Rapid Detection of Rare Geospatial Events: Earthquake Warning Applications

Michael Olson
Caltech
molson@cs.caltech.edu

Annie Liu
Caltech
aliu@cs.caltech.edu

Matthew Faulkner
Caltech
mfaulk@caltech.edu

K. Mani Chandy
Caltech
mani@cs.caltech.edu

ABSTRACT

The paper presents theory, algorithms, measurements of experiments, and simulations for detecting rare geospatial events by analyzing streams of data from large numbers of heterogeneous sensors. The class of applications are rare events - such as events that occur at most once a month - and that have very high costs for tardy detection and for false positives. The theory is applied to an application that warns about the onset of shaking from earthquakes based on real-time data gathered from different types of sensors with varying sensitivities located at different points in a region. We present algorithms for detecting events in Cloud computing servers by exploiting the scalability of Cloud computers while working within the limits of state synchronization across different servers in the Cloud. Ordinary citizens manage sensors in the form of mobile phones and tablets as well as special-purpose stationary sensors; thus the geospatial distribution of sensors depends on population densities. The distribution of the locations of events may, however, be different from population distributions. We analyze the impact of population distributions (and hence sensor distributions as well) on the efficacy of event detection. Data from sensor measurements and from simulations of earthquakes validate the theory.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; G.3 [**Probability and Statistics**]: Experimental Design

General Terms

Algorithms, Design, Experimentation

Keywords

seismology, paas, cloud, sensor networks

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DEBS'11, July 11–15, 2011, New York, New York, USA.
Copyright 2011 ACM 978-1-4503-0423-8/11/07 ...$10.00.

1. INTRODUCTION

A Grand Challenge.

A grand challenge for distributed event-based (DEB) systems is helping communities sense and respond to global calamities such as earthquakes, tsunamis, nuclear radiation, and fires [1]. DEB systems can help save many thousands of lives by rapid, accurate detection of events and dissemination of warnings to people and systems that must respond quickly. This paper presents theory, architecture, and early experience with one grand challenge: applications that help the community respond to earthquakes. Rapid detection of near-field earthquakes also helps in early warning of close offshore tsunamis.

Community-Based Event Detection.

We describe a DEB system in which the community, as a whole, helps to sense and respond to rapidly unfolding events. All members of a community are effected by earthquakes, tsunamis, nuclear reactor meltdowns, and fires. An effective DEB system must involve all members of the community as well as agencies charged with first response. Equipping ordinary citizens with sensors and enabling them to contribute data to a system has problems [2] — the data is likely to be noisy; the same type of sensor, operated by different people, may behave in very different ways; and open systems are also more open to spoofing and attacks. We explore problems of community-based DEB systems.

Detecting Events with Phones and Low-cost Sensors.

The recent trend towards smart phones and other Internet-enabled devices offers unique possibilities for decentralized event detection. Smart phones contain a rich suite of sensors, such as GPS, accelerometers, and cameras that can gather information about a variety of geospatial phenomena. Smart phones, tablets and laptops have accelerometers that are being used by our project and others [3, 4] to obtain seismic measurements.

Cellphone coverage is increasing dramatically in all parts of the world, including regions such as Haiti that suffer from devastating earthquakes and have populations with less disposable income. Further, the cost of MEMS accelerometers and other sensors continues to drop due to their mass-market use in video game systems. We speculate that smart phones will become less expensive and will be adopted in greater numbers in developing economies in the coming decades.

The widespread use of digital communication/computing devices and decreasing costs of sensors allows for the development of dense sensor networks where the sensors are owned and operated by individuals in the community. To capitalize on the pervasiveness of these sensors, systems must be prepared to deal with high sensor densities, and consequently, large numbers of distributed event publishers. The incoming events must be managed accurately and rapidly if the information generated is to be of use in mitigating the damaging effects of disasters.

Event-Based Systems in the Cloud.

Platform-as-a-Service (PaaS) cloud computing systems can be valuable components of DEB systems. An advantage of cloud computing systems is that they are distributed so that events such as earthquakes and tsunamis do not destroy the infrastructure. Just as importantly, cloud computing systems can be accessed by any point on the globe with Internet access. Some regions that have suffered devastating earthquakes — such as Haiti, Lima Peru, and Gujarat State in India — do not have dedicated seismic networks; however, they do have access to the Internet, and as a consequence can use cloud computing systems distributed around the globe for helping to respond to calamities. PaaS systems have limitations too, and this paper describes some of these limitations and how they can be overcome.

Our work focuses on how event detection with very large numbers of publishers can be performed on a distributed detection platform with the imposed architectural limitations described in Section 3. Specifically, we explore how limits on data access and synchronization imposed in the name of scalability affect the algorithms used to perform detection of ongoing events.

Geospatial-Temporal Analysis for Event Detection.

Responding to earthquakes requires analysis of data over multiple complex geospatial and temporal scales. For example, a rupture along the San Andreas fault, on the West Coast of the U.S., can travel hundreds of kilometers, but first responders need information about building damage at the block-by-block level. We present a data structure — the "geocell" — that is well suited to the range of spatial scales. Using the geocell, we present theory and experimental evidence for detecting events over large geospatial regions over time, and shows how events are detected over time series in a distributed manner.

Designs of Event-Based Cyber Physical Systems.

Event detection systems often focus on reliable in-order delivery of events from publishers to ensure that described events are detected. In this paper, we evaluate the use of unreliable sensors as event publishers and explore the process of event detection using out of order messages with no guarantee of delivery. The event publishers all possess different reliability characteristics, owing to differences in both the type of sensor and the environmental conditions in which each sensor is installed. The sensors used in the network are described in detail in Section 6, as well as how their reliability affects detection performance.

