# Cloud Computing Services for Seismic Networks

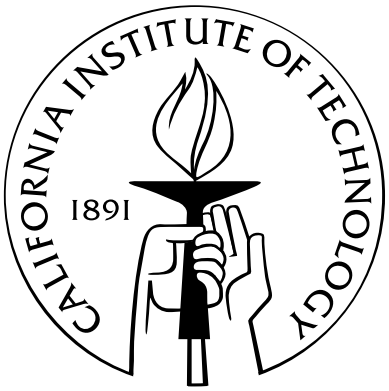
Thesis by

Michael Olson

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy




California Institute of Technology

Pasadena, California

2014

(Defended August 3, 2013)

# Acknowledgements

I would like to acknowledge first and foremost my advisor, K. Mani Chandy, for his encouragement and support through all of my years at Caltech. Without his guidance, this work would not have been possible. To have had the chance to work with and learn from Mani is an opportunity for which I will always be grateful.

I am also fortunate to have had the support of my committee: Greg Billock, Rob Clayton, Tom Heaton, and Adam Wierman. Their encouragement and suggestions for improving my thesis were instrumental in improving the quality of this work.

Much of the data acquired for this thesis could not have been obtained without the commitment of the entire CSN team, which includes Mani, Rob, and Tom, as well as: Michael Aivazis, Julian Bunn, Ming-Hei Cheng, Matt Faulkner, Richard Guy, Monica Kohler, Annie Liu, and Leif Strand. Their commitment to making the Community Seismic Network project a success is what has brought the network to the point that this thesis was possible.

The Computer Science department at Caltech is also owed my thanks. The efforts of the faculty and administrators on the students behalf was always appreciated. I am particularly grateful to Diane Goodfellow and Maria Lopez, whose help made life in the department that much simpler. I am also thankful for Alain Martin and Mathieu Desbrun, whose genuine thoughtfulness made daily life more pleasant.

Finally, my entire time at Caltech would have likely been very different had I not met my wife, Indira Wu. We began our time here together, and we have walked this road alongside one another. Now that we our done, we will have nothing but the fondest memories for our time spent at Caltech, the place where we first learned what it was to be us.

# Abstract

This thesis describes a compositional framework for developing situation awareness applications: applications that provide ongoing information about a user's changing environment. The thesis describes how the framework is used to develop a situation awareness application for earthquakes. The applications are implemented as Cloud computing services connected to sensors and actuators. The architecture and design of the Cloud services are described and measurements of performance metrics are provided. The thesis includes results of experiments on earthquake monitoring conducted over a year. The applications developed by the framework are (1) the CSN — the Community Seismic Network — which uses relatively low-cost sensors deployed by members of the community, and (2) SAF — the Situation Awareness Framework — which integrates data from multiple sources, including the CSN, CISN — the California Integrated Seismic Network, a network consisting of high-quality seismometers deployed carefully by professionals in the CISN organization and spread across Southern California — and prototypes of multi-sensor platforms that include carbon monoxide, methane, dust and radiation sensors.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

This thesis describes a compositional framework for developing situation awareness applications: applications that provide ongoing information about a user's changing environment. I describe in detail how the framework is used to develop a situation awareness application for earthquakes.

A situation awareness application enables users to remain aware of the state of a changing system. Situation awareness applications are used by the military to monitor battlefields, by first responders to keep track of the changing situation during hazards, by businesses that keep track of metrics such as inventories and sales, and by consumer applications, such as an application that keeps drivers in cars informed about changes in road congestion. This thesis describes a compositional, or "plug-and-play" framework for developing situational awareness applications.

The thesis describes the design and implementation of situational awareness applications for earthquakes and gives experimental measurements and performance metrics of the implemented systems. The applications consist of the *CSN* — the *Community Seismic Network* — which uses relatively low-cost sensors deployed by members of the community, and *SAF* — the *Situation Awareness Framework* — which integrates data from multiple sources, including the CSN, CISN — the California Integrated Seismic Network, a network consisting of high-quality seismometers deployed carefully by professionals in the CISN organization and spread across the Southern Region — and carbon monoxide, methane, dust and radiation sensors [1].

This chapter consists of brief descriptions of situation awareness applications, the Community Seismic Network, and the Situation Awareness Framework, followed by a summary of the contribu-

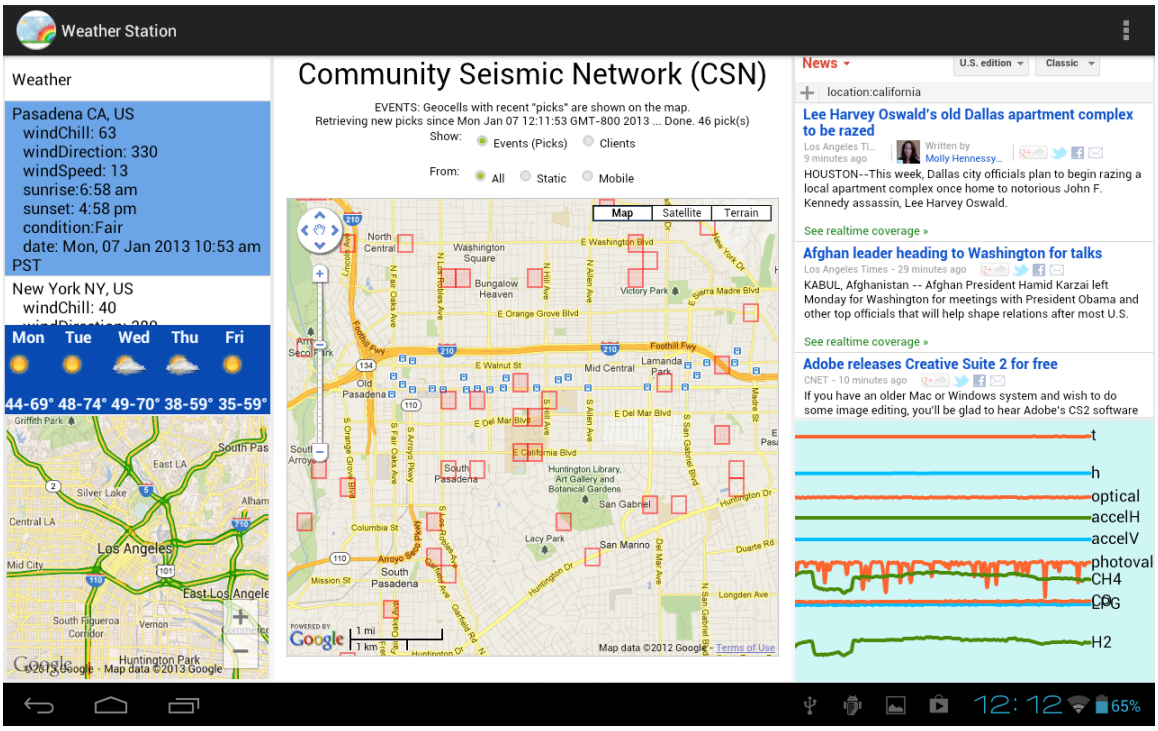



Figure 1.1: A screen shot of a situation awareness application integrated with the Situation Awareness Framework. Screenshot courtesy of Judy Mou [1].

tions of this thesis.

## 1.1 Situation Awareness Applications

Users want to remain aware of situations that are important to them, such as hazards — including fires, earthquakes, and toxic plumes — ongoing communication — such as email and social networks — and real time information streams — such as news and financial market data. The ideal situation awareness application gets a user's attention when, and only when, the user should pay attention. If the trajectory of a situation is such that attention is warranted then the application issues some sort of alert (visual, auditory, or a command to a device) and keeps the relevant information in front of, or readily available to, the user. If, however, the trajectory is consistent with the user's expectation then no alert is generated and situational information is not pushed to the user.

An example of a situation awareness application is described in [1]; this application uses cloud computing services that were designed and implemented as part of the thesis research. Figure 1.1

shows a screen shot of the application. The screen consists of multiple tiles; each tile shows a sequence of images depicting different aspects of the user's situation. For example, a tile may show, in sequence: traffic congestion along the user's commute, news relevant to the user, electrical energy consumption and sensor (carbon monoxide, dust, movement) information from the user's home, and global information about hazards in the area, such as fires, heat waves, and earthquakes. When a user's attention is required, a visual or auditory alert can be provided; for example, if an earthquake is detected, then a shake map — a map showing the intensity of shaking over an area — is displayed continuously. This type of application operates in two modes: a background mode where the screen shows continuously changing relevant information that doesn't require the user's immediate attention, and an alert mode in which information about a critical situation is continuously pushed to the screen.

A situation awareness application consists of data sources such as sensors, a communication medium such as the Internet, a data fusion engine that integrates data streams from all sources and detects critical conditions, and actuators that operate equipment or provide information on computers or phones. These components may be designed and used explicitly for the application, or they may be consumer components used in popular applications. For example, CISN uses high-quality seismometers and special-purpose wholly-owned communication systems and computing services. A typical CISN installation is pictured in fig. 1.2. Likewise, most electric utilities use special meters and their own communication and computing systems to monitor power consumption. By contrast, the CSN uses shared infrastructure components — including volunteer provided Internet service and a public cloud service shared with thousands of other applications — and consumer class electronics, including popular tablets, phones and single-board computers such as the Raspberry Pi. The thesis describes the architecture of a framework for developing situation awareness applications and provides experiments and measurements for applications developed within this framework.

Figure 1.2: A typical CISN installation, in which an Episensor is placed in the ground.

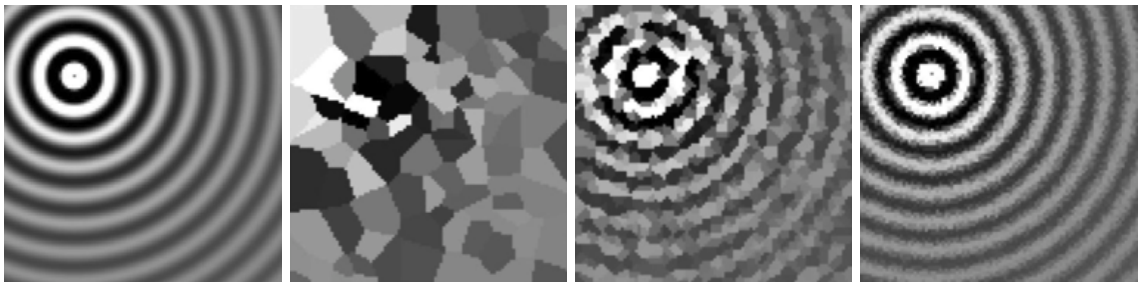
## 1.2 The Community Seismic Network

The Community Seismic Network, described in full in [7], is a dense, relatively inexpensive network of sensors. High densities of sensors have advantages, design consequences, and disadvantages. These are described next, followed by a brief overview of the CSN.

### 1.2.1 Benefits of Dense Networks

There are several advantages to a dense community network. First, higher densities make the extrapolation of what regions experienced the most severe shaking simpler and more accurate. In sparse networks, estimating the magnitude of shaking at points other than where sensors lie must model subsurface properties which may not be known precisely. As you can see in fig. 1.3, a dense network makes visualizing the propagation path of an earthquake and the resulting shaking simpler. A dense network can generate a block-by-block shake map based on measurements taken at more points within the impacted region.

Second, dense networks within a single building can be used to establish whether buildings have



(a) Ideal. (b) Present. (c) 1,000 Sensors. (d) 10,000 Sensors.

Figure 1.3: How sensor density affects the detection of seismic waves.

undergone deformations during earthquakes; often such deformations cannot be visually ascertained. Using community established sensors, rather than instrumenting the structure during construction, makes it possible to establish information about older buildings, and it also makes it possible for building residents to understand the properties of the building - and the ways in which those properties change - whether or not they have access to the readings of any instrumentation put in place by the building owners.

Third, inexpensive networks of small sensors can be deployed rapidly and at relatively low cost in regions of the world that do not have existing seismic networks operated by governmental or other agencies. Also, networks can quickly be deployed to recently shaken regions for data collection. As the network's fusion center exists in the cloud, sensors deployed in any country with access to the Internet rely on existing infrastructure for detection.

### 1.2.2 Design Considerations of Dense Networks

Sensors in dense, low-cost networks must, perforce, be inexpensive. Moreover, the cost of deploying a sensor has to be low. Constraints on both the cost of each sensor and the cost of deployment have critical consequences in design.

**Inexpensive Sensors** Accelerometers used in consumer applications, such as phones, cars, and household robots, are getting better; this is partly a consequence of the demand for more sophisticated consumer applications and the cost savings obtained by manufacturing at scale. Nevertheless, accelerometers for consumer applications have much more electronic noise than seismometers used in

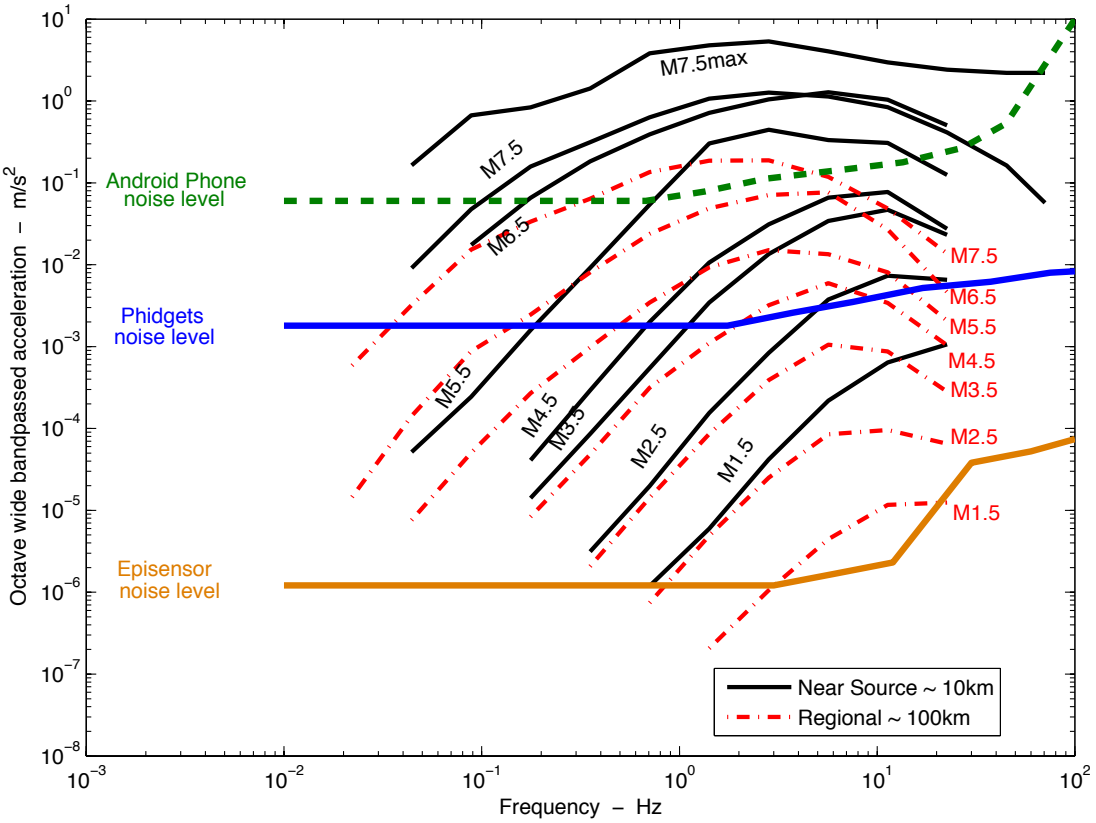



Figure 1.4: Different types of sensors have different sensitivities to seismic waves. This figure shows the distances at which specific types of sensors could expect to detect shaking from the associated event. Figure courtesy of Ming Hei Cheng.

networks such as CISN. Figure 1.4 shows the different sensitivities of an accelerometer in an Android phone, a Phidget used in CSN, and an Episensor as used in CISN. A consequence of the increased electronic noise in inexpensive accelerometers is that data from multiple accelerometers needs to be fused to obtain better estimates of the acceleration at a point; this assumes that electronic noise generated in different sensors are independent.

Each accelerometer is connected to a nearby computer, called the client computer, through a USB cord or other connection. Data from each accelerometer is fed to a program (the client program) running on the local computer. The noise characteristics of the accelerometer depend, in part, on the client computer to which it is attached. Figure 1.5 shows noise from the same model of Phidget accelerometer connected to different notebooks. The Phidgets were placed on a shake table and simultaneous measurements from both devices were taken. The noise level of the sensor connected

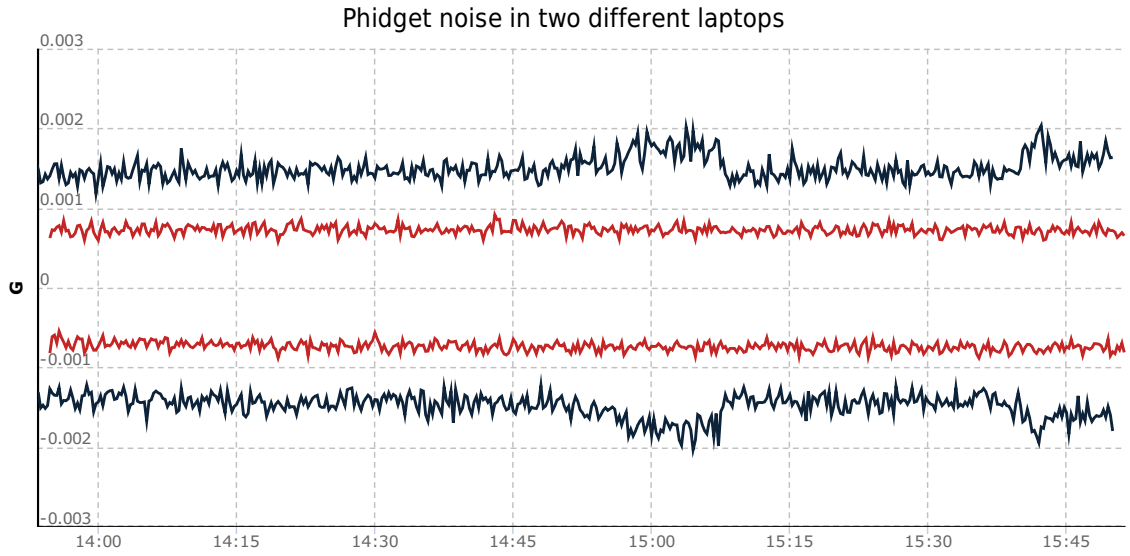



Figure 1.5: Noise from a Phidget device connected to two different hosts. The blue line shows a data envelope computed over the sensor readings when connected to a generic netbook, while the tighter red line shows the data envelope from a sensor connected to an IBM laptop while positioned on the same shake table as the netbook's sensor. Data courtesy of Ming Hei Cheng.

to the netbook is much higher.

The accelerometers used in the CSN have lower sensitivity than those used in CISN. Most CSN accelerometers generate data with at most 16 bits of information whereas CISN seismometers generate data with at least 24 bits of information.

**Inexpensive Deployment** Members of the community purchase or otherwise acquire CSN sensors and then deploy sensors by connecting them to their computers. CSN sensors can also be obtained in a package with a stand-alone single board computer, and in this case the CSN volunteer needs only connect that sensor box to a power socket and a router. Thus, the cost of deploying a sensor is very low.

CSN volunteers are told how to deploy sensors; for example, rather than leaving the sensor on a table, sensors should be affixed to the floor with double-sided sticky tape or other stabilizing mechanism. Volunteers are also asked to register their sensors and provide information about where the sensors are located, e.g., "affixed to the ground floor of a one story wood-frame house", and also provide their address or its latitude, longitude coordinates. Operators of the CSN cannot afford to

monitor the deployment of each sensor. Moreover, volunteers may move their sensors.

CISN seismometers, in contrast to CSN accelerometers, are deployed in a more uniform, consistent way by a crew managed by the organization. "Borehole" installation seismometers are placed in a hole in the ground dug by a backhoe. Uniformity of calibration and deployment of seismometers makes a difference in the quality of inferences obtained by fusing data from multiple sensors. Moreover, once a seismometer has been placed in a receptacle prepared for it, the seismometer remains in place; by contrast, CSN accelerometers can be moved easily and may be turned off by volunteers. Indeed, not knowing precisely how and where CSN sensors are deployed poses a greater challenge than the lower sensitivity and greater noise (when compared with seismometers) of the sensors.

**Inexpensive Communication** Data from a CSN sensor is obtained by the local client computer which sends it through a router to the Internet. In some cases, the local client connects by Wi-Fi to a router, and it is even possible for a sensor to communicate using SMS in 2G or 3G networks [8]. The cost of communication in the CSN is small because the CSN uses conventional Internet access methods that are provided and managed by volunteers. The cost to volunteers is also small; existing network connections are used, and the amount of data transmitted by CSN devices is quite small. Control traffic to the network is quite small, and consumes less than 1 MB per day per sensor. Waveforms are larger, but all waveforms can be transmitted from a CSN device in less than 150 MB per day.

The Internet may fail in the seconds after a major quake, and phone networks may get so congested that SMS messages reach their destination only after the region has suffered damage. Thus, a constraint on the design of the CSN is that the time to get sensor data to the fusion engine and back to actuators must be less than the time before communication networks fail. An advantage of building a special-purpose communication network, such as the network used in CISN, is that it is used only for responding to hazards, it can be hardened to withstand hazards, and it does not suffer from congestion due to traffic from other sources.

**Inexpensive Computation** Data streams from CSN sensors are aggregated and analyzed in a program executing on a commercial, public, cloud service: Google App Engine [9]. Google App Engine's High Replication Datastore [10] stores data in multiple data centers simultaneously, removing the vulnerability of sensor data to localized hazards. The costs of computation for cloud services in general are quite low, owing to economies of scale. Situation awareness applications, which require the most computational power during hazards, are able to consume only the resources they need during periods of low activity. In owning infrastructure, sufficient computational power to operate during hazards would need to be always available (overprovisioning), increasing ongoing infrastructure costs.

The computation engine in CISN, unlike the CSN, is used for a single purpose: monitoring earthquakes. An advantage of not sharing a system among a critical application and other applications is that the behavior of the critical application is more predictable because the system does not have to deal with variations in load from other applications.

The configuration of services in the cloud for the CSN must be done carefully to trade-off predictability of performance with cost.

**Volunteer Concerns** Some volunteer organizations, such as research centers and universities, are willing to have their sensors execute HTTP POST requests on cloud services with messages going *out* from their sites over secure subnetworks set up for this purpose. They may, however, be unwilling to have a cloud service send messages to computers on their networks. Therefore, the system is designed to have all communication initiated by client computers. As a consequence, if a client is unable to communicate with the host service running in the Google App Engine, there is no way for the host service to reboot, or otherwise repair, the client.

For privacy reasons, maps produced by the CSN do not generally show the precise locations of sensors; they show aggregate measurements in an area. If, however, a major earthquake occurs, then first responders are allowed to see precise sensor locations to make use of the accelerations recorded at those locations.

### 1.2.3 An Overview of the Community Seismic Network

The CSN is a dense, but relatively inexpensive, situation awareness application for monitoring earthquakes; major challenges in its design are the constraints imposed by the need to use inexpensive sensors, client computers, communication, and computation for data fusion, and by the need to scale to a large and varying number of sensors.

The CSN is described in detail in section 5.1, here I merely give an idea of how the goals of high density and low cost impact design. Many CSN sensors are connected to volunteers' computers; therefore, the amount of computation on these computers has to be small so that volunteers don't notice an impact in their normal use. Likewise, since communication from CSN sensors to data fusion engines travels along the same connections as the volunteer's other data, the volume of communication sent from the volunteer's computer must be limited.

A typical sensor in the CSN, a PhidgetSpatial 3/3/3, model 1056 [11], measures acceleration in three ($x$, $y$, $z$) dimensions, uses 16 bits for the value in each direction, and is sampled 200 times per second. Though it is possible for thousands of sensors to send their entire waveforms to the fusion engine, we designed the system so that a sensor need only send a message when it detects a local anomaly; such a message is called a "pick". These messages are brief, containing only the sensor's location, anomaly time, and a few statistics such as the peak observed acceleration. A sensor's location is particularly important for data sent from accelerometers in phones. The process of pick detection is discussed in section 6.1. Pick messages are integrated and analyzed in a data fusion engine, running in Google App Engine, to obtain shake maps and detect earthquakes. Shake maps are then pushed to subscribing devices and are made available at the CSN website. A central aspect of the CSN design is the use of widely-used technologies such as the Internet; for example, clients communicate with the data fusion engine using HTTP POST, and maps are visible on web browsers.

An overview of the CSN infrastructure is presented in fig. 1.6. A cloud server administers the network by performing registration of new sensors and processing periodic heartbeats from each sensor. Pick messages from the sensors are aggregated in the cloud to perform event detection.

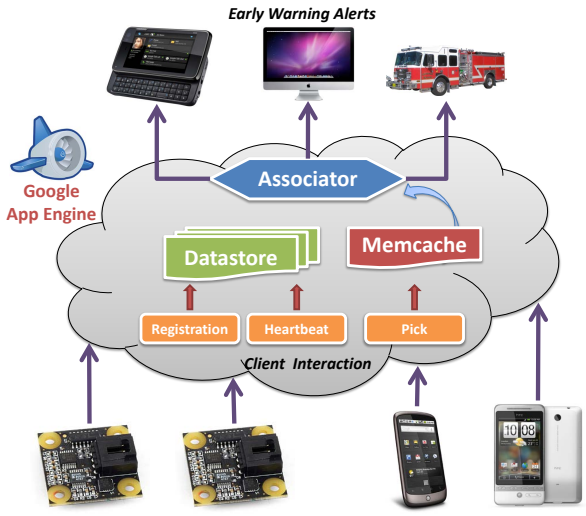



Figure 1.6: Overview of the CSN architecture.

When an event is detected, alerts and shake maps are issued to the community.

## 1.3 The Situation Awareness Framework

### 1.3.1 A Cloud Service for Situation Awareness

I developed the Situation Awareness Framework (*SAF*), which is described in chapter 7. The framework allows different types of data sources to be connected to, and managed by, a cloud service. All data sources — whether sources of news, changes in one's personal social network, or sensors — have a common life cycle. They register with the service; send heartbeats periodically to indicate that they are functioning and to get information from the service; send messages; and finally they de-register and are removed from the application. Likewise, actuators and other receivers of information from the service have a common life cycle. Critical patterns across time and across multiple sensors are detected by the cloud service and alerts and related information are sent to actuators. See section 4.1.2 for the typical message types used by a client. I developed a framework to which new types of sensor and actuators, and software modules for detecting new critical patterns, can be added to a cloud service.

The benefit of this framework is demonstrated by connecting a variety of sensors from a home

hazard monitoring station developed by Sandra Fang and Julian Bunn [12]. The station includes sensors for detecting carbon monoxide, methane, dust, and radiation. I also show how the framework integrates data streams from CSN and CISN sensors. I demonstrate the feasibility of the framework's global reach by showing that sensors from an experimental prototype network installed by IIT Gandhinagar in India can be integrated seamlessly. Preliminary experiments with SAF, described in chapter 6, suggest that it can scale to handle millions of sensors.

### 1.3.2 Integration of Community and Agency Networks

CSN sensors can be deployed in areas that do not have more expensive seismic networks, such as CISN. In cases where the CSN is the only seismic network, it is used both for detecting earthquakes, early warning, and continuing generation of near real-time shake maps.

In addition, the CSN adds value to regions, such as Southern California, that do have CISN-like networks because the CSN adds dense data points, especially around population centers and critical resources. A number of CSN sensors can be placed around a resource such as a transformer or a communication switch to provide local measurements at low cost. The combination of dense local measurements provided by CSN with detection and early warning provided by CISN gives users more accurate information than from CISN alone.

SAF is designed to integrate different types of sensors deployed by different organizations. In the future, we expect to have sensors from organizations besides CSN and CISN integrated with SAF; however, the experiments reported in this thesis were conducted with data from only CSN and CISN sensors.

We use the same algorithm to generate picks from CSN and CISN stations. Figure 1.7 shows the number of picks generated by a CSN and a CISN station during a quiescent period — one in which no earthquake was detected. The picks generated by the CSN station show a pattern of variation over the day and a pattern over days of the week, with fewer picks when there are fewer people and less traffic. Likewise, picks from CISN show diurnal variation and variation over the week.

Figure 1.8 shows the rates of picks from the CSN and CISN during the Sea of Okhotsk earthquake

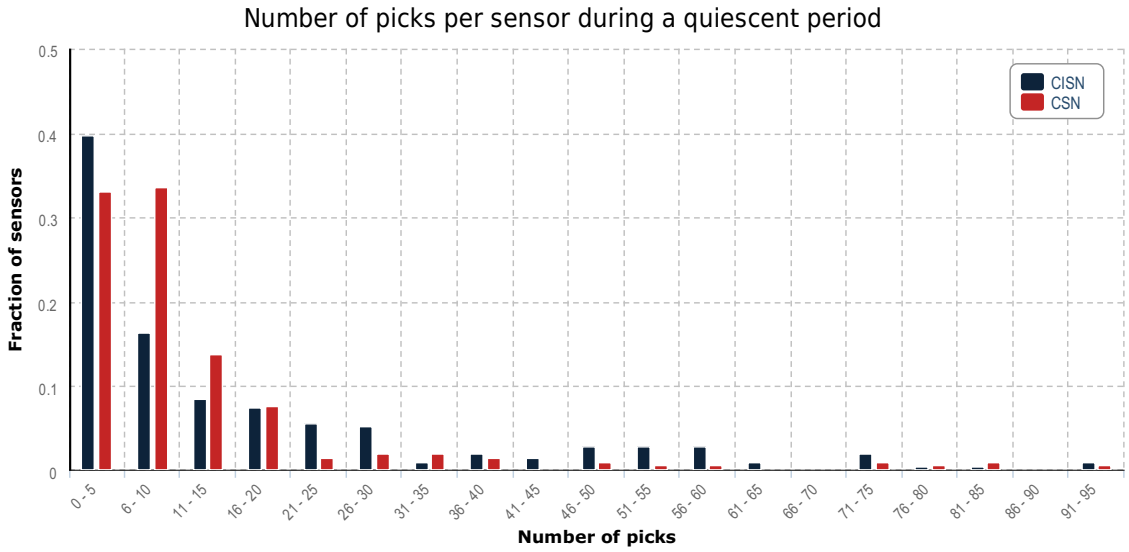



Figure 1.7: Pick rate of CSN and CISN stations during a quiescent period. Picks are generated using the algorithm described in section 6.1.1.2.

in Russia. The excitation of the CISN sensors was sufficient to produce an event notification by USGS for an event in Running Springs of magnitude 4.5, while the CSN sensors could not detect an ongoing event. This is an example where network fusion would be useful; knowing that CSN sensors are capable of detecting a magnitude 4.5 event, it may be possible to determine that the earthquake is out-of-network by noticing that nearby CSN sensors were incapable of detecting the event.

SAF can integrate data from accelerometers and seismometers deployed by CSN, CISN and other organizations, as well as data from smoke, carbon monoxide, and other gas sensors, to provide first responders better situation awareness than is possible with CISN alone.

