Towards a Dependable Architecture for Internet-scale Sensing

Rohan Narayana Murty and Matt Welsh
Division of Engineering and Applied Sciences
Harvard University
{rohan,mdw}@eecs.harvard.edu

Abstract

The convergence of embedded sensors and pervasive high-performance networking is giving rise to a new class of distributed applications, which we refer to as Internet-scale sensing (ISS). ISS systems consist of a large number of geographically distributed data sources tied into a framework for collecting, filtering, and processing potentially large volumes of real-time data. In this paper, we discuss the issues involved in building dependable ISS systems. ISS systems differ from conventional distributed systems in a number of respects, including the number of data sources, differing data quality requirements, and necessity to continue operating despite intermittent link and node failures. Such failures should result in graceful degradation of the quality of the results returned by the system, rather than loss of results.

In this paper, we argue that conventional approaches to achieving consistency do not scale to the requirements of ISS systems. We outline a lightweight approach to dependability based on a set of metrics that reflect on the quality of the answers returned by the system. We argue that answers returned by an ISS system should include a measure of the *harvest* and *freshness* of the data sources participating in the result, and these metrics in turn can be used to drive fault-tolerance mechanisms in the system. We also propose three simple techniques to achieve scalability and graceful degradation in the face of failure.

1 Introduction

The convergence of embedded sensors and pervasive high-performance networking is giving rise to a new class of distributed applications, which we refer to as *Internet-scale sensing* (ISS). An Internet-scale sensing system consists of a number of geographically distributed data sources tied into a networked framework for collecting, filtering, and processing potentially large volumes of real-time data. Data sources include telescopes, satellites, seismometers, or weather stations; corresponding scientific applications include whole-sky surveys [10], automated pulsar detection [8], earthquake detection and characterization [5], and environmental monitoring of large ecosystems [6]. In the distributed systems community, ISS systems are being developed to support network performance monitoring [4], distributed virus and worm detection [7, 21]. The common theme across these systems is the acquisition and processing of real-time data, yielding a macroscopic view of many disparate data sources.

In this paper, we argue that ISS applications have fundamentally different data quality and reliability requirements than conventional distributed systems. In particular, for ISS systems to scale to vast numbers of data sources and simultaneous users, it is simply infeasible to expect the system to achieve "reliability" in the conventional sense. Data sources, network hosts, and network links experience frequent and intermittent failures; yet, the system must continue to produce answers, possibly based on incomplete input data. Failure (and concomitant quality degradation) is the norm, rather than the exception, in such systems.

We claim that ISS systems require a new approach to dependability that eschews the use of complex, heavyweight consistency protocols in favor of leaner mechanisms that scale well with the increasing number of data sources as well as recognize the predominance of transient loss and failure. The real-time nature of stream data processing differs greatly from distributed systems concerned with persistent state. Rather than strive to achieve data consistency, an ISS platform should provide feedback to end users on the *fidelity* and *coverage* of the results returned by the system. Such feedback can be tied to the accuracy of results, and used to drive fault-tolerance mechanisms within the system.

In this paper, we outline broad design principles for ISS systems, and propose three simple techniques to achieve scalability and graceful degradation in the face of failure. These include:

- *Structured operator replication*, which replicates data-processing operators relative to their importance in the query;
- *Free-running operator state*, designed to support operators with a bounded temporal dependence on their input data; and
- *Best-guess reconciliation*, which filters or combines results across multiple, possibly divergent, operator replicas.

In this paper, we outline the requirements for ISS applications, describe previous approaches to building such systems, and explain why these approaches fail to meet the requirements. We then describe a range of data quality metrics for ISS systems, contrasting them to more conventional distributed systems goals. Finally, we outline an approach to architecting a scalable, robust ISS platform that supports

these metrics, representing a significant departure from the classic distributed systems focus on data consistency and availability.

2 Background: Internet Scale Sensing

A broad range of Internet-scale sensing (ISS) systems are currently under development in varying scientific domains. Some important examples of ISS systems currently under development include:

eScience applications, including the EarthScope [5] and NEON [6] initiatives. EarthScope plans to deploy thousands of GPS receiving stations and seismometers across the North American continent to study plate tectonics and earthquakes at unprecedented scales. NEON is deploying networked arrays of mobile and fixed embedded sensors, cameras, and weather stations for monitoring large ecosystems. The increased activity in wireless sensor networks [28, 33] is one driver of this area.