There is a design tradeoff between the amount of data that should be communicated and where data should be stored. For example, a cellphone can send raw accelerometer data continuously to cloud computing servers; this approach is

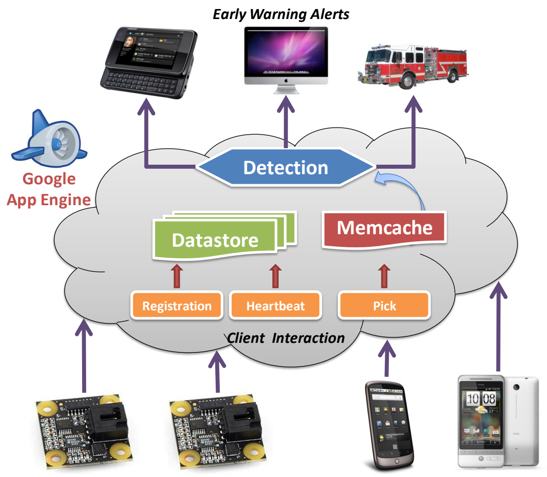


Figure 1: Overview of the CSN architecture.

not cost effective with over a million phones in a region such as Los Angeles. An alternate approach is for a phone to carry out simple event detection and send a short message when it detects an event. Seismologists refer to the detection of an event by a sensor as "picking" an event, and refer to the event data as a "pick." One of the many questions is what information should be sent to servers with each pick.

DEB systems that help respond to physical events such as earthquakes use models of the underlying physical environment — in our case, models and theories of seismological structures. The model of the physical environment is coupled with models of the cyber infrastructure to predict how a given system design will behave. In Section 7, we analyze of the combined cyber and physical layers.

2. COMMUNITY SEISMIC NETWORK

We are building an experimental seismic event detection platform that utilizes a large number of lower quality sensors rather than the small number of high quality sensors traditionally employed by organizations such as the USGS. To obtain the envisioned density of sensors, the CSN recruits volunteers in the community to host USB accelerometer devices in their homes or to contribute acceleration measurements from their existing smart phones. The goals of the Community Seismic Network (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.

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, containing only the sensor's location, event time, and a few statistics such as the peak observed acceleration. The process of pick detection is discussed in Section 5. Due to privacy constraints, decen-

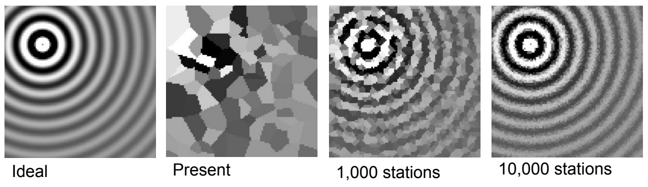


Figure 2: How sensor density affects the detection of seismic waves.

tralized algorithms with a trusted center may be preferred to distributed implementations, where phones are required to know the identity of other members of the network.

An overview of the CSN infrastructure is presented in Figure 1. 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. When an event is detected, alerts are issued to the community.

Dense Community Network.

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, determining the magnitude of shaking at points other than where sensors lie is complicated by subsurface properties. As you can see in Figure 2, a dense network makes visualizing the propagation path of an earthquake and the resulting shaking simpler. With a dense network, we propose to rapidly generate a block-by-block shakemap that can be delivered to first responders within a minute.

Second, community sensors owned by individuals working in the same building can be used to establish whether or not buildings have undergone deformations during an earthquake which cannot be visually ascertained. This type of community structural modeling will make working or living in otherwise unmonitored buildings safer.

Lastly, one of the advantages of relying on cheap sensors is that networks can quickly be deployed to recently shaken regions for data collection or regions which have heretofore been unable to deploy seismic network because of cost considerations. As the infrastructure for the network lies entirely in the cloud, sensors deployed in any country can rely on the existing infrastructure for detection. No new infrastructure will need to be acquired and maintained, rather, one central platform can be used to monitor activity in multiple geographies.

3. SYSTEM INFRASTRUCTURE

Rather than relying on a parallel hardware platform for streaming aggregation[5], our work focuses on the use of the often constrained environments imposed by Platform-as-a-Service (PaaS) providers for event aggregation. In this work, we focus specifically on Google's App Engine[6]. App Engine provides a robust, managed environment in which application logic can be deployed without concern for infrastructure acquisition or maintenance, but at the cost of full control. App Engine's platform dynamically allocates instances to serve incoming requests, implying that the number of available instances to handle requests will grow to match demand levels. For our purposes, a request is an arriving event, so it follows that the architecture can be used to serve any level of traffic, both the drought of quiescent periods and the flood that occurs during seismic events, using the the same infrastructure and application logic.

However, App Engine's API and overall design impose a variety of limitations on deployed applications; the most important of these limitations as it concerns event processing are the following.

3.1 Synchronization limitation

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. However, 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 transactions or synchronized access and that it only supports one atomic operation: increment. Mechanisms for rapid event detection must deal with this constraint of Memcache. More complex interactions can be built on top of the atomic increment 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 timeframe limitation discussed next.

The Datastore supports transactions, but with the limitation that affected or queried entities must exist within the same Entity Group. For performing consistent updates to a single entity, this is not constraining, but when operating across multiple affected entities, the limitation can pose problems for consistency. 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 transaction. If no parent is defined, the entity is a root node. A node can have any number of children, as can its own children.

This imposes limitations because groups can only have one write operation at a time. 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 tradeoff concurrent updates against benefits of transactional integrity. 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.

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.

3.2 Timeframe limitation

Requests that arrive to the system must operate within a roughly thirty-second timeline. 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[7] 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 tolerant to losses of small amounts of information.

3.3 Query limitation

Several query limitations are imposed on Datastore queries. The most important limitation is that 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. We discuss our solution to solving the problem of querying simultaneously by time and location in Section 4. Additionally, the nature of the Datastore makes traditional join-style queries impossible, but this limitation is circumventable by changing data models or data queries.

4. NUMERIC GEOCELLS FOR GEOSPATIAL QUERIES

Since queries on App Engine are limited to using inequality filters on only one property, a different method is needed for any form of geospatial queries. Our solution involves the use of 8-byte long 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.

