SCALING AN URBAN EMERGENCY EVACUATION FRAMEWORK: CHALLENGES AND PRACTICES

Rajasekar Karthik, Wei Lu

Computational Sciences & Engineering Division

Oak Ridge National Laboratory, Oak Ridge, TN 37831

{karthikr, luw4}@ornl.gov

ABSTRACT:

Critical infrastructure disruption, caused by severe weather events, natural disasters, terrorist attacks, etc., has significant impacts on urban transportation systems. We built a computational framework to simulate urban transportation systems under critical infrastructure disruption in order to aid real-time emergency evacuation. This framework will use large scale datasets to provide a scalable tool for emergency planning and management. Our framework, World-Wide Emergency Evacuation (WWEE), integrates population distribution and urban infrastructure networks to model travel demand in emergency situations at global level. Also, a computational model of agent-based traffic simulation is used to provide an optimal evacuation plan for traffic operation purpose [1]. In addition, our framework provides a web-based high resolution visualization tool for emergency evacuation modelers and practitioners. We have successfully tested our framework with scenarios in both United States (Alexandria, VA) and Europe (Berlin, Germany) [2]. However, there are still some major drawbacks for scaling this framework to handle big data workloads in real time.

On our back-end, lack of proper infrastructure limits us in ability to process large amounts of data, run the simulation efficiently and quickly, and provide fast retrieval and serving of data. On the front-end, the visualization performance of microscopic evacuation results is still not efficient enough due to high volume data communication between server and client. We are addressing these drawbacks by using cloud computing and next-generation web technologies,

namely Node.js, NoSQL, WebGL, Open Layers 3 and HTML5 technologies. We will describe briefly about each one and how we are using and leveraging these technologies to provide an efficient tool for emergency management organizations.

Our early experimentation demonstrates that using above technologies is a promising approach to build a scalable and high performance urban emergency evacuation framework that can improve traffic mobility and safety under critical infrastructure disruption in today's socially connected world.

KEYWORDS: emergency evacuation framework, scalability, cloud computing, Node.js, WebGL, HTML5.

1. INTRODUCTION:

Natural or man-made disasters, such as Atlanta ice snow in 2014 and Boston Marathon bombings in 2013 have tremendous impacts on urban transportation systems. It is critical for emergency managers and transportation professionals to have an efficient evacuation plan during these kinds of infrastructure interruptions. Simulation-based studies are commonly used in evacuation planning and decision making processes. Most existing simulation models in transportation evacuation are restricted to certain geographic areas and selected traffic simulation tools, such as OREMS, DYNASMART, VISSIM, etc. [3]. Each tool has its own data format requirement and simulation preferences (macroscopic, mesocopic, or microscopic). This leads to the need of a uniform simulation tool in evacuation planning community, so that researchers can focus on improving traffic models in travel demand, trip distribution, and traffic assignment areas.

This motivated us to develop a computational framework and easy to use system, called World-Wide Emergency Evacuation (WWEE). It is a rapid response emergency evacuation modelling system that estimates evacuation time and travel conditions (Speed, Congestion, etc.) for any geographic locale in the world. WWEE integrates population distribution and urban

infrastructure networks to model travel demand in emergency situations at global level. Also, a computational model of agent-based traffic simulation is used to provide an optimal evacuation plan for traffic operation purpose [1]. In addition, our framework provides a web-based high resolution visualization tool for emergency evacuation modelers and practitioners.

However, there are still some major drawbacks for scaling this framework to handle big data workloads in real time. On our back-end, lack of proper infrastructure limits us in ability to process large amounts of data, run the simulation efficiently and quickly, and provide fast retrieval and serving of data. On the front-end, the visualization performance of microscopic evacuation results is still not efficient enough due to high volume data communication between server and client.

We are addressing drawbacks on back-end by using cloud infrastructure, high performance NoSQL database for storage, and Node.js to solve I/O bottlenecks. On the front-end, to provide rich and blazingly fast interactive visualization, we use next-generation web technologies namely WebGL, Open Layers 3 and HTML5 collection of technologies.

The rest of the paper is organized as follows. Section 2 describes background and motivation for the framework. In Section 3, we describe our system architecture and in detail, various components and technologies we are using and leveraging to scale our framework. Section 4 presents the conclusion of the paper and our future work is described in Section 5.

2. BACKGROUND AND MOTIVATION

Simulation-based evacuation studies give researchers and practitioners great tools to evaluate evacuation strategies. A detailed review of various emergency evacuation models by Alsnih et al. [4] pointed out the three main procedures to devise emergency evacuation plans, including evacuees' behavior analysis, transportation engineering analysis, and the role of government. The interaction and cooperation among these three aspects are needed to provide better solutions of a mass evacuation on the current transport network. Various scales of evacuation areas were conducted to evaluate the evacuation efficiency and systems performance, from the whole state Tennessee to a corridor in Washington D.C. [5, 6]. Microscopic traffic simulation is becoming

more popular than conventional macroscopic traffic simulation in evacuation study due to the cheap but fast computing capacity and expectation of detailed performance. Jha, et al. [7] took advantage of microscopic simulation model (MITSIM) to model the evacuation of a small area - Los Alamos National Laboratory. Cova, et al. [8] presented a method to develop neighborhood evacuation planning with microscopic traffic simulation in the urban – wildland interface. Household-level evacuation planning is implemented in various scenarios. Activity-based traffic simulation provides more realistic simulation at the traffic assignment stage, which is adopted by a population simulation package called Transportation Analysis and Simulation System (TRANSIMS) [9]. Henson et al. [10] reviewed 46 activity-based models and used TRANSIMS to demonstrate their competency for homeland security applications. Despite these existing research efforts in evacuation simulations, most of these studies assume a one-to-one trip assignment to evacuees' full compliance for predetermined destination and route assignments. This motivated us to develop such a computational framework.

The goal of WWEE is to develop an easy-to-use system targeted at emergency response planners. WWEE consists of software and data developed in ORNL combined with readily available data from the open source domain. There are four major components in this system: 1) LandScan Global, a high-resolution population distribution database, developed at ORNL, and a public domain global street network, OpenStreetMap (OSM), 2) a web-based network editing tool that is used to extract the street network from OSM and refine network as needed and include traffic controllers, 3) an open source traffic simulation model (TRANSIMS) in which macroscopic modelling is performed as well as agent based microscopic simulation is performed, an open source time-based microscope traffic simulation model from MITSIM, and 4) a web-based user interface that can display either the agent-based micro individual vehicle analysis or link based macro traffic analysis results.