Network monitoring involving real-time data sources on the Internet. Examples include network telescopes [22], distributed intrusion detection systems [3], and surveillance of blogs, RSS feeds, and chat room activity for law enforcement [1].

Distributed surveillance using networked cameras [15], microphone arrays (e.g., for localizing gunshots) [27], and other sensors. For example, a global array of seismometers, microphones, and radionuclide stations is used to monitor compliance with the Comprehensive Nuclear Test Ban Treaty [2] and locate sources of nuclear explosions.

Existing ISS systems are tied to very specific applications, although each faces similar challenges in terms of scalability, robustness, and performance. To date, solutions have been developed in an *ad hoc* manner, typically by users outside of the distributed systems community. For example, geodesic data from EarthScope is currently hosted at a single FTP site in Colorado, from which users must manually download datasets of interest. Apart from the obvious scalability and reliability concerns, such an approach requires users to download large volumes of data for local processing, which is impractical for real-time applications supporting many simultaneous users.

We believe that the development of a flexible, general-purpose infrastructure to support ISS applications is an exciting research agenda, and one that will increase in importance over time as more scientific disciplines tap into the ability to link multiple data sources over the Internet. Such an infrastructure could support multiple disparate applications in different domains, simplifying application design through a common set of interfaces for data acquisition and in-network processing. By pushing computation on the raw data closer to its sources, network bandwidth can be conserved. In many applications, intermediate results can be shared across multiple users, offering a multiplicative reduction in network load.

Related work

A number of existing systems represent steps towards an Internet-scale sensing platform. In many cases, such as the various astronomical *virtual observatory* efforts [9, 11], the infrastructure is highly specialized to a given application domain. Many systems are currently centralized, as is the case with EarthScope and the nuclear test ban verification network. Numerous research efforts in wireless sensor networks [28, 33, 29, 32] are focused on application-specific deployments focused on data collection, although the potential to link these into a larger network infrastructure has been raised [16, 15, 25]. The network monitoring community has developed a range of distributed systems that can be seen as prototypical ISS platforms, including distributed honeypots [31], network telescopes [22], and intrusion detection systems [3].

Extensive work on streaming database systems, such as Borealis [12], PIER [20], and HiFi [16] look to apply the relational query model to real-time streaming data. However, few of these systems have been concerned with scaling up to large numbers of data sources or simultaneous users. For example, Borealis has mainly been evaluated on small configurations of less than six machines. While PIER [20] is focused on scale, its approach to mapping query operators to hosts using a P2P overlay ignores the impact of increased latency and network load. Moreover, as discussed in the next section, these systems do little to address reliability in environments where failures of hosts and network links are commonplace.

3 Issues and Challenges for Internet-Scale Sensing

In this section, we outline a typical ISS system and consider the challenges in designing such a system. We then discuss to what extent failures affect results returned by the system, and why previously proposed approaches fail to address these problems. We argue that an ISS system must provide feedback on its internal operation that can be used to inform the end user of the quality of the end results, as well as drive fault tolerance mechanisms within the system.

Let us first sketch a concrete ISS system that performs large-scale network monitoring. Such an application would consist of two components: (1) a large number of data sources producing live network data, and (2) multiple queries on the input data. Possible sources of network data include router packet traces, honeypots, network telescopes, BGP feeds, and firewall reports. The system could support queries from a large number of users that mine this rich, real-time data set to understand the network's operation, as well as to detect anomalies, attacks, and virus/worm propagation. Queries are processed by pulling data from the data sources in real time, performing various filtering and aggregation operations, and pushing periodic results to end users. An example of a network monitoring query might be "return the top 1000 destination IP addresses for all packets leaving subnet *a.b.c.d* once every 5 minutes."

