Bulletin of the Technical Committee on

Data Engineering

March 2005 Vol. 28 No. 1 **IEEE Computer Society**

Letters

Special Issue on In-Network Query Processing

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

There are two Data Engineering Bulletin web sites: http://www.research.microsoft.com/research/db/debull and http://sites.computer.org/debull/.
The TC on Data Engineering web page is http://www.ccs.neu.edu/groups/IEEE/tcde/index.html.

Letter from the Editor-in-Chief

International Conference on Data Engineering (ICDE'05)

For one last time, I want to draw your attention to the "call for participation" for ICDE'05 that is on the inside back cover of this issue. ICDE (also called simply the Data Engineering Conference) is sponsored by the IEEE Technical Committee on Data Engineering (TCDE). The 2005 Data Engineering Conference is in Tokyo in April, timed to coincide with the cherry blossoms. I encourage you to find out more about the conference at its web site (http://icde2005.is.tsukuba.ac.jp/), and urge you to attend. You will find it very rewarding, with a great technical program and a wonderful opportunity to see Tokyo and Japan.

The Current Issue

Networks and databases have co-existed for a very long time. We old timers remember CICS (the archetypical TP monitor) and SNA (network architecture), which, together with IMS (the database) formed the heart of IBM's enterprise computing software in the early 1970's. The current world has a much larger diversity of product offerings, from many vendors including startup companies. It is a world of networked computers, grid computing, P2P, sensors, data streams, world-wide web, web services, yada, yada, yada!

This is also a world in which researchers are actively participating. This issue of the Bulletin grew to a rather larger size than is the common case simply because there is so much going on in so many parts of our "database" community. There is interest from database vendors, communications providers, enterprise computing vendors, and a host of web centric companies as well. A cross section of this activity is captured in the current issue.

This issue was very ably assembled by Minos Garofalakis. Minos was very successful in soliciting papers from some of the prime groups doing research in this area. And he shepherded the articles into an issue that truly does represent a snapshot of some of the very latest work in the area. This is a fine example of the role that the Bulletin desires to play, presenting work in progress from leading researchers and practitioners in areas of current interest to our community. I thank Minos for his fine job. I'm confident you will find the issue both engaging and useful.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

As part of the Internet era, most of today's data-management systems need to effectively cope with a networked world. The recent proliferation of sensor networks and Peer-to-Peer (P2P) applications for a variety of data collection and sharing tasks only emphasizes the above thesis, further blurring the traditional boundaries between networking and data management. The efficient operation of the Internet itself relies on data-monitoring applications that continuously gather operational data from numerous points in large ISP networks – clearly, effectively managing the underlying infrastructure and minimizing the monitoring burden on the production network is an important concern here. Compared to conventional distributed databases, this new era of networked data-management systems poses a host of new research challenges, owing to (a) the sheer scale and dynamic nature of distribution (e.g., P2P systems supporting thousands of volatile peers), (b) the large volume of massive, rapid-rate data streams (e.g., large ISPs collecting hundreds of GBs of monitoring data each day), and (c) specific data-source characteristics and limitations (e.g., failure-prone and resource-limited sensor devices). To effectively support a high-level query-processing interface over such systems, it is crucial to intelligently push computation *inside the network*, moving query-processing toward the origin of the data. This requirement represents a major departure from traditional databases, placing issues like routing efficiency, load balancing, and intelligent scheduling of communication at the core of effective query-processing strategies.

The collection of articles in this special issue is only a small sample of different research perspectives and efforts in the vibrant area of *in-network query processing*. The first article by Hellerstein et al. outlines an interesting research vision for a "Network Oracle", a global, real-time monitoring and information infrastructure for the Internet's core state. The authors offer convincing arguments on the importance, timeliness, and feasibility of such an infrastructure, and outline a high-level research agenda with several key challenges for our community. When dealing with distributed data collection and monitoring, communication efficiency is an important concern (e.g., in sensornets where communication is the major source of sensor battery drain). The next four articles discuss different approaches for improving communication efficiency through *approximation*. First, Olston and Widom give an overview of their recent work on approximate replication, an important technique for balancing distributed replica precision against communication efficiency. Second, Deligiannakis and Kotidis discuss data-reduction mechanisms (based on linear regression) and approximate-aggregation techniques for limiting communication overheads in sensornet environments. Third, Kollios et al. propose duplicate-insensitive data synopses for communication-efficient aggregation in sensornets that use broadcast or multi-path routing to improve robustness in the face of failures. Finally, Cormode and Garofalakis outline a general framework and algorithmic approach for continuous, approximate query tracking in distributed settings.