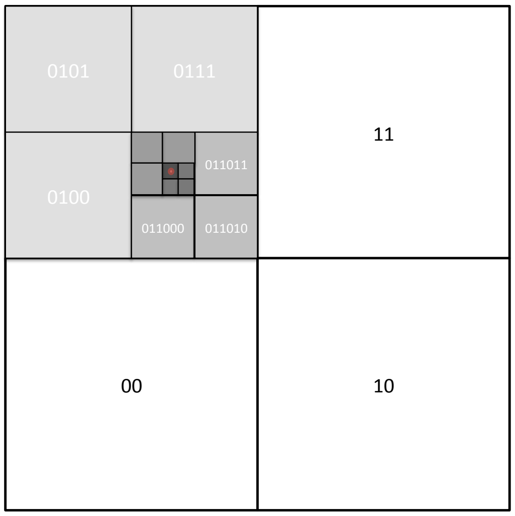


Figure 3: How bounding boxes divide the coordinate space.

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 Figure 3. 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.

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.

4.1 Comparison

Two similar open methods of hashing latitude and longitude pairs into simple strings have been previously proposed: GeoModel[8] and Geohash[9]. 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[10] and MGRS[11] systems not only have a complicated derivation algorithm[12] but also suffer from exceptions to grid uniformity. The GARS[13] and GEOREF[14] system utilize an extremely small number of resolutions: 3 and 5, respectively. The NAC System[15] 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 datapoints, while the focus of the other algorithms is computing an effective hash for datapoints. The path serves at that hash.

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 4.3 contains more information on the selection of geocells to query.

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 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 coordinate's latitude; 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 the following table, 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.

	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

Finally, prefix matching with any of these algorithms suffer 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.

4.3 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

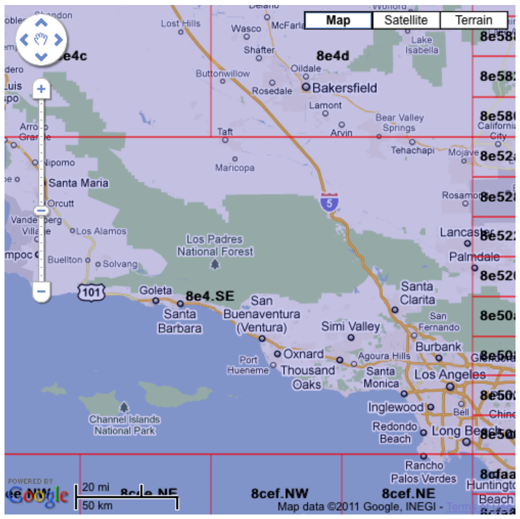


Figure 4: Combining geocells of multiple resolutions to cover an area.

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. 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, Figure 4 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.

5. DECENTRALIZED DETECTION WITH COMMUNITY SENSORS

The CSN system performs decentralized detection of seismic events by allowing each individual sensor to generate picks of potential seismic events and then aggregating these pick messages by geocell in the cloud to determine if and where an event has occurred. The different algorithms used at the sensor and server levels are discussed next.

5.1 Sensor-side Picking Algorithms

Different sensor types are likely to experience different environmental and noise conditions, and so different picking algorithms may be best suited to particular sensor types. We studied two picking algorithms: the STA/LTA algorithm, designed for higher-quality sensors in relatively low-noise environments, and a density-based anomaly detection algorithm suited to handling the complex acceleration patterns experienced by a cell phone during normal daily use. These algorithms are described in detail in [16] and summarized here.

Event Detection using Averages: STA/LTA.

STA/LTA (Short Term Average over Long Term Average) computes the ratio between the amplitude of a short time window (STA) and the amplitude of a long time window (LTA) and decides to "pick" when the ratio reaches above a threshold. In our analysis, we used a short term window size $ST = 2.5$ s and a long term window size $LT = 10$ s. This simple algorithm can detect sudden changes in transients that may indicate the occurrence of an event in a low-noise environment. In an ideal situation where the sensors have fixed orientation, the signal on each axis can be used to derive the direction of the incoming wave. We do not assume consistent orientation here, but instead simply take the L2 norm of all three axes before computing the STA/LTA.

Anomaly Detection using Density Estimation.

Earthquakes are rare, complex natural phenomena, and consequently are difficult to model. Further, heterogeneous community sensors such as cell phones differ widely in quality and reliability due to varying hardware and software platforms, as well as differing environmental conditions. However, sensory data in the absence of earthquakes is plentiful, and can be used to accurately estimate a probability distribution over normal observations. This density estimate is then used to identify anomalies (observations assigned sufficiently low probability by the model) and transmit them as picks to the fusion center.

Phones frequently change their orientation, producing large changes in signal as the sensor rotates relative to gravity. This effect can be removed by automatically determining the phone's orientation. A decaying average is used to estimate the direction of gravity. The data is rotated to point the gravity component in the negative Z direction, and then the acceleration due to gravity is removed. While this process reliably orients the phone's Z axis, the other axes cannot be consistently oriented. Instead, the Euclidean norm $||X, Y||_2$ of the remaining components, which is invariant to rotations about the Z axis, is computed.

Processing accelerometer time series within the computational limits of smart phones requires a concise and informative representation of the raw data. A 3-axis Android accelerometer produces ≈ 50-100 samples per axis each second. The raw acceleration stream is broken into 1 second time windows, and a feature vector is computed to summarize the data in each window. The feature vector is formed by first computing a vector of statistics: 16 Fourier coefficients, the second moment, and the maximum absolute acceleration for both the Z axis and the Euclidean norm $||X, Y||_2$. The final feature vector is obtained by projecting these 36 statistics onto the top 16 principal components computed by PCA, and retaining the projection error as an additional feature. We arrived at this choice of feature vector by extensive cross validation on acceleration data gathered from volunteers' Android phones.

Our anomaly detection requires a density estimate and a probability threshold that separates normal data from anomalies. We use a Gaussian mixture model for density estimation. Model selection (number of Gaussians) and estimation can be computed offline, and uploaded to the phones. Due to the rarity of large earthquakes, nearly all anomalies detected by a phone will be false positives resulting from unusual day-to-day motions. We use this fact to both control the average number of messages sent by each phone and to bound the rate of false positives reported. Each phone uses online percentile estimation to learn a probability threshold that classifies a desired fraction of the data as anomalies. This fraction can be chosen according to bandwidth constraints and is precisely the sensor false positive rate.