The user logs into the main page and defines an evacuation area using several drawing tools (circle, rectangle, polygon, etc.). The system will calculate the number of evacuees within this evacuation area from LandScan Global. The node-link based network is then extracted from OSM and converted it to a traffic-modeling network, which includes lanes, pocket, connectors, intersections, and controllers. If the extracted street network does not present as the real street network, the user can use the graphic network-editing tool to fine-tune the local network

configuration or controllers. The user also has the option to choose one of three traffic simulation methods included (TRANSIMS macro, Router, TRANSIMS agent based, Simulator, or MITSIM micro). Results are displayed on a web-based user interface when the simulation is running.

3. SYSTEM ARCHITECTURE

We have already described about the drawbacks with our framework and our approach to address the same earlier. In this section, we will describe each of the components used in our system architecture. An illustration of our architecture is shown in Figure 1.

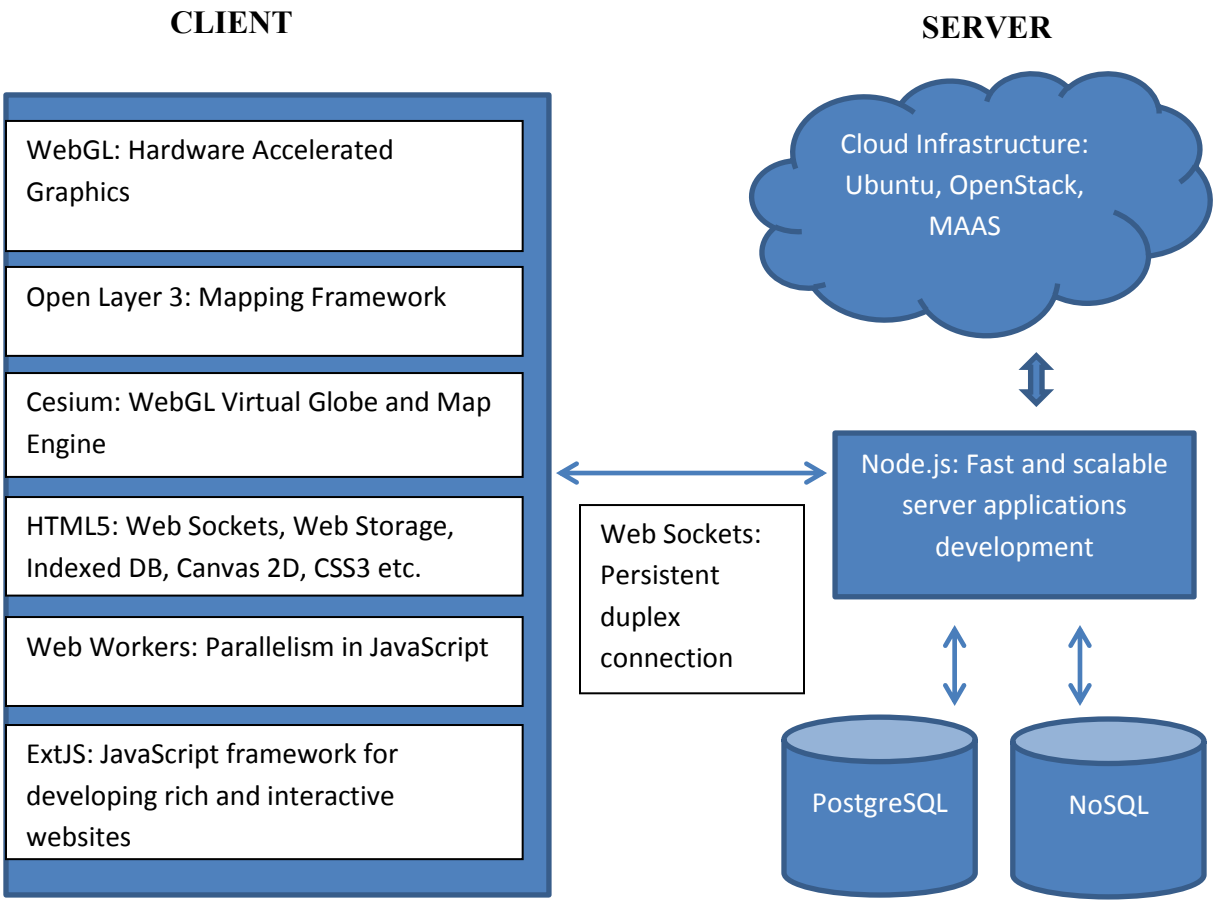

Figure 1: Scalable System Architecture

I. Cloud Infrastructure:

Cloud computing has gained enough awareness that majority of urban planning organizations and researchers know the benefits it can offer to an organization [11, 51]. Many organizations who were asking “Will moving to cloud solve our problems?” are asking “When can we move to the Cloud?” [11, 17]

With increasing adoption of Cloud computing, various infrastructure solutions have been built that ranges from managing nodes to Operating Systems (OS), software platform, applications built on top of them, to networks and storage. One such leading solution that provides these capabilities is called OpenStack.

OpenStack is a free and open source infrastructure solution that powers many of the clouds used in Fortune 500 companies [11]. It provides broad range of components for controlling compute, storage and networking resources needed for massively scaling and deploying resources in global data centers. Originally founded by Rackspace Hosting and NASA, it has grown to be a global active developer community [11, 12].

Due to this broad nature, deploying the plain vanilla distribution of OpenStack is painful and difficult, especially for smaller organizations [13]. It requires dedicated set of expertise and can make moving to cloud very time-consuming. Organizations are typically requiring a bigger bang for their buck, i.e. they want to have most of the features found in large-scale commercial cloud deployments, but with limited time or investment or both [13, 14].

Ubuntu, a leading Linux distribution has been bundling OpenStack with their service since 2011 [13]. Known for making use of Linux simpler on desktops and server, they have done the same with OpenStack. Building a large-scale enterprise cloud, from provisioning to installation, deployment and management has been simplified and made easy to use with Ubuntu “Metal as a Service” (MAAS) tool [15, 16, 17].