The article by Deshpande et al. outlines the design of their *Sensor Control System*, that brings together a number of interesting ideas from statistical models and control theory to enable efficient sensornet management. Shifting to the realm of P2P systems, the next two articles discuss challenges in effective query processing over large, widely-distributed, and dynamic collections of peers: Ganesan and Garcia-Molina propose novel algorithms for mapping data to peers in a manner that ensures efficient, dynamic load balancing and retains locality in the key space for efficient range and similarity queries (unlike conventional DHTs); Felber et al. propose a topology-aware DHT design for peer data management as well as novel indexing schemes for structured XML content and queries. Topology awareness is also the key theme in the article by Ahmad et al. that discusses the issues of query-operator placement and distribution for Internet-scale stream processors. Finally, Gnawali et al. discuss their experiences with implementing SQL functionality on top of a generic sensornet routing platform.

I truly hope that you will find this collection of articles as enjoyable and insightful as I did, and that this special issue will help stimulate new research ideas in this exciting area, at the intersection of networking and data management. My sincerest thanks to all the authors for their contributions.

Minos Garofalakis
Bell Labs, Lucent Technologies
Murray Hill, NJ 07974, USA

The Network Oracle

Joseph M. Hellerstein[* †] Vern Paxson[‡] Larry Peterson[§] Timothy Roscoe[†]
Scott Shenker[* ‡] David Wetherall[¶]

[*]UC Berkeley [†]Intel Research, Berkeley [‡]ICSI [§]Princeton University [¶]University of Washington

Abstract

This paper sets out a high-level research agenda aimed at building a collaborative, global end-system monitoring and information infrastructure for the Internet's core state. We argue that such a system is beneficial, feasible and timely, representing an important area for engagement across database and networking technologies. We start by hypothesizing the benefits to the Internet of a "Network Oracle" that could answer real-time questions about the global state of the network. We then argue that database and networking technology should make it possible to provide a useful approximation of such an oracle today, gathering information from a large number of end hosts and delivering useful views to each end system. We further argue that this can and should be done in a decentralized fashion. We provide an outline of such a system: it employs sensing agents, along with a distributed query/trigger/dissemination engine, and possible attractive end-user applications. A key point of our discussion is the timeliness and importance of a grassroots agenda for Internet monitoring and state-sharing. While significant social and economic barriers to deploying a centralized, public Internet monitoring infrastructure exist, there are corresponding social and economic incentives for a collaborative approach.

1 Vision

There has been increasing interest in recent years in topics at the nexus of distributed databases and core networking, and various agendas have been laid out for exploring synergies ([18, 11, 23, 13]). To date, however, there has been little work on rethinking the Internet architecture in response to these technical directions.

In this paper, we lay out one such agenda in broad terms. We propose that the database, networking, and distributed systems research communities collaborate in building a global end-system monitoring and information infrastructure for the Internet's core state.

Our discussion begins with a thought experiment. Setting aside concerns about social and technical barriers, suppose there existed a Network Oracle: a queryable object that any end-system on the Internet could use to immediately receive information about global network state, from the recent past to real-time updates. This state could include complete network maps (including addressing realms and NAT gateways), link loading, point-to-point latency and bandwidth measurements, event detections (e.g., from firewalls), naming (DNS, ASes, etc.), end-system software configuration information, even router configurations and routing tables.

This is considerably more information than is available to end-systems today. The existence of the Network Oracle would allow end-systems to make more sophisticated decisions about every aspect of their interaction

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

with the network: the parties they communicate with, the routes and resources they use, and the qualities of the various actors in the communication chain. In Section 2 we give concrete examples of how end-systems and end-users could benefit from this information.

How far away from this vision are we today? Network monitoring is not a new activity. Many parties collect significant information in today's Internet, including carriers and large IT departments. However, the information collected is by no means comprehensive; it is chosen with relatively narrow goals in mind, usually with a focus on backbone traffic engineering and academic networking research. Also, since the data is typically collected "in the middle" of the network, it only captures packets as they traverse those links; it misses significant information about the properties and traffic in small Intranets, in switched subnets of large Intranets, in WiFi communities, and in similar rich and evolving "microclimates" at the edges of today's Internet.