## 1.4 Contributions and Structure of the Thesis

**A Compositional Framework** My contribution is the development of a plug-and-play framework based on cloud services for building situation awareness applications; the framework includes support for the following features:

1. Deploying, managing and operating sensors and other data sources.

2. Learning the characteristics of individual data streams.

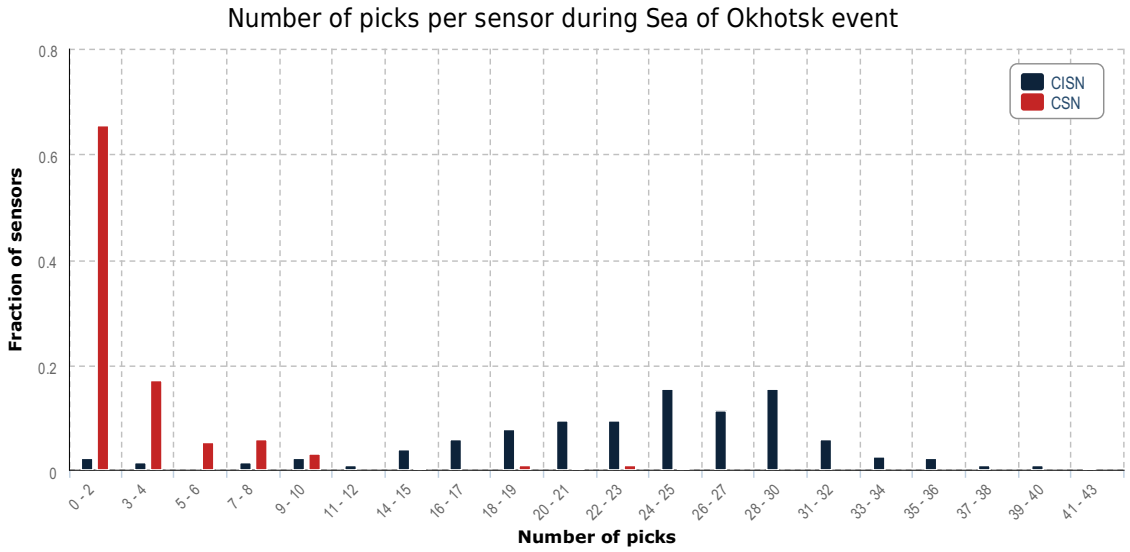



Figure 1.8: Pick rate of CSN and CISN stations during the Sea of Okhotsk magnitude 8.3 earthquake in Russia.

3. Fusing streams of data from multiple sources to detect the presence of hazards and other unusual situations and to provide global situation awareness.

4. Sending data streams to people and equipment to provide ongoing situation awareness as a scenario unfolds.

5. Archiving data in data vaults for further analysis.

The overall goals of the system are described in chapter 2.

**Integrating Consumer Devices into Sense and Response Systems** Scaling to thousands of data sources and millions of people and actuators can be done at reasonable cost by exploiting systems and devices that were developed for consumer applications. Examples of such systems and devices include cloud-computing services like Google App Engine [9], used primarily for web-based applications, sensors such as accelerometers used in consumer electronics, inexpensive single-board computers such as the Raspberry Pi [13], computer tablets, and smart phones. I describe the implementation of a system — the Community Seismic Network — that integrates devices and systems used primarily in consumer applications. I also show how devices used primarily in scientific

applications can be integrated with systems that exploit consumer devices.

The use of these components as both sensors and actuators is discussed in chapter 3.

**Scaling Situation Awareness Applications** Many situation awareness systems used in scientific research and industrial applications use relatively small numbers of sensors [14]. My thesis shows how to scale situational awareness systems in the following ways:

1. **scaling numbers of data sources**: The compositional framework allows large numbers of sensors to be integrated into the application seamlessly. The framework can support a thousand sensors, and I provide data to show that it can scale further as data sources increase.

2. **scaling numbers of types of data sources**: The framework enables rapid integration of different types of sensors by using a common data structure. I demonstrate the feasibility of this framework by fusing data from different sensor types.

3. **rapidly changing numbers of data sources**: In large networks, sensors may fail, and sensors may be turned off and on. Sensors connected to mobile devices may move from one location to another. I show how a cloud-based framework can manage rapidly varying numbers of sensors.

4. **scaling across geographical regions**: I describe a framework that enables all the functionality for situation awareness — data acquisition, data fusion, and situation awareness messages to users — to be carried out across the globe.

   The design of such a system is described in chapter 4, and its implementation is discussed in chapter 5.

**Design, Implementation, Experiments, and Measurements** The thesis specifies the goals of the Community Seismic Network and a generalized Situation Awareness Framework, describes a design that meets those goals, shows how the design was implemented, and provides extensive measurements of a working system.

The results of conducted experiments and measurements of the system are provided in chapter 6, while an overview of the implemented system and remaining work is left to chapter 7.

# Chapter 2

# Goals and Performance Metrics

This chapter first presents goals for situation awareness frameworks and then presents the criteria — the performance metrics — by which the efficacy of the framework is judged. Later chapters describe the design and implementation of the framework, experiments, and performance metrics measured in the implementation. I discuss goals and metrics for situation awareness in general, and then specifically for earthquakes. The discussion for earthquakes first considers systems which only have dense, inexpensive networks (CSN), and later considers systems in which the CSN and CISN are integrated.

## 2.1 Goals

The goal of situation awareness applications is to detect events that are important to each user and to push relevant information to the user's devices.

The goals of CSN, when it is operating without a network, such as CISN, that has accurate seismometers are:

1. Detect earthquakes early and alert users and equipment.

2. Provide shake maps that show ground movement at different locations.

3. Estimate parameters of the earthquake such as its hypocenter, time of initiation, and magnitude.

4. Store sensor data in the cloud and in servers at participating institutions so that scientists can analyze data easily.

5. Provide tools by which scientists and all volunteers in CSN can see shake maps and other maps relevant to the situation on all their computing devices including phones, tablets, and laptops. Likewise, tools for visualizing data such as the stream of values from each sensor.

6. Scale to a worldwide network with millions of customers

Earthquake detections can be broadcast by many mechanisms. In Japan, detections are broadcast by the Japan Meteorological Agency [15]. In Southern California, plans are being made to use the Commercial Mobile Alert System [16] to disseminate alerts quickly. In areas without these kinds of sophisticated dissemination mechanisms, alerts for users are still important. Figure 2.1 shows an application that can receive alerts from multiple sources, including SAF, making it possible for CSN to disseminate alerts in regions without such infrastructure in place. The mechanism used to distribute the alerts can also be set up to communicate with equipment, rather than mobile phones, for the purpose of performing automated actions when events are detected.

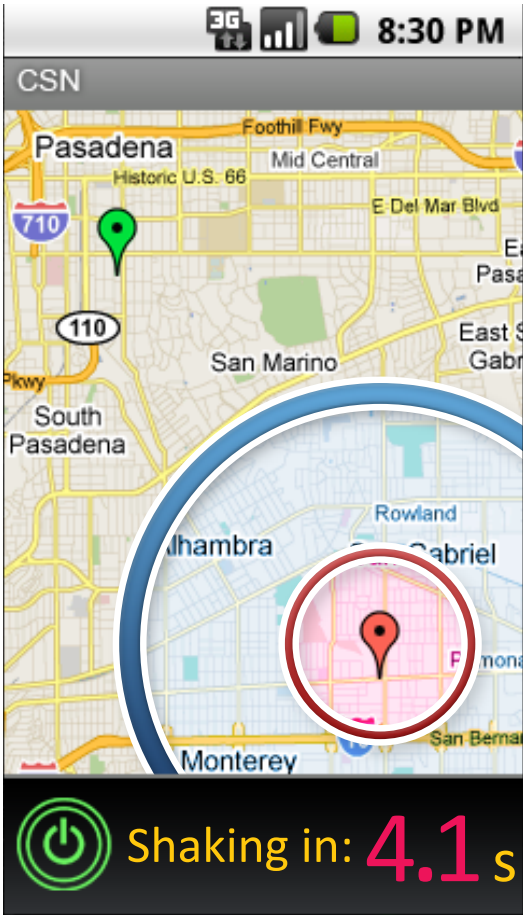



Figure 2.1: An example of how alerts can be delivered to users through their mobile devices. The shown application can deliver alerts generated by SAF or the CISN network. Figure courtesy of Matt Faulkner [2].

Shake maps like those created by the popular "Did you feel it?" website [17] can be created dynamically, and their fidelity is in part based

on the density of the network from which the data was taken. CISN's density is $0.0074/km^2$ over a large area: roughly the size of California, or $423970km^2$. CSN's density is $0.085/km^2$ over a smaller area: about $4380km^2$. CSN adds sensors regularly — see the chart of active sensors in fig. 2.3 — so greater density numbers are coming. Work has already been done on what is possible with an even denser network. A commercial network deployed in Long Beach for surveying purposes had a density of $143/km^2$ over a very small area: $35km^2$. Movies of wave propagation through that network have been produced [3], and the images generated suggest that very small scale shake maps are possible. Figure 2.2 shows an image from the Long Beach array as a seismic wave from the Carson earthquake passes through it.

Section 6.1 discusses earthquake parameter estimation in detail, while section 5.5 discusses tools provided by SAF for both scientists and volunteers to explore generated data. Lastly, section 3.2.3.1 discusses scalability.

### 2.1.1 Network Collaboration

When SAF has access to streams from both dense, inexpensive networks like the CSN and from seismic networks with accurate, low-noise seismometers that have been deployed and studied for many years, different approaches can be used. For example, the geology of Southern California has been studied for decades and CISN has a group of scientists studying sensors and the history of earthquakes in the region. Scientists have developed models that enable them to predict the intensity of shaking based on sensor readings, and they continue to improve and validate those models. SAF can use the history of

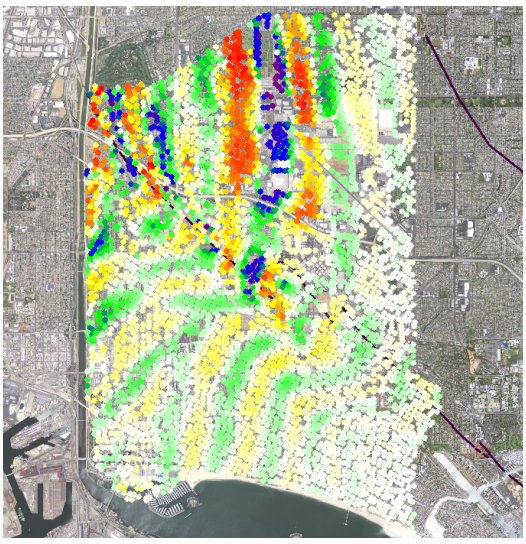


Figure 2.2: A dense accelerometer network in Long Beach provides for accurate visualization of seismic waves. Figure courtesy of Professor Robert Clayton [3].


Figure 2.3: Active CSN clients since late 2010. Downward spikes in the graph indicate data outages rather than client outages.

research on CISN in several ways. The simplest is to provide data about picks from CSN sensors to CISN when CISN detects an earthquake. In this case, CSN provides a great deal of additional data, and this data is particularly valuable around population centers, critical resources, and in CISN "dead zones" — areas that have insufficient CISN sensor coverage. The primary goal in this case is to insert CSN data into the CISN system rapidly.

An alternate approach is to integrate data from all the seismic sensor networks in the area, including CSN and CISN, and also other sensor networks dealing with fire, gas leaks, water supply and sewage. SAF is designed to be extensible and to enable plugging in new types of sensors. SAF's modular structure allows different detection algorithms, such as those used in CISN, to replace existing algorithms easily. The goal in this case is to enable plug-and-play of new sensors and algorithms rapidly.

## 2.2 Performance Metrics

Performance metrics are quantitative criteria by which the efficacy of systems are evaluated. The performance metrics for situation awareness include the following.

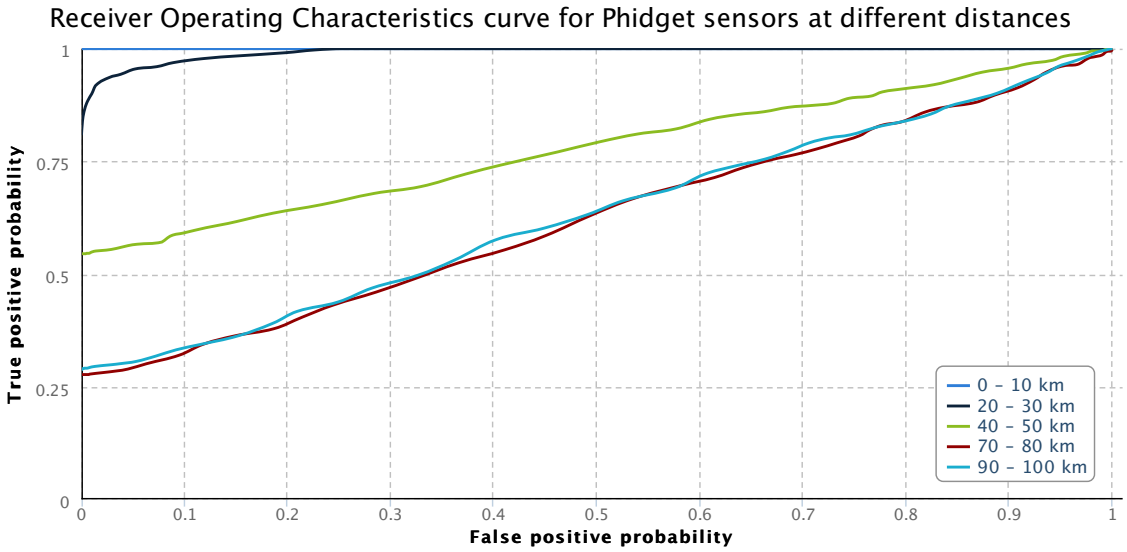



Figure 2.4: This figure demonstrates the explicit trade-off between true positive and false positive rates for earthquakes at different distances from the network [4]. $P_1$ is the true positive rate, while $P_0$ is the false positive rate. The ROC curve was generated using data from CISN downsampled to Phidget rates for earthquakes in the 5.0 to 5.5 magnitude range. Detection probabilities indicate single sensor detection probabilities.

### 2.2.1 Time and Accuracy of Detection

The accuracy of detection is characterized by three parameters:

1. The probability that an alert issued by the application is a false positive (an alert that should not have been issued).

2. The probability of a false negative (an event was not detected by the application).

3. Estimates of the characteristics of the event; in the case of earthquakes these characteristics include the location of the hypocenter, the time of initiation, the magnitude, and the manner in which a fault ruptures.

The accuracy of detection depends on the time that elapses before an alert is sent; accuracy improves with time as more data becomes available, while the speed with which detection occurs gets worse. Ground truth is usually established long after the event has terminated.

The trade-off between false positives and negatives is often represented by a ROC — Receiver Operating Characteristic curve [18], shown in fig. 2.4, with the false positive rate on the x-axis and

the true positive rate on the y-axis. A ROC curve is estimated empirically for a historical time window: the false positive rate is estimated as the fraction of alerts that were false over the time window, and the true positive rate is estimated as the fraction of events for which alerts were issued. The ideal point on the ROC curve is $x = 0$ and $y = 1$, and as time progress the ROC curves typically move towards this point, as shown in the figure.

The quality of the ROC curve typically improves with the intensity of the events for which alerts are issued. Consider two applications, one that issues alerts only for quakes that cause peak accelerations in excess of $0.2g$ and the other that issues alerts for accelerations exceeding $0.02g$. The ROC curve for the first application will be superior to that of the second because more sensors will report picks accurately in the first case.

A more relevant ROC curve, but one that is much harder to obtain empirically, is the ROC curve for a particular location on the ground (or even harder) in a building. In this case, an event is the presence of shaking above a specified magnitude *at that point* as opposed to shaking somewhere in the region. The time delay is the time between the instant at which a device at that location receives an alert and the instant at which intensive shaking at that location occurs.

### 2.2.2 Time and Accuracy of Shake Maps

The delay between the instant at which acceleration is measured by a sensor and the instant at which this value is entered into a data fusion engine is a critical parameter. This delay is a fundamental metric, because this delay impacts all other delays including delays in issuing alerts.

Shake maps that show the amount of acceleration at a point at an instant in time are valuable to geophysicists who study seismic waves. First responders are more concerned about the damage at different locations, and they are more interested in the maximum acceleration that has taken place, or will take place, during an earthquake. The amount of acceleration is a proxy for the degree of damage, and so first responders want maximum figures.

### 2.2.3 Scalability, Reliability, Extensibility

Metrics for scalability can be measured by the number of sensors and number of sensor types that can be incorporated into the system. A measure of reliability is the fraction of time that the system operates with adequate accuracy; this depends on several factors including the number of points of failure of the data fusion engines and the degree of redundancy in sensors. A measure of extensibility is the ease with which new sensors and new sensor types can be integrated into the system, and the effort required to change modules for detecting events, estimating event parameters and issuing alerts and generating shake maps.

### 2.2.4 Effort to Visualize Maps and Analyze Data

A key metric is the effort required by scientists and non-scientists to visualize and experiment with different types of maps and analyze data. Volunteers want to see how their participation is used, and the best ways are for them to have shake maps and other maps pushed to them. Scientists want to visualize and analyze data; for example, how does the estimate of the hypocenter location change with time as more data is gathered? A metric is the ease with which the application can be used to analyze historical data and simulations.

### 2.2.5 Cost

Cost is the metric that constrains the implementation of a situation awareness framework. The cost of a volunteer installing an CSN accelerometer, or connecting an accelerometer in an existing consumer device (such as a phone), is tiny. The costs of developing, implementing and studying a special purpose network of accurate seismometers are much higher. If costs are not a constraint, the best solution is to deploy only accurate seismometers and use information from volunteers as it becomes available. This thesis shows how a situational awareness framework can be used to develop applications to meet a wide variation of cost constraints.

The next chapter deals with the components that comprise a situation awareness system.

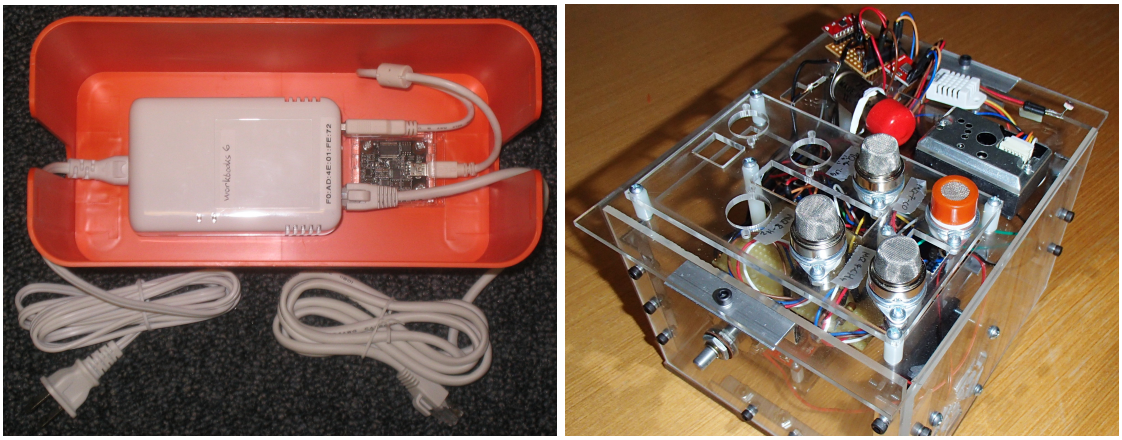
# Chapter 3

# System Components

## 3.1 Sensors

Inexpensive sensors for a wide range of applications have become possible in recent years. Phidgets, the accelerometers used in CSN, while relatively inexpensive, have a noise profile that makes them useful for seismology. Our current deployment of Phidgets in stand-alone configurations looks like the photograph in fig. 3.1a.

Other applications include gas and radiation sensing. Sandra Fang built a prototype home hazard station that can integrate with applications like SAF. The station, depicted in fig. 3.1b, in addition to having an accelerometer, is capable of detecting: Carbon Monoxide, Carbon Dioxide, Methane,



(a) Stand-alone CSN device for deployment where desktop computers are unavailable.

(b) Home Hazard Station prototype with a variety of sensors for detecting changes in the environment.

Figure 3.1: Pictures of sensor boxes for use with SAF.

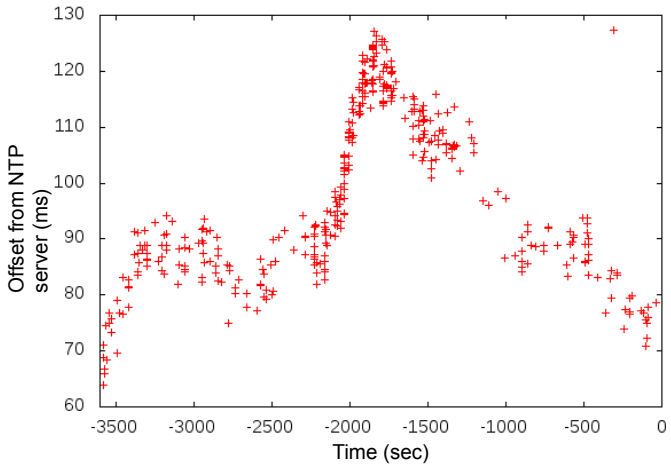



Figure 3.2: This figure shows the time drift for a particular client with respect to the NTP server. Figure courtesy of Leif Strand.

Ozone, temperature, humidity, and radiation.

Client computers for hosting sensors include single board computers, like the one pictured in fig. 3.1a or a Raspberry Pi, traditional laptops and desktops, to which Phidgets can attach directly via USB, and phones and tablets. [1] includes the use of an Android tablet as a host for a Phidget which communicates with SAF, while [19] uses the accelerometers within Android devices themselves as sensors in the CSN network.

The CISN sensors, whose readings are integrated with SAF, use expensive Episensor devices like the one pictured in fig. 1.2. These devices also include GPS clocks, ensuring that the clocks of each sensor are within tight tolerances of each other. Other sensor devices must rely on external clock synchronization mechanisms, like NTP. Even where tight tolerances are not required, our experience has shown that client clocks drift easily and are frequently off by more than an hour. Figure 3.2 shows the drift of an individual client clock over a short period of time. When using a centralized server, or when synchronizing server time, it is sometimes the case that server time can be used. However, as not all readings are transmitted to the server when recorded, this creates problems in establishing the true time of readings taken at a sensor.

## 3.2 Cloud Computing Services

Information Technology has been moving to virtualized resources as a means of reducing cost, decreasing exposure to hardware faults, and increasing flexibility in terms of deployment strategies [20]. Virtualization exists in several forms, from virtual private servers to clusters of virtual machines.

### 3.2.1 IaaS, PaaS, and SaaS

In terms of cloud computing, solutions come in three primary categories [21].

#### 3.2.1.1 Infrastructure-as-a-Service

Infrastructure-as-a-Service (IaaS) is the most basic cloud offering available. Examples include Amazon EC2 [22] and Rackspace Cloud Servers [23]. This type of offering provides a basic infrastructure within which any kind of system can be deployed. The infrastructure provided normally consists of the physical hardware, the network connections between machines and the Internet, and a framework that provides the ability to start up and shut down virtual instances that the customer configures. The basic offering is in many ways similar to virtual private server offerings, or any type of hosted server where the responsibility for the hardware lies with the vendor. IaaS, however, has the advantage that you can easily scale up or down the number of instances in use and, while doing so, pay for only the machines you have currently allocated.

#### 3.2.1.2 Platform-as-a-Service

Rather than allocating virtual machine resources dynamically, Platform-as-a-Service (PaaS) providers provide a more constrained environment where an instance of the environment itself is what must load dynamically. This could be a Java Virtual Machine, along with accompanying tools, or a Python interpreter. Because of the simplified allocation process, application deployment and scaling, possibly within specified bounds, can occur far more quickly than with IaaS. In addition, while IaaS services often utilize an à la carte strategy for what web services a developer may choose to integrate, PaaS often provides a built-in suite of services that integrate with the platform. Use of

those integrated services is not compulsory; choosing not to use the integrated services may help to mitigate the potential for lock-in at the expense of performance or ease-of-use benefits frequently associated with integrated services.

PaaS environments include Google's App Engine (GAE) [9], Heroku [24], and Amazon's Elastic Beanstalk [25]. PaaS environments are more heterogeneous than IaaS environments, because the notion of a platform exists along a finer gradient. Amazon's Elastic Beanstalk, for instance, is really a layer on top of its existing EC2 platform. The difference is that Elastic Beanstalk allocates and starts virtual instances automatically along with the necessary infrastructure for the developer to run a Java Web Application Archive (WAR).

Heroku, on the other hand, provides an environment for running web applications in a variety of languages: Ruby, Node.js, Clojure, Java, Python, and Scala. Unlike Elastic Beanstalk and GAE, however, Heroku does not automatically allocate new instances to match varying load. Instead, it relies on the programmer to alter the number of available web and worker 'dynos'. While this number can be allocated dynamically, it is different than both Elastic Beanstalk and GAE in that the scaling function must be handled by the developer.

App Engine is in many ways a combination of the two approaches. GAE both dynamically allocates processing environments in Java, Python, or Go, but also manages the number of instances available to process incoming traffic. It also allows developers to optionally fine tune how the scaling algorithm will handle their traffic, allowing the minimum and maximum number of idle instances to be set in addition to the maximum time a request will wait before spawning a new instance to serve it.

Because of the notion of what constitutes a PaaS environment differs, it is not possible to say with certainty how fast a PaaS environment will scale. However, since, at their simplest, PaaS environments are based on IaaS environments, we can conclude that they scale at least as fast as IaaS environments given the same conditions for scaling. With specific PaaS environments, such as GAE, their ability to scale in seconds makes it possible to handle more swiftly varying load structures than IaaS environments without over provisioning.

#### 3.2.1.3 Software-as-a-Service

Software-as-a-Service (SaaS) is both the most sophisticated cloud-based offering and also the most restrictive. It is the most actively used model for cloud-based computing as the typical use case for SaaS is consumer facing products. Examples of consumer facing SaaS products include Gmail [26], Photoshop Online [27], and Zoho Office Suite [28]. It is also one of the most common distribution methods of enterprise software, such as accounting, customer relationship management, and enterprise resource planning.

This model is gaining popularity for developer facing products as well, such as for storage, messaging, and database platforms. Much of Amazon Web Services [29] can be construed as SaaS for developers. That is, the products are multi-tenant, all users see the same version of the application, the application has very specific interface points, and the internal workings are opaque. The main difference is that the product, rather than providing a web-based interface, offers only programmatic access. That access, however, allows critical pieces of functionality to be outsourced to a cloud provider.

In this way, users of IaaS or PaaS can have specific pieces of their infrastructure managed by another provider.

### 3.2.2 Public and Private Clouds

The above referenced services all utilize what is referred to as a Public cloud — that is, the services are managed by the provider and delivered for use to the consumer over the public Internet. Private clouds, by contrast, are operated for the sake of an individual organization, whether they are managed by a third party or not. Computing solutions exist to run architectures similar to EC2 [30] and GAE [31] within a privately owned data center. This model simultaneously increases flexibility and privacy in exchange for the work needed to manage the data center and a more fixed cost of operation, as opposed to the usage based billing typical of public cloud offerings.

### 3.2.3 PaaS for Sense and Response

PaaS providers have characteristics that make them useful for sense and response systems. Many sense and response systems have variable load; cloud systems adapt gracefully to variable load, acquiring resources when load increases and shedding resources when they are no longer necessary. This feature introduces added complexity; in the absence of over provisioning, it requires resource management in the form of an algorithm which dictates when resources should be added to or removed from the system, how added resources are integrated with existing resources, and how to avoid addressing resources which have been removed from the resource pool. Some PaaS providers enable carefully designed programs to execute the same code efficiently when serving thousands of queries per second from many agents or when serving only occasional queries from a single agent (see section 6.2.4). While some projects [32, 33] exist to help deal with scalability, IaaS providers normally leave it up to the client to determine how resources are added to and removed from the resource pool as well as relegating dealing with the complexities associated with these varying levels of resources to the developer.

Sense and response applications are often required to execute continuously for years. PaaS providers remove the need for routine maintenance from the system, making supporting systems over years simpler. These providers also offer the ability to easily deploy new versions of an application so that users see the new version with no perceived downtime, making transitioning between application versions painless.

Many sense and response applications are required to grow over time as larger parts of the physical infrastructure become cyber-physical. For instance, more bridges and buildings will have sensors added to help manage energy and security; so, the IT part must grow with need. The cost structures of PaaS platforms and IaaS providers help sense and response systems grow cost effectively because an application pays only for what it consumes.

Cloud computing has disadvantages as well as advantages for sense and response applications. One concern is vendor lock-in. In our research we explore the use of widely used standards for PaaS providers running sense and response applications; these standards allow an application to be ported

to other providers. Of course, porting an application, even when standards are identical, is timely and expensive.

A major concern for outsourcing IT aspects of sense and response applications, since they are mission-critical, is reliance on third parties. Durkee [34] discusses the frequent lack of meaningful service level agreements available from current cloud computing providers. This concern must be balanced against the benefits of cloud computing systems.

#### 3.2.3.1 Automatic Scaling

Because scaling up instances with IaaS requires allocating and booting up virtual machine images, it is best suited to slowly varying demand, otherwise requiring the over provisioning common with owned infrastructure. According to Amazon's FAQ, 'It typically takes less than 10 minutes from the issue of the `RunInstances` call to the point where all requested instances begin their boot sequences.' [35]. Demand structures that require resource changes in periods shorter than 10 minutes will be unable to rely on just in time provisioning without accurate demand forecasts at least ten minutes into the future.


Figure 3.3: An existing warning application for use with CISN's ShakeAlert system.

PaaS architectures do not necessarily have this limitation. While Amazon's Elastic Beanstalk, built on EC2, inherits the limitation of the architecture on which it is implemented, Heroku and Google App Engine, which require more narrowly crafted applications, make it possible to provide additional computational resources more quickly. While we do not provide scalability numbers for Heroku, we explore the speed with which additional computation resources can be

allocated by App Engine in section 6.2.4.

#### 3.2.3.2 Load Balancing

PaaS providers handle load balancing on behalf of the user, and some IaaS providers also provide services that can manage load balancing for the user. These load balancing services are sufficient for most users' needs, as they often come with a small number of configurable parameters to control the way that requests are allocated. For the purposes of situation awareness, we consider the load balancing provided by PaaS providers perfectly adequate, and utilizing it removes additional complexity from the system.

Some users may wish to implement more sophisticated load balancing, such as changing the incoming request queueing strategy. For those users, pre-build load balancers may be inadequate, and this could be a reason to avoid PaaS environments where replacing the load balancer, or altering its behavior to that extent, is typically not possible.

#### 3.2.3.3 Security

Some situation awareness applications have very high requirements for security. These applications benefit from the immense amounts of money, time, and energy spent by the largest cloud providers on security infrastructure and monitoring. PaaS applications benefit more than IaaS applications, because more potential security problems — such as unpatched software in particular — are now the responsibility of the provider.

### 3.2.4 Using the Cloud

A decision of whether to host a sensor network application in the cloud depends primarily on two factors: scalability and cost. The measurements made as part of this thesis do not provide a definitive answer, but the data is consistent with the view that a cloud-based system can scale provided that the system is provisioned properly. Cloud systems, such as the Google App Engine, provide mechanisms for provisioning an application to handle surges in load; such provisioning requires additional funds.

We have not made a detailed comparison of costs for hosting the application on the App Engine versus hosting an application on distributed sites managed by an organization; we note, however, that the costs paid for hosting the service on the App Engine have been very low, and the entire SAF application was developed in less than six months by primarily one graduate student focusing on it only part time.

Topologies for sensor networks traditionally fuse data as it flows through the network. This thesis describes a different architecture in which data from sensors flows into a cloud service. A goal of this architecture is to get as much data out of a hazard zone and into a reliable cloud service before the communication network and possibly power fails in the hazard region. Results of analysis in the cloud are sent back to the hazard region as communication lines open up; meanwhile these results are sent to federal and state agencies that can receive communication. This architecture is predicated on the assumption that enough useful data can be sent to cloud services away from hazard zones in the seconds before the hazard destroys communication and power; the measurements carried out in this thesis suggests that this assumption is reasonable.

## 3.3 Phones and Actuators

Figure 3.3 shows an example of a phone being used as a mechanism for alerting users in the event of a crisis. Figure 1.1 showed a persistent awareness application running on a tablet. These applications help demonstrate the potential that exists to utilize existing consumer electronics to increase awareness of the world around us both by providing additional sensors and by providing additional endpoints for delivering important information.

Beyond phones and tablets, hardware such as elevators, trains, and fire station doors can also be controlled via actuators that respond to alerts distributed by a trusted network.

# Chapter 4

# Design

## 4.1 Data Structure for Sensors

To describe the architecture we use for attaching sensors to the network, we use three terms:

**Host** The physical hardware on which the client software runs and to which the sensor hardware is connected. This could be a single board computer, laptop, mobile phone, or other hardware capable of running client software on its own.

**Client** The software running on the host that processes sensor readings, identifies local events, and submits those events to the network.

**Sensor** The physical hardware, possibly integrated with the host, that makes detection of environmental conditions possible.