OpenStack and Ubuntu have a long history of tight and continuous integration and rigorous testing, making their combination one of the fastest and reliable ways to build a cloud. Additionally, Ubuntu provides a five year support of this combination, with their Long Term Support (LTS) releases [15, 16].

Ubuntu, the most popular OS and widely used developer OS for OpenStack, has been historically used for powering WWEE simulator [15, 16]. Ubuntu supports conventional and old hardware pretty well [17, 18]. Due to the above mentioned benefits, we decided to adopt Ubuntu OpenStack as our cloud infrastructure.

Ubuntu MAAS:

Ubuntu MAAS is a quick and easy-to-use tool for transforming machines into a cloud [17]. It makes it easy to set up hardware on which to run "Ubuntu's OpenStack cloud infrastructure" [16, 17]. Machines (nodes) can be added, provisioned, commissioned, and deployed without interrupting rest of the network. Machines can be retired as well without any interruptions. It supports dynamically provisioning or re-allocation of physical resources to handle "Big Data" workloads [15, 16, 18].

Key components of MAAS tool are:

1. Region controller: consists of
    a. Web User Interface (UI) for easy and quick management of resources,
    b. REST API for integrating MAAS with third-party tools and workflows,
    c. Metadata server for cloud initiation, and
    d. DNS Server (Optional) [18, 19].
2. Cluster controller(s): controls provisioning and management of a cluster (or group of Nodes). It consists of
    a. TFTP (PXE) Server: booting Node(s) via network.
    b. DHCP Server (Optional): for dynamically assigning and managing IP addresses.
    c. Power management on Node(s): starting and stopping Node(s) [18, 19].
3. Nodes: Each node is an individual machine

DNS and DHCP are optional services because existing services used within an organization could be potentially used.

A typical MAAS deployment (as illustrated in Figure 2) has one Region controller, at least one Cluster controller, and Nodes needed to handle your workloads [18, 19]. Provisioning process is explained in detail in [19].

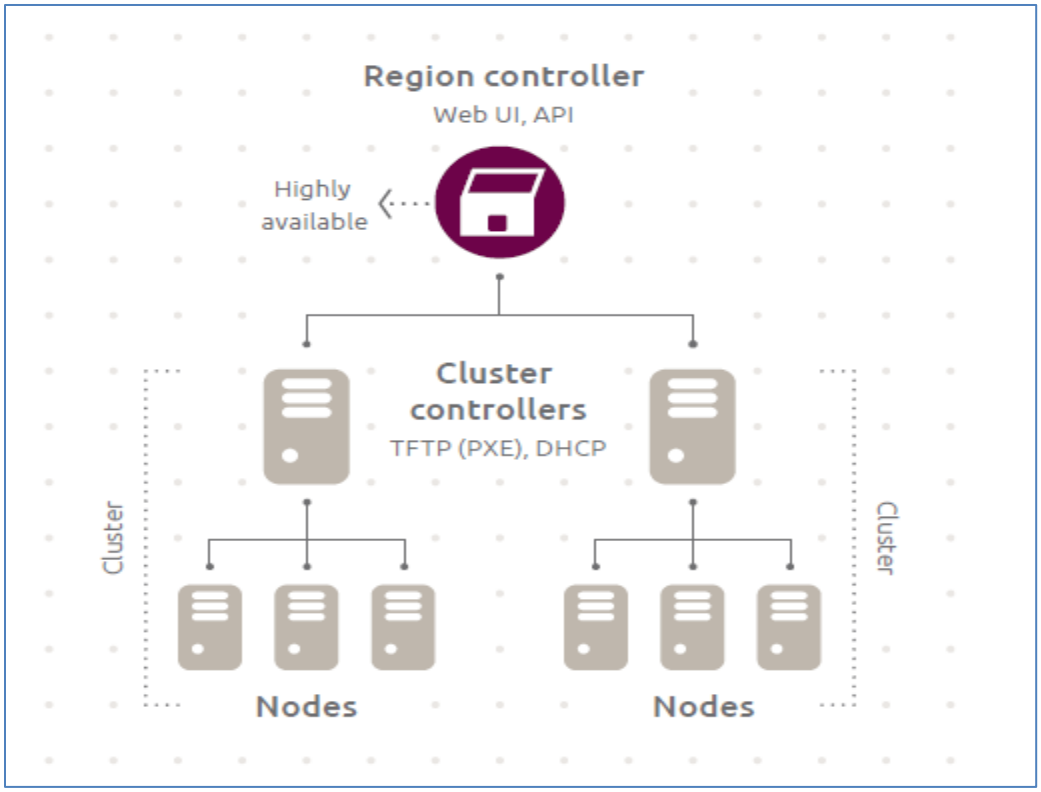

Figure 2: Key components in Ubuntu MAAS - Region Controller, Cluster Controller(s), and Nodes [18, 19]

Experiment:

We will discuss the cloud infrastructure setup process with MAAS, challenges faced and solutions to handle the same. Our experiment consisted of 3 physical machines with configurations and their use in our cloud (as specified in Table 1.)

Machine	Processors	RAM	Storage	Type
gcloud2	Intel Core™ 2 Quad core CPU Q6700 – 2.66GHz	4 GB	430 GB	Region and Cluster Controller
gcloud3	Intel Core™ 2 Quad core CPU Q8400 – 2.66GHz	4 GB	320 GB	Node
gcloud1	Intel Xeon™ Octa core CPU X5460 – 3.20GHz	16 GB	3 TB	Node

Table 1: Machine (Node) configurations used for our experiment

Prerequisites: We started with installation of latest Ubuntu 14.04 MAAS distribution on one of the machines. The installations were always unsuccessful due to failing hard drives. We diagnoses the hard drives with SMART utility provided along with the hard disks and removed failing ones. As the other two machines were also old, we decided to check for hardware (memory and hard drive) failures before we proceeded with setting up cloud.

Region and Cluster Controller: We again started with installation of MAAS on gcloud2. The Region and Cluster controller was setup on the same machine. This is the ideal deployment strategy recommended for initial cloud setup [16, 18, 19]. Then, we completed post-install tasks like adding users and getting boot images. This process was smooth and easy due to Ubuntu's commitment towards making the cloud infrastructure setup simpler. Also, the documentation by Ubuntu community helped as well.