Furthermore, current network monitoring systems focus on data collection, but ignore public-access query or dissemination facilities. Information gathered by today's network monitors is generally available neither to end-users nor their protocols or applications. This places inherent limits on innovation. Making this information widely available in near-real-time can significantly change the protocol and distributed system design landscape, in a way that offline centralized analysis cannot. Today's Internet was designed under the assumption that it is not feasible to gather and disseminate such information at scale, and researchers and developers of end-user network applications constrain their design space accordingly. We argue below that this assumption no longer holds. Eliminating these constraints can open up new opportunities for significant innovation in network functionality and robustness – opportunities that span traditional boundaries between data management, networking, and distributed systems.

2 What For?

We conjecture that a Network Oracle would have multiple benefits for various parties at differing timescales. These include (a) social benefits such as raised awareness of security risks and the importance of end-system maintenance, (b) medium-term network engineering benefits including localization of performance problems and identification of malevolent actors, and (c) long-term design opportunities including the development of end-to-end network protocols and distributed applications that leverage the global information. In this section we present several examples.

Performance Fault Diagnosis. There are very few options for diagnosing performance faults within the Internet today. When a site loses connectivity, it is possible to use tools such as traceroute to determine whether the problem is likely local or remote. But when the Internet performs poorly from a given site it is difficult to determine whether the problem lies with that site or elsewhere, and whether there is in fact a problem to be corrected rather than a temporary overload. The key difficulty is that there is often no baseline that the site can use to gauge what level of performance it should obtain.

The Network Oracle we envision addresses this difficulty directly by allowing a site to compare and contrast its performance with that of other sites, both in the past and at the given moment in time. For instance, a site might note suspect destinations for which it obtains persistently low performance, and query the oracle for these suspect destinations. If these destinations subscribe to the oracle, then current and historical information will be available to assess the overall level of performance of the destination and whether it matches past performance. Recent falloffs suggest a problem with the site; consistent performance suggests a problem elsewhere, and if all other sites reporting information about the suspect destination are receiving equally poor performance, then the problem likely lies with the site itself which is simply overloaded. This same test can be applied to regions of the network between the site and destinations. The effect is to narrow the region of performance faults, facilitating correction, as well as to identify alternative paths that offer improved performance. It is the sharing of performance data across sites facilitated by the oracle that makes this possible.

Tracking attacks. Hosts on the Internet experience incessant attack [20]. This engenders an edginess in

Internet users, much of which boils down to questions like: *(i)* Is this remote host attempting to contact me a bad guy? *(ii)* Am I being targeted, or just enduring what everyone else endures? *(iii)* Is there a new type of attack going on? *(iv)* Is something happening on a global Internet scale?

By providing insight into activity experienced by one's peers and Internet sites in general, the Network Oracle can help answer questions both about types of activity, and which hosts have been seen doing what. This kind of shared information is not only interesting, but potentially actionable. As a simple example, studies have demonstrated that very few source IPs generate a significant fraction of port scans [32]. A real-time distributed query could compute the global "worst offenders" list, and allow firewalls at the endpoints to adaptively change their packet filters to track changes in the list. Placing the Network Oracle at the core of a global security agenda raises major research issues concerning attacks on and subversion of the infrastructure and its data, and in general ensuring that the information has an "actionable" degree of fidelity. We return to this point in Section 5.

Network Routing Protocols. The current routing infrastructure computes routing tables that, to a first approximation, are applied to all flows headed to the same destination. However, it is clear that no uniformly applied routing protocol, no matter how well designed, can accommodate every flow's policy and/or performance requirements. Starting with source routing, and continuing with more recent designs such as NIRA [31] and TRIAD [12], there is a large literature about mechanisms that would allow flows to choose their own route. Research has typically focused on mechanisms for expressing and implementing the desired route, but much less attention has been paid to how flows (or the hosts acting on their behalf) could determine which route would best serve their needs. If only a very small fraction of flows were making such individualized route choices, then fairly primitive, and bandwidth-expensive, route exploration mechanisms could be used. But if source-specified routing became commonplace, then a more scalable approach would be needed. In particular, one would want the information used to decide routes to be shared, rather then individually discovered. The Network Oracle approach, in which general classes of information are gathered and made available, could provide a virtual repository for the relevant information. Initial work (e.g., [16]) suggests that such a repository, along with a general querying facility, could be used for such route computation.