3.1 ISS system architecture

Given the large number of data sources and potentially large number of concurrent queries in an ISS system, it is infeasible for each user to download all data of interest for local processing. To support such massive scale, a number of systems have investigated the use of an overlay network of hosts that collect, process, and deliver real-time data [12, 15, 25]. While the details of these systems differ, their high-level architectures are very similar: a user constructs a query that is typically realized as a tree of operators that filter and aggregate streaming data. The query pulls data from multiple sources, where it flows up the query tree until results are delivered to the end user.

By leveraging an overlay network and pushing query processing closer to the data sources, the total network load consumed by a set of queries can be greatly reduced. This is especially important for data sources or users that have low-bandwidth connections to the Internet (e.g., seismic sensors connected via expensive and low-bitrate satellite telemetry). In addition, numerous optimizations can be performed within the overlay network. For example, queries within similar data requirements and processing can be merged, reducing computational load. Likewise, the placement of operators within the overlay network can be tuned based on latency, bandwidth, or load [25].

3.2 ISS challenges

The massive scale, data fidelity requirements, and real-time nature of ISS systems give rise to a unique set of challenges that differ somewhat from more traditional client/server or peer-to-peer distributed systems. We outline these challenges below.

Scaling to support large numbers of data sources and users: Harnessing data from vast numbers of real-time sources is made difficult by intermittent failures of sources and network links, as well as the difficulty of diagnosing such failures (e.g., determining whether a source is offline, or whether a link is faulty). Varying data rates, link latencies, and bandwidth capacities raise concerns of operator placement and link saturation. Scaling up to support many simultaneous users and queries raises questions of effective load balancing and bandwidth sharing across queries.

Failures and their effect on query results: The failure of an overlay host will affect the ability of an operator to consume input data and produce results to push up the query tree, leading to gaps or delays in the result stream. Failures also affect the internal state of an operator. A stateful operator (e.g., one maintaining an average value over some time window) may lose its state following a failure, leading to errors in future query results after it recovers. Likewise, network partitions lead to data loss even if the overlay hosts themselves are reliable. Moreover, failures in an ISS system have a cascading effect in that the inputs to downstream operators in a query are impacted. As an example, the failure of a node in the query tree can knock out an entire subtree of the query, deeply affecting the quality of results.

3.3 Previous approaches

A natural starting point for increasing robustness in an ISS query is to replicate query operators across multiple physical hosts, using geographic diversity to avoid outages due to network link failures. However, replication alone does not address the problem of *state divergence*, which can cause different replicas to report different results. An alternate approach is to make use of state replication schemes, which attempts to keep replicas in a consistent state. This is the classic approach in managing replicated state in distributed systems, though we argue it is inappropriate for ISS systems due to its high overhead, potentially requiring message exchange on every tuple arriving at an operator.

While there exists a great deal of prior work on replication techniques and consistency protocols, we advocate the use of lightweight techniques for the purposes of achieving fault tolerance in ISS systems. The Tandem system [14, 18] proposed using passive standby involving primary-backup pairs to insure against system failures. Stronger versions of this approach, involving checkpoint primary-backup state, have since been proposed in the literature [26]. We believe for the purposes of an ISS system, this approach suffers from low availability and high overhead.

In an *active replication* approach, all replicas receive the same set of inputs (with ordering guarantees) and compute the same state. This approach does not require replicas to synchronize state. However, it hinges on the assumption that all incoming data are delivered in order to all replicas. In an ISS system with intermittent node failures, asymmetric link failures (possibly caused by routing anomalies), network partitions, processing delays, and variable network latency, we explore the possibility of using active replication to provide high availability but with relatively lower overhead. In particular, we relax the requirement of in-order delivery at all replicas, which reduces overhead but can lead to state divergence.

The Borealis [12, 13] system makes use of replication but leverages *eventual consistency* semantics: an operator recovering from a failure continues to report answers that are marked as "tentative" until its input state is brought up to date with its replica group. Each upstream operator must buffer its history of output tuples in order to replay them against downstream operators that recover from a failure.

The Borealis approach trades off sophisticated consensus protocols with a buffer-and-replay strategy following failures. There are several problems with this approach in an ISS context. First, the history of output tuples that must be stored by an operator could be arbitrarily long, and must be recorded in persistent storage in case of failure. Second, this approach assumes that failures are infrequent, requiring an expensive replay of past tuples against a recovering operator. In a system with frequent failures and high churn, we expect such an approach will not scale well. Third, Borealis does not attempt to bound the error for so-called "tentative" tuples, so users have little information about the quality of these results.