DHCP and DNS Server: Next, we proceeded with setting up our own DHCP and DNS Servers. This was needed as to not conflict with our organization's DHCP and DNS servers. All our nodes had to be on a private network, i.e. gcloud1 and gcloud3. Only our region controller, i.e. gcloud2 could be connected to our organization's network. All the internet traffic from individual nodes had to be routed via gcloud2 via proxy.

We could not setup DHCP server initially via MAAS UI. So, we manually installed isc-dhcp – server, configured its properties and had it running as a separate DHCP server, disconnected with MAAS. Then, we shut down the separate DHCP server, replicated the same configurations in MAAS UI to have MAAS – DHCP server working together.

Proxy Configuration: We ran across an infamous problem with Ubuntu MAAS – If a private network is used, Proxy and DNS servers had to be manually configured with the private network addresses used. Detecting this problem was time-consuming. But, once we had figured it out, we will able resolve it quickly.

Provisioning: gcloud3 node was provisioned first and went through the following three steps:

Enlistment: When gcloud3 started, it obtained IP address from our DHCP server. Then, it was booted via PXE and retrieved Ubuntu boot image from the Cluster controller. The image ran an

initial discovery procedure to collect and send gcloud3's configuration to the Cluster controller and registered itself in MAAS with a "Declared" state [19].

Accept and Commission: Once gcloud3 was enlisted, we commissioned the node. Base OS was installed on OS. But, it failed to get latest packages from internet. It was during this process, we realized this was due to above mentioned "Proxy Configuration" problem. Once we resolved the same, and re-ran the same process, it got and installed the latest packages from internet. Its state was changed to 'Ready', meaning it was ready for deployment [19].

Deployment: gcloud3 was allocated to one of our users.

We are conducting experiments on the best way to allocate our resources for various needs – one node for retrieving, processing and storing data; another node for communicating with simulation framework; and last node for managing our cloud infrastructure.

II. Databases

Our datasets consists of: 1) LandScan Global, a high-resolution population distribution database, developed at ORNL, and a public domain global street network, OpenStreetMap (OSM). Currently, these datasets are stored in PostgreSQL traditional relational database management system (RDBMS). The case study data of population and networks are preprocessed because PostgreSQL used in our framework cannot handle volume of datasets, primarily LandScan global population and OpenStreetMap network data.

NoSQL represents a radical paradigm shift designed to overcome certain limitations of RDBMS [20]. While RDBMS excels for applications that requires ACID transactions, it either provides very poor performance or fails in handling huge datasets with limited available hardware. Also, RDBMS is not well suited for real-time analytical needs [20, 21]. NoSQL solutions apply novel methods to not only overcome these limitations, but also provide a fast linear performance and are Cloud-capable as well [20, 21]. We are analyzing various types of NoSQL solutions to find the one best that can handle our data workloads and real-time analysis. Based on our preliminary analysis, we are inclining towards MongoDB.

III. Node.js

Node.js is a server-side JavaScript environment for easily developing scalable network applications. It's built on top of Chrome's V8 engine [22, 23]. Node.js uses an event-driven programming model, which is different from traditional multi-thread programming [24, 25, 26]. To easily understand comparisons between these models, let's take the case of web servers as our example.

Traditional web servers require large amounts of I/O and multi-threading is an efficient way of using available processors (one thread per processor in a multicore system.) But, the downside is that cost of context switching between the threads is expensive, and thus limiting the speed and performance of the server [22, 26]. Event-based programming aims to provide an efficient alternative to address such drawbacks [26]. It utilizes a single event loop and uses asynchronous or non-blocking I/O model so execution do not get blocked when waiting for I/O operations [26, 28].

Node.js uses the event-based programming model to provide the power of multithreading, but with low overhead and a lock free design [28, 30]. It provides more control over the execution flow and makes it very attractive for developers as they do not have to handle the hard complexity of multi-thread programming mentioned above [30]. Various studies discussed in [28], [29] and [31] shows how Node.js outperforms traditional web servers such as Apache Web Server.

Node.js is increasing becoming popular for use on the server-side, due to its lightweight nature [22]. It's ideal for (1) data-intensive real-time applications that run across distributed machines, (2) Websocket server like chat server, (3) Fast file upload client etc. [25].

For scaling WWEE, we are using JavaScript as one of the primary languages for both client and server-side. JavaScript has been traditionally used in client side, i.e. browsers for displaying webpages. Server-side applications require another programming language such as Java or PHP. But, Node.js uses JavaScript as its programming language. By using Node.js, we are writing both server and client side application with just one language and one codebase [22, 30].

On our server-side, we use JavaScript along with Node.js modules for retrieving and serving data quickly from the database; message and data passing to WWEE simulator as well as storing and caching the output data from the simulator; routing our web service requests to appropriate controllers; and administration of our application including logging. Our client-side visualization stack is built based on JavaScript libraries such as OpenLayers, ExtJS, Cesium, Three.js etc. We have detailed discussed about them and its use in our application below.

Also, we will discuss about Web Sockets, also belonging to HTML5 family of technologies, and how Node.js improves I/O performance on server side below.

IV. WebGL:

WebGL is a JavaScript API for rendering rich and interactive graphics including 3D in a browser without the need for plug-ins. It's based on widely adopted OpenGL ES graphics library [33]. Full power of "Graphics Processing Unit" (GPU) accelerations can be harnessed for processing graphics computations and rendering them using just JavaScript and WebGL [34, 35]. Before WebGL, plug-ins or native applications were needed to be downloaded and installed to get a 3D experience [35, 36].

WebGL is one of the popular technologies within HTML5 family. Majority of browsers in desktop and mobile already support WebGL [35, 36].

WebGL is a low-level API and since it is increasingly being adopted, many libraries have been built on top of WebGL that provides various rich set of features and common utilities needed to produce high-end visualizations. While there are tons of libraries being developed, we will mainly discuss about three libraries used for our needs: (1) Open Layers 3, (2) Cesium, and (3) Three.js.