Adaptive Applications. A natural extension of having collected a wealth of information about the health of the network is for applications to adapt; to react to this information by selecting alternative protocols, alternative routes, or even alternative sources for the content they are trying to access. While adaptation might happen purely at the end system (e.g., selecting a variant of TCP most suitable for the current end-to-end path), it is easy to imagine the emergence of way-stations that help end systems avoid network trouble spots and rapidly recover from failure [25, 1], as well as more globally coordinated network services that are able to distribute network load over a wider swath of the Internet [19, 9].

An Internet Screensaver. Distributed computation projects like SETI@Home [2] have demonstrated that individuals will contribute private computing resources to the common good – particularly if rewarded with the right combination of entertainment (e.g., interesting screensavers) and community-building tools (e.g., the SETI "leader board" listing the top contributors). Given mounting press and public concern about Internet viruses and worms, the time seems ripe to build an Internet Screensaver – a peer-to-peer application of end-hosts monitoring the network for security events and performance anomalies. Such an application could have multiple benefits. First, it could serve as an attractive, sizable testbed for a prototype Network Oracle, measuring the network from the richness of a variety of "last-mile" endpoints. Second, if properly designed it could engender a unique culture of enlightened vigilance, with client machines swapping notes on anomalous traffic for a variety of purposes. For example, end-users could set up social networks for "community watch", actively probing each other's machines for vulnerabilities. They could swap notes on passively-monitored undesirable traffic (worms, port-scans, spam), to help configure firewall rules. They could compare performance across ISPs. While they may not provide the most accurate measurements or the most effective security measures, these techniques would give Internet users insight into their own experience and incentive to control it more carefully.

Serendipity. While the previous scenarios described practical uses for the Network Oracle, its relevance to networking and database research should not be neglected. The network measurement literature is huge, and continues to grow at an astounding rate, so the field is hardly lacking for interesting questions to ask and relevant data to answer them. Database technologies like streaming query systems are finding some of their most compelling applications in these contexts. However, much of the networking research analyzes data that was gathered with the specific question in mind, or at least in a specific context with limited scope; for example, a routing study might collect BGP routing tables but would be unlikely to also simultaneously collect data from firewalls or application logs at points nearby the relevant routers. Thus, current measurement studies have a naturally limited ken which may prevent certain networking questions from being answered, or even being asked. This presents a chicken-and-egg problem with respect to database research ideas as well; the lack of Internet-scale distributed database products prevents network researchers from asking these kinds of questions, which in turn leaves (some) database researchers and developers unmotivated to design systems for this purpose. The result is that a ”virtuous cycle” of collaboration between the communities has been relatively slow to emerge. We hope and anticipate that, should it be built, a general-purpose Network Oracle could open up surprising new connections – both for research and for application.

3 Why Now?

The vision of making detailed, relevant information about global network state available at all endpoints was not realistic several years ago. We argue that the vision is now realistic, and is becoming easier.

The Social Case. Our first point is social, not technical. A widespread monitoring infrastructure will only be achieved (and sustained) if the sensors are widely desired and deployed. We believe that the time is ripe for encouraging end-users to install software to help improve the Internet.

We are at a point in Internet history where the need for protective action has hit the common consciousness. Spam, viruses and worms have led even unsophisticated users to take non-trivial technical steps to try and ameliorate these problems. Simultaneously, news stories about identity theft, Internet credit card fraud and the U.S. Patriot Act have raised popular sensitivity to the importance of the Internet in their lives. We believe that many users are not only willing and ready, but thirsty to install applications that can improve their trust in the Internet, and better inform them of risks.

Working against this, of course, is concern with individual privacy – often argued to be in tension with safety. A recent article [4] suggests that the strategic need to secure the Internet, combined with the ease of surveillance as a tool for doing so, will lead to an Internet that is a “broadly surveilled police state”. Is it possible for the community to be vigilant without compromising their privacy?