The network makes certain assumptions about this configuration, depicted in fig. 4.1. It assumes that a client operates on at most one host. It does not, however, make any assumptions that a

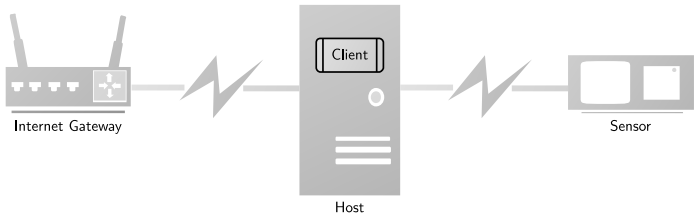



Figure 4.1: Diagram showing the relationship between host, client, and sensor. All hosts also need access to the Internet.

host operate only one client. In some cases, it makes sense for many clients to be operated from an individual host.

Any number of sensors can be controlled by one client. Every client is assumed to have only one location; sensors attached wirelessly or over long distances and operating in capacities where precise location estimates are necessary violate this assumption. This assumption was made to simplify location algorithms and could be trivially corrected or worked around by utilizing multiple clients on the same host.

### 4.1.1 Time Synchronization

In the case of CSN, and any network that requires precisely aligned events in time, time synchronization is very important. Our network operates on the assumption that clients regularly synchronize their clocks with a time server. Note that this does not require a host clock to be correct; the normal CSN software merely calculates and keeps up to date a difference between real time and host time.

This is important for several reasons. First, server times cannot be trusted to be in sync. App Engine provides no guarantees about the closeness of instance clocks, and dialogue between instances for ascertaining a correct time is complex.

Additionally, even if perfect server clocks were possible, clients are transmitting events that happen locally; for those times to be of optimal use to the network, the desired timestamp is an accurate time taken at the client at the time that the event occurred.

### 4.1.2 Client Messages

To circumvent problems with firewalls and NAT traversal, the network makes the assumption that it cannot contact the client software directly. In many community hosted scenarios, this is a valid assumption. Making this assumption also makes security guarantees with respect to the client software simpler; there are no incoming avenues of attack added by including SAF client software in a network short of replacing the client code itself.

The possible client message are listed below; they are defined by SAF and stored with the SAF

documentation [36].

**Client Creation** This message is only sent when a client first registers with the network. The client is provided with a secret on registration that is used to hash the message id of future sent messages to validate the origin of the messages sent.

**Client Update** This message is sent when a client wants to update the metadata the server has stored about it.

**Heartbeat** This message is sent periodically — currently every ten minutes — to inform the server that the client is still alive. In the absence of event messages, this is the only way the server can know whether or not a previously registered client is still attached to the network.

Clients that do not transmit a heartbeat for a configured expiry period are presumed inactive. Clients can also send a Heartbeat message to indicate that they are going offline, keeping the system information more current that if it were to wait for the last heartbeat time to pass the expiry window.

As heartbeats are the only means by which the server can guarantee contact with the client, the Heartbeat response is also the means by which requests and notifications are transmitted to the client. At present, this includes requests for data from specific periods of time, usually corresponding to an event, and notification of metadata changes.

**Event** This message is sent to inform the server of local events at the client. These are the messages collected to perform aggregated event detection, either over multiple sensors, clients, or both.

**Data Offer** These messages are sent in response to a data request or in the presence of a request for continual data. They contain the complete logs of the data from a given sensor.

**Fetch Metadata** A client can request the metadata that the server has stored about it. This make it possible for a client to rely exclusively on the server for metadata storage, and is also necessary when the client metadata has been updated remotely, such as using the client editor tool.

**Fetch Token** This message allows a client that does not necessarily have a GUI to create a URL from which the client's metadata can be altered. This is done by using the client secret to confirm the message origin and transmitting a URL with an embedded token which, when accessed, provides authenticated access to the client originating the token request.

## 4.2 Geocells

Seismic networks deal with geospatial data, as do other situation awareness applications such as air pollution monitoring, toxic plume detection, and radiation leakage alerting. For that reason, we require an efficient mechanism for querying for sensor information from specific geographic areas.

Since queries on App Engine are limited to using inequality filters on only one property (see section 5.2.6), a different method is needed for any form of geospatial queries. Our solution involves the use of 8-byte integer objects to encode latitude and longitude pairs into a single number. This single number conveys a bounding box rather than a single point, but, at higher resolutions, the bounding box is small enough that it can be used to convey a single point with a high degree of accuracy. We define the resolution of a numeric geocell to be the number of bits used to encode the bounding box. A resolution 14 geocell uses 14 bits, 7 for latitude and 7 for longitude, to encode the resolution. A resolution 25 geocell uses 25 bits, 12 for latitude and 13 for longitude.

It's important to note that the ratio of the height to the width of a bounded area depends on the number of bits used to encode latitude and longitude. For even-numbered resolutions, an equivalent number of latitude and longitude bits are used. For odd numbered resolutions, one additional longitude bit is used. This permits bounding boxes with different aspect ratios. An odd numbered resolution at the equator creates a perfect square, while an even-numbered resolution creates a rectangle with a 2:1 ratio of height to width.

### 4.2.1 Creating geocells

Geocells are created using a latitude, longitude pair. This is done by dividing the world into a grid and starting with the (90°S, 180°W), (90°N, 180°E) bounding box, which describes the entire world.

Each additional bit halves the longitude or latitude coordinate space. Odd numbered bits, counting from left to right in a bit string, convey information about the longitude, while even-numbered bits convey information about the latitude. After selecting an aspect ratio by choosing even or odd numbered resolutions, geocells are made larger or smaller in increments of 2. This means that each larger or smaller geocell selected will have the same aspect ratio as the previous geocell.

For this reason, each bit pair can be thought of as describing whether the initializing point lies in the northwest, northeast, southwest, or southeast quadrant of the current bounding box. Subsequent iterations use that quadrant as the new bounding box. To work a simple example, consider (34.14°N, 118.12°W). We first determine whether the desired point lies east or west of the mean longitude and then determine whether it lies north or south of the mean latitude. If the point lies east of the mean longitude, the longitude bit is set to 1, and if the point lies north of the mean latitude, the latitude bit is set to 1. In our example, the point lies in the northwest quadrant, yielding a bit pair of 01 for a resolution of 2. Iterating through the algorithm yields the following bits for a resolution of 28:

0100110110100000010000000101

For an illustration of how increasing resolution divides the coordinate space, see fig. 4.2. Because these representations are stored as fixed-size numbers, the resolution of the geocell must also be encoded. Otherwise, trailing zeros that are a result of a less than maximum resolution would be indistinguishable from trailing zeros that represent successive choices of southwest coordinates. Therefore, the last 6 bits of the long are used to encode the resolution. The other 58 bits are available for storing resolution information.

For space considerations, we also have an integer representation capable of storing resolutions from 1 to 27 using 4 bytes, as well as a URL-safe base64 string based implementation that uses a variable number of bytes to store resolutions from 1 to 58. The integer implementation uses 5 bits to store the resolution, and the remaining 27 bits are available for resolution information. The string based implementation always encodes the resolution in the final character, occupying 6 bits of information in 1 byte, while the remaining characters encode the resolution information.

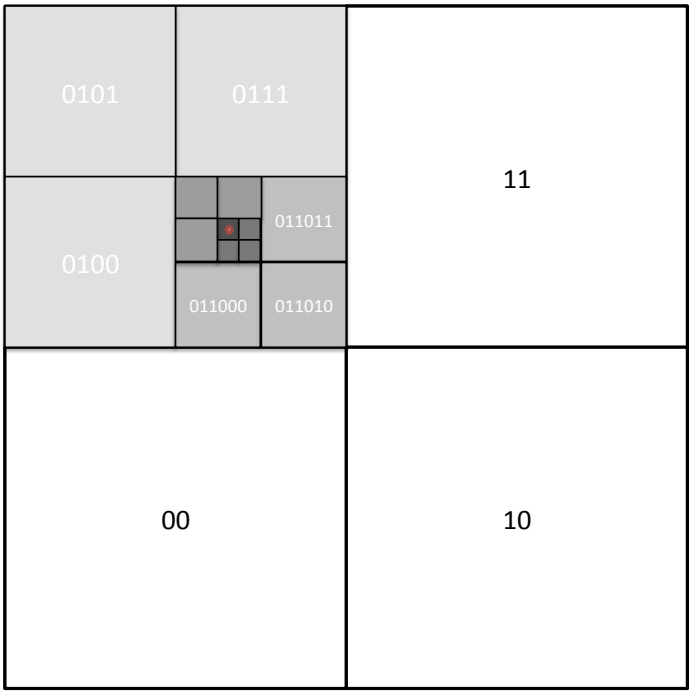



Figure 4.2: How bounding boxes divide the coordinate space.

Our analysis of geocell sizes at various resolutions led us to the conclusion that the most useful geocell sizes for event detection were resolutions 12 through 28. Resolution 29 ranges from 1.5 kilometers square to 0.65 kilometers square depending on the point on earth (see Limitations) and is too small to be useful for aggregation in all but the densest networks. Resolution 12 is quite large, encompassing anywhere from 84,000 square kilometers to 195,000 square kilometers. This resolution is still useful for aggregation of extremely rare events that may be spread out over a large region.

The maximum resolution that SAF will provide regarding client locations for general clients is 35.

Algorithms for working with Geocells have been created in Java, Python, and JavaScript. The process of creating a Geocell in Python is excerpted in appendix A.3. The full library is available online; see appendix A.1 for details.

### 4.2.2 Comparison

Two similar open methods of hashing latitude and longitude pairs into simple strings have been previously proposed: GeoModel[37] and Geohash[38]. Our algorithm is capable of translating to and from representations in both systems. Numerous other systems exist; however, many are variations on a similar theme, and the earlier systems not designed for computer derivation each suffer from different shortcomings. The UTM[39] and MGRS[40] systems not only have a complicated derivation algorithm[41], but also suffer from exceptions to grid uniformity. The GARS[42] and GEOREF[43] system utilize an extremely small number of resolutions: 3 and 5, respectively. The NAC System[44] is proprietary and has different aims, such as being able to encode the altitude of a location.

GeoModel, Geohash, and our own system all bear similarity to the well known quad tree algorithm for storing data. All of these algorithms rely on dividing the plane into sections: quad tree algorithms divide the plane into quadrants, our own algorithm divides the plane into 2 sections per resolution while GeoModel and Geohash divide the plane into 16 and 32 sections respectively. While the algorithm for finding a storage point in a quad tree is the same, what the other algorithms actually compute is equivalent to the path to that storage point in a quad tree with a storage depth equal to the resolution. The focus of the quad tree method is on the in-memory storage of spatial data points, while the focus of the other algorithms is computing an effective hash for data points. The path serves at that hash.

In addition, navigating between points at the same depth in a quad tree is difficult, as it requires traversal of parent nodes. In the geocell model, neighboring nodes can be retrieved by simple bit manipulation.

Our numeric geocells have one key advantage over the Geohash and GeoModel algorithms: the numeric representation allows for the description of a broader range of resolutions. GeoModel and Geohash encode 4 and 5 bits of information per character, respectively, using the length of the character string to encode the resolution. Numeric geocells therefore have 4 to 5 times more expressive power in possible resolutions.

Resolution density has a strong impact on the number of cells required to cover a given region

or the amount of extra area selected by the cells but not needed. When selecting cells to cover a region, it is possible that several smaller geocells could be compressed into one larger geocell. This can happen more often when more resolutions are available. For instance, 16 GeoModel cells and 32 Geohash cells compress into the next larger cell size, where only 4 numeric geocells compress into the next larger numeric geocell (when maintaining aspect ratio). This comes at the expense of having to store more resolutions in order to perform the compression. Section 4.2.4 contains more information on the selection of geocells to query.

Further, as a consequence of the parity of 5, Geohash strings with an odd length encode one more bit of longitude information than latitude information, while even length Geohash strings use equal number of latitude and longitude bits. This means that as the parity of the length of the string varies, so too does the aspect ratio of the described bounding box. Since Geohash was originally purposed for encoding specific points on earth, it should not be a surprise that it has limitations when encoding bounded areas.

If space is a consideration, numeric geocells represented by integers are helpful for the most useful range of resolutions. Both GeoModel and Geohash use strings to convey bounding boxes. GeoModel uses a base 16 character encoding and thus encodes 4 bits of information per character. Geohash uses a base 32 character encoding and thus encodes 5 bits of information per character. Our integer representation has a fixed size of 4 bytes and is capable of storing resolutions from 1 to 27, and our long representation has a fixed size of 8 bytes for possible resolutions from 1 to 58. While both GeoModel and Geohash have an unlimited potential resolution, resolutions beyond 58 are of limited utility. In Jakarta, which is near the equator, where geocells are the largest, a resolution 58 geocell is 7.42 cm x 3.73 cm.

At a resolution of 16 our integer representation and GeoModel's representation use the same number of bytes, while for resolutions greater than 16 our integer representation is more space efficient. Similarly, our integer representation uses the same number of bytes as a Geohash representation at resolution 20 and is more space efficient for resolutions greater than 20. Our long representation, using 8 bytes, is only of greater space efficiency than GeoModel and Geohash for resolutions

| | Jakarta | Caltech | London | Reykjavik |
|---|---|---|---|---|
| A:Jakarta | 1 | 0.83 | 0.63 | 0.44 |
| H:W Even | 1.99 | 1.66 | 1.24 | 0.87 |
| H:W Odd | 0.99 | 0.83 | 0.62 | 0.44 |

Table 4.1: The aspect ration and area of resolution 16 Geocells at different points on Earth.

greater than 32 and 40 respectively. While the long representation maintains the other advantages of numeric geocells, it loses in space efficiency for all but the highest resolutions. If encoding points rather than bounding boxes, the long representation is the logical choice.

Space filling curves, such as the Hilbert curve, can provide similar advantages by using an algorithm to ascribe addresses to all the vertices in the curve. Whatever advantage these curves might have derives from their visitation pattern, which can yield better aggregation results for queries that rely on ranges. Our query model utilizes set membership testing for determining geographic locality, which means that we cannot derive a benefit from the visitation pattern of space filling curves. We rely on the simpler hash determination method used in quad trees instead.

### 4.2.3 Limitations

Because they rely on the latitude and longitude coordinate space, numeric geocells and similar algorithms all suffer from the problem that the bounded areas possess very different geometric properties depending on their location on Earth. The only matter of vital importance is the latitude value of the coordinates; points closer to the equator will have larger, more rectangular geocells while points farther from the equator will have smaller, more trapezoidal geocells.

Algorithms which rely on the geometry of the geocells, if applied globally, will not operate as expected. Instead, algorithms must be designed without taking specific geometries into account, or must be tailored to use specific resolutions depending on the point on earth. In table 4.1, we compare the size, in terms of area, of four different locations. The area is expressed as a ratio of the size to Jakarta, the site used with the largest geocells. Geocells of any resolution converge to this ratio between sizes beginning with resolution 16. The ratio of the height to the width is also included for both even and odd resolutions.

Finally, prefix matching with any of these algorithms suffers from poor boundary conditions. While geocells which share a common prefix are near each other, geocells which are near each other need not share a common prefix. In the worst case scenario, two adjacent geocells that are divided by either the equator, the Prime Meridian or the 180th Meridian will have no common prefix at all. For this reason, geocells are used exclusively for equality matching.

### 4.2.4 Queries

When querying for information from the Datastore or Memcache, geocells can be used to identify values or entities that lie within a given geographic area. A function of the numeric geocell library allows for the southwest and northeast coordinates of a given area, such as the viewable area of a map, to be given and returns a set of geocells which covers the provided area. Given that no combination of geocells is likely to exactly cover the map area, selecting a geocell set to cover a specified area is a compromise between the number of geocells returned and the amount of extraneous area covered.

With smaller geocells, less area that is not needed will be included in the returned geocells, however, more geocells will be required to cover the same geographical area. Larger geocells will require a smaller number of geocells in the set, but are more likely to include larger swaths of land that lie outside the target region. Balancing these two factors requires a careful choice of cost function which takes into account the cost of an individual query for a specific size, which depends on the network density.

With a low density, smaller numbers of queries across larger parcels of land are optimal as discarding the extraneous results is less costly than running larger numbers of queries. With very high sensor densities, too many extraneous results may be returned to make the extra land area an efficient alternative to a larger number of queries, and so reducing the size of the geocells to help limit the area covered is helpful.

Another feature of numeric geocells is that smaller cells can be easily combined to form larger cells. If an object stores the geocells that it exists within at multiple resolutions, then any of those resolutions can be used for determining whether or not it lies within a target geographical area.

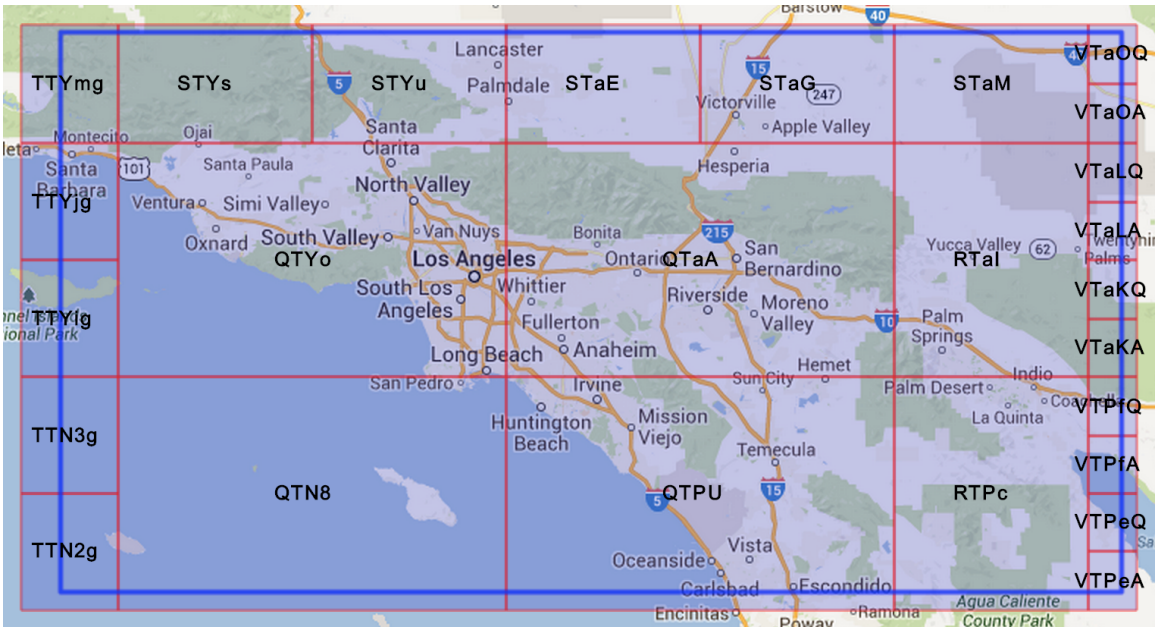



Figure 4.3: A screenshot from the Geocell Map [5] application written to make visualizing Geocells easier. The application shows how a given bounded area can be covered with geocells of multiple resolutions.

The algorithm for determining the set of geocells to query can then combine several smaller geocells into larger geocells, which allows larger geocells to be used in the interior of the map with smaller geocells along the exterior.

For instance, fig. 4.3 shows how smaller geocells can be combined into larger geocells of varying sizes. Importantly for our purposes, the determination of neighboring geocells is a simple and efficient algorithm. By using minor bit manipulations, it is possible to take a known geocell and return the geocell adjacent to it in any of the four cardinal directions. This means that if an event arrives at a known location, not only can the cell that the event belongs to be easily identified but also the neighboring cells. This factors in to our event detection methods, which are described next.

The next chapter deals with our design for utilizing these components in situation awareness applications.

# Chapter 5

# Implementation

This chapter explores the work that went in to building the CSN server, and the tools and patterns that were exposed while architecting SAF.

## 5.1 Community Seismic Network

The Community Seismic Network (CSN) project [7] recruits volunteers in the community to host USB accelerometer devices in their homes or offices or to contribute acceleration measurements from their existing smart phones. The goals of the CSN include measuring seismic events with finer spatial resolution than previously possible and developing a low-cost alternative to traditional seismic networks, which have high capital costs for acquisition, deployment, and ongoing maintenance.

fig. 5.1 shows places where CSN clients have registered worldwide; most clients outside of the United States are registered through the CSN Android application [2], excepting a pilot installation at IIT Gandhinagar.

The CSN is designed to scale to an arbitrary number of community-owned sensors, yet still provide rapid detection of seismic events. It would not be practical to centrally process all the time series acceleration data gathered from the entire network, nor can we expect volunteers to dedicate a large fraction of their total bandwidth to reporting measurements. Instead, we adopt a model where each sensor is constrained to send fewer than a maximum number of simple event messages — "picks" — per day to an App Engine fusion center. These messages are brief; they contain only the sensor's location, the event time, and a few statistics such as the peak observed acceleration.


Figure 5.1: This figure shows places where clients have registered worldwide, indicating worldwide interest in community sensing projects.

The process of pick detection is discussed in section 6.1.1.2, while an overview of the architecture was presented in fig. 1.6. A distributed server running on Google App Engine administers the network by performing registration of new sensors and processing periodic heartbeats from each sensor. The reasoning behind the choice of platform and some of the consequences of that choice are discussed in the following section, section 5.2.

Pick messages from sensors are aggregated in the App Engine server to perform event detection; the algorithm for detection is discussed in section 6.1.2. When an event is detected, alerts are issued to the community through actuators like those mentioned in section 3.3.

### 5.1.1 CSN Pasadena Array

Figure 5.2 shows the density CSN has achieved in the Pasadena area; the network currently has over 300 registered sensors and maintains an active sensor count over 250. Since September 2011, the network has observed many small seismic events. Table 5.1 below shows earthquakes of magnitude 3 or higher within $100km$ of the cluster since September 2011. During this period the network had at least 30 sensors online. In the table the distance is in kilometers and the impact is the manifestation

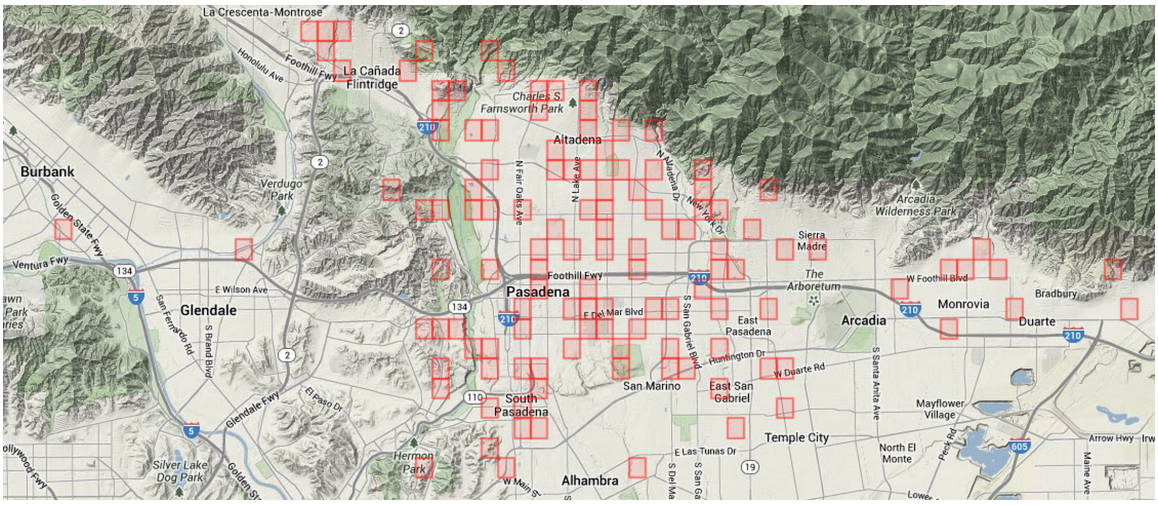



Figure 5.2: This figure shows a zoomed in view of the Pasadena area, demonstrating the density that CSN has achieved.

| Location | Magnitude | Distance | Impact |
|---|---|---|---|
| Newhall | 4.2 | 37.7 | 80 |
| Yucaipa | 4.1 | 100 | 15 |
| Irvine | 3.5 | 65 | 7 |
| Saugas | 3.3 | 46.9 | 7 |
| Ontario | 3.5 | 52.6 | 9 |

Table 5.1: Small earthquakes measured between September, 2011 and August, 2012. These quakes were detected using a simple cell-based aggregation model discussed in section 6.1.2.

of the event measured in terms of relative acceleration according to the Kanamori model [45].

All these events were detected by the system using a simple cell-based aggregation model discussed in section 6.1.2 within seconds of the wave reached the cluster. One false alert was detected on December 15, 2011; a strong clap of thunder was able to sufficiently excite nearby sensors and generate a system alert with probability 0.69.

## 5.2 Google App Engine

The server software running CSN has been running on Google App Engine since the middle of 2009. The choice of platform came about in part because, as a small research project with minimal IT resources, operating a 24x7 situation awareness platform is a daunting task. Because we do not know when an event of interest will occur, it is of paramount importance that the system be available at

all times.

### 5.2.1 Platform advantages

Selecting a PaaS architecture enabled CSN to entrust uptime and performance to Google App Engine's System Reliability Engineers (SREs). Although this reliance has trade-offs, it comes with large upsides. In the words of Peter Magnusson [46], Google App Engine's Engineering Director,

> "[Application developers] don't need to worry about firewalls, most denial of service attacks, viruses, patches, network configurations, failover, load balancing, capacity planning, OS patches and upgrades (in particular security related), hardware upgrades or fixes, certification levels, most security issues, routing, etc."

AppScale, in replicating App Engine's service, uses no fewer than 7 open source systems [31] and 300,000 lines of their own code [47]. Those systems would all normally have to be installed, configured, and kept up to date with security patches or feature upgrades. In addition, every system would have to be monitored for availability and performance.

Relying on a PaaS system largely removes all of these requirements, but at a cost; I discuss some of the liabilities of this architecture in section 6.2.2.

### 5.2.2 Structure of GAE

Google App Engine is divided into three main components: [6]

**Frontend** — the load balancer.

**App Master** — the resource allocator.

**App Server** — the runtime environment host.

Requests to App Engine are split into two types: static requests and dynamic requests. Static requests never reach the App Master or App Server, and I do not discuss them further. They are handled by the Frontend server through a specialized handler.

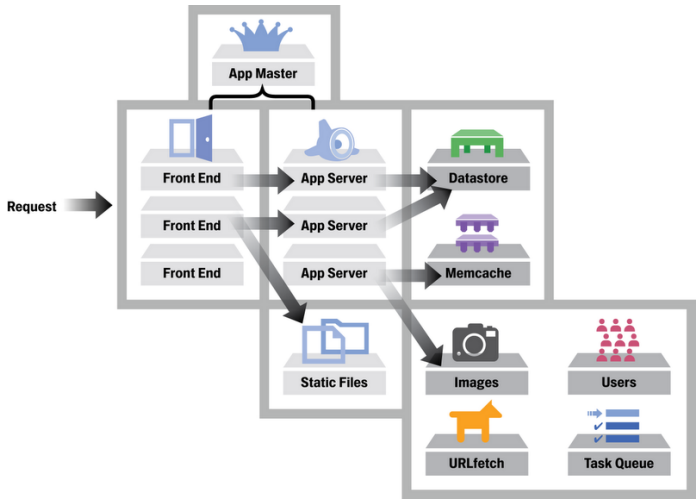



Figure 5.3: A diagram showing the flow of a request within App Engine. Figure from Alon Levi [6].

The App Master is responsible for allocating runtime instances on App Servers; it is the black box that is responsible for how App Engine scales an application. We cannot measure its mechanisms directly — information on instance allocation and destruction is not available — but I attempt to measure the allocation process indirectly by observing the instances to which requests are routed in section 6.2.4.

Dynamic requests are handled by a runtime instance on an App Server. The majority of our analysis in section 6.2.2 focuses on the performance of the runtime instances.

### 5.2.3 GAE Components

GAE consists of a large number of provided services. I reference many of them throughout the thesis and so describe the referenced components here to make the meaning clear.

#### 5.2.3.1 Memcache

Memcache is Google's implementation of the popular open source product Memcached [48], similar in many ways to Amazon's implementation named Amazon ElastiCache [49]. Google's class-based interface to Memcache exists to provide API compatibility with third party Memcached clients.

Memcache is designed to be faster than the datastore and is used as a lossy cache for frequently accessed data.

#### 5.2.3.2 Datastore

"Datastore" is a term used to refer to both the original Master-Slave Datastore, which debuted with App Engine in 2008, and the newer High-Replication Datastore (HRD), which debuted in 2011 [50]. Where the distinction is unimportant, Datastore is used to refer to the two types of data stores interchangeably.

The Datastore is built on top of Google's BigTable [51] technology. It is a key-value store approximated by open source projects including: Apache Cassandra [52], Apache HBase [53], and Hypertable [54]. Other "NoSQL" key-value stores, column oriented databases, and document databases also provide similar functionality.

Google App Engine is currently the only mechanism by which developers outside of Google can utilize BigTable.

#### 5.2.3.3 Task Queue

App Engine's Task Queue is a distributed task queue [55] in which work units, or tasks, can be placed in a queue to run asynchronously from the request that enqueued them, either immediately or in the future. The service is similar to the open source projects Celery [56] and beanstalkd [57] as well as Amazon Simple Queue Service [58].

The Task Queue API provides two kinds of queues: Push Queues and Pull Queues. Push Queues operate by running tasks as they enter the specified queue, while obeying the rates specified in the queue configuration and the task configuration. Pull Queues, on the other hand, provide a repository for work that is later retrieved by another process by request only. Pull Queues were implemented in the Datastore as a form of fan-in referred to as a fork-join in [59], and were formally introduced in May of 2011 [60].

#### 5.2.3.4 Blobstore

App Engine's Blobstore is a simple storage repository for large data objects. When Blobstore first launched, it was the only way to store data objects exceeding 1 MB. All data objects stored in the

Datastore have always been restricted to 1 MB, and that restriction is why the first version of the CSN server could not store sensor readings exceeding that size.

#### 5.2.3.5 Namespaces

App Engine provides an API called "Namespaces" which enables multi-tenancy in a single application. Utilizing the Namespaces API makes it possible to force all data in other App Engine APIs, including Datastore, Memcache, but excluding Task Queues and Blobstore, into a data partition identified by the name of the Namespace.

Namespaces are a useful separation for organizations that require strict isolation of data for security, privacy, or billing reasons. They impose a burden when data across namespaces needs to be aggregated, however, so their use for casual isolation — such as data for a single building — is not recommended. Instead, metadata for clients or sensors can be used to achieve that type of data isolation.

Namespaces are particularly appropriate for data groups that do not anticipate a need for data aggregation. Managing data within a namespace is simpler when extraneous data is not available. If the data in the two namespaces need not interact then, with respect to each other, they are not relevant.

### 5.2.4 Synchronization options

Processes which manage requests are isolated from other concurrently running processes. No normal inter-process communication channels are available, and outbound requests are limited to HTTP calls. To establish whether or not an event is occurring, it is necessary for isolated requests to collate their information. The remaining methods of synchronization available to requests are the use of the volatile Memcache API, the slower but persistent Datastore API, and the Task Queue API.

Memcache's largest limitations for synchronization purposes are that it does not support conventional transactions or synchronized access and it only supports one atomic operation: increment. In

August of 2011 [61], the GAE team added 'compare-and-set', which provides a form of synchronization that allows a client to guarantee that a value has not been changed before setting a new value. Mechanisms for rapid event detection must deal with these constraints of Memcache if guarantees about the accuracy of cached data are necessary.

More complex interactions can be built on top of the atomic increment operation and the compare-and-set operation, but complex interactions are made difficult by the lack of a guarantee that any particular request ever finishes. This characteristic is a direct result of the time frame limitation discussed in section 5.2.6.

The Datastore supports transactions, but originally launched with the limitation that affected or queried entities must exist within the same entity group. The High Replication Datastore was given the ability to support 'Cross-Group' transactions on up to five entity groups in October of 2011 [61]. For performing consistent updates to a single entity, the original limitation was not constraining, but, when operating across multiple affected entities, the limitation could pose problems for consistency. In the High Replication Datastore, the cross-group transaction ability exchanges the original limitation for a performance penalty, so applications must explicitly denote when they intend to cross entity groups with their transactions.

Entity groups are defined by a tree describing ownership. Nodes that have the same root node belong to the same entity group and can be operated on within a regular transaction. If no parent is defined, the entity is a root node. When no parent entities are declared, all entities are root nodes and only a single entity can be operated upon outside of a cross-group transaction. When defining ownership, there must always be a root node with no parent. A node can have any number of children, as can its own children.