5.2 Server-side Pick Aggregation

Picks generated by the sensors are sent to the App Engine server. These simple events are aggregated using the numeric geocells described in Section 4. However, a few factors complicate complex event association and detection.

First, the time of App Engine instances is not guaranteed to be synchronized with any degree of accuracy. This means that relative time determination within the network must be handled by clients if any guarantees about clock accuracy are to be made. This is currently done through the inclusion of an NTP client in the sensor software which determines the drift of the host computer to the network time at hourly intervals.

Second, requests may fail for a variety of reasons. We have previously estimated that as many as 1% of requests on App Engine will fail for reasons beyond the control of the developer. These kinds of errors include requests that wait too long to be served, hard deadline errors, and serious errors with the App Engine servers. In addition to these system errors, clients may go offline without notice due to a software error or something as simple as the host computer going to sleep.

These system conditions mean that the detection algorithm must be: insensitive to the reordering of arriving messages, which occurs by variations in processing or queueing time or by inconsistent determination of network time, and insensitive to the loss of small numbers of messages either due to client or server failures.

The server's job is to estimate complex events such as the occurrence of an earthquake from simple events that indicate an individual sensor has experienced seismic activity. This is done by estimating the frequency of arriving picks by allocating arriving picks to buckets. Buckets are created by rounding the pick arrival time to the nearest two seconds and appending the geocell to the long representation of the time. This gives a unique key with which a bucket is created that all arriving picks in the same time window and region will use to create estimates of the number of firing sensors at that point in time. For instance, an example bucket name would be '12e55d89260-4da040500000001c'.

This bucketing necessarily removes any ability to detect events based on arrival order, but permits event detection based on both arrival frequency and the content of arriving events. Whenever a pick arrives, the appropriate bucket name is calculated and the number of arriving events for that bucket is incremented. The number of active clients for location identified by the bucket's key is also retrieved, which makes it possible to determine the ratio of clients that have experienced a seismic event. For each arrival, the contents of the buckets of the current time window and the surrounding time windows are summed to help manage inconsistencies in arrival time and time of computation.

The sum of the arriving picks across a known time interval is then divided by the number of active clients to determine whether or not a specific geocell has exceeded a threshold level of activity to perform further computation. This is the first trigger which generates a complex event that a given geocell has activated. Activation of the geocell is managed by a Task Queue job which is created to proceed with further analysis. The job is named, which means that for any number of arriving picks in the same time window, only one job will be created per geocell per time window.

The execution of the named job involves probing the surrounding geocells to determine what other geocells have recently fired. The total number of sensors reporting seismic activity in any region for a given time window can be computed by calculating the bucket names under which those events would have been aggregated and summing their contents. The sequence of activation is then used to extrapolate what kind of event the network is experiencing. Of particular importance is the reliability of this detection, which is discussed in Section 6.

6. ESTIMATING SENSOR PERFORMANCE

A question we are trying to answer is: are inexpensive sensors capable of detecting seismic events? The CSN currently uses two types of sensors: the accelerometers in Google Android cell phones and 16-bit MEMS accelerometer USB sensors manufactured by Phidgets, Inc. Measurements from a variety of Android phones while at rest showed device noise with standard deviation of ≈ 0.08 m/s^2. Phidgets at rest showed device noise with standard deviation ≈ 0.003 m/s^2. For reference, earthquakes of Gutenberg-Richter magnitude 4 produce accelerations of approximately 0.12 m/s^2 close to the earthquake's point of origin. These numbers suggest that cell phone accelerometers should be sensitive enough to be able to detect large earthquakes.

Without waiting to observe several large earthquakes, we can only estimate the performance of each picking algorithm under several moderately large seismic scenarios. Detection performance on these simulated event recordings should provide a lower bound on sensor performance during larger events; this claim is experimentally validated in [16].

We obtain simulated acceleration time series recordings for both Phidget and Android sensors by combining historical earthquake recordings from the USGS Southern California Seismic Network (SCSN) with noise recordings from volunteers' Phidget and Android sensors. We collected a set of 54 SCSN records of magnitude M5-5.5 earthquakes from seismic stations between 0-100 km. The SCSN recordings are down-sampled to 50 samples per second to be comparable with low-end consumer accelerometers, and are then overlaid with Phidget or Android recordings from the volunteer data set in order to obtain a realistic noise profile.

6.1 Lower Bounds for Sensor Performance

Receiver Operating Characteristic (ROC) curves are used to gain insight into the performance of a binary classifier. The curve plots true positive rate (TPR) against false positive rate (FPR) for each possible decision threshold. ROC curves allow us to estimate the obtainable TPR of a sensor, given a constraint on its FPR, such as a limit on the average number of pick messages per day.

Using the data set of synthetic historical recordings, we can compute ROC curves for each sensor type, under a variety of seismic scenarios. From a data set of magnitude $M = 5 - 5.5$ earthquakes, we extract 5 sets of records, containing data from stations at varying distances away from the epicenter. The data sets correspond to distance ranges d in kilometers, $d = \{0-10, 20-30, 40-50, 70-80, 90-100\}$. Figure 5 illustrates the performance of the the STA/LTA algorithm (evaluated on synthetic records made with volunteers' Phidget data), and the anomaly detection algorithm (evaluated on synthetic records made with volunteers' Android data). These ROC curves demonstrate that the Phidgets - higher resolution sensors that are typically not subjected to user motion - obtain superior performance to the Android sensors at all distance ranges. The curves also reflect the $1/r^2$ decay rate of shaking intensity, where r is distance from the quake epicenter.

