and Configuration***

***Preliminaries***

1. First, you need Java JRE, Java JDK and Git installed and in your user's PATH

**Install Storm**

1. git clone git://github.com/apache/storm.git && cd

storm/examples/storm-starter

2. mvn compile

4. mvn package -DskipTests

5. mvn install -DskipTests

6. Place lib directory in root of storm directory.

**Install Zookeeper**

1. Download a stable ZooKeeper release from <http://zookeeper.apache.org/releases.html>

2. Unpack it and cd to the root

3. Create a configuration file in **conf/zoo.cfg**:

tickTime=2000  
dataDir=/var/zookeeper  
clientPort=2181

4. Start ZooKeeper using command: bin/zkServer.sh start

**Install Maven**

1. Download the Maven zip file from http://maven.apache.org/download.cgi

2. Unzip the distribution archive, i.e. apache-maven-3.3.1-bin.zip to the directory you

wish to install. These instructions assume you chose C:\Program Files\Apache

Software Foundation. The subdirectory apache-maven-3.x.x will be created from the

archive.

3. Add the unpacked distribution's bin directory to your user PATH environment

variable by opening up the system properties (WinKey + Pause), selecting the

"Advanced" tab, and the "Environment Variables" button, then adding or selecting

the PATH variable in the user variables with the value C:\Program Files\Apache

Software Foundation\apache-maven-3.3.1\bin.

4. In the same dialog, make sure that JAVA\_HOME exists in your user variables or in the

system variables and it is set to the location of your JDK, e.g. C:\Program

Files\Java\jdk1.7.0\_51.

5. Open a new command prompt (Winkey + R then type cmd) and run mvn

--version to verify that it is correctly installed.

**Install Storm Deploy Alternative**

1. git clone <https://github.com/KasperMadsen/storm-deploy-alternative.git>
2. mvn install
3. edit conf/configuration.yaml file. Below is an example:

#

# Amazon EC2 example cluster configuration

#

mycluster:

- provider "aws-ec2"

- storm-version "0.9.3"

- zk-version "3.4.6"

- image "us-west-1/ami-56120b13"

- region "us-west-1"

- t1.micro {MASTER, UI}

- t1.micro {ZK}

- t1.micro {WORKER}

- t1.micro {WORKER}

- remote-exec-preconfig {cd ~, echo hey > hey.txt}

- remote-exec-postconfig {}

4. edit conf/credential.yaml file. You must enter your EC2 identity and credential

**Launch Remote Cluster**

1. java -jar storm-deploy-alternative-1.jar deploy NAME\_OF\_CLUSTER
2. IP addresses of all remote nodes will be written to $HOME/.storm/storm.yaml

and $HOME/.storm/supervisor.yaml

3. To scale cluster, run java -jar storm-deploy-alternative.jar scaleout

CLUSTER\_NAME #NumInstances INSTANCE\_TYPE

**Deploy StormCV topologies**

**DISCLAIMER: This has only been tested on a Windows 8.1 Pro OS**

1. Create a development branch from Github repository located at <https://github.com/yglee/StormCV.git>
2. Clone from your new branch into the root of your Storm directory.
3. Edit your conf/storm.yaml file and enter the IP address of all your remote nodes. Example below:

storm.zookeeper.servers:

- "52.8.118.54"

nimbus.host: "52.8.118.58"

ui.host: "52.8.118.58"

cluster: "mycluster"

storm.supervisor.servers:

- "52.8.118.57"

- "52.8.118.55"

4. mvn clean install

5. Install Eclipse IDE for Java Developers from <http://eclipse.org/downloads/>.

6. Run Eclipse. File-> Import->Existing Maven Packages->Next

7. Browse to root directory of StormCV-master where pom.xml file is located

and click Finish.

8. Be sure to use pom.xml file located at the github URL mentioned in Step 1 in

order to avoid configuartion deployment errors.