This structure imposes limitations; an entity can only have one parent, and entity groups can only have one write operation at a time. Choices must be made as to what entity group a given entity should belong to, such as a multi-author post. Large entity groups may result in poor performance because concurrent updates to multiple entities in the same group are not permitted. Designs of data structures for event detection must trade-off volume of concurrent updates against speedy single

group transactions. High throughput applications are unlikely to make heavy use of entity groups because of the write speed limitations.

Task Queue jobs provide two additional synchronization mechanisms. First, jobs can be enqueued as part of a transaction. For instance, in order to circumvent the transactional limitations across entities, you could execute a transaction which modifies one entity and enqueues a job which modifies a second entity in another transaction. Given that enqueued jobs can be retried indefinitely, this mechanism ensures that multi-step transactions are executed correctly. Therefore, any transaction which can be broken down into a series of steps can be executed as a transactional update against a single entity and the enqueueing of a job to perform the next step in the transaction. We discuss this pattern explicitly in section 5.4.1.

Second, the Task Queue creates tombstones for named jobs. Once a named job has been launched, no job by that same name can be launched for several days; the tombstone that the job leaves behind prevents any identical job from being executed. This means that multiple concurrently running requests could all make a call to create a job, such as a job to generate a complex event or send a notification, and that job would be executed exactly once. That makes named Task Queue jobs an ideal way to deal with the request isolation created by the App Engine framework.

### 5.2.5 Resource Allocation

GAE allocates processing resources in units called 'instances'. Our concern within this work is the speed with which new computational resources can be allocated. I explore this issue by simultaneously exploring the issue of latency.

Many applications are concerned with the overall latency experienced by a user, and sense and response applications in some domains are especially sensitive to processing delays. Domains where processing delays on the order of seconds matter include earthquake and radiation detection. In other domains, such as air quality monitoring and forest fire detection, latency at the second level is immaterial to the overall efficacy of the application.

Latency will be used to describe the total amount of time required to process a request — that is,

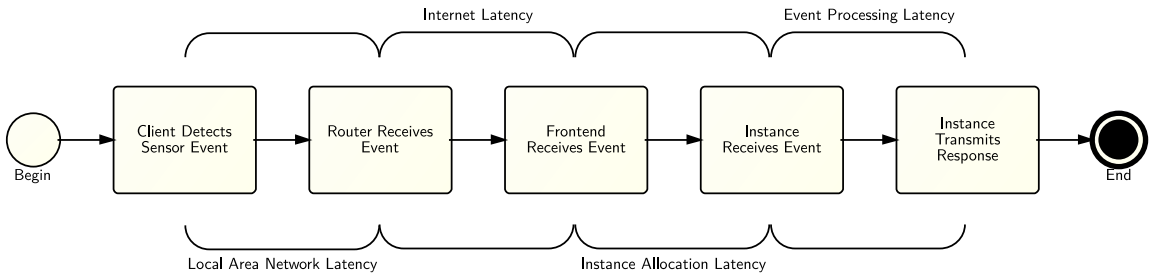



Figure 5.4: A depiction of the different kinds of latency that contribute to delay in event detection. The situation awareness application can only control the event processing latency at the server.

the time between receipt by the Frontend and transmission of the response, as depicted in fig. 5.4 — rather than the latency experienced by a user or sensor, which is subject to network latency beyond that which is due to network components controlled by the provider. App Engine contains internal heuristics, referred to as the scheduler, which determine when a particular application should have more instances allocated to it. Incoming requests may be sent to newly created, or so called 'cold' instances. When a request is sent to a cold instance, that request is referred to as a 'loading request' or 'cold start'.

It is difficult to separate the processing time penalty of a loading request. A loading request requires that the incoming request wait not only for the normal response time of the application but also for the additional latency incurred when starting with a cold instance. This includes loading compiled files into the runtime, reading configuration files if necessary, and other startup operations.

We attempt to measure the processing penalty of a cold instance by subtracting our normal processing time from the processing time experienced by loading requests. Loading requests introduce artificial lag into the system between the time when a stimulus is detected by a sensor and the time when the system is able to respond to it. Applications that are extremely sensitive to latency must therefore take measures to minimize the loading request time or avoid it entirely.

#### 5.2.5.1 Loading request performance by language

App Engine supports three different programming environments: Java, Python, and Go. Because Python is a scripting environment, it is presumed that Python performs better than Java for loading request duration. To test this, I ran an experiment in which the simplest possible Java application

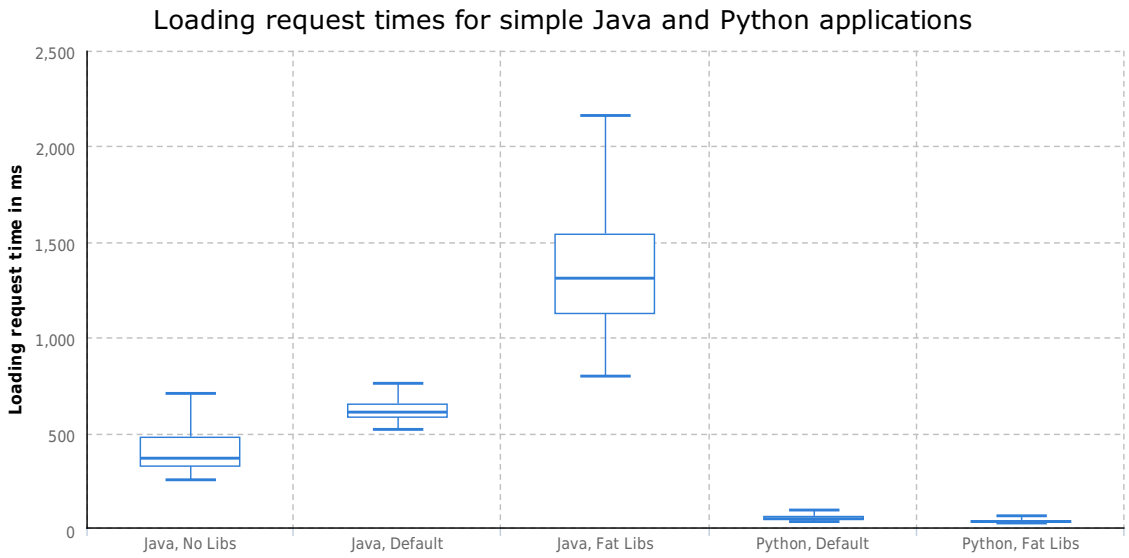



Figure 5.5: Differences in the loading request times for a "hello world" style application in both Java and Python. Min and max values for the whiskers only represent values within a maximum of $1.5 \times IQR$, and outliers are not plotted.

was uploaded; it printed a 'Hello world' style response to every single request. I then constructed a similarly simple Python app using the webapp framework [62]. While Python users are not compelled to use webapp, since Java users must rely on HttpServlet it seemed reasonable — particularly given that most Python users will use webapp or another framework — to use it in our application. As you will see, Python's performance did not suffer unduly from this requirement.

Figure 5.5 shows the differences in loading request times between the Java and Python applications. The applications shown are:

- *No Libs:* in this Java application, all libraries were stripped from the war's library directory and the application was uploaded that way.

- *Default Libs:* in this application, the libraries in the application's library directory were only those placed there by default by the Google plugin for Eclipse.

- *100 MB Libs:* in this application, 100 MB worth of jar files were added to the application's library directory. These libraries were not referenced by the 'hello world' application in any way, and were only added to estimate their impact on application load time.

- *Python:* in this Python application, the only file uploaded was the .py file for the 'hello world' application.
- *Python 100 MB:* in this Python application, 100 MB of additional .py files were added to the application directory. These additional files appear to have been optimized out by App Engine.

From the data we can see that the startup time of Python instances is substantially better than Java instances for very simple applications. The median loading request response time of the simplest Java application was 369 milliseconds, while the median response time of the Python application was 54 milliseconds. For applications that are not particularly sensitive to latency, this initial difference is relatively insubstantial, however, the test with additional libraries added is more worrisome for Java users that rely on substantial frameworks. The median response time for a loading request sent to the Java application with an artificially inflated library folder was 1,341 milliseconds.

It is worth noting here that the additional latency experienced for the 100 MB Libs application is roughly 1 second, which would correspond to about 1 Gbps when factoring in additional latency for finding and accessing the appropriate files. What this means is that most of the increased latency is likely attributable to being forced to download the entire application before the servlet could start. This is notable because it is clear from our tests that the equivalent Python application suffered no such delay. While the application package was still 100MB in size, the webapp was able to start before the entire application was downloaded. This is a substantial performance advantage for Python users.

To test this, I also constructed a similar Java application; one which had 100MB of `.class` files after compilation but had no additional library files beyond the default. This application also required more than 1 second to start up, indicating that Java simply cannot optimize the files necessary to start the JVM in the same way that Python can start with a subset of the application files. For that reason, it is clear that Java users that intend to endure many loading requests should avoid large codebases, while language agnostic users needing to deal with loading requests may be best served by choosing Python. The final and most obvious takeaway from our data is that loading requests should be avoided where possible.

#### 5.2.5.2 Avoiding loading requests

Loading requests occur for one of several reasons:

1. The application had no traffic at all for a period of time, did not permit idle instances, and was unloaded.

2. The application experienced a small spike in traffic and did not permit idle instances or no idle instances were available.

3. The application experienced a larger spike in traffic and it was faster to send requests directly to the new instances than to wait for new instances to warm up.

Applications can permit or require that idle instances are available. This makes it possible for a user to define how many instances will always be available to service incoming requests. Given that demand does not exceed the rate at which these idle instances are provisioned, loading requests can be nearly eliminated. Applications which are more resilient to latency are unlikely to be bothered by the latency of loading requests, particularly since those requests are guaranteed to succeed (see results in section 6.2.2).

Applications that expect to see larger spikes in traffic should be aware, however, that if the observed increase in traffic cannot be accommodated by the number of idle instances, then loading requests will still be generated to handle the spike in traffic. For this reason, driving down loading request times and planning for them is a more robust solution than hoping to avoid them.

In addition, an outstanding issue — Issue 7865 [63] — shows that requests can be sent to cold instances even while idle instances are available. This is an acknowledged defect.

### 5.2.6 Limitations

#### 5.2.6.1 Request time limit

Requests that arrive to the system must operate within a roughly thirty-second time limit. Before requests hit the hard deadline, they receive a catchable `DeadlineExceeded` exception. If they have

not wrapped up before the hard deadline arrives, then an uncatchable `HardDeadlineExceeded` exception is thrown which terminates the process. Prior work [64] indicates that factors outside of the developer's control can create a timeout even for functions which are not expected to exceed the allocated time. Therefore, it is quite possible for a `HardDeadlineExceeded` exception to be thrown anywhere in the code, including in the middle of a critical section. For this reason, developers must plan around the fact that their code could be interrupted at any point in its execution. Care must be taken that algorithms for event detection do not have single points of failure and are persistent in retrying failures at all levels or tolerant to losses of small amounts of information.

Individual operations within the task queue have a more generous deadline of 10 minutes, and tasks can themselves spawn other tasks creating a chain that can be as long as is necessary to accomplish a given job.

#### 5.2.6.2 Query restrictions

The GAE Datastore architecture imposes several limitations on queries. Most developers first notice the lack of an RDBMS-style join operation, in addition to missing operations like count and sum. These operations do not translate well to the BigTable data model; instead, denormalized data is expected and operations like count and sum can use counters for real time availability or MapReduce for generating statistics on a dataset.

Of particular importance to situation awareness applications, at most one property can have an inequality filter applied to it. This means, for instance, that you cannot apply an inequality filter on time as well as an on latitude, longitude, or other common event parameters. I discuss our solution to solving the problem of querying simultaneously by time and location in section 4.2.

#### 5.2.6.3 Downtime

Scheduled maintenance periods for App Engine put the Master Slave Datastore into a read only mode for usually on the order of half an hour to an hour. Sensor networks without sufficient memory to buffer messages for that period of time will lose data during any maintenance period. Operations can

still be performed in memory, however, so sensor networks can still receive and perform calculations on data that do not require persisting results to the datastore. Scheduled maintenance periods occurred 8 times in 2009 and 8 times in 2010 in addition to 9 outage incidents in 2009 and 5 in 2010 [65]. These outages can be mitigated by using App Engine's newer and more expensive high replication datastore which makes continued operation through planned maintenance possible.

#### 5.2.6.4 Errors

The error rate is described as errors on the part of the cloud service provider, that is, errors that would not have been expected when operating owned infrastructure. For App Engine, these include errors in the log marked as a 'serious problem', instances where App Engine indicates it has aborted a request after waiting too long to service it, and `DeadlineExceededExceptions`. I include deadlines as errors because, for a properly configured application with a predictable input set, if the mean processing time for a single request lies substantially below the deadline time, then the substantial increase in processing time required to drive the request to generate a `DeadlineExceededException` is due to factors not under the developer's control.

Deadlines and other errors illustrate that applications which utilize App Engine for collation of sensor data must either be insensitive to the loss of small amounts of data or resilient to transient errors.

## 5.3 Global Data Structures

Association algorithms often rely on measurements taken over a partition of the network; tracking these measurements requires data structures that make selecting the correct partition and ascertaining the sensor properties of the partition members efficient. SAF utilizes a data structure that is sharded [66] by Geocell at a resolution specified at the system level. To collect statistics over a partition of the network, a simple covering of Geocells at the system data resolution is used to query for the necessary cells by name.

The data structure used embeds the current number of active sensors in the region, as well as the

event rates and most recent events submitted by each sensor. This denormalization of event data makes it possible to perform association based only on the data contained at the Geocell level.

Figure 5.8 shows how events and heartbeats enter the system and how that information is propagated to the cell-based data structures. The next section, section 5.4, describes the algorithm by which the data is distributed through the system and justifies the distribution methods.

## 5.4 Cloud-Based Software Patterns

The data structures referenced in section 5.3 require careful management to ensure that high volumes of events or large numbers of clients do not impair the ability of the system to detect events. In this section, we describe software patterns used to manage the data in an efficient way, but which have general applicability when designing distributed data structures under load.

### 5.4.1 Transactions as components

As a mechanism for circumventing App Engine's query restrictions (see section 5.2.4) or merely for transaction continuation out of line with incoming requests, breaking transactional operations into components is an effective strategy on App Engine.

The essential idea is to separate a transaction into stages that can operate independently. Depending on the need for consistency, the stages must be idempotent. The App Engine documentation [67] notes:

> If your application receives an exception when committing a transaction, it does not always mean that the transaction failed. You can receive `Timeout`, `TransactionFailedError`, or `InternalError` exceptions in cases where transactions have been committed and eventually will be applied successfully.

That is, even in light of an apparent failure, a transaction may have been committed. In the face of a need for strict accuracy, this prohibits breaking a normally idempotent transaction into components where the components would not be idempotent.

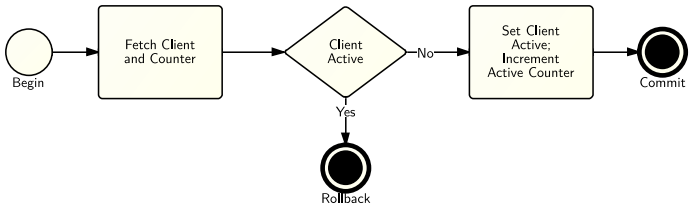



Figure 5.6: How a cross-entity group transaction would modify a counter based on the activity status of a client.

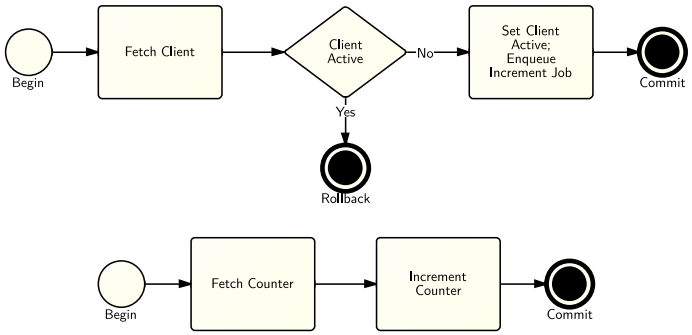



Figure 5.7: How a cross-entity group transaction could be split into two jobs, while failing to guarantee real-time accuracy.

For instance, a transaction might seek to increment a counter when a client's activity status changes. Suppose the initial transaction looks like fig. 5.6. The depicted operation is an idempotent transaction that works across entity groups; it can be run any number of times, and its net result will always be the same. It cannot change the system other than in the way it was intended to.

This mechanism has drawbacks; all jobs seeking to modify either the client or the counter must queue to perform their transactions. If, in this example, entries to and exits from the system occur regularly, then we can expect the affected counters to be under heavy contention. Even in the absence of contention, we can expect the cross-entity group transaction to take longer than an operation on a single entity.

Figure 5.7 shows how the same transaction could be run as two separate jobs. The first job is idempotent; the client can only be set active if it is not presently active. The second job, however, will operate on the system independent of other factors. Since we know the first job was committed, if we can guarantee that the second transaction is only run once, then this separation is fine; exactly one increment will fire in response to exactly one status change.

Given App Engine's documentation, however, we must assume that it is possible for a transaction to be executed multiple times. If strict real-time accuracy is necessary, then this separation into jobs will not work where idempotency of components cannot be guaranteed. If, however, partial accuracy or eventual consistency is fine, then the separation maintains several benefits over the single cross-entity group transaction.

The separated transaction can benefit from fan-in. If instead of pushing the increment job to a push queue, the increment job is pushed to a pull queue, then a worker can consolidate all outstanding increments/decrements for a given counter and apply them simultaneously. This dramatically reduces contention and increases throughput on the affected counters, in addition to consuming less resources overall (for a system with high throughput).

#### 5.4.1.1 Counter snapshot

The question remains as to how accuracy can be restored. The only way to properly restore accuracy in the counter example is to take a snapshot of the system and set the counter to the correct value.

To do this, we introduce the notion of a snapshot id. Enqueued jobs to increment/decrement a counter must include the snapshot id of the client that triggered the operation. All jobs to modify the counter that have a different snapshot id than the current snapshot id of the counter are rolled back rather than committed. Snapshot ids must be ordered in time, however; the easiest way to achieve this it to make the id an incrementing integer. Jobs to modify the counter with an older update id are abandoned after rollback. Jobs to modify the counter that have a newer id than the current id of the counter are deferred until the counter is updated. This indicates that a snapshot is in progress.

We then construct a MapReduce operation either across all counters and affected entities or over a specific counter and all observed entities. The MapReduce job is assigned a new snapshot id that is different from the current snapshot id of all counters being included in the job. The map job accumulates the count of each entity while modifying that entity's snapshot id to the new value. The reduce stage counts the affected entities and stores the new count and snapshot id in each counter.

This guarantees that the counter, together with any pending jobs that have the current snapshot id, contains a complete and accurate snapshot of the system.

### 5.4.2 Separately managed objects

Data can have different tolerances to inaccuracy of missing records, and yet these varying data types may both be necessary for the same operations. One option is to store and manage them separately; this works well, but may introduce overhead if the objects are always retrieved together.

Another option is to store the objects together but manage them separately. This is one of the options used in SAF for management of active client statistics and active client counters. Client statistics can be easily regenerated from client data, but such regeneration takes time. For that reason, backing the statistics up to the Datastore is preferable. Loss of small amounts of data is acceptable, however, and the data is highly transient, making it a natural fit for Memcache.

Client counters are used in tandem with client statistics to perform more accurate detection across a geographic area. Client counters have a high need for accuracy, however, and operations that manipulate them should work exclusively with the Datastore.

For that reason, SAF treats Memcache as the most accurate record for the current client statistics, and the Datastore as the most accurate record for current client counters. The objects are managed together by using two separate types of fan-in simultaneously. The first type of fan-in was discussed in section 5.4.1. The second type involves backing up the current client statistics stored in Memcache to the Datastore. After any update to the statistics, a copy in the Datastore is preferred.

Executing a copy operation for every incoming pick is wasteful, however, and would place the datastore object in which statistics are gathered under high contention. Instead, Task Queue jobs to update the statistics are launched with a name based on the current update id of the statistics. Task Queue jobs created with names leave tombstones behind that prevent subsequent jobs with the same name from launching. This ensures exactly once semantics for jobs with a given name.

This means that, until the last updated id of the Memcache entity is updated, no more jobs to update the Datastore statistics can be created.

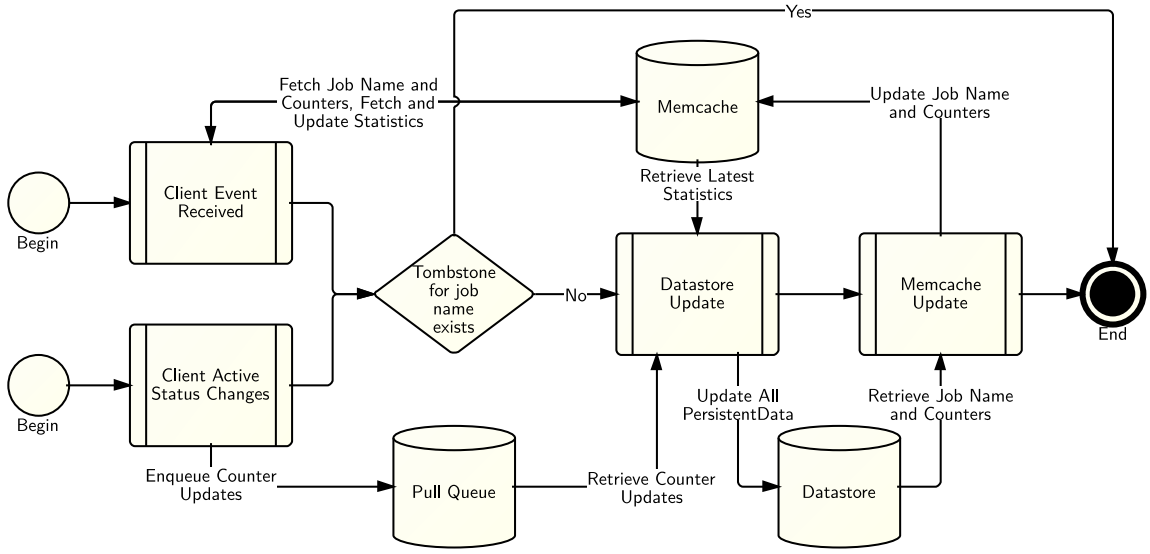



Figure 5.8: This figure shows how updates to the Datastore and Memcache are performed in a cycle. Each box represents an individual process that contributes to the overall flow. While the job name remains unchanged, the update sequence cannot be initiated.

### 5.4.3 Cyclic updates

Finally, we can combine the patterns from the previous two sections in a form of cyclic update that takes an object that is managed partially in the Datastore and partially in Memcache, provides fan-in mechanisms for both types of updates to the object and, in the process, ensures that contention is minimized while throughput is maximized.

We do this by initiating a cyclic update to the Memcache and Datastore entities. When counters are updated, they only update the client and enqueue two more jobs: one, as a part of that transaction, to a pull queue for updating the counter, and a second, after the transaction has been committed, which is a named job to pull the jobs from the queue. The name is derived from the current last update id, meaning that named jobs to update the datastore from client activity changes use names equivalent to named jobs from updates to client statistics.

As mentioned previously, updates to client statistics also enqueue a named job to update the Datatstore. When the update job runs, it leases all appropriately tagged jobs from the pull queue, condensing all counter updates into one operation that also includes the statistic updates from recent client events. This datastore update, as part of the transactional update, enqueues a job to update

Memcache. When that job runs, it modifies the last update id in Memcache, which permits the cycle to begin again.

This sequence, depicted in fig. 5.8, ensures that all client statistic and client counter updates written to the datastore occur in one operation that is renewed at regular intervals. During periods of high volume and absent the presence of rate limiters, it can be easily assumed that one such cycle will always be ongoing. The rate at which the updates occur can be controlled by introducing intentional delays into the cycle, such as before the run of the Datastore or Memcache jobs, which would guarantee a specific rate of Datastore updates.

## 5.5 Tools

In the process of making CSN and SAF possible, various tools have been developed to simplify the management of a situation awareness application. These tools, as part of SAF, are freely available; for details, see appendix A.1.

### 5.5.1 Storing and Processing Data

SAF currently accepts data in a format agnostic way, requiring only that the client application indicate the format when transmitting the message. Unprocessed sensor readings are then collected in the Blobstore. Sensor readings can be directly accessed from the Blobstore, but a tool was developed — `readings_downloader` — to collect all available sensor readings from a specified namespace and application.

SAF also makes extensive use of system logs and data collected from those logs. The `log_downloader` is responsible for periodically fetching logs for a given application and organizing them by date. The `log_parser` makes it possible to extract details from the logs and separate the logs by various attributes, such as namespace, client id, or type of request.

Companion tools to the parser exist that can convert parsed log entries to database entries, transmit or receive the logs over a local network, and apply arbitrary functions to log collections.

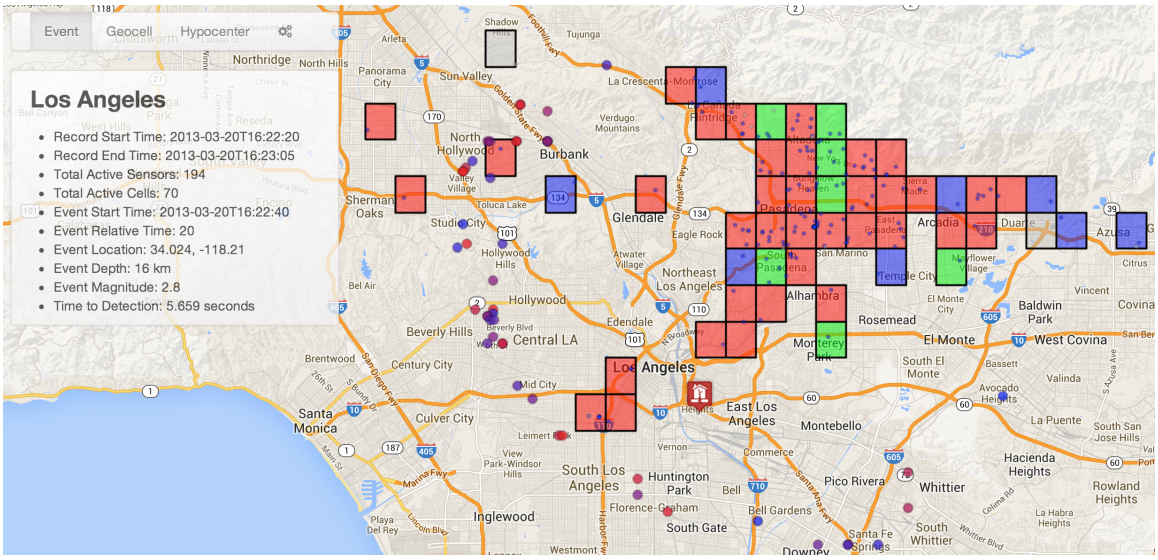



Figure 5.9: A screenshot of the event viewer, which enables inspection of network behavior during earthquakes.

These tools provide for the use of a secondary server for data analysis, as is currently done with CSN.

### 5.5.2 Visualizing Data

A tool for visualizing events was developed; the event viewer is an interactive map that makes it possible to select events that have occurred in the past and view information about how the network behaved when the event occurred. It shows which geocells generated alerts, whether or not the system detected an event, how long from the event origin it took until detection took place, estimates of the event location, and insight into the probability maps that led to the estimates chosen. An example output from the tool is seen in fig. 5.9.

A waveform visualization tool, pictured in fig. 5.10, was developed to make it possible for community volunteers to visualize waveforms without access to specialized software. The tool can be used within a web browser, making it possible to work through a library of available waveforms and visualize each one. The tool also makes it easier to reason about how pick algorithms and sensors are behaving.

The next chapter deals with implementation of theses designs on cloud computing services.

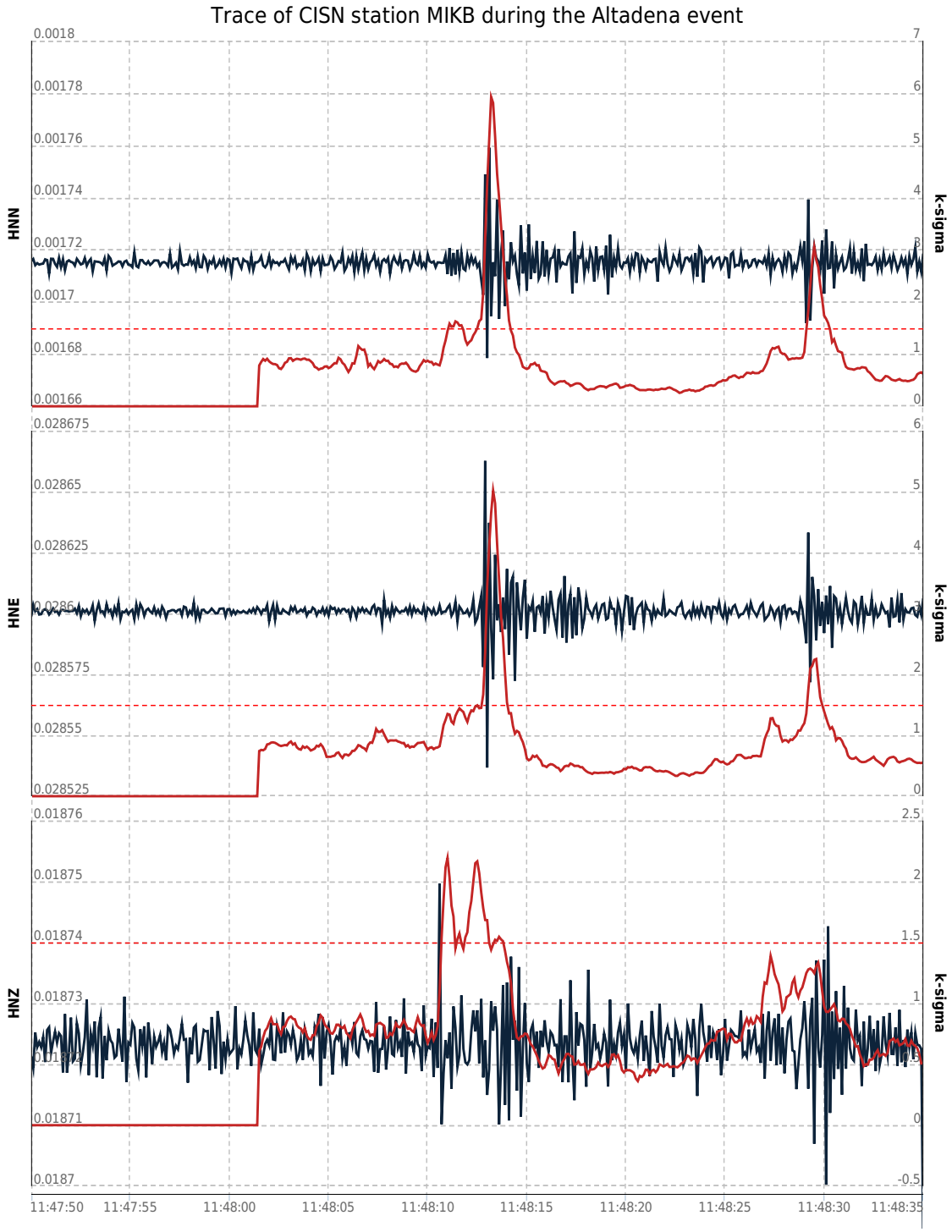



Figure 5.10: An example from the waveform visualization tool developed to make it possible for community volunteers to visualize the waveforms provided from their own house without access to special software.

# Chapter 6

# Experiments

## 6.1 Detection

The primary contribution of this thesis is a framework including a collection of tools that help in implementing situation awareness applications. A module in these applications detects or predicts events. In the case of earthquakes, the challenge is to detect earthquakes early. I show how the framework is used to develop modules for detecting earthquakes. The algorithms for detection are modified as more experiments are conducted; therefore, the key goals of the framework is flexibility in generating modules coupled with visualization and analysis tools for evaluating different algorithms.

I study detection at three levels of locality: how accurately does a set of sensors detect an earthquake when the set consists of:

1. a single sensor,

2. the set of sensors in a small region (a geocell), and

3. all the sensors in the network?

The answers depend on several factors including the intensity of the earthquake and its location, the magnitude of the acceleration at a sensor location, and the time that can elapse between the initiation of an earthquake and its detection.