In TRAPP [23], the authors explore the tradeoffs between precision and availability. Similar in spirit to our argument, the system permits the user to specify constraints on the desired correctness of the end answer as well as the availability of the system. Based on the set of constraints, the system attempts to mix and match cached data as well as fresh data from the sources. This approach has been studied assuming a fixed number of operator replicas. Also, it is unclear how well TRAPP would scale to meets the needs of an ISS system, given the large number of data sources involved. TRAPP requires data sources to provide upper and lower bounds on the numerical values produced and these values are cached at replicas in the system. Hence there is a significant amount of state that needs to be stored and processed when the system has a large number of data sources. This poses a challenge to its ability to scale.

Bayou [24] deals with weakly connected replicas that are kept consistent using epidemic protocols. Similarly, Astrolabe [30] makes use gossip protocols to achieve eventual consistency among replicas. TACT [19] explores a continuous consistency model that allows one to study the effects of protocols trading off availability for consistency. Similar techniques could potentially be used in an ISS system and we address this in the discussion section.

4 New Dependability Approaches for ISS

Internet-scale sensing systems demand new approaches to dependability that take into account the scale and data-quality requirements of end-user applications. We claim that building an ISS system to conform to the traditional metrics of availability and data consistency is neither feasible nor desirable. Failures and intermittent outages are the norm, rather than the exception. Rather, the focus in ISS should be on scaling to increasing numbers of data sources and end users and graceful degradation of query results as failures occur. Rather than strive to always report the "correct" answer to a query, the ISS platform should provide the user feedback on the fidelity of the query result. Every query result returned by the ISS system should carry with it a set of *quality metrics*.

The quality metrics exposed by an ISS platform can be divided into two categories: data-centric and operational. *Data-centric metrics* are those that directly represent the accuracy, timeliness, or certainty of a given result. In contrast, *operational metrics* represent the internal operation and behavior of the ISS system in processing the input data, such as the network latency experienced by tuples flowing over a query tree.

A wide range of metrics could be supported by an ISS system. There is a clear tension between exposing many low-level details of the query's operation (which may have little meaning to an end user) and providing high-level feedback on data quality (that may be too abstract to be useful). Depending on the query semantics, it may be possible for the system to directly estimate confidence intervals or an error envelope, although in general this requires extensive knowledge of the data sources and query operators.

Two specific quality metrics that we believe will be useful in a wide range of circumstances are defined below:

Harvest: The fraction of data sources represented in an answer to a query [17]. Harvest represents the coverage of a query result and is negatively affected by failures or loss within the query tree. Note that harvest is not directly related to correctness, since a result with a harvest of 100% can still experience errors due to state divergence of operators. However, a high value for harvest indicates good confidence that the answer represents all of the input data sources.

Freshness: The age of the input data tuples represented in the answer. Freshness is negatively affected by network latency, system load, and buffer-induced delays within the query tree. Freshness is bounded below by the network diameter. In addition to reporting the oldest tuple, the *distribution* of the ages of input tuples can provide feedback on the timing variance of operators in the query.

While these metrics are primarily operational, they are straightforward to measure online and somewhat intuitive. Using an appropriate model of the data sources and query semantics, it is possible to translate these values into data-centric quality metrics such as error bounds.

4.1 Towards a new ISS system design

Through initial experiments with a large-scale network monitoring application running on PlanetLab, we have derived a simple set of design principles for architecting a scalable, robust ISS platform. The techniques described here are intended to sidestep the complexity of traditional replication and consistency mechanisms, in recognition of the differing data needs of ISS applications. We are currently developing an ISS platform based on these design principles.

Structured operator replication

The first technique that we employ is *structured operator replication*, in which each operator in the query tree can be replicated either proactively (in anticipation of a failure) or reactively (in response to a failure). Replication can increase harvest and freshness considerably, though at the cost of increased resource consumption (of overlay hosts and network bandwidth). Increased replication also has the potential to aggravate the effects of state divergence.