6.2 Geocell Detection Performance

From the ROC of a single sensor, we can analyze the collective behavior of a group of sensors. Consider a number of sensors occupying a relatively small geocell (e.g. several street blocks). Inside this cell, each sensor experiences similar seismic shaking during an event, and independent noise (such as motions caused by a cell phone's user) in the absence of an event. We can roughly say that all sensors within a cell have the same signal to noise ratio (SNR) and that their picks can be well approximated as independent, identically distributed binary random variables when conditioned on whether an event has occurred or not. By fixing the decision rule for each sensor, and a decision rule for cell-wide event detection, we can evaluate the event detection performance as a function of the number of sensors in the cell.

The sensor decision rules can be specified by constraining the maximum allowable rate of false positive picks. Here, we constrain the Phidget USB sensors to produce at most 1 false pick per hour, and constrain the Android sensors to at most 1 false pick per 5 minute interval. The cell-wide false positive rate is constrained to no more than 1 per year, on average. Figure 6(a) and Figure 6(b) show cell detection performance as a function of sensor density, generated from synthetic M5-5.5 records. These results indicate that a cell containing 30 Phidgets or 100 Androids could reliably detect a moderately large earthquake at a distance of 50km from the epicenter, and that the higher-quality Phidget sensors are capable of detecting the signal from up to 100km.

6.3 System-wide Detection

While sensors within several kilometers of each other are likely to experience similar shaking during an event and thus have similar SNR, sensors distributed across a city or other large region can be expected to behave quite differently. In Section 7, we evaluate how system-wide detection performance is impacted by aggregating picks using a grid of geocells.

7. EXPERIMENTS

In this section, we describe simulation results that study the detection performance of dense, heterogeneous seismic networks during several earthquake scenarios. To simplify the discussion, we restrict our focus on detection to a single subcontinental area, e.g. the Greater Los Angeles Area rather than the North hemisphere. This is reasonable, as one can identify a priori clusters of sensors in different geographic regions so that there is little correlation between clusters. We experimentally evaluate an event association algorithm which aggregates picks by geocell, and compare its performance against a baseline naive event association algorithm which aggregates all pick messages within the region.

7.1 Simulation Platform

An earthquake is a complex event that differs in every occurrence depending on where in the world it occurs. Simulators developed by seismologists often have a large number of input parameters. In comparison, this study focuses on capturing the behaviors of a large-scale network of noisy sensors. Without loss of generality, we assume a much simpler seismic model that includes a point source (i.e. the epicenter is a single point) which is isotropic (i.e. the wave travels in all directions with equal speed).

Based on the sensor characterizations described in Section 6, we can begin to study the detection performance of a total system with a given deployment of sensors. To aid these studies, we have developed a simulation platform that allows time series of sensor picks to be simulated, based on a set of specified sensor TPR,FPR operating points. These operating points can be chosen to maximize detection performance, while satisfying the per-sensor bandwidth constraints.

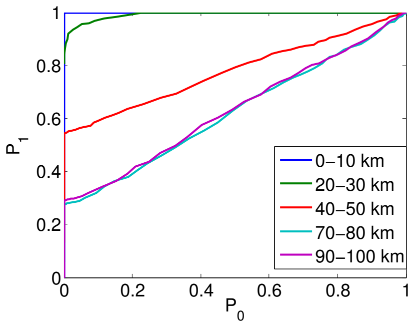
Given the location of each sensor in the network and the origin of a seismic event, the program computes the probability that each sensor picks during each time instance, and generates picks with these probabilities to produce a time series of pick messages. Figure 8 shows a snapshot of simulated detection of a M5.5 event. The snapshot is taken 20 seconds after the event occurs. For this simulation, the Phidget FPR is set at 1 pick per hour and the Android FPR is set to 1 pick every 5 minutes. The pick messages are timestamped after factoring in network delays. The counts of message arrived at the server in this specific simulation run is shown in Figure 6(c) as a function of time from the start of the event.

7.2 Naive Event Association

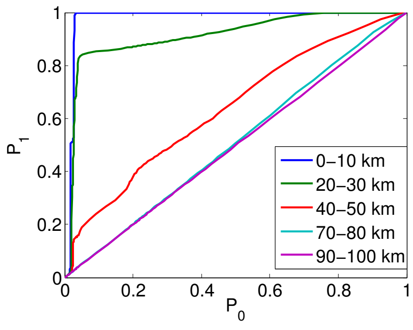
Section 5.2 describes a procedure for identifying if a single geocell contains a significant number of picks within a small interval of time. While the activation of one geocell could be used to generate system-wide alerts, such a policy neglects the physical laws governing how earthquakes spread.

In this section, we study the system level detection performance, starting with a simple event association algorithm.

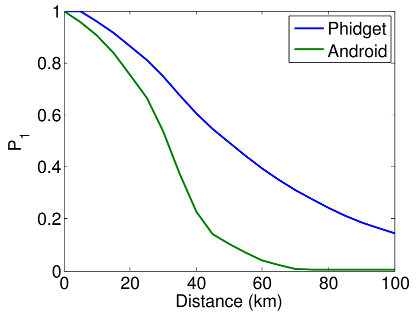
At each time step, the system decides whether an event has occurred based on the pick messages it has received so



(a) Phidget

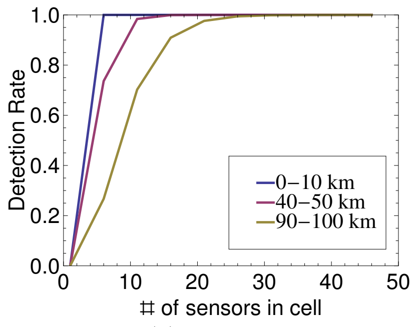


(b) Android

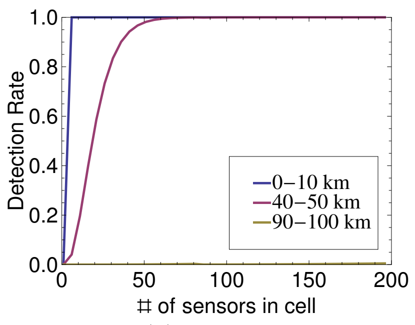


(c) Detection vs. Distance

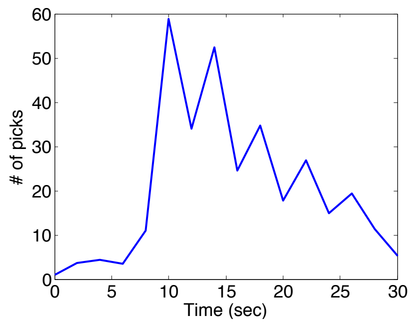
Figure 5: ROC curves for (a) Phidget and (b) Android. (c) Detection performance vs. distance from epicenter under the guarantee of at most 1 false message per hour for the Phidget and 1 false message every 5 minutes for the Androids.