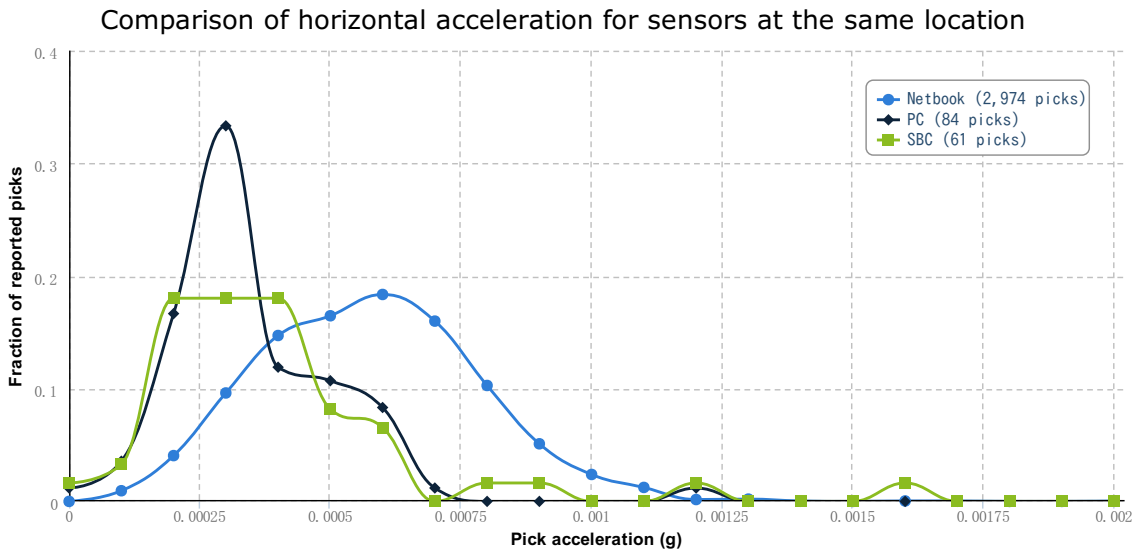



Figure 6.1: This figure demonstrates how sensor response can vary even in a single location. The result is similar to that demonstrated in fig. 1.4. Data courtesy of Julian Bunn.

### 6.1.1 Single Sensor

#### 6.1.1.1 Detecting Anomalies based on Maximum Acceleration

One approach is for a sensor to "pick" when its acceleration exceeds a threshold. We first look at measured accelerations for different sensors. Figure 1.4 shows how sensors even at a single location can have different acceleration response profiles. Analysis of reported accelerations of sensors across the network suggests that much of the deviation between sensors in the network appears to be a result of constant bias. Constant bias is normally a function of sensor orientation or electronic noise, and in many of the sensors is quite large.

To attempt to make comparing acceleration values more reasonable, we employed a filter that cuts off zero-frequency, i.e. the filter subtracts any constant bias, including gravity effects, from the sensor components. This allows us to observe the acceleration as a function of deviation from any bias. Absent a bias, the acceleration values before and after the zero-pass filter should be identical. We then take the absolute value of the filtered acceleration stream and use that as a starting point for comparisons.

Figure 6.2, based on these filtered values, suggests that an approach based on acceleration values

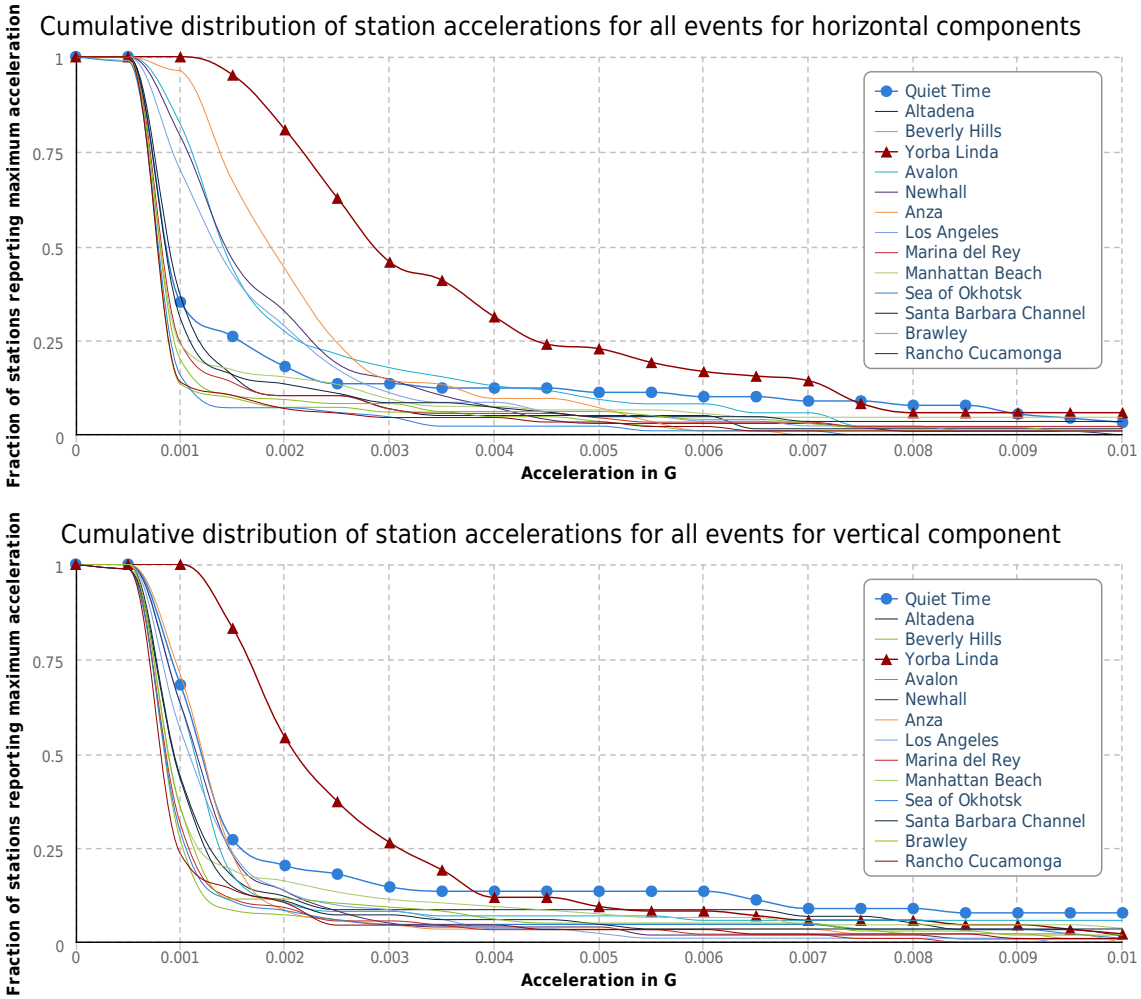



Figure 6.2: The figures show the cumulative distribution of maximum acceleration values for 166 sensors within a 5 km radius of Caltech. The acceleration values are created by using a zero-pass filter on the entire event sample and returning the absolute value of the reported accelerations.

will not work for small earthquakes. The figure represents samples taken from 166 sensors within a 5 kilometer radius of Caltech.

The limited separation between events makes the figures difficult to read, but such a limited separation combined with a very low maximum acceleration value shows that performing detection using any references to absolute acceleration is unlikely to work for events with small shaking. The measured acceleration in the vertical axis for the quiescent period exceed the accelerations reported from every event in the last 18 months except for the Yorba Linda event.

In the horizontal component, things are more promising, and the fraction reporting acceleration

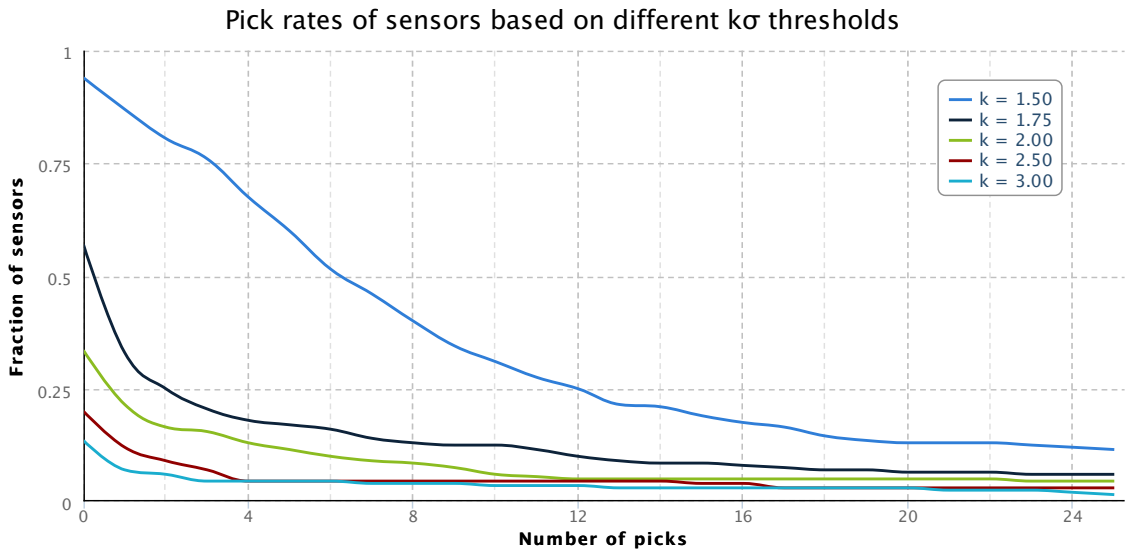



Figure 6.3: This figure shows the fraction of sensors out of 200 that generated at least a given number of picks during a 10 minute long quiescent period. Modifying $k$ alters the pick rate of sensors substantially.

values is greater for five events than for quiet time — but only up to an acceleration value of 0.004g, when quiet time again dominates all events but Yorba Linda. 0.004g is such a small value that no meaningful metric could be based off it.

#### 6.1.1.2 Detecting anomalies based on number of standard deviations

An alternate approach is to detect an anomaly when the acceleration exceeds $k$ standard deviations above the mean, where the standard deviation and the mean are measured over a time window. The value of $k$ can be set so that the average rate of picks across all sensors is a desirable value, say 1 every 2 minutes. We will see later that a moderate pick rate helps the effectiveness of the detection engine. If $k$ is low then the pick rate is high and the high throughput through the cloud service has concomitant costs. If $k$ is high so that the pick rate is low, then the system may not detect earthquakes that are far away or that have low magnitude. After some experimentation, we set $k = 1.5$. We selected such a small value so that each sensor would pick about once every minute to five minutes. Figure 6.3 shows how pick rates change during a sampled quiescent period as the value of $k$ is adjusted. Picks at these high rates are almost always due to noise; fusing data from

multiple sensors reduces noise. Figure 6.4 shows how choosing different threshold values can result in dramatically different noise levels. In section 7.2.5 we discuss future work that might more closely explore fine tuning of detection parameters.

We can operate at different points on the Receiver Operating Characteristics (ROC) curve by changing $k$; if the client generates picks when $k$ is low then the probability of false negatives is low but the probability of false positives is high. We can lower the rate of false positives by increasing $k$, but doing so increases the probability of false negatives.

As figs. 6.5 and 6.18 show, the noise in a sensor varies with the time of day and day of the week. The noise in a CSN sensor due to ambient vibrations can change quickly as when a person walks in to an empty house. The algorithm for calculating the standard deviation is designed to deal with rapid changes in sensor behavior.

Ideally, CSN sensors are mounted horizontally, and a compass is used to orient all sensors in the same way with respect to due North; however, we cannot be sure how volunteers orient their sensors. The vertical axis can be estimated based on the presence of gravity; but the algorithm makes no assumptions about horizontal orientation. The client computes the horizontal acceleration by combining the acceleration along two horizontal axes (North-South and East-West), and thus generates only two channels (horizontal and vertical) from the three channels supplied by the sensor.

The mean $m_1$ and standard deviation $\sigma$ of the absolute value of the raw acceleration are calculated over a 10 second interval for each of the two channels. A mean $m_2$ is then calculated over a subsequent 0.5 second interval for each of the channels. The sensor detects an anomaly (generates a pick) on a given channel if, for that channel:

$$m_2 > m_1 + k\sigma$$

A client sends a pick to the data fusion engine if a pick is generated on the horizontal or vertical channel. A pick message includes the acceleration values measured on each channel.

The tail end of the 10-second window can influence the mean calculated in the following 0.5-second window; for example, if the acceleration at the end of the 10-second window is very low, then the acceleration in the following 0.5-second window starts at that low value. We use a 1-second gap

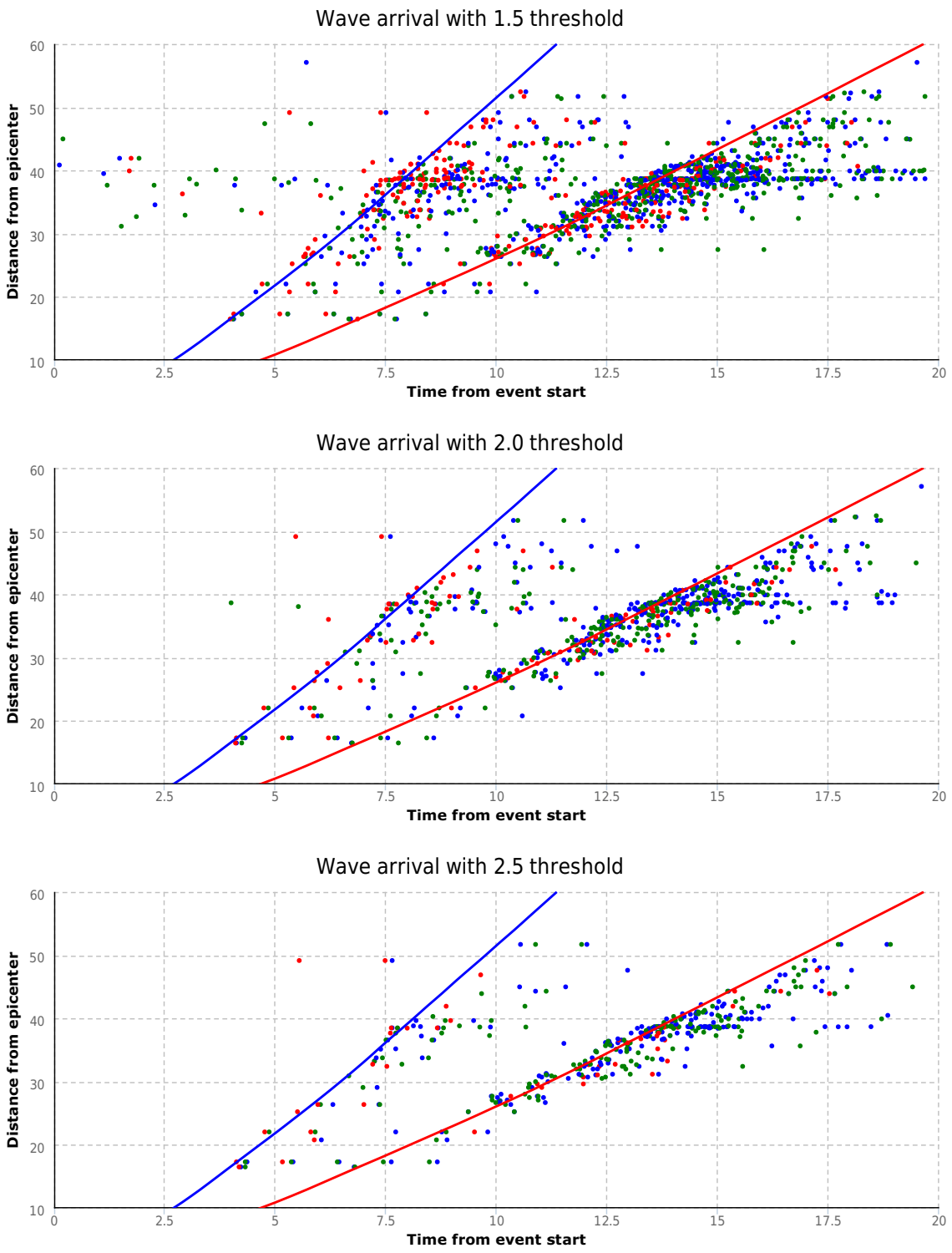
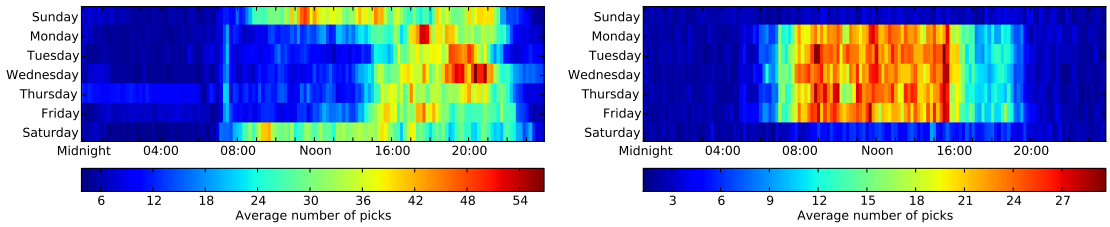



Figure 6.4: The figures show how adjusting the threshold value in the $k\sigma$ algorithm determines the noise levels and response rates of sensors during seismic events. The particular event pictured here is the magnitude 3.9 Newhall event.




(a) Pick frequency for a home sensor

(b) Pick frequency for an office sensor.

Figure 6.5: Pick frequency as a function of time of day and day of week for a typical home sensor.

between the two windows as indicated in the figure which shows the 10-second window which is used to calculate the mean and standard deviation, followed by a 1-second gap, followed by a 0.5-second window which is used to determine the current value. The gap and associated averages can be seen in fig. 6.6.

Figures 6.7 and 6.8 show picks from CISN and CSN sensors for a horizontal channel. The plots show that CISN sensors are much more accurate than CSN sensors, reporting values with far less variance. Figure 6.8 shows the substantial variation among CSN sensors. CSN sensor S1141 generates many more picks during quiet (non-earthquake) periods than CSN sensor S1139 and CISN sensors MIK and MIKB.

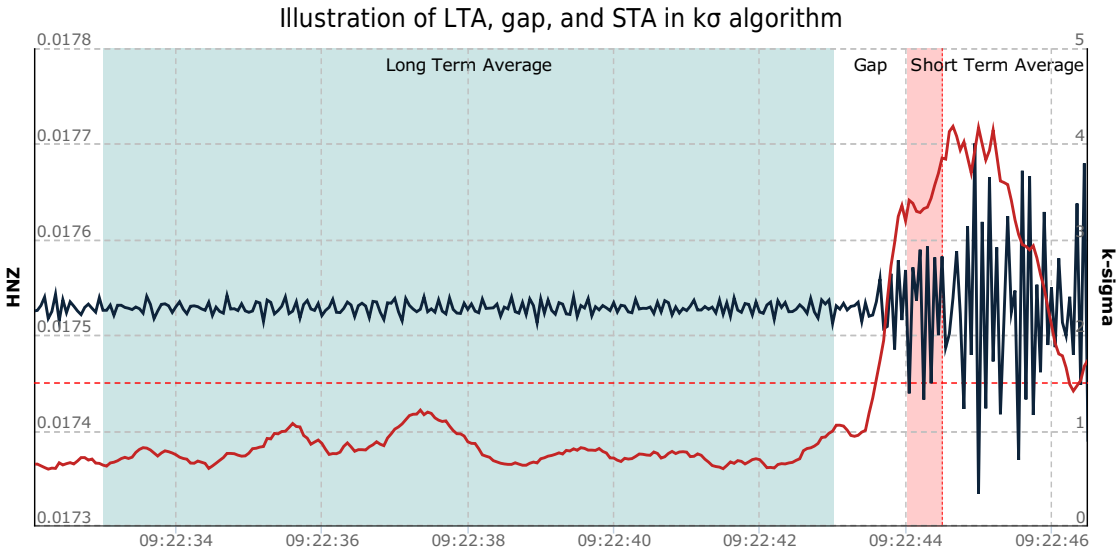
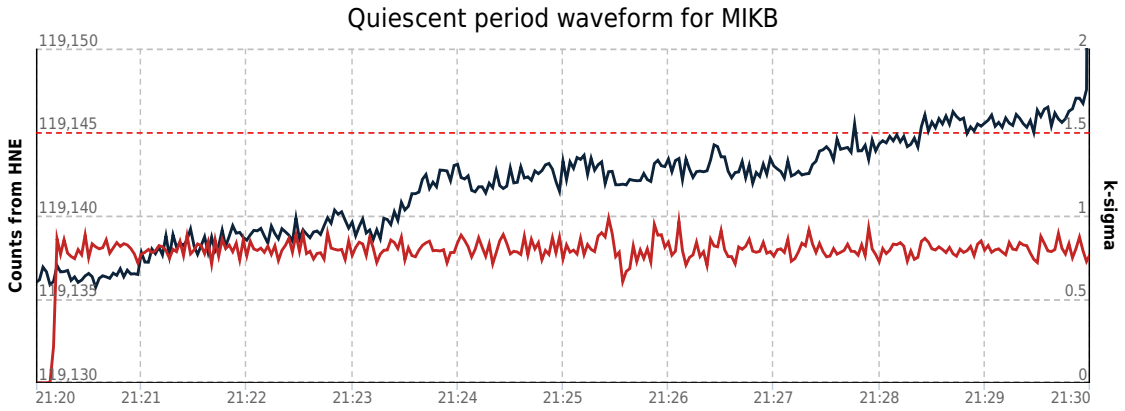
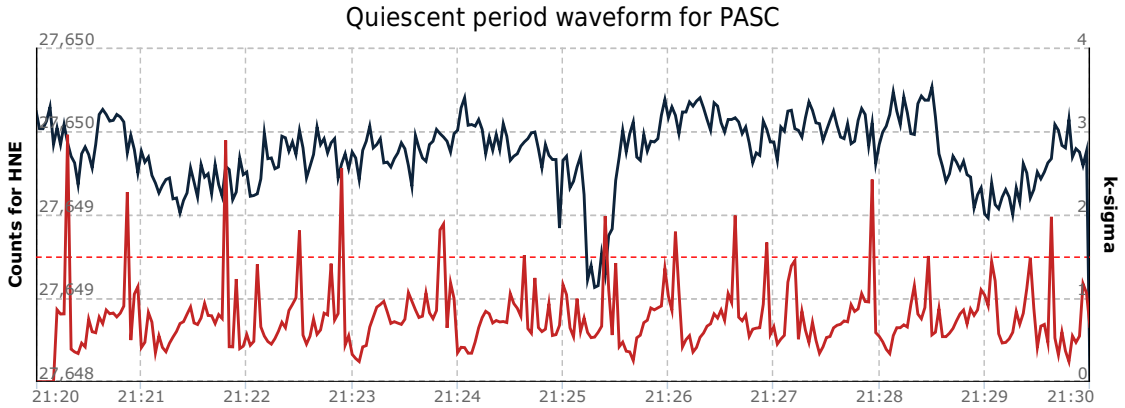



Figure 6.6: Diagram illustrating how the $k\sigma$ algorithm operates on a waveform. The dashed red line at the leading edge of the short term average indicates the instantaneous value of the algorithm for the point in time outlined by the windows shown.




(a) CISN station MIKB remains steady during the entire quiescent period.



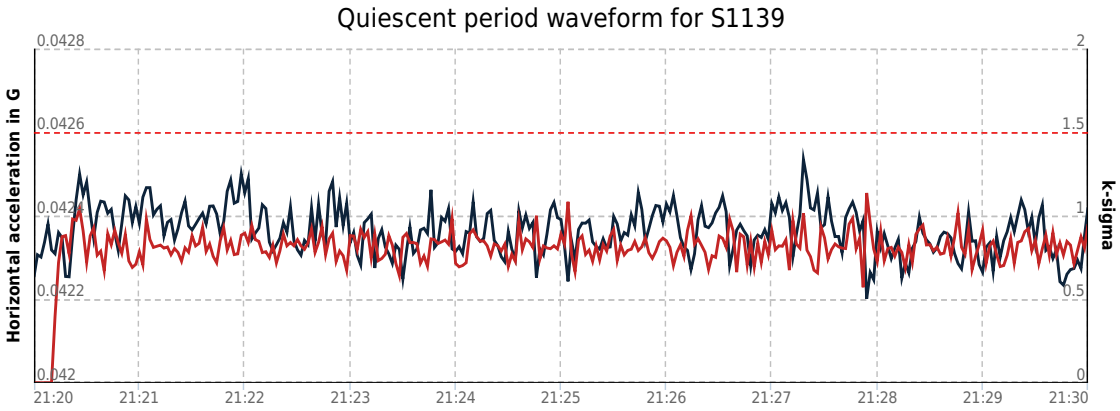

(b) CISN station PASC:00 shows signs of non-seismic influence during the quiescent period.

Figure 6.7: The figures show the waveforms of two CISN stations during a quiescent period with no known events.

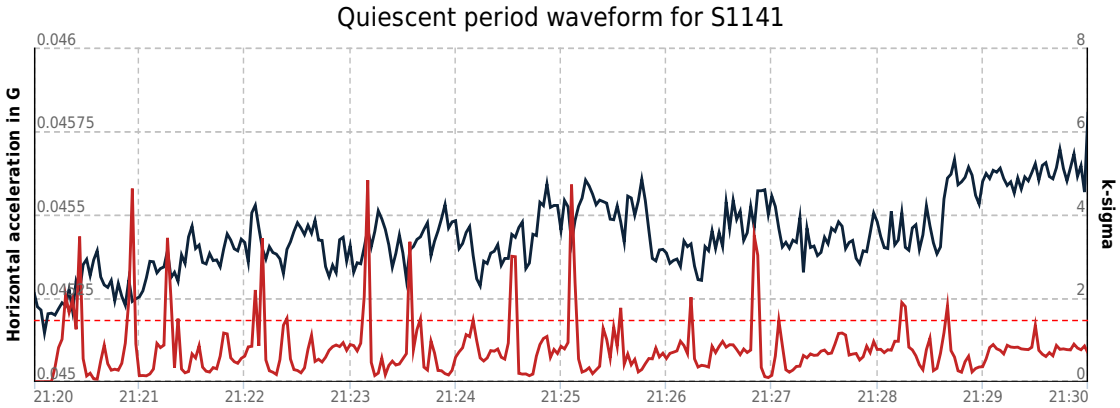
Sensors generate some picks due to electronic noise or shaking that is not due to earthquakes; such picks can be considered to be noise picks. The rate of noise picks varies over the time of day and days of the week for both CSN and CISN sensors. The differences in fig. 6.9 show the effects of vehicular traffic on roads on the CISN sensors and the effect of people moving around offices and homes for the CSN sensors.

I begin by first describing detection algorithms using only CSN sensors and then applying the same algorithm to both CSN and CISN sensors.

The data that the fusion engine receives in a pick message consists of the identity and location of the sensor, the time (as determined by the client) that the pick message was generated, and the maximum magnitude in each of the three axes. The problem is to determine whether shaking from




(a) CSN station S1139 remains steady during the entire quiescent period.




(b) CSN station S1141 shows wild fluctuations in signal even during the quiescent period.

Figure 6.8: The figures show the waveforms of two CSN stations during a quiescent period with no known events.

an earthquake is going on based on the sequence of pick messages from a sensor. In terms of the wave arrival plots, we look at a single horizontal line corresponding to the given sensor and based on the vertical blue and red dashes (the picks) we determine whether that sensor is shaking due to an earthquake or not.

Since the algorithm must detect earthquakes as the system is running, a detection must be based only on the history (the line segment) up to the current time. Since data from the distant past is less relevant to whether an earthquake is currently in progress, the detection algorithm makes its decision based on a short, moving time window rather than on the entire history. We discuss the appropriate length of the time window later.

The client algorithm is designed so that a client sends at most one pick message in a second.

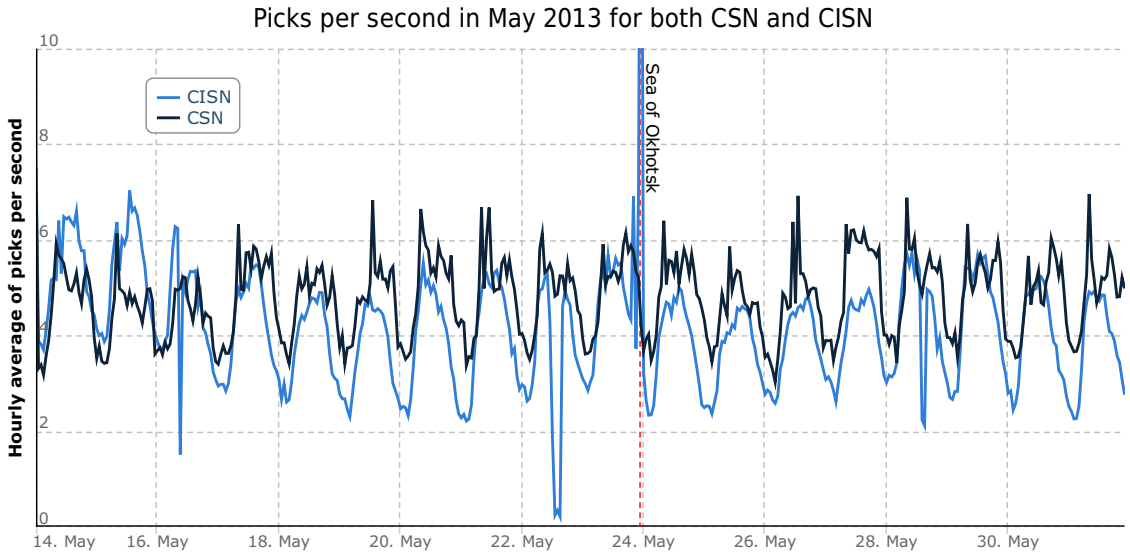



Figure 6.9: This figure shows the picks per second for both CSN and CISN in late May of 2013. The dual diurnal pattern of the two networks is clearly visible, as is the response of CISN to the Sea of Okhotsk event, and CSN's inability to detect the arriving wave.

This feature is to prevent a client from sending too many messages to the fusion engine causing an overload. In a 4-second window, a client can send 0, 1, 2, 3, or 4 pick messages.

The information in a 4-second window from a single sensor is the number of picks in the window and the maximum accelerations of each pick in each axis. Let *obs* be the information from a time window; for example, *obs* may be 4 picks each with acceleration exceeding 0.3g for one time window and *obs* may be 0 picks during a different time window. In the former case, the data *obs* indicates that an earthquake is in progress with high probability whereas in the latter case, *obs* indicates that with high probability no earthquake is in progress. Let

$$P(obs|quake)$$

be the conditional probability of a time window with information *obs* given that the time window occurred during a period of shaking from a quake. Similarly, let

$$P(obs|\neg quake)$$

be the conditional probability of a time window with information *obs* given that the time window occurred during a quiet period. Let

$$P(quake|obs)$$

be the probability that there is a quake going on given that the current time window has information *obs*. From Bayes' Law, $P(quake|obs)$ is a monotone increasing function of the ratio:

$$\frac{P(obs|quake)}{P(obs|\neg quake)}$$

Let us call this ratio $\alpha(obs)$:

$$\alpha(obs) = \frac{P(obs|quake)}{P(obs|\neg quake)}$$

The information *obs* generated by a sensor during a quake depends on the amount of shaking at the sensor. We set thresholds so that during *strong* motions the information *obs* generated by a low-noise sensor is typically such that the ratio exceeds $10^6$, and during quiet ($\neg quake$) periods the ratio is less than $10^{-2}$.