V. Open Layers 3 (OL3)

OL3 is a cutting edge and high-performance mapping library, being developed by OSGeo to provide rich user experience including 3D using HTML5, WebGL, Cesium, CSS3 and Closure

[41, 22]. OL3 is re-written from scratch with a clean and modular design. But, at the same time, rich set of features and utilities available in Open Layers 2 are being ported to OL3 with the new design. OL3 supports mobile platforms and devices out of the box [45].

OL3 uses various techniques for creating high-performance maps such as better Garbage Collection (GC), re-using objects, uses rAF, avoiding unnecessary boxing or unboxing operations, and re-drawing as few pixels as possible [45]. Also, Google's closure compiler is used for advanced optimizations such as code checking and dead code removal to produce high performance, lightweight and extremely compressed JavaScript file [44].

VI. Cesium

Cesium is a virtual globe and map engine based on WebGL. Built for precision and performance, it supports visualization of dynamic geospatial data and worldwide terrains [41, 42]. With just one API, three views – 2D, 2.5D Columbus and 3D Globe views can be created and switched easily. Cesium is free, open source and has an active community supporting it [46]. With WebGL and Cesium, OL3 is being built to support very large vector layers, 2D tilted and rotated views, and rich collection of 3D capabilities such as 3D globes creation and terrain in local projection [42, 43].

Three.js: Our application uses another WebGL based library called Three.js, as it supports various common utilities and helper functions such as textures, lighting, shaders etc. that makes use of WebGL simpler. [37].

VII. HTML5

HTML5 introduces (1) next version of HTML5 specification, and (2) Large collection of HTML5 Technologies (as illustrated in Figure 3) [32] used to enable rich and powerful applications. We will describe some of these technologies we use below.

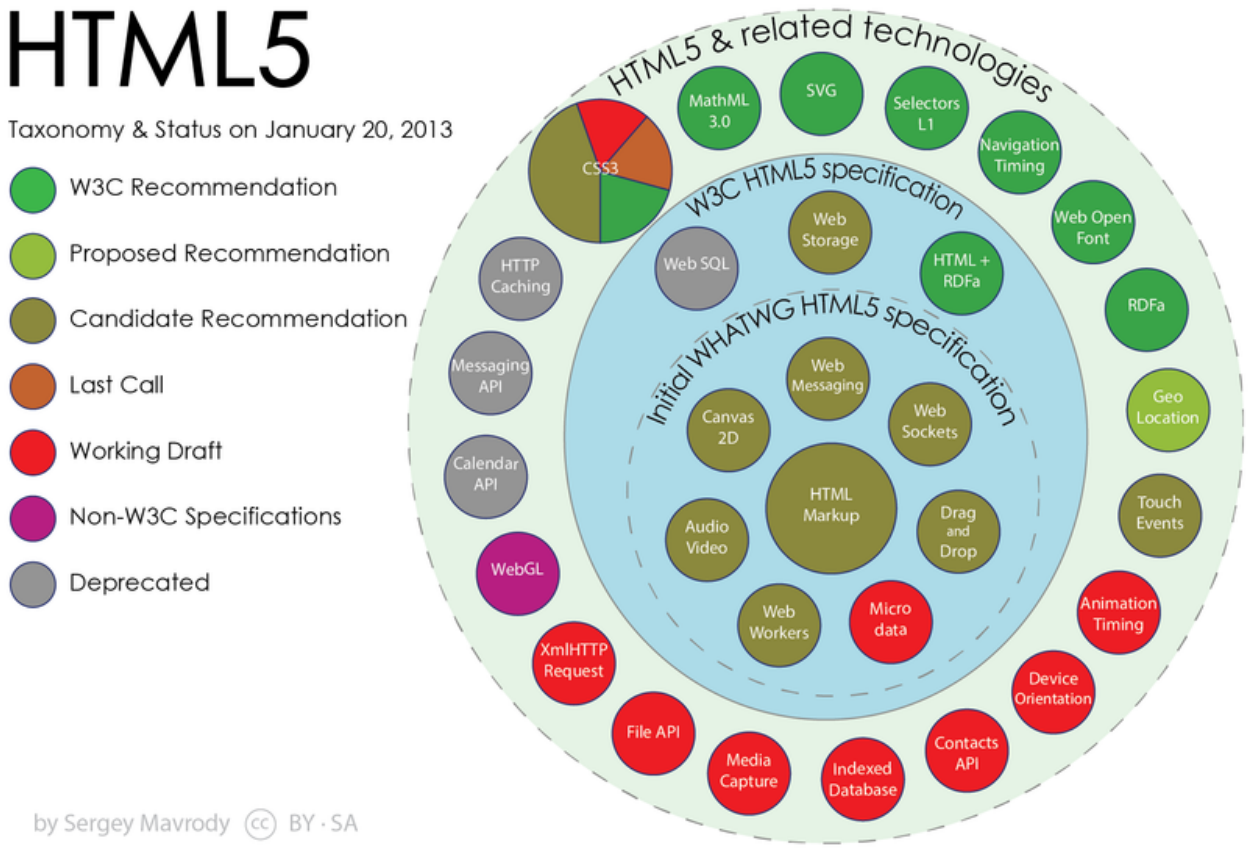

Figure 3: HTML5 specifications and related technologies [32].

Web Storage:

There are primarily three options in HTML5 for storing data on the client, i.e. browser: (1) Session Storage, (2) Local Storage, and (3) Indexed DB. We plan to use all of them for various purposes and details on them and its use are discussed below.

Session Storage: is a HTML5 technology used to solve the storage problems associated with cookies. Two main problems with the cookies are: (1) Separate instances of the web application cannot run, without interfering with each other typically, and (2) Storage limit is few kilobytes [38, 39]. Session Storage is fundamentally designed to solve both these problems. Data is stored in key / value pairs and its lifetime is limited to per tab or window only [38, 39]. We plan on using this storage for saving map and visualization states, including the currently selected area, visualization variables, zoom and resolution levels and device orientations.

Local Storage: is another technology for storing data in key / value pairs – but, data persists in the client beyond a page refresh [38]. As data is stored in client and does not need to be retrieved from the server after the first time, network traffic can be greatly reduced. Also, data is available for processing immediately. Majority of the browsers already support it [38, 39]. We plan to use this storage for storing visualization results and cached tiles.