This can be construed as a technical or social challenge; ultimately it is both. The technical agenda in privacy-preserving information sharing is heating up (see the proceedings of any database conference) but is in its infancy, and it may be years before practical guarantees can be made about the privacy of information in a Network Oracle. We conjecture that the deployment of an early-stage Network Oracle need not wait for such technology to mature. Many users may be willing to opt into a decentralized Network Oracle, even if they would not do so with a centralized system offered by a large organization (commercial or non-profit), based on the user’s presumption that no single malicious individual would see much of their information. This soft attitude toward privacy is surprisingly widespread – witness the number of people who identify their IP address to unknown peers in order to acquire copyrighted material illegally, or the often vigorous participation in pseudo-anonymous fora such as newsgroups and chat rooms. The social issue of privacy is less contentious when large interests (companies, governments, standards bodies) are removed from the debate. We explore the implications of the requirement for decentralization below.

Technology Trends. Our second point concerns the technical feasibility of disseminating adequate information about the state of the Internet. We believe that technology trends are working in our favor.

We conjecture that *the "metadata" of the Internet's behavior is shrinking relative to data being shipped across the Internet.* The bandwidth required to ship useful measurement data around the network to end users, as a fraction of the total bandwidth available to end users, is decreasing. This makes it more attractive to start making measurement data available to all end systems.

We note that while both bandwidth and flow size are increasing, the data required to describe a flow is not. Available network bandwidth is increasing, both in residential (DSL, Cable Modems) and business settings (100–1000 Mbps Ethernet), a trend even more pronounced in technologically advanced countries like South Korea and Japan. At the same time, the size of data objects transferred through these larger pipes is also increasing (video streams, large downloads, VoIP sessions, even web-page objects are becoming larger), while the amount of data required to describe this activity is staying relatively constant – it scales with the number of end systems, rather than the available link capacity. While there are counter-examples (such as the proliferation of instant-message traffic), in general the principle holds: the amount of data one needs to know at endpoints to make informed decisions is growing more slowly than the access bandwidth at these end systems.

Moreover, processing power in end systems (indeed, systems anywhere in the network) is outstripping the increase in available network bandwidth. We are a long way from the days when 80% of the CPU cycles of an Alto were devoted to running a 3Mb/s Ethernet interface. A modern IA32 server can comfortably handle a 1Gb/s Ethernet interface at line rate with most of its CPU cycles to spare. The conclusion is that systems can afford to give much more consideration to traffic (in particular, its temporal characteristics) than is typically assumed in protocols and implementations. A corollary is that users with more cycles to throw at the problem may be capable of extracting more value (per b/s) from their network link.

Technology Innovations. Our final point is that we are seeing the appearance of technologies that can take this opportunity and make it useful. In recent years the building blocks for delivering a modest but non-trivial approximation of the Network Oracle are falling into place, with contributions from a number of research communities: for example, content-addressable networks, distributed query processing, statistical data reduction techniques, and statistical machine learning techniques. We return to these in Section 5.

4 Why Us?

The research community is uniquely positioned to realize an initial approximation to the Network Oracle. Other parties have no incentive to pursue this agenda: while the long-term benefits are significant, neither network providers nor equipment vendors gain obvious advantage from investing in such an endeavor at this stage. By contrast, this is an opportunity for researchers: while the Network Oracle vision represents a shift in emphasis for some networking research groups, substantial new research agendas and synergies exist in this direction.

Historically, ISPs have been resistant to the sharing of measurement information, except as marketing and sales aids. Where they have instrumented their networks, it has been with the internal goal of traffic engineering. Carriers have little interest (for sound commercial reasons) in disseminating end-to-end performance or security measurements. Consequently, equipment vendors have little interest in the problem; indeed, a shared, global management infrastructure may be a disruptive technology that threatens their market position.

In short, we are in a situation where commercial benefits of change are indirect but communally valuable, however the commercial threats are direct. The Network Oracle is not going to happen commercially at first. However, the situation is very different for the networking research community. In particular, the fields of network measurement and security have much to gain from realizing a Network Oracle.

Internet measurement in academia has been heavily restricted by the forms of measurement to which it has access, with rising security and privacy concerns making this increasingly difficult rather than the situation easing over time. This often leads to work that is implicitly driven and shaped by (shrinking) measurement opportunities. Internet security research, on the other hand, has struggled with the rise of rapid, automated attack technology, the loss of a defensible "perimeter" with mobile Internet devices, and an inability to adequately track and share information about miscreants. Both communities stand to gain by realizing a Network Oracle.