(a) Phidget



(b) Android



(c) Expected Traffic

Figure 6: (a) Single cell of varying number of Phidgets observing 3 levels of seismic events of M5.5 and lower. (b) Single cell of varying number of Androids observing 3 levels of seismic events of M5.5 and lower. (c) Number of pick messages received by the system as a function of time since the event started.

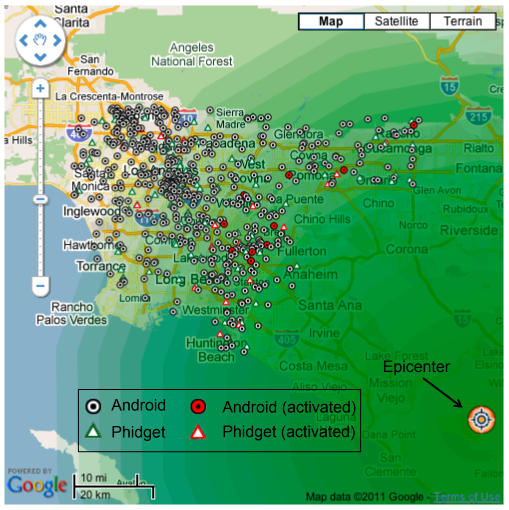


Figure 8: Snapshot of simulated detection of a M5.5 event 80 km outside the great Los Angeles area. There are 100 Phidgets and 1000 Androids distributed according to population density. The snapshot is taken 20 seconds after the event occurred.

far. One naive decision rule is to perform hypothesis testing on the aggregated pick counts in the past few seconds, that is, to compute the ratio of likelihood for the two hypothesis: 1) that there is an event ($p = p_1$), and 2) that there is no event ($p = p_0$). In other words, the naive decision rule performs the test

$$\frac{Binomial(k; n, p_1)}{Binomial(k; n, p_0)} \geq r$$

where n is the total number of sensors in the system, k is the number of picks observed in the past few seconds and r is the decision threshold chosen to satisfy a constraint on the system false positive rate. If the inequality holds, the system declares detection.

This algorithm is naive because it disregards the varying strengths of geospatial correlation between sensors as well as the pattern in which seismic waves travel. Depending on the distance and direction of the wave relative to the region of sensors, different number of sensors are affected at a given time. Therefore it may not be reasonable to consider measurements from all sensors equally at all times. We will present a different association algorithm that exploits these behaviors in the next section. Here we study the naive algorithm as a lower bound for the system level detection performance.

We collected 1000 sets of measurements from 2000 Androids and 20 Phidgets separately during a simulated M5.5

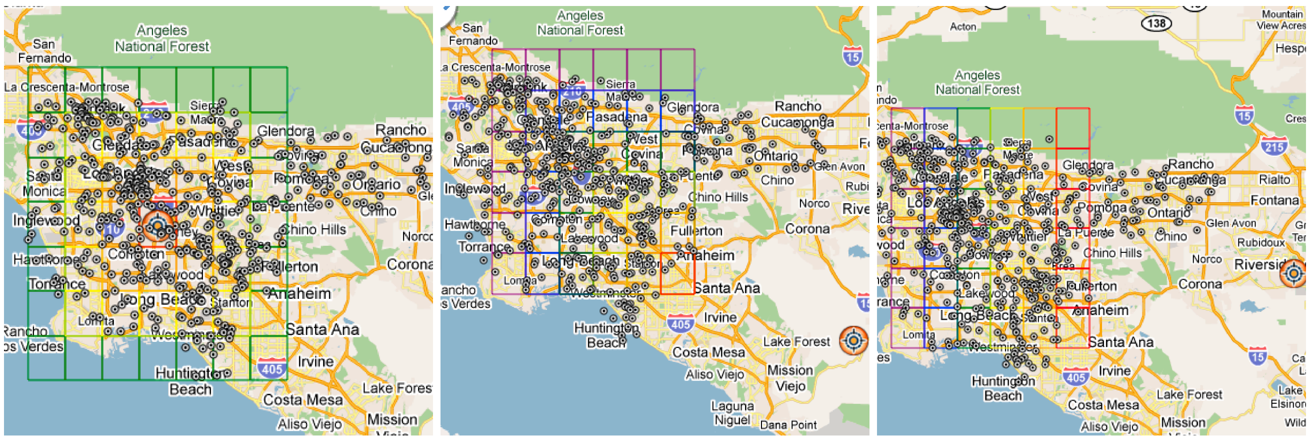

(a) Central Model (b) Corner Model (c) Side Model

Figure 7: Regions of different sizes and shapes are activated in different sequence for each of the three scenarios. The rainbow-colored rings indicate the order of activation. Red: first. Purple: last

event within 60 km of downtown Los Angeles. During a period of $T = 0 - 10$ seconds after the event occurs, we perform the naive association algorithm on each 2-second interval and compute the system level detection rate while maintaining the guarantee of at most 1 false alarm per year. The results are shown in Figure 9 as the lower bounds for the two types of sensors.