But typically, Local Storage's limit is capped around 5MB (depending on the browser) [38, 39]. So, as the size and number of our visualization results grows, we are forced to either discard or overwrite previous visualization data, or worse cannot use Local Storage at all. Hence, in the future, we plan to use another storage called Indexed DB that can solve this problem.

Indexed DB: is an upcoming API that provides a high-performance NoSQL-like data store for storing large amounts of structured data [40]. Indexed DB is also a low level API that allows you to create databases, data stores, indexes, populate and iterate data [39]. It can also support data revisions [39, 40]. Typically, there is no storage limit, making it suitable to store large datasets.

While this seems to solve our need for storing large amounts of data in the client side, this technology is very new. API is complex, difficult and likely subject for further revisions [39, 40]. Besides, older browsers and mobile devices do not support it [39]. Due to these drawbacks, it is too early to adopt this technology. But, we plan to use it in future, once it's mature and stable.

CSS3 can be used to create rich animations. By moving the animations from scripts to styles, websites can be built with a cleaner codebase for visuals [42, 52]. **Web Messaging** and second version of **XMLHttpRequest (AJAX)** specifications support cross-origin communications. Additionally, standardizations for **device orientations, notifications and touch events** are in progress [42, 52]. We will use these upcoming technologies for rich visuals and supporting mobile devices once they are standardized.

Web Workers and **Web Sockets**, also part of HTML5 family of technologies are discussed in next sub-sections.

VIII. Web Workers

We have already discussed the problems of concurrency and complexity of multi-thread programming on server side earlier. This sub-section will discuss bringing threading to JavaScript on the client side.

Imagine a website that does common tasks - query and process large datasets, control DOM, and handle User Interface (UI) actions and events as well. As JavaScript is single threaded, so multiple JavaScript programs cannot run at the same time [47]. If they can be run concurrently, we can easily reduce the time needed.

Developers are already using few workarounds to handle this problem by using AJAX, setTimeout(), and event handlers [47]. Though these features are non-blocking and run asynchronously, they do not necessarily mean concurrency, i.e. only when one script has yielded, asynchronous events are processed [47, 48].

Web Workers in HTML5 aims to solve the problem, by bringing threading to JavaScript. Web Workers allows concurrent execution by breaking up huge tasks into small sets, so computationally intensive tasks such as large datasets processing and image filtering can be processed in the background in separate thread(s), without blocking the UI thread [47, 49].

To reduce typical concurrency problems such as deadlocks and starvation, Web Workers cannot access non-thread safe components or DOM [48]. The complexity of multi-thread programming and overhead to deal with common pitfalls such as deadlocks, starvation and race conditions makes it very hard [30, 31, 49]

However, there are few downsides of using Web Workers, due to memory cost and as it cannot access DOM as well. There are under active areas of research currently and resolutions to fix the same will be drafted in the near future [49, 50]. With JavaScript engines getting faster day by day, memory cost would come down. In future, we can even build faster websites, by running WebGL contexts in separate threads [49, 50].

IX. Web Sockets

The visualization performance of microscopic evacuation results is still not efficient enough due to high volume data communication between server and client. We have earlier discussed about our various approaches to solve this problem using: (1) Data storage on the client (browser) to store map tiles and simulation results, to reduce network traffic and improve performance, (2) Cloud Infrastructure, so dedicated nodes are available for sending data to client, and does not interfere with nodes used for simulation, and (3) Using fast read performance data storage system like NoSQL for retrieving and sending the data to the client.

There is another HTML5 technology called Web Sockets, which can improve our network performance as well. Typically, a client starts an HTTP connection to the server and keeps this connection alive until response has been sent. This is called “long polling” [52, 53]. But, the bottleneck is the overhead of HTTP, making it not well suited for low latency applications [53].

Web Sockets solves this problem by creating a persistent duplex connection between the server and the client [53]. Web Sockets enables server side applications to initiate and send data to the clients. Majority of the browsers already support it [52, 53].

Traditional web servers such as Apache Web Server use one thread per connection, and make them not well suited for large number of Web Socket connections [27, 53]. To support such large number of connections require a non-blocking architecture that provides high concurrency at low I/O cost - this architectural design is what is used in Node.js [27, 53].

Our Server Side application that is being built with Node.js, uses Web Sockets to send visualization data to the connected clients, instead of the clients constantly connecting, requesting data and overwhelming the server. With this approach, we hope to further save some network costs.

X. ExtJS

Last, but not the least, we will discuss about ExtJS, one of the leading JavaScript application frameworks, that is being used to create our interactive visualization web application. ExtJS 5

uses Model-View-ViewModel (MVVM) for building cleaner and modular codebase. It provides rich set of features – layout managers, data packages, two way data binding, delegated event model, highly optimized class system, and so on [41, 54]. Tons of plugins for quick creation of grids and trees, charting, and drawing are available [41]. ExtJS 5 is designed to support both mobile and web devices with one codebase [54]. Other notable benefits include dynamic loading, cross-browser compatibility, localization, unit and interface testing tools, debugging tools, good documentation, and active support community [41, 54].

4. CONCLUSIONS

In this paper, we have described various challenges and practices used to scale WWEE emergency evacuation framework. WWEE is used for simulations of urban transportation systems under critical infrastructure disruption in order to aid real-time emergency evacuation. On the back end, we have described need for an infrastructure to process large amounts of data, run the simulation efficiently and quickly, and provide fast retrieval and serving of data. We have addressed this drawback by busing cloud infrastructure, high performance NoSQL database for storage, and Node.js to solve I/O bottlenecks. On the front-end, the visualization performance of microscopic evacuation results is still not efficient enough due to high volume data communication between server and client. We are using next generation web technologies such as WebGL, Open Layers 3 and HTML5 collection of technologies to provide rich and blazingly fast interactive visualization. Our early experimentation demonstrates that using above big data and web technologies is a promising approach to build a scalable and high performance urban emergency evacuation framework and an efficient tool for emergency management agencies to improve traffic mobility and safety in emergency evacuation scenarios.

5. FUTURE WORK

We have built the base cloud infrastructure using Ubuntu MAAS. We have discussed about various approaches to scale back-end and front-end system. Next, we would like to scale our

middleware, i.e. simulator using our cloud infrastructure and other cutting-edge technologies. We are currently analyzing the challenges involved in the same.