The Network Oracle would act as a rallying point where unnecessarily divergent research thrusts can be brought back together: network measurement, security, distributed systems, distributed databases, and statistical methods. Network measurement researchers would gain access to much larger, shared datasets than available to them today. Researchers in data management and analysis would achieve renewed relevance by taking their technologies out of endpoint data systems and rethinking them at Internet-scale. Similarly, security researchers would gain the ability to directly tackle problems at global scale, and with global resources.

Research in turn performs a bootstrapping function. The Network Oracle puts monitoring, diagnosis, and measurement functionality directly into end systems. If it can demonstrate value to end users in this way, it provides a path by which many measurement, monitoring, and diagnostic techniques can achieve a critical deployment mass without first requiring productization. Organizations with strategic interests in the deployment of such techniques can thus derive immense benefit from "plugging into the information substrate." Products follow deployment, rather than the other way around.

5 How?

The Network Oracle will not appear overnight. Here we highlight some challenges in realizing this vision.

Adoption and Uptake. The Network Oracle is characteristically a shared infrastructure. For reasons we have already outlined, we expect, at best, limited data from ISPs to start with. Consequently, the Network Oracle will rely at first on end-systems for both data sources and applications (consumers of data).

Projects like NETI@home [24] and DIMES [26] are exploring large-scale network measurement from end hosts, and the DShield project [8] warehouses firewall data sent in from many sources. Bundling a sensor with a global query visualization like an Internet Screensaver seems like a good incentive here, with a quid-pro-quo opt-in model: for the features you publish, you can see distributed results involving such features.

Moreover, another rich source of endpoints is "dark" IP address space, as monitored by Network Telescopes. While the traffic at these addresses is idiosyncratic, it is of interest to many parties, and could serve as "seed" data to populate the screensavers of early adopters.

The Network Oracle must also be able to locate and interface with large curated databases of information as well as distributed real-time sources. This includes slowly-changing network data (e.g., WHOIS) and archival data warehouses of traffic information (e.g., RouteViews).

Finally, we note that the definition of end systems expands to include large distributed services like P2P networks and CDNs. Public-minded instances of these services can share interesting traffic data with the Network Oracle, along the lines of PlanetSeer [33].

Architecture and Deployment. We envision a healthy diversity of popular sensors targeted at different network features, sharing an integrated query processing backplane. The function of this backplane is the processing (filtering, summarization, correlation) of data from many sources, and the delivery of relevant results to interested end systems. The amount of source data involved means that computation needs to occur "in the network", along the network path from the data source to the consumer.

The deployment of this computational infrastructure could evolve in a number of ways. We do not foresee a centrally administered solution succeeding. One option is to have a consortium manage a well-provisioned infrastructure, conceivably federated in nature like the Internet today. The recent success of PlanetLab is encouraging in this regard, but it remains unclear whether a consortium can maintain a production service like the Network Oracle without sustainable, measurable benefit to the institutions hosting the machines.

An alternative is an organic p2p deployment, with the query processor being bundled with the sensors. This is easier to deploy than the "distributed platform" approach, and is in an important sense more self-sustaining: the system remains up as long as sensors are deployed. It has a populist flavor that may allay some concerns about privacy and control. Of course, robustness in the p2p approach raises many technical challenges of scale, management, and resistance to attack or manipulation.

Technical Approach. Challenging as the Network Oracle may seem from a technical standpoint, we think most of the pieces of the puzzle have fallen into place in recent years, and a focused research effort could bring together a usable system in the spirit of a Network Oracle in short order. We present a brief selection here.

First, distributed query processing and content-addressable networks (DHTs) are getting much better at providing the right information to the right place at the right cost. P2p systems like PIER [14] push query processing into the network to reduce the data shipped during query answering. In addition to queries, triggering functionality akin to active databases is also possible, even with conditions that span distributed data [15]. A key tenet is the *data independence* that underpins relational databases: the physical organization (e.g., network location) of data should be separate from the logical data model and query interface [13]. Queries can be posed on data regardless of its location, and data can be reorganized without requiring changes in queries or applications that embed them.