9. Go to Package Explorer->right-click on stormcv-deploy->Run As->Maven

build…

10. Type “package” for Goals->click Apply->click Run  
 11. cd to ~\storm\StormCV-master\stormcv-deploy\target where

stormcv-deploy-0.0.1-SNAPSHOT-jar-with-dependencies is located

12. Run “storm jar stormcv-deploy-0.0.1-SNAPSHOT-jar-with-dependencies

nl.tno.stormcv.deploy.NAME\_OF\_TOPOLOGY\_YOU\_ARE\_RUNNING

NAME\_OF\_YOUR\_CLUSTER”

13. Ignore SLF4J Warning. The warning emitted by SLF4J is just that, a warning.

Even when multiple bindings are present, SLF4J will pick one logging

framework/implementation and bind with it. The way SLF4J picks a binding

is determined by the JVM and for all practical purposes should be considered

random.

14. Wait several minutes. Go to NIMBUS\_IP\_ADDRESS:8080 to see Storm UI.

15. Go to NIMBUS\_IP\_ADDRESS/ganglia to see Ganglia UI

***Analysis***

A Storm cluster executing StormCV topologies on an Amazon EC2 cloud platform has the following benefits:

1. Distributed video processing
2. Simulation of a real world mobile vehicular cloud
3. Tracking, graphing, analysis, and sharing of performance metrics
4. Unit test deployment infrastructure
5. Free or cheap if performance is not an issue
6. Very fast if cost is not an issue

***How-to Demo***  
Please see the following link for a how-to video for setting up a remote cluster, deploying topologies on this remote cluster, analyzing the metrics of our performance, viewing the output using a web service, and dynamically scaling the cluster thereby simulating a real world environment, such as when 2 new vehicles enter our mobile cloud infrastructure.

AWS: <https://www.youtube.com/watch?v=PPHUY6cyMkk>

## 

## Conclusions and future work

The team has worked on bringing the proof of concepts for distributed querying and processing, as well as featuring a vehicle recognition ability in the topology. While demonstrating, the team was able to confirm that implementing a mobile cloud system is a very promising field of technology to provide safety, entertainment, and convenience to multi-purpose drivers in a practical sense.

Obviously there are integration work with another CMUSV team working on the mobile cloud architecture and an actual Toyota engineering team to bring this proof of concept to life. Our implementations to demonstrate the potentials of this topology also need to be enhanced to support a real-life use case. More tests and analyses need to be developed to ensure the robustness of this framework. However, with the current progress in mind, the team strongly believes that this mobile cloud system will be a huge enhancement to our driving experience in everyday lives.

## Appendix

## Logistical Information

Toyota Practicum Team

|  |  |
| --- | --- |
| **Name** | **Email** |
| Hyo Jeong | hyo.jeong@sv.cmu.edu |
| Ambarish Karole | ambarish.karole@sv.cmu.edu |
| Grace Lee | grace.lee@sv.cmu.edu |
| AJ Ruiz | aj.ruiz@sv.cmu.edu |
| Harikumar Sulochana | harikumar.kumar.sulochana@sv.cmu.edu |

SupervisingFaculty

|  |  |
| --- | --- |
| **Name** | **Email** |
| Prof. Patrick Tague | patrick.tague@sv.cmu.edu |
| Prof. Jia Zhang | jia.zhang@sv.cmu.edu |

Client

|  |  |
| --- | --- |
| **Name** | **Email** |
| Toyota Automobile Company as represented by Professor Patrick Tague | patrick.tague@sv.cmu.edu |

**Source Code Repositories and Demo URLs:**

1. Github for Android JGroups Demo:

<https://github.com/karoleam/CMUToyotaAndroidJGroups/tree/master>

Video Demo:

<https://www.youtube.com/watch?v=ujcDQe4IB1U>

1. Github for JGroups API/Demo Application:

<https://github.com/asiandrummer/toyota-jgroups>

1. Github for StormCV:

<https://github.com/yglee/StormCV.git>

StormCV Demo: <https://www.youtube.com/watch?v=4PlpfhPExww>

AWS Demo: <https://www.youtube.com/watch?v=PPHUY6cyMkk>