We plan to measure the performance of our front-end application using Stats.js - frames per seconds (FPS), milliseconds needed to render frame (MS) and allocated memory (MB) [55].

Our application use JSON to retrieve data from our servers, do post processing, and finally copy data to WebGL buffers. This approach is problematic for large-scale visualizations. WebGL's rise in popularity has created a need for a new format that can represent rich data, and requires only minimal extra processing to be rendered [56].

glTF, the GL Transmission Format, is an upcoming runtime asset format specification, which aims to fill above need by directly loading data to WebGL buffers and designed to support rich data [56]. "Typed Arrays" is another technology that aims to solve the same problem without the complexity of glTF. It is part of next version of ECMAScript 6 specification and similar to how arrays work in C, it has an efficient way to support fast loading of binary data in WebGL [57]. We are analyzing both these upcoming specifications to support data exchange in our application.

6. ACKNOWLEDGEMENT

The authors would like to acknowledge Oak Ridge National Laboratory for providing the LandScan (2011)™ High Resolution USA Population Dataset, which is copyrighted by UT-Battelle, LLC, operator of Oak Ridge National Laboratory under Contract No. DE-AC05-00OR22725 with the United States Department of Energy. The United States Government has certain rights in this data set. Neither UT-Battelle, LLC nor The United States Department of Energy, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of the data set. The authors would like to acknowledge the support extended by Dr. Devin White and Dr. Cheng Liu.

7. REFERENCES

1. Lu, W., Han, L., Liu, C., Tuttle, M., and Bhaduri, B. (2013). Rapid Traffic Assignment with Agent-based Models and High Resolution Data: A Case Study of Emergency Evacuation Planning. Proceedings of Conference on Agent-based Modeling in Transportation Planning and Operations.
2. Lu, W., Han, L., Liu, C., Tuttle, M., and Bhaduri, B. (2014). Evacuee Compliance Behavior Analysis using High Resolution Demographic Information. Proceeding of 93th Annual Meeting of the Transportation Research Board.
3. Pel, A., Bliemer, M., and Hoogendoorn, S. (2012). A review on travel behaviour modelling in dynamic traffic simulation models for evacuations. Transportation, 39 (1) 97-123.
4. Alsnih, R., and Stopher, P. (2004). Review of procedures associated with devising emergency evacuation plans. Transportation Research Record: Journal of the Transportation Research Board, 1865 (1), 89-97.
5. Han, L., Yuan, F., Chin, S., and Hwang H. (2006). Global optimization of emergency evacuation assignments. Interfaces, 36 (6), 502-513.
6. Liu, Y., Chang, G., and Lai, X. (2008). Corridor-based emergency evacuation system for Washington, DC: system development and case study. Transportation Research Record: Journal of the Transportation Research Board, 2041 (1), 58-67.
7. Jha, M., Moore, K., and B. Pashaie. (2004). Emergency evacuation planning with microscopic traffic simulation. Transportation Research Record: Journal of the Transportation Research Board, 1886(1), 40-48.
8. Cova, T., and Johnson, J. (2002). Microsimulation of neighborhood evacuations in the urban-wildland interface. Environment and Planning A, 34(12), 2211-2230.
9. Smith, L., Beckman, R., and Baggerly, K. (1995). TRANSIMS: Transportation analysis and simulation system, Los Alamos National Lab., NM (United States).
10. Henson, K., and Goulias, K. (2006). Preliminary assessment of activity analysis and modeling for homeland security applications. Transportation Research Record: Journal of the Transportation Research Board, 1942 (1), 23-30.
11. OpenStack Open Source Cloud Computing Software (2014). Retrieved from https://www.openstack.org/

12. Wen, X., Gu, G., Li, Q., Gao, Y., and Zhang, X. (2012)..Comparison of open-source cloud management platforms: OpenStack and OpenNebula. Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery, May 29-31, IEEE Xplore Press, Sichuan, pp: 2457-2461. DOI:10.1109/FSKD.2012.6234218
13. 6 OpenStack Distributions that Help Enterprises JumpStart Private Cloud Deployment (2014). Retrieved from http://yourstory.com/2014/02/6-openstack-distributions-help-enterprises-jumpstart-private-cloud-deployment/
14. Deploying OpenStack Is Difficult? It's A Myth!! (2011). Retrieved from http://www.cloudave.com/15100/deploying-openstack-is-difficult-its-a-myth/
15. OpenStack with Ubuntu. (2014). Retrieved from http://www.ubuntu.com/cloud/ubuntu-openstack
16. Ubuntu OpenStack. (2014). Retrieved from http://resources.canonicalwebteam.com/wp-content/uploads/Ubuntu-OpenStack.pdf
17. MAAS. (2014). Retrieved from http://www.ubuntu.com/cloud/tools/maas
18. The top 10 questions about MAAS. (2014). Retrieved from http://resources.canonicalwebteam.com/wp-content/uploads/Top-10-Questions-about-MAAS.pdf
19. Deploying workloads with Juju and MAAS in Ubuntu 13.04. (2013). Retrieved from http://linux.dell.com/files/whitepapers/Deploying_Workloads_With_Juju_And_MAAS.pdf
20. mongoDB - A BRIEF INTRODUCTION. (2014). Retrieved from http://www.10gen.com/static/downloads/mongodb_introduction.pdf
21. Karthik, R. (2013) PASS-CS: A Promising Advance in Personalized, Spatialized, and Scalable Content Search. Presentation on Association of American Geographers Annual Meeting, Los Angeles, CA, United States.
22. Paudyal, U. (2011). Scalable web application using node.JS and CouchDB. Uppsala University.
23. Node.js. (2014). Retrieved from http://nodejs.org/
24. Cluster Node.js. (2014). Retrieved from http://nodejs.org/api/cluster.html
25. Intro to Node.js. (2014). Retrieved from http://www.slideshare.net/ArtemisaYescasEngler/about-nodejs