Bayes Law states:

$$P(quake|obs) = \frac{P(quake)P(obs|quake)}{P(quake)P(obs|quake) + P(\neg quake)P(obs|\neg quake)}$$

where $P(quake)$ is the *a priori* probability of a quake in the specified time interval (usually over a 4-second window). Hence:

$$P(quake|obs) = \frac{P(quake)\alpha(obs)}{P(quake)\alpha(obs) + 1 - P(quake)}$$

Assuming that $P(quake)$ is very small:

$$P(quake|obs) \approx \frac{P(quake)\alpha(obs)}{P(quake)\alpha(obs) + 1}$$

Consider an observation *obs* that is typical of observations when there is a quake occurring. For a given observation *obs*, the probability of a quake given the observation exceeds some threshold $q$, if:

$$P(quake)\alpha(obs) \geq \frac{q}{1-q}$$

For example, if $\alpha(obs) = 10^{30}$ and $q = 0.99$, then the probability of a quake given the observation exceeds the threshold 0.99 if the *a priori* probability of a quake in the time window exceeds about $10^{-28}$. If, however, $\alpha(obs)$ is smaller, say $\alpha(obs) = 10^6$, and the threshold is the same (0.99), then the *a priori* probability of a quake in the time window must exceed about $10^{-4}$. Thus, the decision about whether to take action in response to an observation *obs* is less sensitive to the *a priori* probability of a quake if $\alpha(obs)$ is high. We would like our sensors to be low-noise and accurate, and to have enough sensor density so that $\alpha(obs)$ is very high; as a consequence decisions about whether to issue an alert are insensitive to the usual *a priori* probabilities of an earthquake.

Likewise, consider an $obs'$ which is typical of observations during quiet times (i.e., no quake in progress). The decision to (correctly) take no action in response to observation $obs'$ is less sensitive to the *a priori* probability of a quake if $\alpha(obs')$ is low.

Next, we analyze empirical data to record observations during quiet times and earthquakes and estimate the values of $\alpha$ for these observations.

The data shows that CSN sensors usually pick multiple times as the S-wave passes through during a significant quake. Let the information *obs* obtained for a 4-second time window from a sensor be of the form: "*the sensor generated j picks in 4 seconds*" where $j$ is in $\{0, 1, 2, 3, 4\}$. (Since a sensor can pick at most once a second, the maximum number of picks in 4 seconds is 4.) Thus the information *obs* is characterized by an integer in the interval $[0, 4]$.

The system measures the rate at which each sensor generates picks. Let the probability that a given sensor generates picks in any second be $Q$. Assuming that picks generated during quiet periods are due to random noise, the probability of $j$ in a 4-second interval is a given by the

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Quiet Sensor | $\approx 1$ | $6.6 \times 10^{-3}$ | $1.7 \times 10^{-5}$ | $1.8 \times 10^{-8}$ | $7.7 \times 10^{-12}$ |
| Noisy Sensor | 0.93 | 0.06 | $1.6 \times 10^{-3}$ | $1.8 \times 10^{-5}$ | $7.7 \times 10^{-8}$ |

Table 6.1: This table shows the probability of a given number of picks in a four second window for a quiet and noisy sensor.

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 0.1 | 0.2 | 0.2 | 0.3 | 0.2 |

Table 6.2: Probability of j picks during 4 seconds in a quake.

binomial probability mass function:

$$\binom{4}{j} Q^j (1-Q)^{4-j}$$

These probabilities are given in table 6.1 for a quiet sensor which generates picks at the rate of a pick in 10 minutes, and a noisier sensor which generates picks at the rate of a pick per minute.

We estimated the probability that a sensor would pick $j$ times during a 4-second window in an earthquake by analyzing the picks generated during a 60 second window while a quake was in progress for sensors experiencing moderate shaking. Table 6.2 shows the empirically estimated probabilities. These estimates were made for earthquakes and sensors in geocells where the sensors picked multiple times; examples are sensors 20 - 40 km away from the epicenter for the Newhall 2 and Yorba Linda 2 events.

Table 6.3 shows values of the ratio $\alpha(obs)$ for the quiet and noisy sensor where *obs* is the number of picks in 4 seconds; table 6.3 is table 6.1 divided by table 6.2.

The tables show that if a quiet sensor generates 4 picks in 4 seconds then the ratio is very high, and this ratio is also high for the noisy sensor. If a sensor generates no picks in 4 seconds then the ratio is low, as it should be. The table shows that for a quiet sensor the values increase very rapidly as the number of picks increase. Even for the noisy sensor, the values increase quite rapidly, with high values for 3 and 4 picks.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Quiet Sensor | 0.11 | 63 | $1.9 \times 10^4$ | $1.1 \times 10^7$ | $1.3 \times 10^{10}$ |
| Noisy Sensor | 0.11 | 6.3 | $1.9 \times 10^2$ | $1.1 \times 10^4$ | $1.3 \times 10^6$ |

Table 6.3: This table shows the value of the ratio $\alpha(obs)$ for a quiet and noisy sensor.

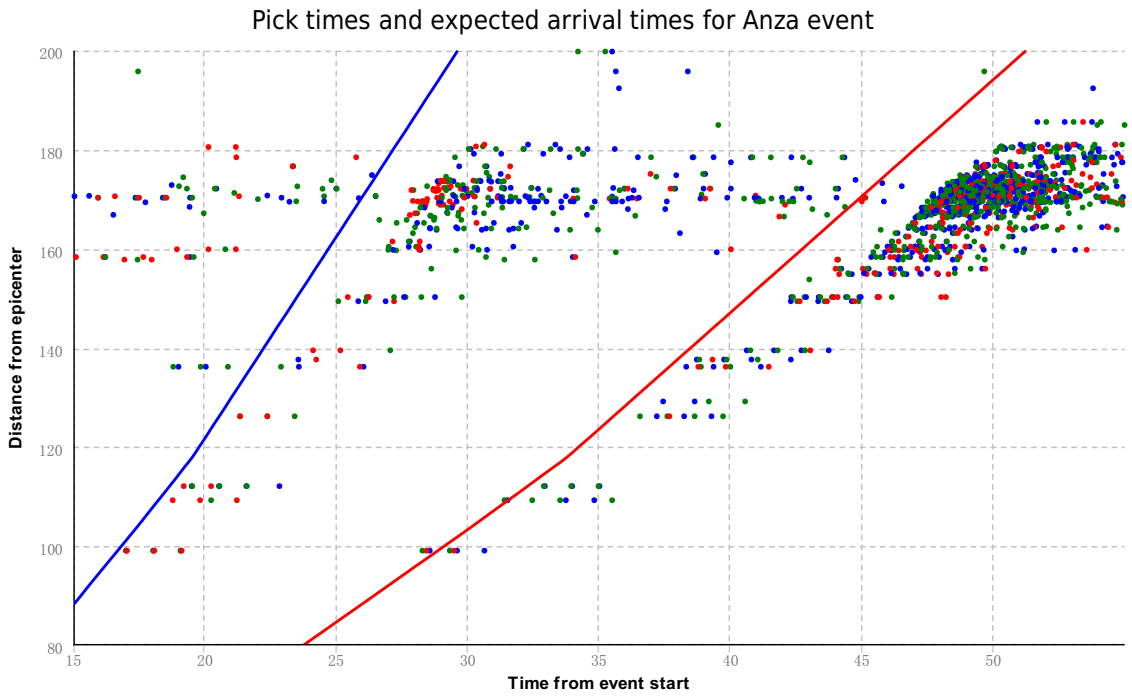



Figure 6.10: Wave arrival plot for the 4.7 magnitude Anza event.

The Python implementation of the $k\sigma$ algorithm presented above can be seen in appendix A.2.2.3.

### 6.1.2 Fusing data from all sensors in a small Geocell

An anomaly reported by a single sensor could be due to electronic noise in that sensor or due to sensor acceleration. The acceleration may be due to proximate physical activity such as somebody walking by the sensor or slamming a door nearby, The activity may also be a geographically widespread phenomenon such as loud thunder, cannon shots (particularly in Caltech at the end of term), and earthquakes. Anomalies reported simultaneously by multiple sensors separated by 50 meters or more are unlikely to be due to purely local phenomenon.

Figures 6.10 to 6.14 show picks for sensors during earthquakes and quiet periods. In these plots a sensor can generate at most one pick for each channel in a second; this a single sensor can generate up to 3 picks in each second. The horizontal axis in the chart is the time elapsed since the initiation of the earthquake. Each sensor's picks are represented as individual dots. Picks made on horizontal channels are represented as blue and green dots, while picks made on the vertical

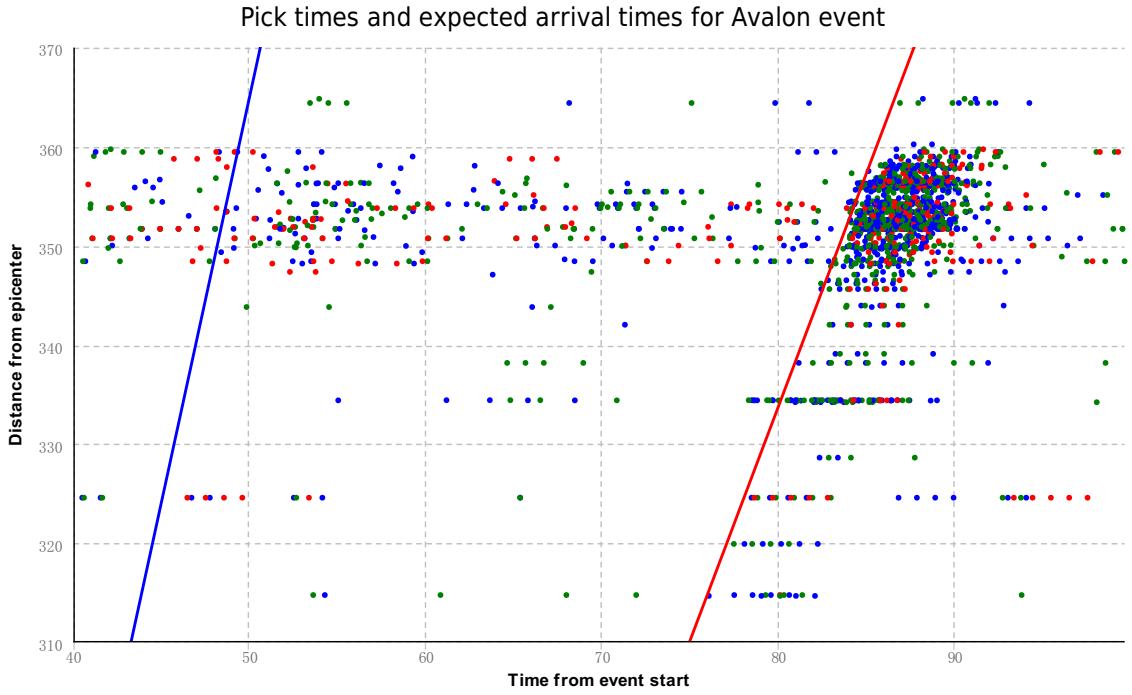



Figure 6.11: Wave arrival plot for the 6.3 magnitude Avalon event. Note the high sensor distance from the epicenter.

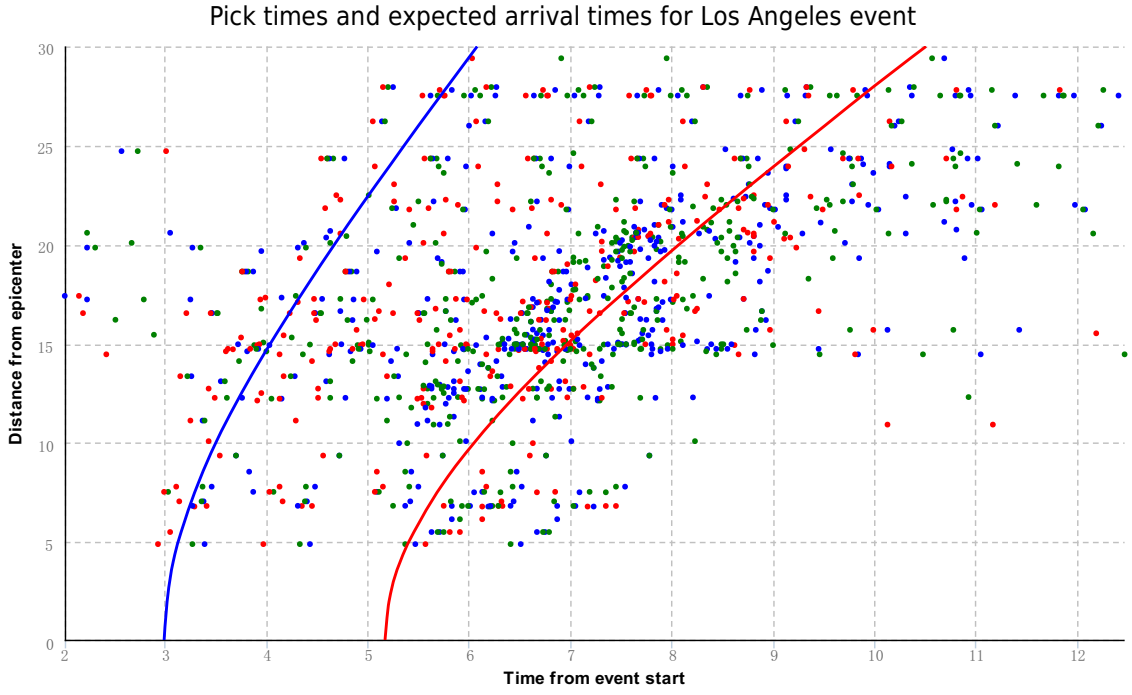



Figure 6.12: Wave arrival plot for the 2.8 magnitude Los Angeles event.

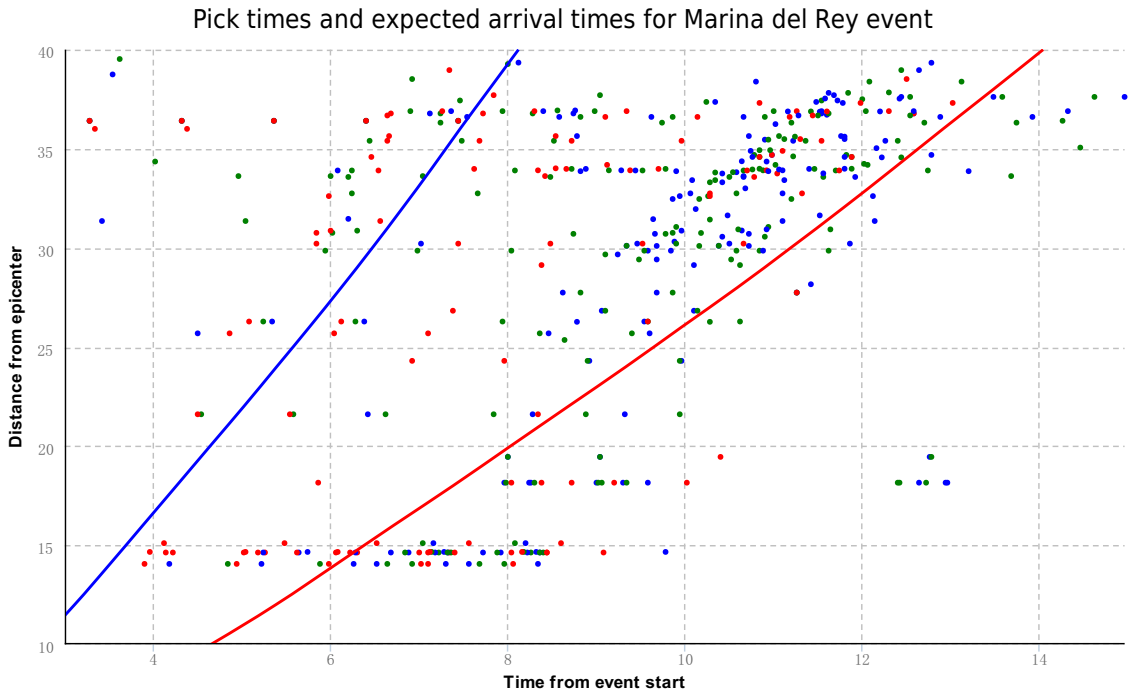



Figure 6.13: Wave arrival plot for the 3.2 magnitude Marina del Rey event.

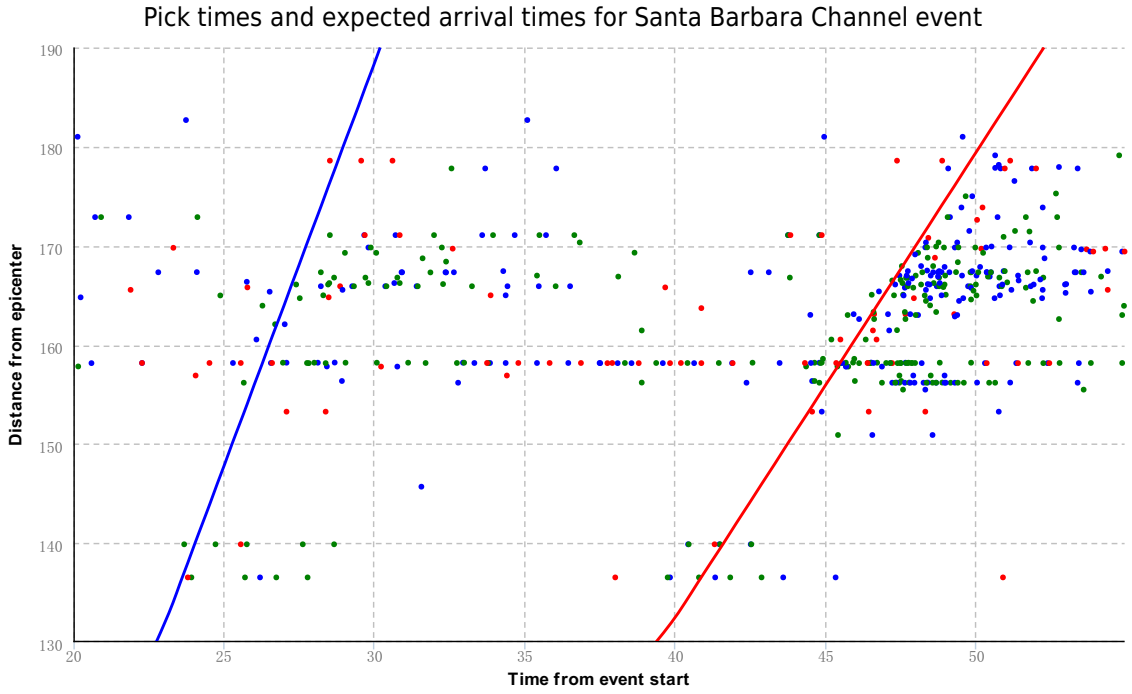



Figure 6.14: Wave arrival plot for the 4.9 magnitude Santa Barbara event.

channel are represented as red dots. The blue line (the left-hand line) through the graph indicates the estimated arrival time of the P-wave, while the red line through the graph (the right-hand line) indicates the estimated arrival time of the S-wave. The yellow line indicates the time at which the P-wave should reach each sensor and the green line indicates the same for the S-wave.

The challenge of distinguishing periods of shaking due to earthquakes from periods of quiet (non-earthquake) times can be cast as the problem of detecting patterns such as those shown in the wave arrival plots during earthquakes from the patterns evidenced during quiet times.

We begin the development of detection algorithms by asking the questions: how should picks from sensors in a single geocell be aggregated so as to improve the accuracy and ROC curve of the entire geocell treated as a single unit?

We assume that all proximate sensors undergo shaking at the same time during an earthquake. We also assume that picks due to electronic or local noise are independent. We compute the ratio:

$$\alpha(obs) = \frac{P(obs|quake)}{P(obs|\neg quake)}$$

where *obs* is the information from all the sensors in the (small) geocell. Assuming independence, $\alpha(obs)$ for the geocell is the product of the ratios for the sensors in the geocell.

For example, if a geocell of size 1 km square has 4 quiet sensors (1 pick per 10 minutes), and if all sensors pick 4 times in 4 seconds then the ratio $\alpha$ is $\approx 1$ and if none of the sensors pick the ratio is $9.5 \times 10^{-10}$. The corresponding values for a group of 4 noisy sensors (1 pick per minute) are 0.79 and $4.6 \times 10^{-9}$.

The algorithm for calculating the cell level probability can be observed, as implemented in SAF, in appendix A.2.2.1.

### 6.1.3 Fusing data from all sensors in the system

A geocell generates a pick message if the system detects anomalous heavy shaking when the system fuses data from all the sensors in a (small) geocell. A geocell may generate a pick due to some

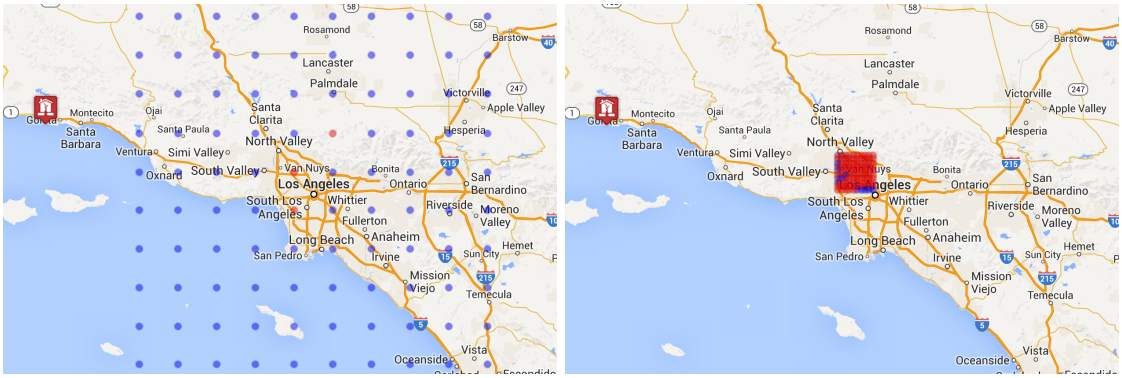
| | | | Active | | Detection | | Arrival | |
|---|---|---|---|---|---|---|---|---|
| | Mag. | Dist. | Sensors | Cells | Sim. | Time | P | S |
| Altadena | 2.1 | 5.4 | 96 | 46 | S/1.0 | 7.60 | 3.14 | 5.44 |
| Anza | 4.7 | 168.5 | 192 | 70 | P/1.0 | 29.32 | 25.75 | 44.54 |
| Avalon | 6.3 | 351.9 | 183 | 65 | S/1.0 | 84.38 | 33.82 | 55.59 |
| Beverly Hills | 3.2 | 26.3 | 175 | 63 | P/1.0 | 6.76 | 5.81 | 10.04 |
| Brawley | 5.5 | 268.4 | 203 | 65 | S/1.0 | 54.82 | 30.03 | 51.96 |
| Los Angeles | 2.8 | 15.1 | 194 | 70 | P/1.0 | 5.66 | 4.05 | 6.99 |
| Manhattan Beach | 3.3 | 50.2 | 241 | 81 | S/1.0 | 15.24 | 9.09 | 15.72 |
| Marina del Rey | 3.2 | 34.5 | 202 | 76 | P/1.0 | 8.74 | 7.23 | 12.50 |
| Newhall (2) | 3.9 | 39.7 | 226 | 78 | P/1.0 | 7.88 | 8.08 | 13.97 |
| Quiescent | - | - | 200 | 71 | - | - | - | - |
| Rancho Cucamonga | 1.9 | 52.5 | 179 | 61 | - | - | 9.55 | 16.53 |
| Sea of Okhotsk | 8.3 | 3.6K | 222 | 75 | - | - | - | - |
| Santa Barbara Channel | 4.9 | 168.5 | 216 | 73 | S/1.0 | 48.38 | 27.55 | 47.66 |
| Yorba Linda (2) | 4.3 | 39.6 | 187 | 69 | P/1.0 | 8.16 | 7.45 | 12.90 |

Table 6.4: This table lists the earthquakes used in simulation experiments. Distances and arrival times measured from cell bTaBAQ, which includes the eastern half of Caltech. The "Sim." column captures whether the simulation run of the experiment detected the quake, on which wave the detection appears to have occurred, and the probability of the associated alert. The "Time" column records the distance in time from the event origin before the simulation detection occurred.

intense local activity such as the demolition of a building. Therefore, we seek corroboration from sensors further away from that geocell.

Consider an earthquake with a specified hypocenter. Assume that S-waves propagate according to a known model. The pattern of quakes that we should expect to see is shown in fig. 6.10 which depicts the Anza event. The x-axis is time and the y-axis is distance from the epicenter. To distinguish a period during which shaking from an earthquake is going on from quiet periods we compare the probability that the patterns of picks generated by all sensors was generated during an earthquake versus the probability that the pattern was generated randomly.

If we had unlimited computing power then we would calculate the pattern of picks that would be generated by an earthquake for every possible location $(x, y, z)$ of the hypocenter and earthquake origination time $t$. Even with a grid granularity of 1 km and time granularity of 1 second, and a $100 \times 100 \times 30$ km-cubed region and time running over 50 seconds, the number of possible values for which S-wave profiles are computed exceeds $10^7$. Therefore, we use a multi-grid approach. We start with a coarse grid of 400 points in space over $100 \times 100 \times 20$ km cubed area, use 1 second time steps over a 20 second window, and calculate the point on the grid that is most likely to be




(a) Hypocenter fit 1st pass.

(b) Hypocenter fit 2nd pass.

Figure 6.15: The dots in these figure show the $x, y$ coordinates that are searched for a first and second pass through the hypocenter fit algorithm, respectively. The first past diagram demonstrates why the fit algorithm had no chance to find the far out of network Santa Barbara Channel event.

the hypocenter that generates the observed pattern of picks. The first pass of this grid is visualized for the $x, y$ coordinates only in fig. 6.15a.

After the first pass, we consider a finer grid around that point and repeat the process. Pictured at the same scale, fig. 6.15b shows how much smaller the area scanned in the second pass is. The second pass searches the same 400 points using 1 second time steps over a 20 second window, but in a $10 \times 10 \times 20$ km cubed area.

The highest probability point from the second pass is then selected as the fit. Many fits may be performed, and how to combine information from fits that are close in time, is a subject for future examination. In production, the system naively limits the rate at which fits will be performed. The simulation system will perform a pick every time that the probability threshold for a level 2 alert is exceeded.

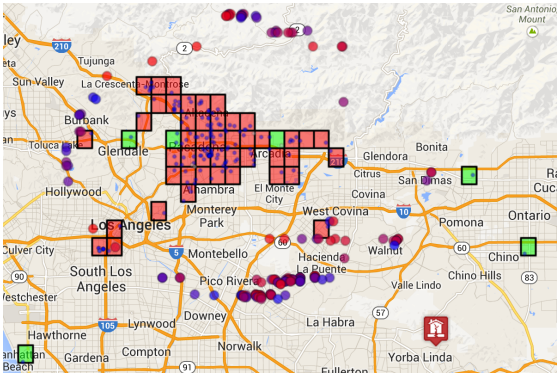
The algorithm for iterating across the grid and finding the highest probability points is included in part in appendix A.2.3.

## 6.2 CSN Performance

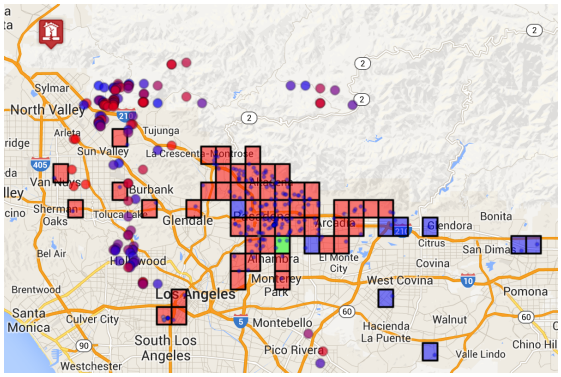
Section 5.1 discusses the Community Seismic Network in detail. This section highlights observations of the performance level experienced by CSN's App Engine application. PaaS systems were inves-

tigated in general, and Google App Engine selected in particular, because of the ability to scale in small amounts of time from using minimal resources to consuming large amounts of resources. App Engine's architecture is discussed in detail in section 5.2.

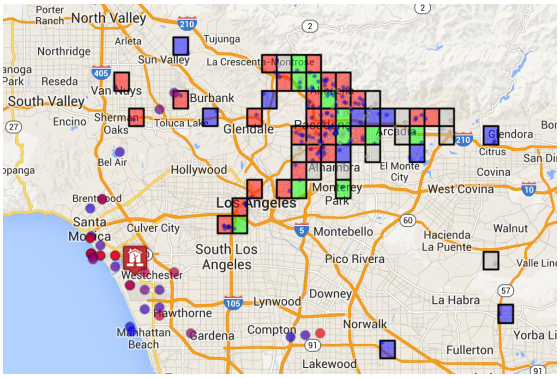
Note that the measurements in this section were taken without allocating any reserve instances to the system; that is, no excess capacity beyond what was required to serve the current load was normally available to the system. This was an intentional configuration choice to test the speed with which the system could adapt without explicit reservation of additional resources. This network



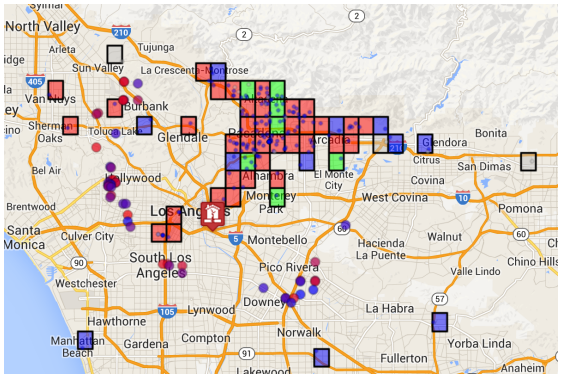

(a) An attempt to fit the magnitude 4.3 Yorba Linda event. As a function of the tight cluster of CSN sensors, the algorithm cannot discern the direction of the arriving wave.




(b) An attempt to fit the magnitude 3.9 Newhall event. The algorithm successfully discerns the direction of the event, but not its distance from the network.




(c) An attempt to fit the magnitude 3.2 Marina del Rey event. The fits are clustered directly over the true epicenter.




(d) An attempt to fit the magnitude 2.8 Los Angeles event. The algorithm is able to figure out the overall direction, but forms a fan around the true epicenter.

Figure 6.16: In these figures, fits are attempted in simulation when level 2 alerts in the network are triggered. The house symbol represents the epicenter as reported by USGS. Large colored circles represent fit attempts, and get brighter colored as they move forward in time. Small blue dots represent picking sensors, while small grey dots represent non-picking sensors. Red cells indicate cells which triggered level 2 alerts, green cells triggered level 1 alerts, blue cells picked with no alerts, and grey cells had no picks at all.

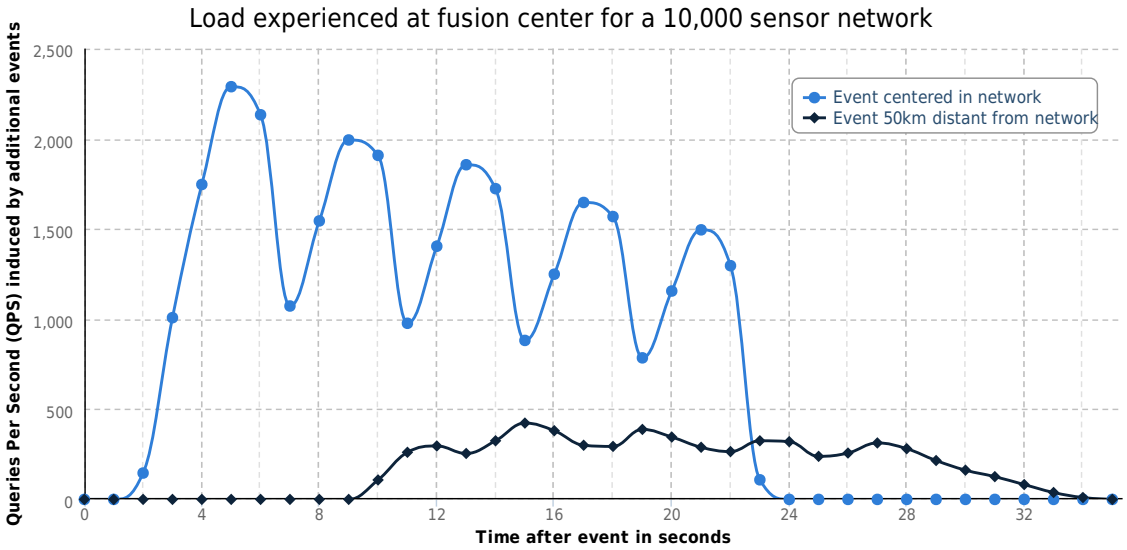


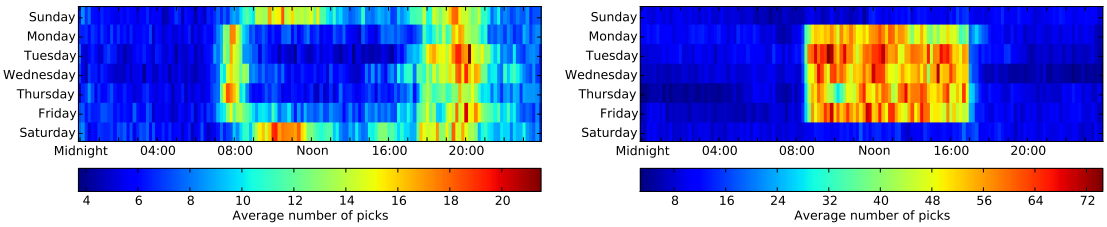

Figure 6.17: Estimates of the amount of server traffic generated by a magnitude 5 earthquake at different distances from the sensor network [4].

would benefit from a modest number of reserved instances, which would improve the average response rate of the system and the size of the request rate spikes that it could absorb without incident.