7.3 Geocell-based Regional Event Association

A seismic event can be very coarsely modeled as an isotropic point source event that travels in all directions at a constant speed. With this assumption in mind, we can break down the detection problem into a few case scenarios in terms of how the incoming waves come to contact with the identified cluster of sensors. By exploiting sensor co-activation patterns in these scenarios, one can design a more logical online event association algorithm. Figure 7 shows three such possible cases after we pre-gridded the area into geocells — (a) the epicenter is inside the cluster, (b) the epicenter is diagonally away from the cluster, and (c) the epicenter is on the side and away from the cluster. In each of these cases, regions of different sizes and shapes will be activated in different sequences during san event. While it is computationally nontrivial to partition a 2-dimensional space into arbitrary regions, the geocell library provides the tools to compute these regions efficiently. We can perform hypothesis testing in parallel for each possible regions to improve the system-wide detection performance.

We computed the system-wide detection performance for each of the scenarios illustrated in Figure 7 with either 20 Phidgets or 2000 Androids distributed according to the population density in the area. The regions in terms of activation sequence are identified a priori using the geocell library. A region consists of multiple nearby geocells. Each geocell is $\approx 10 \times 10$ km in size, which is approximately how far the shock wave travels in 2 seconds. Two seconds is also roughly the short-term integration window used in both STA/LTA and anomaly detection algorithm. We can thus safely assume that all sensors in the same region have the same SNR and model them as independently identically distributed random variables, following the analysis in Section 6.2. In each time step of 2 seconds, we perform hypothesis testing on each of the regions and compute the system-wide ROC curves.

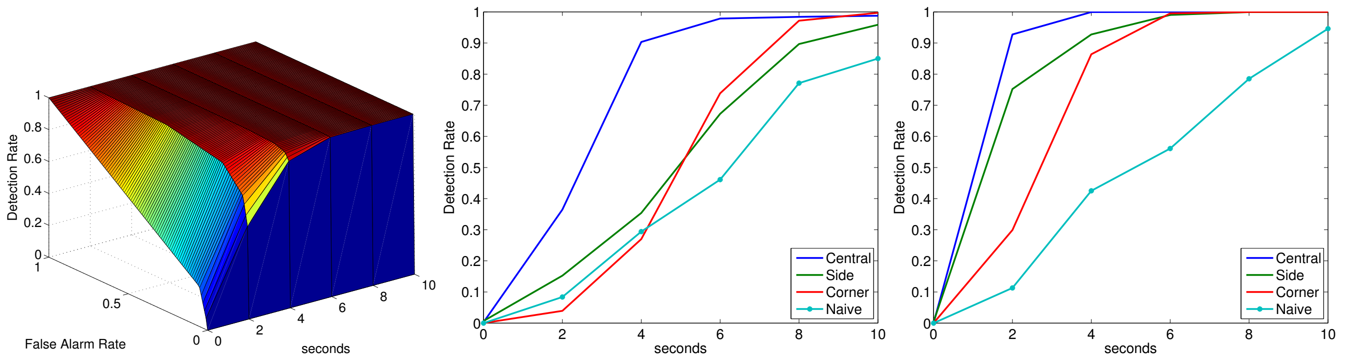
Figure 9(a) shows an example of how ROC curves of 2000 Androids evolve in time for the corner case illustrated in Figure 7(b). We slice the surface of this figure at the false alarm rate of 1 per year and retrieve the detection rate as a function of time in Figure 9(b) and 9(c). The results are compared to the baseline results computed using the naive total association algorithm discussed in Section 7.2. These results clearly highlight the benefit of intelligent event association by locality. They also touch on the tradeoffs between delayed decision making and gain in detection confidence. In the case with 2000 Androids (Figure 9(b)), we can fire off alarm at T=2 second that allows us to give 10s of seconds of early warning to surrounding cities such as Santa Barbara or San Diego but with only 20% confidence. Or we can wait till T=10 second or after to fire the alarm with $\approx 100\%$ confidence but give slower warnings.

While making the assumption of simple geometry about a highly complex event such as earthquake, this study serves as insights to the intelligent association choice based on geospatial relationship of the sensors and the event.

8. RELATED WORK

Seismic Networks.

The Quake Catcher Network[3] is closely related to CSN; QuakeCatcher shares the use of cheap MEMS accelerometers in USB devices and laptops, however, our system differs in its use of algorithms designed to execute efficiently on cloud computing systems and its incorporation of statistical algorithms for detecting rare events. The incorporation of mobile phones, which pose their own challenges distinct from those of USB devices, also distinguishes this work. The NetQuakes project[17] is related in its deployment of expensive stand-alone seismographs at the homes of community volunteers. The more expensive devices employed by NetQuakes make different tradeoffs between cost and accuracy than the sensors used by CSN.



(a) ROC transition in the corner scenario with 2000 Androids.   (b) 2000 Androids   (c) 20 Phidgets

Figure 9: Detection of a M5.5 event with (b) 2000 Androids and (c) 20 Phidgets in the three scenarios described in Figure 7. This result guarantee at most 1 false alarm per year at the system-wide level. Results computed using the geocell-based association algorithm are compared to those using the naive algorithm.

Community and Participatory Sensing.

CSN is not alone in its use of sensors owned and operated by citizen scientists to aid in research. Some projects [18, 19] have incorporated mobile phones to monitor traffic and road conditions while others [20, 21] use community sensors for environmental monitoring by obtaining up-to-date measurements of the conditions participants are exposed to. Like CSN, these applications stand to be benefit from high sensor densities, but their aims of monitoring ongoing phenomena rather than detecting rare events makes them distinct.

Distributed and Decentralized Detection.

The classical hierarchical hypothesis testing approach has been analyzed by Tsitsiklis [22]. Chamberland et al. [23] study classical hierarchical hypothesis testing under bandwidth constraints. Their goal is to minimize the probability of error, under constraint on total network bandwidth. Martinic et al. [24] perform distributed detection on multi-hop networks by clustering nodes into cells, and comparing observations within a cell to a user-supplied "event signature." The communication requirements of these detection algorithms cannot be met in community sensing applications since sensors cannot communicate with neighbors due to privacy and security restrictions.

Anomaly Detection.