We propose an approach to structured replication that recognizes the varying impact of failures of different operators in an ISS system. In particular, the *depth* of an operator in the query tree is directly related to its potential impact on harvest; failures of operators at the leaves of the tree only affect a few sources, while failures higher in the tree can greatly reduce harvest. In our approach, operators higher in the tree are given preference for replication, striking a balance between resource requirements and resulting data quality. Operators at the highest levels of the tree are replicated proactively, while lower-level operators may only be replicated following a failure.

Free-running operators

Rather than maintaining strong consistency between operator replicas, we advocate allowing operators to "free run," updating internal state independently as incoming tuples arrive. This approach obviates the need for expensive consistency protocols, although each operator must push its output tuples to *all* of its replicated downstream operators, increasing bandwidth usage. This approach also simplifies operator startup in case of failure: the recovering operator starts with an empty internal state.

The downside to this approach is that an operator's state may diverge from its replica peers due to missing or delayed inputs and operator failure. However, we observe that in ISS queries, typical operators have a finite (and often short) *causality window* that defines the set of past input tuples that affect its internal state. For example, an operator performing a windowed average of the last 30 sec of data has a causality window of only 30 sec. For typical streaming query operators, the causality window is usually the size of the operator's input window, although it may be a more complex function of the operator semantics. In general, an operator with a causality window of w sec will become consistent with its replica peers after running for w sec, assuming all replicas receive all input tuples and no additional failures occur during this window. Therefore, the ISS system uses a form of eventual consistency across operator state without explicit synchronization.

Best-guess reconciliation

A set of replicated operators will experience occasional failures and restarts. As described above, eventual consistency is achieved across the replicas because of the bounded causality window. However, when replicas do diverge, we face the problem of determining which of the results from the replica set represents the "best" answer to push downstream. Our approach is to make use of a *reconciliation* mechanism that filters the values from multiple replicas before passing them into the next operator in the query tree. We are currently investigating a range of reconciliation policies, including:

- Choose the result from the replica with the longest uptime. The intuition here is that as long as the node's uptime exceeds the causality window w, the operator should be reporting the "correct" result, although intermittent losses can still result in errors.

- Choose the result from the replica reporting the highest harvest. This mechanism favors replicas culling input data from a greater number of data sources. A challenge arises in breaking ties if two replicas have similar harvest but non-identical input sets (say, due to network partitions).

- A voting scheme could be used that chooses the result from the majority of replicas in agreement with each other. This approach only works if there happens to be a majority set of replicas reporting identical (or very similar) results. A failure affecting multiple replicas can remove this condition.

- Finally, results from multiple replicas can be combined into a single result, if it makes sense to do so given the semantics of the operator in question. For example, for certain operators, it may be appropriate to take the average or median value across all replicas.

Reconciliation pushes the results reported by a replicated set of operators towards convergence. However, as the number of replicas increases, so does the state divergence across them. A useful operational metric for measuring the effectiveness of reconciliation is *group spread*, which we define as the degree to which replicas differ in their results. A large group spread indicates that the replicas as highly divergent and lowers confidence that the reconciliation is choosing "correct" values. Low group spread implies that replicas are reporting consistent results.

5 Discussion and Conclusion

Collecting and processing vast amounts of real-time data is an important direction for distributed systems research. At the same time, Internet-scale sensing raises new challenges for dependability. The transient and real-time nature of stream data processing differs greatly from distributed systems dealing with persistent state. We argue that ISS systems should be designed to offer feedback to end users on the fidelity and coverage of the results returned by the system, and make use of simple, lightweight replication techniques. This is in stark contrast to current work on replicated distributed systems that rely on heavyweight protocols to achieve consistency.

This paper explores one end of the spectrum of operator replication in stream-processing systems. We believe that given the scale of ISS systems, overall system availability should be achieved by tolerating graceful degradation in the fidelity of the answers produced. This focus motivates the need for lightweight concurrency mechanisms. We have proposed the use of active replication, using free-running operators coupled with a reconciliation mechanism that can be generalized across various application domains. The current proposal for replication does not affect the internal state of the replicas, relying instead upon the bounded causality window to achieve eventual consistency. One possible refinement is to introduce periodic synchronization across operators, which is particularly attractive when causality windows are large.