### 6.2.1 Traffic levels

While during quiescent periods the only data sent on the network is control traffic, which amounts to very little, the data sent during seismic events is quite substantial. We ran simulations to estimate the traffic load of a dense network, which you can see the results of in fig. 6.17. For a network of 10,000 sensors, we expect the number of queries per second (QPS) the server must handle when sensors detect a magnitude 5 earthquake to reach a maximum rate of 423 QPS for an earthquake 50 km distant to the center of the network and a maximum rate of 2,289 QPS for an earthquake centered relative to the sensor network.

Sensor applications in general, and CSN in particular, often have strong diurnal patterns in their readings. Because CSN's sensors communicate when changes in the environment are sensed, pick rates tend to be higher — often substantially so — during hours when community members are active. Figure 6.18 shows differences between an office sensor, with strong patterns during weekday working hours, and a typical home sensor, with more varied pick frequency patterns.




(a) Pick frequency for a home sensor

(b) Pick frequency for an office sensor.

Figure 6.18: Pick frequency as a function of time of day and day of week for a typical home sensor. Rates are calculated over ten minute intervals.

While the community nature of CSN makes it more susceptible to such environmental influences, these same patterns can be derived for CISN's carefully deployed network as well. In this case, the attribution is to increased human activity in general, such as road traffic, which CISN sensors are sensitive enough to pick up on, rather than the actions of a small number of individuals, as might be the case with CSN sensors. In fig. 6.9, you can see the diurnal pattern in picks generated by examining the CISN waveforms.

### 6.2.2 Fluctuations in performance

The lack of responsibility for PaaS component performance is both boon and bane. The benefits of abdicating responsibility are discussed in section 5.2, while here we discuss the liabilities.

When systems are independently managed, shortfalls in performance can be analyzed and corrected. When utilizing a provided service, shortfalls in performance can often be inexplicable, and, even if a way exists to remediate these problems, it may not be clear what the mechanism is without insight into the root cause.

For example, CSN experienced highly elevated latency for two extended periods in 2012. Both periods arrived and departed with no code changes committed to create or address the performance issue. Figure 6.19 shows the periods of elevated latency for a single request type with code deployments shown as vertical red lines.

Additionally, using public cloud services in general, and PaaS systems in particular, means that performance can fluctuate as a function of load put on the system by other users — a factor which

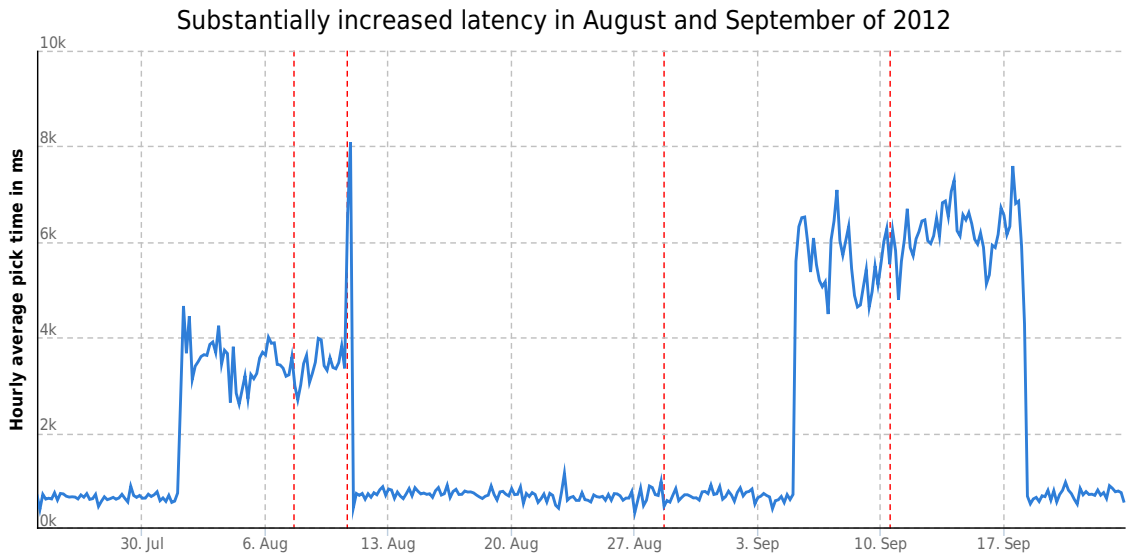



Figure 6.19: Poor average performance in App Engine for two extended periods in 2012. Vertical red lines show code deployments.

cannot be deduced or accounted for. Figure 6.20 shows the average processing time for CSN over a long time interval along with the application traffic level at those points in time. A cursory examination of the graph reveals that request volume and processing time do not appear to be related.

### 6.2.3 Instance startup time

Figure 6.21 shows the difference in the amount of time required to process a pick sent to a cold instance for both the original CSN server and SAF. The data supports the conclusions drawn by the simple experiments in section 5.2.5 with respect to Java versus Python startup times, but in this case for real world applications operating at roughly 10 QPS and 7 QPS respectively. The average Java startup times for simple applications studied were indicative of the trend, but not the extent to which startup times would grow for both languages.

Between January, 2011 and July, 2013, the startup times for the original CSN server have increased dramatically. Figure fig. 6.22 shows the progression in startup times. The total lines of Java code in January, 2011 was 17,338, while the code base grew to 26,488 lines by 2013. This change in code volume hints at a change in application complexity that cannot be directly measured, but no

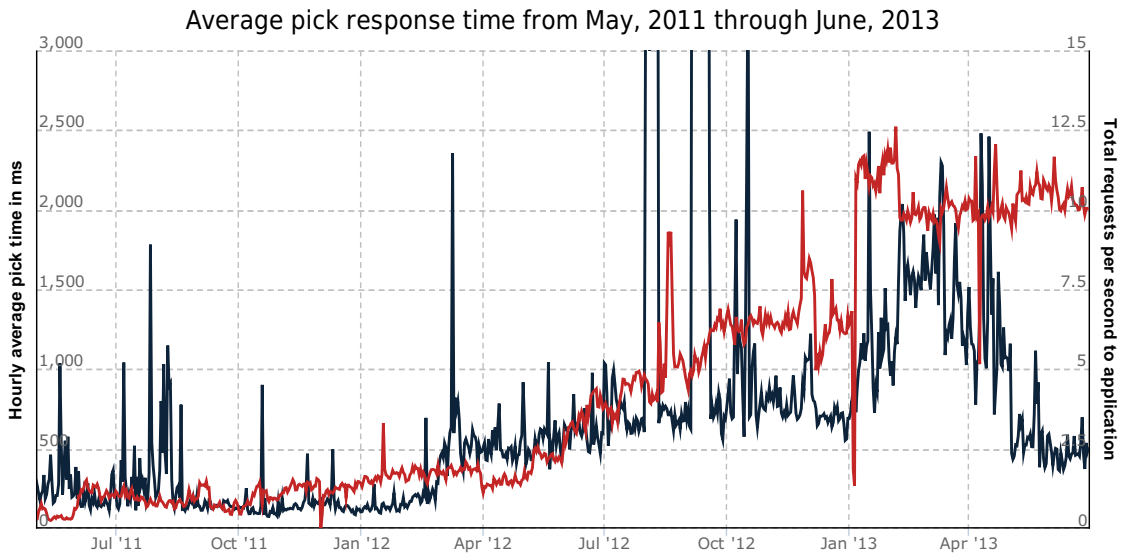



Figure 6.20: Shows the average processing time on App Engine over a long interval. The total volume of requests per second to the application is shown in red.

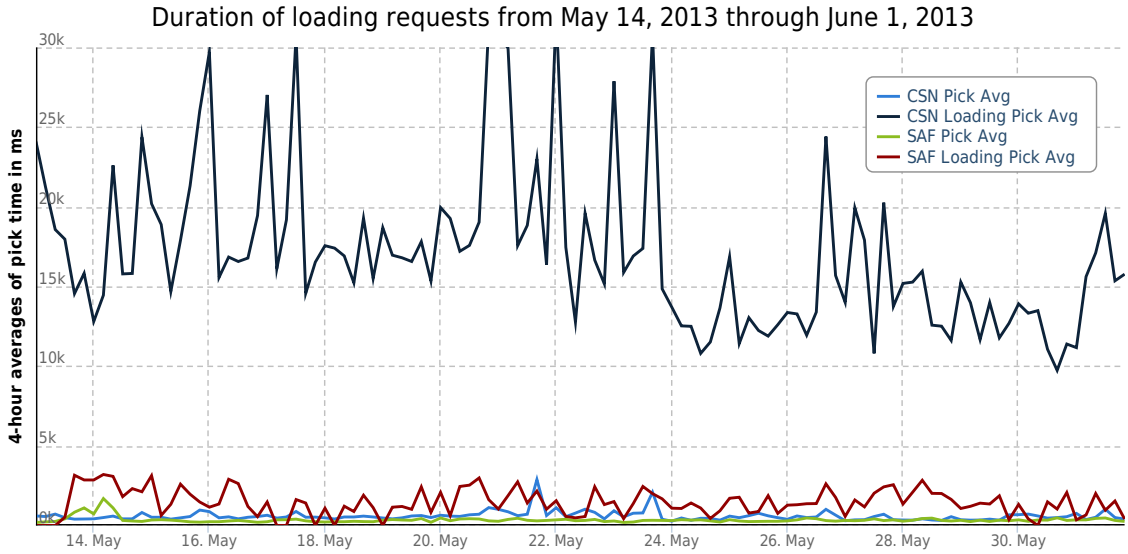



Figure 6.21: This figure shows the average time to process a pick request sent to a cold instance — a loading request — and the average time to process picks sent to a warm instance for both the original CSN server and SAF.

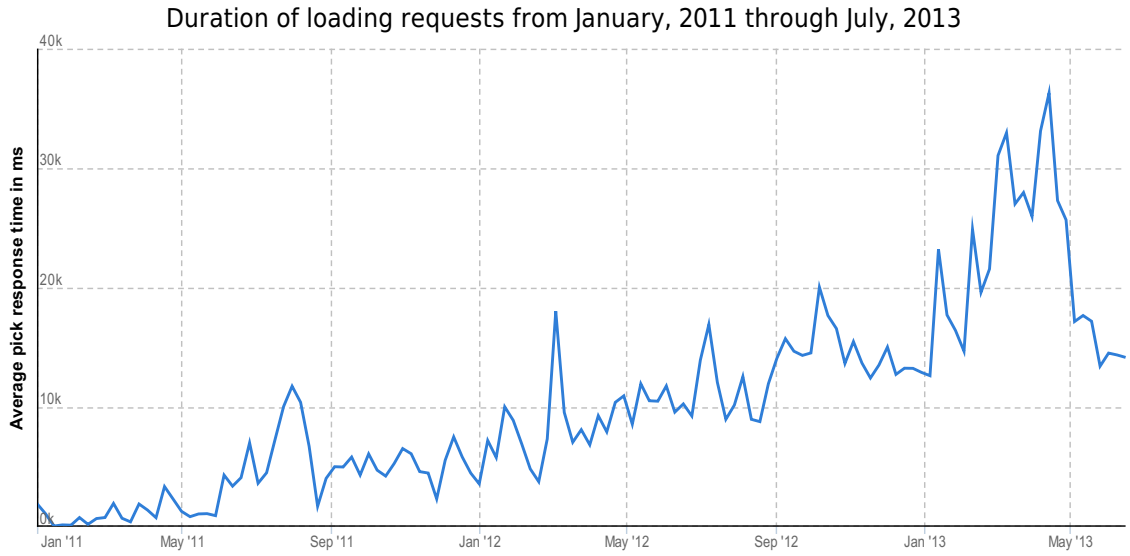



Figure 6.22: This figure shows the change in loading request processing times for the original CSN server between January of 2011 and July of 2013. The data is averaged over one week intervals to remove sharp spikes from the graph.

relationship with the change in startup time can be established.

These figures provide further evidence for the assertion made in section 5.2.5: applications that are sensitive to latency should not rely on dynamically commissioned instances on App Engine when using Java and, if particularly sensitive, should not rely on them at all. Future experiments should evaluate the performance of Go when running an application of similar complexity as well as the impact of splitting an application into "Modules", as provided for in the latest release of App Engine on July 17 [68].

### 6.2.4 Scalability

Other applications — including "Hope for Haiti Now: A Global Benefit for Earthquake Relief" [69], Wolfire Games's "Humble Indie Bundle" [70], Earth Hour [71] and WhiteHouse.gov's "Town Hall Meeting" [72] — have reported useful numbers demonstrating platform scalability; the reported statistics are summarized in table 6.5. The statistics provided are quite shallow, however, and do not reflect interesting details such as the number of instances used to serve the applications or the rate at which those instances were commissioned. We explore those figures for both CSN and CISN

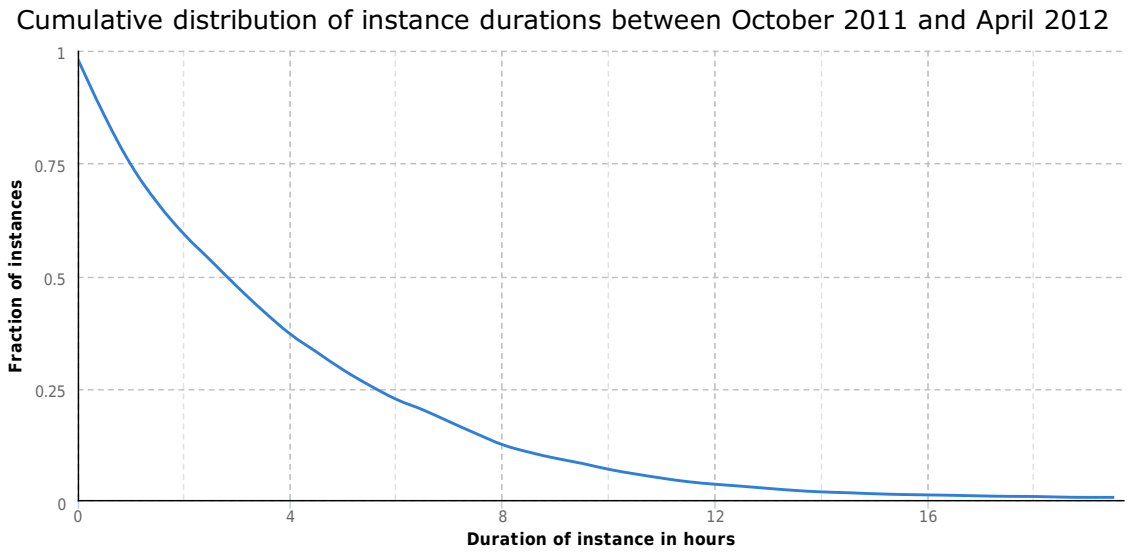



Figure 6.23: This figure shows the number of instances that were serving requests for specific lengths of time between October 2011 and April 2012. The histogram is generated based on data from 122,000 instance records.

| | Peak QPS | Daily |
|---|---|---|
| Hope for Haiti | 1,600 | n/a |
| White House | 700 | 1.8m+ |
| Earth Hour | 450 | n/a |
| Humble Bundle | n/a | 283k+ |

Table 6.5: Statistics that applications have reported publicly regarding peak QPS and daily volumes for App Engine applications.

as well as more closely examining relationships between load and other system statistics for the largest spike in traffic the network has observed to date.

One facet of the scaling algorithm in App Engine is that under normal configuration, it moves relatively quickly to load and unload instances. This is a desirable feature when load varies not only between periods of time with events and quiescent periods, but also between times of the day. Figure 6.23 shows a histogram of the length of time that instances were available for serving requests. Because information on the creation and decommissioning of instances is not available directly from App Engine, this data is constructed using the MapReduce job depicted in appendix A.4; it does not show the period of time after commission but before the first request and before decommission but after the last request.

Ten minutes after the Sea of Okhotsk event in Russia, the wavefront arrived in Southern Cali-

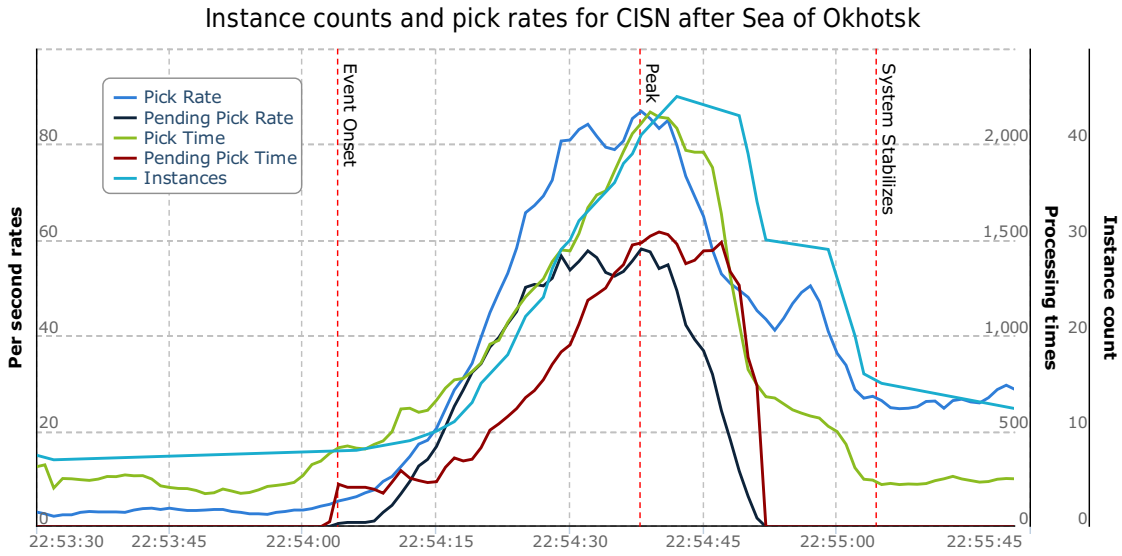



Figure 6.24: This graph shows the change in picks per second, pick processing times, and instance counts after the May 2013 Sea of Okhotsk event. The rate of picks that were forced to queue — pending picks — and the amount of time they queued for is also displayed. These values represent ten second averages sampled every second.

fornia. Despite its distance from California, this wave resulted in a large increase in traffic from the CISN network. Traffic levels in SAF are depicted in fig. 6.24 and show the corresponding increase in instance numbers. The number of instances rose from 7 to 45 at the peak traffic rate, while the pick rate climbed from an average of around 3 picks per second to an average of 87 picks per second.

During this time, the rate of picks that were pending — queued for processing at an instance — also rose as new cold instances were created to process requests (see section 5.2.5 for more on cold instances). The average pick processing time rose from just over 200 ms per pick to just over 2,000 ms at the peak traffic time, but that number does not include the pending time. For the fraction of picks that were pending, roughly two thirds at the peak pick rate, the average pending time of 1,500 ms must also be added in, giving an average processing time at peak of 3.5 seconds for pending picks.

Because of the dramatic increase in processing times, we can say that during this rapid increase in rates the system performance destabilized. It is worth noting, however, that the pick error rate during that time remained at 0; no picks were terminated with a 500 error. App Engine's ability to scale quickly is demonstrated by the fact that the system performance stabilized again at the new

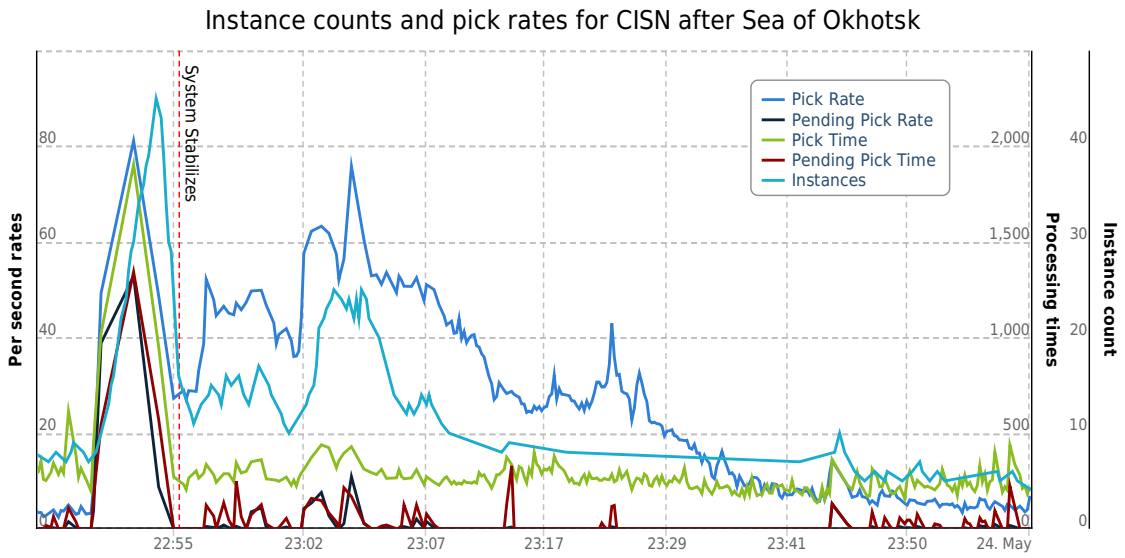



Figure 6.25: This graph is similar to fig. 6.24 but shows more time after the event. A second peak in pick rates occurs approximately five minutes later.

rates within less than a minute. The system stabilized at a new average pick rate of 27 picks per second with 15 instances, at which time the average processing time had returned to just over 200 ms and the fraction of picks pending had returned to 0.

The time from the onset of increased rates to system stabilization is approximately one minute. During that time, the system continued to perform, but with substantially elevated latency. The speed with which the system stabilized is quite fast, particularly in comparison to other mechanisms for scaling discussed in section 3.2, but may not be adequately fast for sense and response applications, like seismology, that require low latency levels at all times. Although algorithms can be used that operate with missing data, elevated latency will always drive up detection times.

About five minutes after the first spike, a second pick rate spike, pictured in fig. 6.25, occurred and reached a higher average rate of 94 picks per second. Because of the increased number of instances, which were now warm, the fraction of picks pending and the amount of time they spent pending did not increase appreciably. This indicates that the stabilization of the system after the first spike was not ephemeral, and under sustained load would continue to behave as expected.

### 6.2.5 Worldwide deployment

Earthquake detection currently exists at the limits of what the network in its current form is capable of. The data in the previous section, however, shows that a modest increase in the number of available instances makes it possible to absorb large spikes in traffic. This indicates that if the network existed in multiple locations, then the number of instances required to operate the network with nominal traffic would require enough instances to absorb the observed traffic spike without ill effects. That is, the performance of the network as a whole improves with its geographic scope. If the CSN's goal of performing earthquake detection in different parts of the world is achieved, then performing detection in any part of the world will be made easier by the infrastructure necessary to support the worldwide network.

# Chapter 7

# The Situation Awareness Framework

## 7.1 Generalized framework

The Situation Awareness Framework is a reimplementation and generalization of the first CSN server, which began its life in early 2009. SAF adds many features on top of what the original server was capable of, some of which are discussed at length below. More briefly, SAF: transitions the CSN client base to the high replication datastore, preventing network outages as a result of maintenance time; utilizes the faster startup times of Python for faster scaling (see section 6.2.4); and broadens the scope of what the server on which CSN runs is capable of doing.

### 7.1.1 Multiple sensors and sensor types

The original server's protocols were all designed around submission of seismic information from seismic sensors, where each host client had exactly one sensor. SAF supports those types of messages, but expands by enabling clients to have multiple sensors and to identify the types of sensors that are attached to them. More generalized data streams can be submitted by those sensors to permit non-seismic data to be entered into the network. Integration with a multi-sensor box, the Home Hazard Station, has been tested, and such devices are part of a hopeful future state for a larger situation awareness network of which CSN is a component.

### 7.1.2 Multi-tenancy

The server is capable of isolating data using App Engine's Namespace API (see section 5.2.3.5). This makes it possible to:

- Fragment the network in logical ways — such as dividing CSN geographically to simplify association algorithms.
- Compartmentalize data for organizations that have high privacy concerns.
- Construct isolated test or simulation namespaces that operate identically to production namespaces but without impacting or altering production data or operation.

In one use case, we can now isolate the clients, readings, maps, and configuration of our sites in Gandhinagar, India from the operations in Southern California, while still making it possible to fuse the readings if desired.

### 7.1.3 Enhanced Security

The Community Seismic Network allows any volunteer to register with the network and provide sensor readings. This, however, leads to complex issues around security and how to avoid malicious event collections being transmitted to the fusion center. SAF now requires clients to provide a hash of the current message id and the client secret, which is provided to the client only on registration. This mechanism provides a low-bandwidth low-CPU mechanism for verifying the origin of messages, preventing an attacker from using a list of known network clients to transmit bad data to the fusion center.

### 7.1.4 User Privacy

The previous server allowed any user to view or manipulate information related to any sensor that the user knew the id or name of. In part, this was a function of a desire for user privacy; since we intended from the start to respect volunteer privacy as much as possible, we have never wanted

to compel volunteer registration. SAF respects this directive but still restricts access to sensors by providing a two-pronged authentication approach.

On the one hand, a conventional, generic authentication framework based on the SimpleAuth [73] library for OAuth is available that allows users to register with the network if they so desire. However, a second authentication mechanism relies on the security mechanism discussed previously in section 7.1.3. Given that we can now trust the origin of client messages, a client can request a URL from the SAF server that can be used to modify the client and sensor details. In this way, users can still manage their sensors anonymously by obtaining an access token from the computer to which their sensor is connected. This token is also the means by which a user proves that they own or have access to a given client. If a user is logged in when using the token for access, they can choose to associate the client with their username to make it possible for them to access and change their client information at any time.

The privacy system is also more flexible. All client data has been considered private by default, but, under the old system, there was no way to opt-in to sharing information for a particular client with no privacy concerns, such as many of the public clients around Caltech. Users can now elect to make their client information public; this allows the SAF framework to display basic information about the client to any user that requests it, and also makes the sensor readings from that client available. Given that many of the sensors in the CSN network are hosted in community areas where privacy is not a significant concern, this ability to opt-in to making information public makes it possible for non-CSN participants to view network data to become more interested in the project.

The software adapts to the authentication level of the user. Figure 7.1 shows the client editor viewing a public client, accessible at the SAF Client Editor [74] by entering client id 3001. The first two tabs, “Basic Data” and “Sensors”, are visible to any user that tries to access the sensor. Authenticated users with permission to access the sensor or users in possession of an authentication token can see the third tab, “Edit Client”, which enables manipulation of client data. The fourth tab, “Admin Data”, is only available to network administrators.

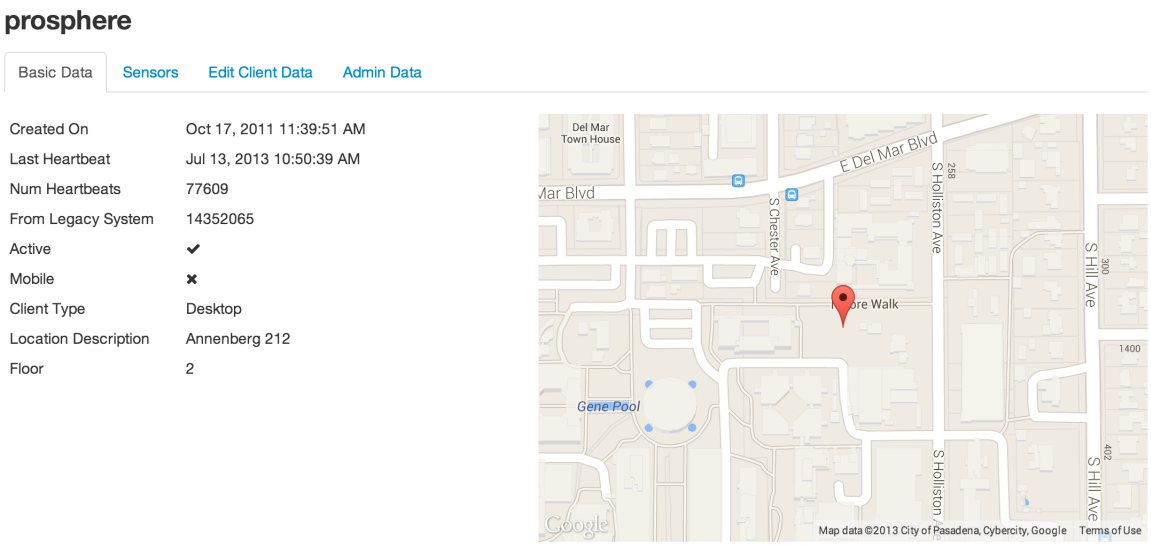



Figure 7.1: A screenshot of the SAF client editor being used by an administrator to view public client data.

### 7.1.5 Third Party Data Usage

SAF provides JSON endpoints for common client software actions, such as receiving the latest event information sent to the system. This means that other software applications can construct applications using data from SAF. In addition, using the authentication mechanisms noted previously, requests to retrieve or change sensitive data, such as sensor metadata, can be made, allowing entirely third-party interfaces to be developed.

The provided tools include a simple JSON client interface that makes it possible to get client information and sensor readings or tap into the stream of event data used to generate the map of ongoing sensor events. The client program uses the same JSON endpoints used by the sensor map, and could be used to provide other visualizations of ongoing network activity or to try out alternative fusion algorithms relying only on the event stream.

A simple retrieval of ongoing events would look like this:

```
import json_client
client = json_client.Client(server='http://csn-server.appspot.com',
                            namespace='csn')
event_id = client.get_event_id()['event_id']
events = client.get_events(event_id)['events']
for event in events[:10]:
```

```
    print ('{client_id} sent {readings} at {date} '
           'with rate {event_count}/{time_window}').format(**event)
```

```
2633006 sent [0.00299935, 0.0, 0.0] at 2013-07-26T18:38:05.308 with rate 7/600
2326012 sent [0.0, 0.00315111, 0.0] at 2013-07-26T18:38:04.968 with rate 36/600
2517020 sent [0.0, 0.00657836, 0.0] at 2013-07-26T18:38:09.928 with rate 3/600
2339025 sent [0.0, 0.0, 0.00073715] at 2013-07-26T18:38:06.898 with rate 73/600
2497011 sent [0.00430092, 0.0, 0.0] at 2013-07-26T18:38:17.568 with rate 42/600
2426018 sent [0.0, 0.0027518, 0.0] at 2013-07-26T18:37:59.269 with rate 49/600
2188018 sent [0.0086085, 0.0, 0.0] at 2013-07-26T18:38:15.349 with rate 9/600
2633004 sent [0.0, 0.0, 0.0148647] at 2013-07-26T18:38:11.878 with rate 25/600
2633004 sent [0.0100959, 0.0, 0.0] at 2013-07-26T18:38:11.648 with rate 24/600
2379014 sent [0.0099794, 0.0, 0.0] at 2013-07-26T18:38:14.478 with rate 10/600
```

### 7.1.6 Reference Client

SAF includes a fully functional reference client, which is used to connect the CISN network sensors to SAF. This code can be reused to connect new sensors to the system, or simply used as a template for how to construct a client for a situation awareness application.

### 7.1.7 Event Simulation Platform

SAF provides offline mechanisms to run collections of events through the same code that is exercised when clients are submitting events to the network. This makes inspection of network behavior during events simpler, and aids in the development of improved algorithms. The event viewer is tied into this platform and visualizes the output of event detection algorithms.

## 7.2 Further work

### 7.2.1 More deployments

To fully test all of the algorithms incorporated in the Situation Awareness Framework, larger, denser deployments spread over large geographic regions are necessary. The results for the network so far are promising, but include a density at a fraction of the hopeful eventual state for the overall system.