26. Tilkov, S., and Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. Internet Computing, 14, 6 (2010) 80-83.
27. What is Node.js? (2014). Retrieved from http://stackoverflow.com/questions/1884724/what-is-node-js
28. McCune, R. (2011). Node.js Paradigms and Benchmarks. STRIEGEL, GRAD OS F'11, PROJECT DRAFT.
29. Chaniotis, I., Kyriakou, K-I., and Tselikas, N. (2014). Is Node.ks a viable option for building modern web applications? A performance evaluation study. Computing. DOI: 10.1007/s00607-014-0394-9
30. Pasquali, S. (2013). Mastering Node.js. Packt Publishing. ISBN **:** 1782166327
31. Hartweg, J. (2012). Web Application development with Node.js: Can it already compete with the Apache web server? Mid Sweden University.
32. HTML5 – Wikipedia. (2014). Retrieved from http://en.wikipedia.org/wiki/HTML5.
33. L. Xue, Q. Yangming, and L. Lei. (2012). Visualization of Geomagnetic Environement Based on WebGL. Computational Intelligence and Design (ISCID).
34. Cantor, D., Jones, B. (2012). WebGL Beginner's Guide. Packt Publishing.
35. Parisi, Tony. (2012). WebGL Up and Running. O'Reilly Media, Inc.
36. Engel, W. (2013). GPU Pro 4: Advanced Rendering Techniques, Volume 4. CRC Press.
37. Three,js – JavaScript 3D library. (2014). Retrieved from http://threejs.org/
38. THE PAST, PRESENT & FUTURE OF LOCAL STORAGE FOR WEB APPLICATIONS. (2014). Retrieved from http://diveintohtml5.info/storage.html
39. HTML5 Browser Storage: the Past, Present and Future. (2014). Retrieved from http://www.sitepoint.com/html5-browser-storage-past-present-future/
40. IndexedDB. (2014). Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
41. Myers, A. Movva, S., Karthik, R., Bhaduri, B., White, D., Thomas, N., and Chase, A. (2014). BioenergyKDF: Enabling Spatiotemporal Data Synthesis and Research Collaboration. Submitted to 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems 2014.
42. Karthik, R., Patlolla, D., Sorokine, A., White, D., and Myers, A. (2014). Building a Secure and Feature-rich Mobile Mapping Service App Using HTML5: Challenges and Best

Practices. Submitted to 12th ACM International Symposium on Mobility Management and Wireless Access (MobiWac) 2014.

43. Why we are building OpenLayers 3? (2012). Retrieved from http://openlayers.org/blog/2012/11/14/why-are-we-building-openlayers-3/
44. Closure Tools. (2014). Retrieved from https://developers.google.com/closure/compiler/.
45. Open Layers 3. (2013). Retrieved from http://www.slideshare.net/camptocamp/ol3-26732198
46. Cesium. (2014). Retrieved from http://cesiumjs.org/
47. The Basics of Web Workers – HTML5 Rocks. (2010). Retrieved from http://www.html5rocks.com/en/tutorials/workers/basics/
48. Harjono, J., Ng, G., Kong, D., and Lo, J. (2010). Building smarter web applications with HTML5. CASCON.
49. Pilgrim, M. (2010). HTML5: Up and Running. O'Reilly Media, Inc.
50. Cozzi, P., and Riccio, Christophe. (2012). OpenGL Insights. CRC Press.
51. Yang, C., Raskin, R., and Goodchild M. (2010). Geospatial cyberinfrastructure: Past, present and future. Computers, Environment and Urban Systems, 34(4), 264–277.
52. Lubbers, P., Albers, B., and Salim, F. (2010). Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. Apress.
53. Introducing Web Sockets. (2010). Retrieved from http://www.html5rocks.com/en/tutorials/websockets/basics/
54. Ext JS Guides | What's New in Ext JS 5. (2014). Retrieved from http://docs.sencha.com/extjs/5.0.0/whats_new/5.0/whats_new.html
55. JavaScript PErformance Monitor. (2014). Retrieved from https://github.com/mrdoob/stats.js/
56. Parisi, T. (2014). Programming 3D Applications with HTML5 and WebGL : 3D Animation and Visualization for Web Pages. O'Reilly Media, Inc.
57. Typed Arrays: Binary Data in the Browser. (2012). Retrieved from http://www.html5rocks.com/en/tutorials/webgl/typed_arrays/

GLOSSARY:

- Cesium: a virtual globe and map engine based on WebGL [46].
- Cluster controller(s): controls provisioning and management of a cluster (or group of Nodes) [18, 19].
- gLTF: an upcoming runtime asset format specification, which aims to directly load data to WebGL buffers, avoiding unnecessary processing [56].
- ExtJS: one of the leading JavaScript application frameworks that is being used to create our interactive visualization web application using MVVW architecture [41, 54].
- LandScan: a high-resolution population distribution database, developed at ORNL [1, 2].
- MAAS: a quick and easy-to-use tool for transforming machines into a cloud [17].
- Node: an individual machine. This is different from Node.js technology.
- Node.js: a server-side JavaScript environment for easily developing scalable network applications using asynchronous or non-blocking I/O model [26, 28].
- MongoDB: one of the popular NoSQL solutions [20, 21].
- OpenStack: "is the world leading open-source cloud platform and provides all the components needed to build and deploy an operational open-source cloud" with compute, storage and network components [16].
- OpenStreetMap (OSM): a public domain global street network [1, 2].
- Open Layers 3: a cutting edge and high-performance mapping library, being developed by OSGeo to provide rich user experience including 3D using HTML5, WebGL, Cesium, CSS3 and Closure [41, 22].
- Postgres: one of the leading RDBMS [20, 21].
- Region controller: Easy and quick management of resources in the cloud via Web UI [18, 19].
- Three.js: provides various common utilities and helper functions to make use of WebGL simpler [37].
- Typed Arrays: ECMAScript 6 specification that aims to support fast loading of binary data in WebGL, without the complexity of glTF [57].
- Ubuntu Server: is the operating system of the cloud [16].

- WebGL: a JavaScript API for rendering rich and interactive graphics including 3D in a browser without the need for plug-ins [33].
- Web Workers: a HTML5 specification that brings threading to JavaScript [47, 49].
- Web Sockets: a HTML5 specification that provides persistent duplex connection between the server and the client [53].
- Web Storage:
    - Session Storage: a HTML5 technology used to solve the storage problems associated with cookies [38, 39].
    - Local Storage: another HTML5 technology for storing data in key / value pairs – but, data persists in the client beyond a page refresh [38].
    - Indexed DB: an upcoming HTML5 API that provides a high-performance NoSQL-like data store for storing large amounts of structured data [40].