We are currently developing an ISS platform based on our previous work on the stream-based overlay networks [25]. While many of the underlying techniques for data collection, processing, and query optimization have been explored by our work and others [12, 15, 20, 16], to date no system has been proposed for Internet-scale sensor networking that is designed to handle the scale and robustness requirements outlined in this paper. This is a fertile area for future research and requires taking a broad view of data quality metrics, fault tolerance strategies, and distributed state management.

References

[1] Chatguard. http://www.chatprotection.com/.

[2] Comprehensive test ban treaty organization (ctbto). http://www.ctbto.org/.

[3] Distributed intrusion detection system. http://www.dshield.org/.

[4] Distributed monitoring framework (dmf). http://dsd.lbl.gov/DMF/.

[5] Earth scope. http://www.earthscope.org/.

[6] National ecological observatory network. http://www.neoninc.org/.

[7] Netbait. http://netbait.planet-lab.org/.

[8] Rapid telescope for optical response. http://www.raptor.lanl.gov/.

[9] Skyview: The internet's virtual telescope. http://skyview.gsfc.nasa.gov/.

[10] Sloan digital sky survey. http://www.sdss.org/.

[11] Us national virtual observatory. http://www.us-vo.org/.

[12] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[13] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-tolerance in the borealis distributed stream processing system. In *ACM SIGMOD Conf.*, Baltimore, MD, June 2005.

[14] J. Bartlett, J. Gray, and B. Horst. Fault tolerance in tandem computer systems. In A. Avizienis, H. Kopetz, and J.-C. Laprie, editors, *The Evolution of Fault-Tolerant Systems*, pages 55–76. Springer-Verlag, Vienna, Austria, 1987.

[15] J. Campbell, P. B. Gibbons, S. Nath, P. Pillai, S. Seshan, and R. Sukthankar. Irisnet: an internet-scale architecture for multimedia sensors. In *Proc. the 13th annual ACM international conference on Multimedia*, November 2005.

[16] O. Cooper, A. Edakkunni, M. J. Franklin, W. Hong, S. R. Jeffery, S. Krishnamurthy, F. Reiss, and E. Wu. Hifi: A unified architecture for high fan-in systems. In *Proc. the 30th International Conference on Very Large Data Bases*, August 2004.

[17] A. Fox and E. A. Brewer. Harvest, yield and scalable tolerant systems. In *Proc. the 1999 Workshop on Hot Topics in Operating Systems*, Rio Rico, Arizona, March 1999.

[18] J. Gray. Why do computers stop and what can be done about it? In *Symposium on Reliability in Distributed Software and Database Systems*, pages 3–12, 1986.

[19] Haifeng Yu and Amin Vahdat. The Costs and Limits of Availability for Replicated Services. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[20] R. Huebsch, B. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of pier: an internet-scale query processor. In *Proc. the Second Biennial Conference on Innovative Data Systems Research*, January 2005.

[21] Hyang-Ah Kim and Brad Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *13th Usenix Security Symposium (Security 2004)*, August 2004.

[22] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Network telescopes: Technical report. Technical Report UCB/CSD-00-1096, Cooperative Association for Internet Data Analysis, April 2004.

[23] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *The VLDB Journal*, pages 144–155, 2000.

[24] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating SystemsPrinciples (SOSP-16), Saint Malo, France*, 1997.

[25] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *Proc. the 22nd International Conference on Data Engineering (ICDE'06)*, August 2006.

[26] A. Ray. Oracle data guard: Ensuring disaster recovery for the enterprise. Technical report, An Oracle white paper, March 2002.

[27] G. Simon et al. Sensor network-based countersniper system. In *Proc. ACM SenSys '04*, November 2004.

[28] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proc. Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[29] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proc. the Third ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, November 2005.

[30] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. In *ACM Transactions on Computer Systems*, volume 21, pages 164–206, May 2003.

[31] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the Potemkin virtual honeyfarm. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 148–162, New York, NY, USA, 2005. ACM Press.

[32] H. Wang, D. Estrin, and L. Girod. Preprocessing in a tiered sensor network for habitat monitoring, 2002.

[33] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing, Special Issue on Data-Driven Applications in Sensor Networks*, March/April 2006.