There has also been a great deal of research on anomaly detection in the statistics, machine learning, and complex-event processing communities. Yamanishi et al. [25] develop the SmartSifter approach that uses Gaussian or kernel mixture models to efficiently learn anomaly detection models in an online manner. Davy et al. [26] develop an online approach for anomaly detection using online Support Vector machines. One of their experiments is to detect anomalies in accelerometer recordings of industrial equipment. They use produce frequency-based (spectrogram) features, similar to the features we use. However, their approach assumes the centralized setting. Subramaniam et al. [27] develop an approach for online outlier detection in hierarchical sensor network topologies. This approach is not suitable for the community sensing communication model, where each sensor has to make independent decisions. Onat et al. [28] develop a system for detecting anomalies based on sliding window statistics in mobile ad hoc networks (MANETs). However, their approach requires nodes to share observations with their neighbors, and so may not be suitable under the privacy constraints inherent in community sensing.

9. CONCLUSION

This paper presents results from an ongoing multi-year project carried out by researchers in geology, civil engineering, and computer science. We presented initial steps at meeting a grand challenge: building DEB systems that save thousands of lives by improving responses to disasters. The architecture, analysis of tradeoffs, and new theory presented in this paper were applied to the specific problem of responding to earthquakes. The results are, however, applicable to responses to many rapidly evolving crises.

The unfolding catastrophe due to the earthquake, tsunami, fires, and nuclear reactor explosions in Japan provides an immediate and stark example of information technology saving lives. The community, as a whole, participated in the response. This paper described how community-based event-detection systems, in which individual members of the community install and operate sensors and responders, can help society to deal with disasters collectively.

Important trends over the last five years include widespread use of cellphones and cloud computing services in all parts of the world. Even people in remote rural parts of developing economies use phones and cloud services such as those that provide weather information. This paper showed how these trends can be exploited to deploy event-based applications anywhere in the world, especially economically disadvantaged areas.

10. ACKNOWLEDGMENT

The authors would like to thank the following Caltech collaborators who are all working to make the Community Seismic Network a success: Prof. Robert Clayton and Prof. Jean-Paul Ampuero from the Seismo Lab; Prof. Tom Heaton, Dr. Monica Kohler, and Ming-Hei Cheng from Earthquake Engineering; Prof. Andreas Krause and Rishi Chandy from

Computer Science; Dr. Julian Bunn, Michael Aivazis, and Leif Strand from the Center for Advanced Computing Research.

This research is supported in part by the National Science Foundation Cyber-Physical Systems program.

11. REFERENCES

[1] K. M. Chandy, O. Etzion, and R. von Ammon, "10201 Executive Summary and Manifesto – Event Processing," in *Event Processing*, ser. Dagstuhl Seminar Proceedings, no. 10201. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2011. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2011/2985

[2] A. Campbell, S. Eisenman, N. Lane, E. Miluzzo, R. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn, "The rise of people-centric sensing," *Internet Computing, IEEE*, vol. 12, no. 4, pp. 12 –21, 7-8 2008.

[3] E. Cochran and J. Lawrence, "The quake-catcher network: Citizen science expanding seismic horizons," *Seismological Research Letters*, vol. 80, p. 26, Jan 2009.

[4] (2011, 3) Measuring shaking intensity with mobile phones. [Online]. Available: http://ishakeberkeley.appspot.com/mission

[5] S. Schneidert, H. Andrade, B. Gedik, K.-L. Wu, and D. S. Nikolopoulos, "Evaluation of streaming aggregation on parallel hardware architectures," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '10. New York, NY, USA: ACM, 2010, pp. 248–257.

[6] (2011, 3) Google app engine. [Online]. Available: http://code.google.com/appengine/

[7] M. Olson and K. M. Chandy, "Performance issues in cloud computing for cyber-physical applications," in *Proceedings of the 4th IEEE International Conference on Cloud Computing*. IEEE, 2011.

[8] 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

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

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

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

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

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

[14] (2011, 3) Georef. [Online]. Available: http://en.wikipedia.org/wiki/Georef

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

[16] 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.

[17] (2011, 3) Netquakes. [Online]. Available: http://earthquake.usgs.gov/monitoring/netquakes/

[18] R. Herring, A. Hofleitner, S. Amin, T. Nasr, A. Khalek, P. Abbeel, and A. Bayen, "Using mobile phones to forecast arterial traffic through statistical learning," *Submitted to Transportation Research Board*, 2009.

[19] A. Krause, E. Horvitz, A. Kansal, and F. Zhao, "Toward community sensing," in *Proceedings of the 7th international conference on Information processing in sensor networks*. IEEE Computer Society, 2008, pp. 481–492.

[20] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda, "Peir, the personal environmental impact report, as a platform for participatory sensing systems research," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*. ACM, 2009, pp. 55–68.

[21] P. Völgyesi, A. Nádas, X. Koutsoukos, and Á. Lédeczi, "Air quality monitoring with sensormap," in *Proceedings of the 7th international conference on Information processing in sensor networks*. IEEE Computer Society, 2008, pp. 529–530.

[22] J. Tsitsiklis, "Decentralized detection by a large number of sensors," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 1, no. 2, pp. 167–182, 1988.

[23] J. Chamberland and V. Veeravalli, "Decentralized detection in sensor networks," *Signal Processing, IEEE Transactions on*, vol. 51, no. 2, pp. 407–416, 2003.

[24] F. Martincic and L. Schwiebert, "Distributed event detection in sensor networks," in *Systems and Networks Communications, 2006. ICSNC'06. International Conference on*. IEEE, 2006, p. 43.

[25] K. Yamanishi, J. Takeuchi, G. Williams, and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," *Data Mining and Knowledge Discovery*, vol. 8, no. 3, pp. 275–300, 2004.

[26] M. Davy, F. Desobry, A. Gretton, and C. Doncarli, "An online support vector machine for abnormal events detection," *Signal processing*, vol. 86, no. 8, pp. 2009–2025, 2006.

[27] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 187–198.

[28] I. Onat and A. Miri, "An intrusion detection system for wireless sensor networks," in *Wireless And Mobile Computing, Networking And Communications, 2005.(WiMob'2005), IEEE International Conference on*, vol. 3. IEEE, 2005, pp. 253–259.