### 7.2.2 Sensor weighting

Much work can be done, particularly with machine learning, to develop models of how sensors behave over time. Work has been done in this area with mobile devices [19], but has not explored the potential with fixed sensors in particular or in a mixed sensor environment in general. It remains to be seen how much about a sensor's installation parameters and response fidelity can be discerned by examining the data it returns.

In addition to learning information about individual sensors, as more types of sensors, and, in particular, more models of sensors, enter the network, it is known that different sensor models have different responses to the same event. Incorporating this information in some fashion is likely to be a fruitful avenue for further exploration.

Finally, with respect to the security issues mentioned in section 7.1.3, sensors should be weighted based on their tenure in the network and whether or not they seem to be transmitting data in line with expectations of a sensor of that class. This helps prevent sensors that have newly joined the network from overwhelming network algorithms with bad data.

### 7.2.3 Network fusion algorithms

Network aware algorithms that capitalize on known factors of member networks. Much research has gone into the response of CISN during earthquakes, for example; algorithms that take advantage of that work are more likely to reach the performance frontier.

### 7.2.4 Different network topologies

For reasons of security and privacy, the CSN has always guaranteed that client software does not accept inbound connections. Instead, all communication from CSN clients involves outbound connections to the cloud fusion center. It would be possible to have CSN-controlled clients, or clients that opt-in to participation, to serve as redundant local managers that can coordinate sensors from an immediate geographic area.

There are advantages to the network if locally managed sensors can arrive at a reasonable es-

timate of the environment during a network partition. This type of locally concentrated analysis is particularly valuable, and more feasible, when viewed in respect to a single building where a particular client in that building oversees and manages the other building sensors.

In addition, local managers can be used to further reduce load at the cloud fusion center in situations where the local managers are trusted and any induced latency by aggregation is less important than the load reduction made possible by the new topology.

It is also the case that some algorithms may benefit from all of the readings within a given building being aggregated rather than receiving many picks from the same location; those picks are correlated in a way that makes their aggregate contribution less valuable than the same number of picks arriving from different buildings.

### 7.2.5 Parameter tuning

Different parties have different sensitivities to false positives and false negatives. Parameters such as the picking threshold outlined in section 6.1.1.2 dramatically effect the detection properties of the system as a whole, and also the cost to operate that system. Finding the right balance of parameters to satisfy all parties is difficult, but one mechanism for finding that balance is to provide a multi-tier association mechanism. For instance, the lowest tier might accumulate all sensor events, while the next tier would discard events below a secondary threshold.

### 7.2.6 Delayed analysis of low-priority information

Offline analysis would benefit from more information from sensors, particularly in the absence of full waveforms. One option would be to record all local events that cross a low k-sigma threshold, such as 1.25 or lower, but only transmit local events that cross a higher k-sigma threshold, such as 1.75. The lower threshold events would then be queued for transmission at heartbeat intervals and could be analyzed later by the system. These lower threshold events are not likely to correspond to a high priority event, so processing them in real time is not necessary; however, the events may still contain correlated information of use in detecting very small events, so transmitting the information

is still useful for analysis purposes. There may be other metrics that are also worth recording at regular intervals, but which do not require immediate transmission to the network.

# Appendix A

# Code Samples

## A.1 Code Availability

The Situation Awareness Framework repository is available at [75].

The Geocell Library repository, which contains the Java implementation of the Geocell algorithms, is available at [76]. The Python and Javascript implementations are components of the SAF repository.

## A.2 Event Detection

### A.2.1 Java detection algorithm from original CSN server

```
static void pasadenaPicks(Pick pick, ClientWithId client)
        throws InvalidData, DatastoreProblem {
    if (!isPickOfInterest(pick, client)) {
        return;
    }

    String geostring = pick.getGeostring();
    Date pickDate = pick.getReceiptDate();
    Long clientId = client.getId();

    Set<ActiveClients> activeClients =
            ActiveClientsCache.getActiveClients(SOCAL);
    int numActive = 0;
    int numPicks = 0;
    int numPicksAcc = 0;
    double logGamma = LOG_BETA;
    double logGammaAcc = logGamma;

```

```
    for (ActiveClients ac : activeClients) {
        numActive += ac.getCount(SensorType.STATIC);
        if (ac.getGeostring().equals(geostring)) {
            if (!ac.getSensorStats().containsKey(clientId)) {
                ac.getSensorStats().put(clientId, new SensorStats());
            }
        }
        for (Entry<Long, SensorStats> e :
                ac.getSensorStats().entrySet()) {
            SensorStats stats = e.getValue();
            if (e.getKey().equals(client.getId())) {
                stats.updateWithPick(
                        pick.getReceiptDate(),
                        pick.getMagnitude() > ACC_THRESHOLD ? true : false);
                ac.setModified(true);
            } else {
                if (stats.updateWithoutPick()) {
                    ac.setModified(true);
                }
            }

            double lambda = stats.getLambda();
            double lambdaAcc = stats.getLambdaAcc();
            double q = 1 - Math.exp(-lambda * (double)SECONDS_PER_BIN);
            double qAcc = 1 - Math.exp(-lambdaAcc * (double)SECONDS_PER_BIN);
            double sAcc = (1 - qAcc) / (1 - P);

            boolean highAcc = false;
            switch (stats.pickThisBin()) {
            case NONE:
                double s = (1 - q) / (1 - P);
                logGamma += Math.log(s);
                logGammaAcc += Math.log(sAcc);
                break;
            case HIGH_ACC:
                highAcc = true;
                numPicksAcc++;
                double rAcc = qAcc / P;
                logGammaAcc += Math.log(rAcc);
            case LOW_ACC:
                numPicks++;
                double r = q / P;
                logGamma += Math.log(r);
                if (!highAcc) {
                    logGammaAcc += Math.log(sAcc);
                }
                break;
            default:
                break;
            }
        }
    }
    double gamma = Math.exp(logGamma);
    double prob = 1 / (1 + gamma);
```

```
73
74      double gammaAcc = Math.exp(logGammaAcc);
75      double probAcc = 1 / (1 + gammaAcc);
76
77      log.info(numActive + " active; L1/" + prob + " L2/" + probAcc);
78
79      if (prob > PROBABILITY_THRESHOLD && numActive > MINIMUM_ACTIVE_CLIENTS) {
80          alert(numActive, numPicks, numPicksAcc, prob, probAcc);
81      }
82
83      updateStats(
84              activeClients, geostring, pickDate, clientId,
85              pick.getMagnitude() > ACC_THRESHOLD ? true : false);
86  }
```

### A.2.2 Python detection algorithm from SAF

#### A.2.2.1 Cell level probability

```
1   def calculate_probability(active_stats_list, event_date, start_time=None):
2       """Calculates probability that an individual cell picks."""
3       if start_time:
4           # Find events in larger time span before current event.
5           time_delta = (event_date - start_time).total_seconds()
6       else:
7           # Find events in :py:const:`SECONDS_PER_BIN` time before current event.
8           start_time = event_date - SECONDS_PER_BIN_DELTA
9           time_delta = SECONDS_PER_BIN
10
11      total_sensors = 0
12      total_non_picking = 0
13      sum_log_a = 0
14
15      # TODO: This ignores the impact of multiple contributions from one client.
16      # Need to check sensor stats keys if concerned about consolidating multiple
17      # sensor contributions from a single client.
18      for active_stats in active_stats_list:
19          total_sensors += active_stats.static_count
20          num_static = 0
21          for sensor_stats in active_stats.sensor_stats.viewvalues():
22              if sensor_stats.mobile:
23                  continue
24              num_static += 1
25              num_events = sensor_stats.events_in_window(start_time, event_date)
26              prob_over_window = prob_ratio_from_single_sensor(
27                  sensor_stats.rate, time_delta, num_events)
28              sum_log_a += math.log(prob_over_window)
29
30          #if num_static < active_stats.static_count:
31              ## Provide a base rate for sensors with no picks.
32              #prob_over_window = prob_ratio_from_single_sensor(
```

```
                #EPSILON_RATE, time_delta, 0)
            #no_picks = active_stats.static_count - num_static
            #total_non_picking += no_picks
            #sum_log_a += no_picks * math.log(prob_over_window)

    try:
        c = math.exp(sum_log_a)
    except OverflowError:
        result = 1.0
    else:
        result = P_QUAKE * c / (P_QUAKE * c + 1)
    return result, total_sensors, total_sensors - total_non_picking
```

#### A.2.2.2 Single sensor probability

`prob_no_quake` is the mirror image of `PROB_QUAKE`. The index is the same as the index of `PROB_QUAKE`. `prob_no_quake[n]` is the probability of `n` picks in `time_delta` seconds given that there is no quake, for `n` running over `range( MAX_EVENTS_IN_PERIOD )`. `prob_no_quake[n]` is the probability of `n` *or more* picks in `time_delta` seconds given that there is no quake.

Assert: sum over all `n` of `prob_no_quake[n]` is `1.0`.

Assume that quiescent picks occur at random times; so picks occur in a Poisson process.

The probability, `prob[n]`, of `n` picks in time `t`, in a Poisson process with rate $\lambda$ is:

$$\text{prob}[n] = \frac{\rho^n \cdot e^{-\rho}}{n!}$$

where $\rho = \lambda \cdot t$

Use `rate` and `time_delta` for $\lambda$ and $t$.

Calculate `prob[n]` recursively using:

$$\begin{aligned} \text{prob}[0] &= e^{-\rho} \\ \text{prob}[n] &= \frac{\text{prob}[n-1] \cdot \rho}{n} \end{aligned}$$

```
def prob_ratio_from_single_sensor(rate, time_delta, num_events):
    r"""
```

```
    The ratio of the probability of a quake to the probability of no quake
    given data from a single sensor.

    Parameters
    ----------
    rate : float
        Picks per second observed for this sensor in the past time window.
    time_delta : float
        The length of the time interval in seconds for which picks have been
        counted.
    num_events : int
        The number of picks observed in the last time_delta seconds.

    Returns
    -------
    probability : float
        Probability of a quake divided by probability of no quake.

    """
    rho = rate * time_delta

    # Calculating Poisson probabilities iteratively according to the formula
    # above.
    prob_no_quake = range(MAX_EVENTS_IN_PERIOD + 1)

    # Note: `prob_no_quake[n]` is the probability of a quake given `n` or more
    # picks in `time_delta` time.
    prob_no_quake[0] = math.exp(-rho)
    sum_prob_no_quake = prob_no_quake[0]

    for n in range(MAX_EVENTS_IN_PERIOD - 1):
        prob_no_quake[n + 1] = prob_no_quake[n] * rho / (n + 1)
        sum_prob_no_quake += prob_no_quake[n + 1]

    assert sum_prob_no_quake <= 1.0
    prob_no_quake[MAX_EVENTS_IN_PERIOD] = 1.0 - sum_prob_no_quake
    num_events = min(num_events, MAX_EVENTS_IN_PERIOD)
    return PROB_QUAKE[num_events] / prob_no_quake[num_events]
```

#### A.2.2.3 Client side picking algorithm

```
def find_events(sac_data,
                time_lta=TIME_LTA, time_sta=TIME_STA, time_gap=TIME_GAP):
    if sac_data.get('kcmpnm' == 'HNZ'):
        magnitudes = [1.0 - abs(mag) for mag in sac_data.data_points]
    else:
        magnitudes = [abs(mag) for mag in sac_data.data_points]
    delta = sac_data.get('delta')
    num_samples_lta = int(time_lta / delta)
    num_samples_sta = int(time_sta / delta)
    gap_for_lta = int(time_gap / delta)
    max_mag = None
```

```
    min_mag = None
    startup_samples = gap_for_lta + num_samples_lta
    lta_window = collections.deque(maxlen=num_samples_lta)
    lta_window_sqrd = collections.deque(maxlen=num_samples_lta)
    sta_window = collections.deque(maxlen=num_samples_sta)
    picks = []
    ksigma_history = []
    lta_history = []
    picking = False
    for index, magnitude in enumerate(magnitudes):
        # Compute rolling sums over num_samples_lta data magnitudes.
        if index >= gap_for_lta:
            gap_index = index - gap_for_lta
            lta_window.append(magnitudes[gap_index])
            lta_window_sqrd.append(magnitudes[gap_index] ** 2)
        # Can only compute lta after startup_samples have elapsed.
        if index >= startup_samples:
            max_mag = max(max_mag, magnitude) if max_mag else magnitude
            min_mag = min(min_mag, magnitude) if min_mag else magnitude
            lta = sum(lta_window) / num_samples_lta
            lta_sqrd_data = sum(lta_window_sqrd) / num_samples_lta
            if lta_sqrd_data <= lta ** 2:
                stdev = EPSILON
            else:
                stdev = math.sqrt(lta_sqrd_data - (lta ** 2))
        # Set default values until startup_samples have elapsed.
        else:
            stdev = EPSILON
            lta = 0.0
        lta_history.append(lta)
        # Compute rolling sum over num_samples_sta data magnitudes.
        sta_window.append(abs(magnitude - lta))
        # 100 == hack. ksigma very high before startup_samples + some value.
        if index >= startup_samples + 100:
            sta = sum(sta_window) / num_samples_sta
            ksigma = sta / stdev
            # No new pick can be generated until PICK_INTERVAL time elapses.
            if not picking:
                # New pick.
                if ksigma > PICK_THRESHOLD:
                    pick_start_index = index
                    picking = True
                    max_pick_mag = magnitude
            # Only check for max acceleration if ksigma exceeds threshold.
            elif ksigma > PICK_THRESHOLD:
                max_pick_mag = max(max_pick_mag, magnitude)
            # Record pick and allow a new pick to be generated.
            if picking and (index - pick_start_index) * delta > PICK_INTERVAL:
                picking = False
                picks.append((pick_start_index, index, max_pick_mag))
        else:
            ksigma = 0.0
        ksigma_history.append(ksigma)
    if picking:
```

```
        picks.append((pick_start_index, index, max_pick_mag))
    return (min_mag, max_mag, ksigma_history, picks)
```

### A.2.3 SAF hypocenter fitting algorithm

```
@ndb.tasklet
def check_geostr(pick_time, geostr, active_geocells_future):
    cell = geostring.to_geocell(geostr)
    bounds = geo.Bounds(*geocell.get_bounds(cell))
    center = bounds.get_center()
    active_geocells = yield active_geocells_future

    nearby_bounds = geo.bounds_from_pt_and_distance(
        center, ACTIVE_CELL_RADIUS)
    nearby_active = earthquake_model.get_nearby_active(
        geostr, active_geocells, region_bounds=nearby_bounds)
    logging.info('Fitting based on %s active cells near cell %s.',
                 len(nearby_active), geostr)

    # Compress {geostr: ActiveClients} map into list of SensorStats
    # objects for speed purposes.
    nearby_stats = []
    nearby_cells = yield models.active_clients.get_multi_by_geostring_async(
        nearby_active)
    non_picking = []
    for active_stats in nearby_cells.viewvalues():
        if active_stats:
            num_static = 0
            for sensor_stats in active_stats.sensor_stats.viewvalues():
                if not sensor_stats.mobile:
                    num_static += 1
                    nearby_stats.append(sensor_stats)
            stats_non_picking = active_stats.static_count - num_static
            if stats_non_picking:
                active_cell = geostring.to_geocell(active_stats.key.id())
                active_bounds = geo.Bounds(*geocell.get_bounds(active_cell))
                active_center = active_bounds.get_center()
                non_picking.append((active_center, stats_non_picking))

    region_bounds = geo.bounds_from_pt_and_distance(
        center, LARGE_REGION_RADIUS)
    prob, (lat, lon), depth, time = iterate_over_bounds(
        1, pick_time, nearby_stats, non_picking, region_bounds)
    logging.info('Pass 1: best prob=%s at (%s, %s) with d=%s '
                 'and t=%s.', prob, lat, lon, depth, time)
    highest_prob_bounds = geo.bounds_from_pt_and_distance(
        geo.Point(lat, lon), SMALL_REGION_RADIUS)
    prob, (lat, lon), depth, time = iterate_over_bounds(
        2, pick_time, nearby_stats, non_picking, highest_prob_bounds)
    logging.info('Pass 2: best prob=%s at (%s, %s) with d=%s '
                 'and t=%s.', prob, lat, lon, depth, time)

```

```
48
49  def iterate_over_bounds(_, event_time, nearby_stats, non_picking,
50                          region_bounds):
51      lats = numpy.linspace(region_bounds.sw.lat, region_bounds.nw.lat,
52                            REGION_GRANULARITY, endpoint=True)
53      lons = numpy.linspace(region_bounds.nw.lon, region_bounds.ne.lon,
54                            REGION_GRANULARITY, endpoint=True)
55      points = numpy.transpose(
56          [numpy.tile(lats, len(lons)), numpy.repeat(lons, len(lats))])
57      time = event_time - TIME_WINDOW
58      best_prob = 0
59      best = None
60      while time <= event_time:
61          for depth in DEPTHS_TO_CHECK:
62              depth_table = taup.ARRIVAL_S[taup.depth_model_index(depth)]
63              for point in points:
64                  prob, active, picking = check_for_hypocenter(
65                      time, point, depth_table, nearby_stats, non_picking)
66                  if prob > best_prob:
67                      best = (prob, point, depth, time, active, picking)
68                      best_prob = prob
69          time += TIME_STEP
70      return best
71
72
73  def check_for_hypocenter(time, point, depth_table, nearby_stats, non_picking):
74      """
75      This is a derivative of earthquake_model.calculate_probability.
76
77      Notes
78      -----
79      The main difference is that different time windows are used per sensor
80      depending on the expected wave arrival time. An argument to
81      calculate_probablility that is used to calculate the time window could
82      unify the two functions, but it does not seem worthwhile at this juncture.
83
84      Important updates to either function should probably be mirrored.
85
86      """
87      center = geo.Point(*point)
88      sum_log_a = 0
89      num_active = 0
90      num_picking = 0
91      for sensor_stats in nearby_stats:
92          if center.distance_to(sensor_stats.location) > FIT_REGION_SIZE:
93              continue
94          num_active += 1
95          distance = center.approx_distance(sensor_stats.location)
96          arrival_time = time + datetime.timedelta(
97              seconds=taup.distance_arrival(depth_table, distance))
98          window_end = arrival_time + earthquake_model.SECONDS_PER_BIN_DELTA
99          rate_over_window = (
100             sensor_stats.rate * earthquake_model.SECONDS_PER_BIN)
101         num_picks = sensor_stats.events_in_window(
```

```
            arrival_time, window_end)
        if num_picks > 0:
            num_picking += 1
        prob_over_window = earthquake_model.prob_ratio_from_single_sensor(
            rate_over_window, earthquake_model.SECONDS_PER_BIN, num_picks)
        sum_log_a += math.log(prob_over_window)

    for location, count in non_picking:
        if center.distance_to(location) <= FIT_REGION_SIZE:
            # Provide a base rate for sensors with no picks.
            prob_over_window = earthquake_model.prob_ratio_from_single_sensor(
                earthquake_model.EPSILON_RATE,
                earthquake_model.SECONDS_PER_BIN, 0)
            sum_log_a += count * math.log(prob_over_window)
            num_active += count

    try:
        c = math.exp(sum_log_a)
    except OverflowError:
        prob = 1.0
    else:
        prob = ((earthquake_model.P_QUAKE * c) /
                (earthquake_model.P_QUAKE * c + 1))
    return prob, num_active, num_picking
```

## A.3 Geocell Creation

```
def create(point, resolution):
    """
    Create a new geocell and return it and the bounds that describe it.

    """
    return increase_resolution(0, point, WORLD_BOUNDS, resolution)


def increase_resolution(geocell, point, bounds, new_resolution):
    """
    Increase the resolution of a geocell and return the new geocell and bounds.

    """
    point_lat, point_lon = point
    ((sw_lat, sw_lon), (ne_lat, ne_lon)) = bounds
    odd_or_even = 0 if not geocell else get_resolution(geocell) % 2
    for i in range(new_resolution):
        geocell = geocell << 1
        if i % 2 == odd_or_even:
            lon_midpoint = (sw_lon + ne_lon) / 2
            if point_lon < lon_midpoint:
                ne_lon = lon_midpoint
            else:
                sw_lon = lon_midpoint
```

```
                geocell += 1
        else:
            lat_midpoint = (sw_lat + ne_lat) / 2
            if point_lat < lat_midpoint:
                ne_lat = lat_midpoint
            else:
                sw_lat = lat_midpoint
                geocell += 1
    geocell = (geocell << _shift_distance(new_resolution)) + new_resolution
    return geocell, ((sw_lat, sw_lon), (ne_lat, ne_lon))
```

## A.4 Instance Lifetime

```
// Instance startup time mapper.
function () {
    var result = {
        startup: false,
        conflict: false,
        conflicts: new Array(),
        first_seen: this.timestamp,
        last_seen: this.timestamp,
        requests: {},
        total_requests: 1
    };
    if (this.loading_request) {
        result.startup = {
            'timestamp': this.timestamp,
            'path': this.path,
            'ms': this.ms,
            'pending_ms': this.pending_ms
        };
    } else {
        result.requests[this.path] = {
            count: 1,
            ms: this.ms,
            pending_ms: this.pending_ms
        };
    }
    emit(this.instance, result);
}

// Instance startup time reducer.
function (key, values) {
    var result = {
        startup: false,
        conflict: false,
        conflicts: new Array(),
        first_seen: new Date(3000, 0, 0),
        last_seen: new Date(0, 0, 0),
        requests: {},
        total_requests: 0
```

```
    };
    values.forEach(function (value) {
        if (value.startup !== false) {
            if (result.startup !== false) {
                result.conflict = true;
                result.conflicts = result.conflicts.concat(value.startup);
            } else {
                result.startup = value.startup;
            }
        }
        for (var key in value.requests) {
            if (key in result.requests) {
                result.requests[key].count += value.requests[key].count;
                result.requests[key].ms += value.requests[key].ms;
                result.requests[key].pending_ms += result.requests[key].pending_ms;
            } else {
                result.requests[key] = value.requests[key];
            }
        }
        result.conflict = result.conflict || value.conflict;
        result.conflicts = result.conflicts.concat(value.conflicts);
        result.total_requests += value.total_requests;
        if (value.first_seen < result.first_seen) {
            result.first_seen = value.first_seen;
        }
        if (value.last_seen > result.last_seen) {
            result.last_seen = value.last_seen;
        }
    });
    return result;
}
```

# Bibliography

[1] J. Mou, "Situation awareness application," Master's thesis, California Institute of Technology, 2013. [Online]. Available: http://thesis.library.caltech.edu/7907/

[2] M. Faulkner. csn-android. [Online]. Available: https://play.google.com/store/apps/details?id=edu.caltech.android

[3] R. W. Clayton, D. Hollis, and M. Barba. Earthquake motions recorded on a 5000-node seismic network in long beach, california. [Online]. Available: http://www.gps.caltech.edu/~clay/EQmovies/EQmovies.html

[4] A. Liu, M. Olson, J. J. Bunn, and K. M. Chandy, "Towards a discipline of geospatial distributed event based systems," in *DEBS '12: Proceedings of the Sixth ACM International Conference on Distributed Event-Based Systems*. ACM, 2012.

[5] M. Olson. Geocell map application. [Online]. Available: http://geocell-map.appspot.com/

[6] A. Levi, "From spark plug to drive train: Life of an app engine request," in *Google I/O 2009*, 5 2009. [Online]. Available: https://www.google.com/events/io/2009/sessions/FromSparkPlugToDriveTrain.html

[7] R. W. Clayton, T. Heaton, M. Chandy, A. Krause, M. Kohler, J. Bunn, R. Guy, M. Olson, M. Faulkner, M. Cheng, L. Strand, R. Chandy, D. Obenshain, A. Liu, and M. Aivazis, "Community seismic network," *Annals of Geophysics*, vol. 54, no. 6, 1 2012.

[8] N. N. Rao, "Sensor platforms for deployment in rural areas," California Institute of Technology, Summer Undergraduate Research Fellow Report, 9 2012.

[9] Google app engine. [Online]. Available: http://code.google.com/appengine/

[10] M. W. Alfred Fuller, "More 9s please: Under the covers of the high replication datastore," in *Google I/O 2011*, 5 2011. [Online]. Available: https://www.google.com/events/io/2011/sessions/more-9s-please-under-the-covers-of-the-high-replication-datastore.html

[11] Phidgetspatial 3/3/3. [Online]. Available: http://www.phidgets.com/products.php?product_id=1056_0

[12] S. Fang, "The home hazard weather station: Home sensor platforms for environmental hazard detection and response," California Institute of Technology, Summer Undergraduate Research Fellow Report, 9 2012.

[13] Raspberry pi. [Online]. Available: http://www.raspberrypi.org/

[14] United states geological survey. [Online]. Available: http://www.usgs.gov/

[15] Japan meteorological agency. [Online]. Available: http://www.jma.go.jp/jma/en/Emergency_Warning/ew_index.html

[16] Commerical mobile telephone alerts (cmas). [Online]. Available: http://transition.fcc.gov/pshs/services/cmas.html

[17] D. L. Smith, "Did you feel it?" *Engineering and Science*, vol. 62, no. 4, pp. 34–38, 1999.

[18] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[19] M. Faulkner, M. Olson, R. Chandy, J. Krause, K. M. Chandy, and A. Krause, "The next big one: Detecting earthquakes and other rare events from community-based sensors," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 2011.

[20] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34 – 41, April 2005.

[21] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," *International Joint Conference on INC, IMS and IDC*, Jan 2009.

[22] (2011, 2) Amazon ec2. [Online]. Available: http://aws.amazon.com/ec2/

[23] (2011, 2) Rackspace cloud servers. [Online]. Available: http://www.rackspacecloud.com/cloud_hosting_products/servers

[24] (2011, 3) Heroku. [Online]. Available: http://www.heroku.com/

[25] (2011, 3) Amazon web services elastic beanstalk. [Online]. Available: http://aws.amazon.com/elasticbeanstalk/

[26] (2011, 3) Gmail. [Online]. Available: http://mail.google.com/

[27] (2011, 3) Photoshop online. [Online]. Available: http://www.photoshop.com/tools?wf=editor

[28] (2011, 3) Zoho office suite. [Online]. Available: http://www.zoho.com/

[29] (2011, 2) Amazon ec2 faqs. [Online]. Available: http://aws.amazon.com/

[30] (2013, 7) Eucalyptus: Open source, aws compatible, private clouds. [Online]. Available: http://www.eucalyptus.com/

[31] (2013, 7) Appscale: The open source implementation of google app engine. [Online]. Available: http://www.appscale.com/whyappscale

[32] Y. Jie and Q. Jie. . . , "A profile-based approach to just-in-time scalability for cloud applications," *Cloud Computing*, Jan 2009.

[33] L. Vaquero and L. Rodero-Merino. . . , "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer . . .*, Jan 2011.

[34] D. Durkee, "Why cloud computing will never be free," *Queue*, vol. 8, no. 4, pp. 20:20–20:29, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1755884.1772130

[35] (2011, 2) Amazon ec2 faqs. [Online]. Available: http://aws.amazon.com/ec2/faqs/

[36] M. Olson. Situation awareness framework documentation. [Online]. Available: http://csn-server.appspot.com/docs/index.html

[37] S. S. Roman Nurik. (2011, 3) Geospatial queries with google app engine using geomodel. [Online]. Available: http://code.google.com/apis/maps/articles/geospatial.html

[38] geohash.org. (2011, 3) Geohash. [Online]. Available: http://en.wikipedia.org/wiki/Geohash

[39] *DMATM 8358.2 The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS)*, Defense Mapping Agency, Fairfax, VA, 9 1989.

[40] *DMATM 8358.1 Datums, Ellipsoids, Grids, and Grid Reference Systems*, Defense Mapping Agency, Fairfax, VA, 9 1990.

[41] Locating a position using utm coordinates. [Online]. Available: http://en.wikipedia.org/wiki/Universal_Transverse_Mercator

[42] L. Nault, "Nga introduces global area reference system," *PathFinder*, 11 2006.

[43] (2011, 3) Georef. [Online]. Available: http://en.wikipedia.org/wiki/Georef

[44] N. G. P. Inc. (2011, 3) The natural area coding system. [Online]. Available: http://www.nacgeo.com/nacsite/documents/nac.asp

[45] T. Hanks and H. Kanamori, "A moment magnitude scale," *Journal of Geophysical Research*, 1979.

[46] (2013, 7) Lock in, what lock in? [Online]. Available: https://plus.google.com/u/0/110401818717224273095/posts/Uoj3pmhbCkH

[47] (2013, 7) Appscale code repository. [Online]. Available: https://github.com/AppScale/appscale

[48] (2013, 7) Memcached. [Online]. Available: http://memcached.org/

[49] (2013, 7) Amazon elasticache. [Online]. Available: http://aws.amazon.com/elasticache/

[50] K. Gibbs. (2011, 5) Announcing the high replication datastore for app engine. [Online]. Available: http://googleappengine.blogspot.com/2011/01/announcing-high-replication-datastore.html

[51] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267308.1267323

[52] (2013, 7) Apache cassandra. [Online]. Available: http://cassandra.apache.org/

[53] (2013, 7) Apache hbase. [Online]. Available: http://hbase.apache.org/

[54] (2013, 7) Hypertable. [Online]. Available: http://hypertable.com/

[55] E. Gonina and J. Chong, "Task queue implementation pattern," *UC Berkeley ParLab*, 2008.

[56] (2013, 7) Celery: Distributed task queue. [Online]. Available: http://celeryproject.org/

[57] (2013, 7) Beanstalk: a simple, fast work queue. [Online]. Available: http://kr.github.io/beanstalkd/

[58] (2013, 7) Amazon simple queue service. [Online]. Available: http://aws.amazon.com/sqs/

[59] B. Slatkin, "Building high-throughput data pipelines with google app engine," in *Google I/O 2010*, 5 2010. [Online]. Available: https://www.google.com/events/io/2010/sessions/high-throughput-data-pipelines-appengine.html

[60] (2011, 5) App engine 1.5.0 release. [Online]. Available: http://googleappengine.blogspot.com/2011/05/app-engine-150-release.html

[61] (2013, 6) Google app engine python sdk release notes. [Online]. Available: http://code.google.com/p/googleappengine/wiki/SdkReleaseNotes

[62] (2011, 2) The webapp framework. [Online]. Available: http://code.google.com/appengine/docs/python/tools/webapp/

[63] (2012, 7) User-facing requests should never be locked to cold instance starts. [Online]. Available: https://code.google.com/p/googleappengine/issues/detail?id=7865

[64] M. Olson and K. M. Chandy, "Performance issues in cloud computing for cyber-physical applications," in *CLOUD '11: Proceedings of the Fourth IEEE International Conference on Cloud Computing*. IEEE, 2011.

[65] (2011, 2) Google app engine downtime notify. [Online]. Available: http://groups.google.com/group/google-appengine-downtime-notify

[66] J. Gregorio. (2008, 10) Sharding counters. [Online]. Available: https://developers.google.com/appengine/articles/sharding_counters

[67] (2013, 6) Google app engine transaction documentation. [Online]. Available: https://developers.google.com/appengine/docs/python/datastore/transactions

[68] (2013, 7) Google app engine. [Online]. Available: http://googlecloudplatform.blogspot.com/2013/07/google-app-engine-182-released.html

[69] (2010, 2) Scalability means flexibility. [Online]. Available: http://googleappengine.blogspot.com/2010/02/scalability-means-flexibility.html

[70] (2010, 18) How app engine served the humble indie bundle. [Online]. Available: http://googleappengine.blogspot.com/2010/06/how-app-engine-served-humble-indie.html

[71] (2013, 7) Earth hour 2009 site hosted on app engine. [Online]. Available: http://googleappengine.blogspot.com/2009/05/web2py-support-new-datastore-backend.html

[72] (2009, 3) Google developer products help whitehouse.gov connect with america. [Online]. Available: http://googlecode.blogspot.com/2009/04/google-developer-products-help.html

[73] Simple authentication for python on google app engine. [Online]. Available: https://github.com/crhym3/simpleauth/

[74] M. Olson. Situation awareness framework client editor. [Online]. Available: https://csn-server.appspot.com/#/clients/csn/

[75] ——. Situation awareness framework repository. [Online]. Available: https://bitbucket.org/molson/saf

[76] ——. Geocell-java repository. [Online]. Available: https://bitbucket.org/molson/geocell-java/