DHTs are the first technology that provide this kind of data independence at Internet scale. PIER's "flat" DHT infrastructure is potentially a better fit for the Network Oracle than hierarchically organized systems; it does not depend on a small number of "roots" for the information and processing as DNS does, nor does it restrict the system to queries that traverse the hierarchy (as does, e.g., Astrolabe [28] or IrisNet [7]).

Second, the practical application of statistical methods in systems has been maturing over the last decade, in particular in computing approximate answers to queries (e.g. [3, 10]). Early exploration of these techniques in distributed settings are promising (e.g., [17, 6]). Also, distributed implementations of techniques like graphical models are emerging in the machine learning community, largely in the sensor network space (e.g., [21]). These approaches model statistical correlations in data, and use the models to predict data values and quantify uncertainty; this is useful both for predicting missing data, and for "cleansing" noisy acquired data.

Third, the security and trustworthiness of the Network Oracle implies several key challenges: validating the fidelity of data and computations, managing resource consumption at the end-hosts, providing accountability of misbehaving components, ensuring that the system itself is not used as an attack platform, and a viable and enforceable privacy framework. Security in p2p systems has been a topic of interest in recent years, with progress being made on topics including self-certifying data, secure routing, and fair sharing of work (e.g. [29]).

Finally, the research community now has the resources to do non-trivial test deployments of global-scale systems in controlled, repeatable "laboratory" settings [30, 27] and more permanently in the real Internet [22].

We view the development and deployment of a limited-function but useful Network Oracle as more actionable – and hence more fertile – than the recent proposal for a "knowledge plane" [5]. That vision is rather more broad, but also more vague. We favor an approach centered on workable designs and working implementations, which can spur organic community collaboration and follow-on work.

6 Conclusion

In speculating about a Network Oracle, our goal is not to map out an unattainable research ideal. On the contrary, a useful approximation of the vision is broadly desirable, and eminently feasible.

A research agenda for the community in this direction must encompass a range of areas, including overlay routing, query processing and database management, network measurement techniques, distributed intrusion detection, distributed statistical methods, and a broad set of issues in security and privacy. To achieve impact, this research must be informed and complemented by a thoughtful strategy for gaining and sustaining uptake in the system infrastructure. Such an effort would benefit significantly from the research community rallying around the problem, seeding the space with interesting data streams and useful compute resources, and working together toward common techniques and protocols.

References

[1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *ACM SOSP*, 2001.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *CACM*, 45(11):56–61, November 2002.

[3] D. Barbará, *et al.* The New Jersey Data Reduction Report. *IEEE Data Eng. Bull.*, 20(3), 1997.

[4] S. Berinato. The future of security. ComputerWorld, `http://www.computerworld.com/printthis/2003/0,4814,88646,00.html`, December 2003.

[5] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A Knowledge Plane for the Internet. In *ACM SIGCOMM*, 2003.

[6] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

[7] A. Deshpande, S. K. Nath, P. B. Gibbons, and S. Seshan. Irisnet: Internet-scale resource-intensive sensor services. In *SIGMOD*, 2003.

[8] DShield: Distributed Intrusion Detection System. `http://www.dshield.org/`, June 2004.

[9] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *USENIX/ACM NSDI*, 2004

[10] M. Garofalakis and P. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, 2001. Tutorial.

[11] S. D. Gribble, A. Y. Halevy, Z. G. Ives, M. Rodrig, and D. Suciu. What can database do for peer-to-peer? In *WebDB*, 2001.

[12] M. Gritter and D. R. Cheriton. An architecture for content routing support in the internet. In *USITS*, 2001.

[13] J. M. Hellerstein. Toward network data independence. *SIGMOD Record*, 32(3):34–40, 2003.

[14] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *VLDB*, 2003.

[15] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *HotNets-III*, Nov. 2004.

[16] B. T. Loo, J. M. Hellerstein, and I. Stoica. Customizable routing with declarative queries. In *HotNets-III*, Nov. 2004.

[17] S. Nath, P. B. Gibbons, Z. Anderson, and S. Se. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys*, 2004.

[18] J. C. Navas and M. J. Wynblatt. The network is the database: Data management for highly distributed systems. In *SIGMOD*, 2001.

[19] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the web: An open proxy's view. In *HotNets-II*, Nov. 2003.

[20] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. In *IMC*, 2004.

[21] M. A. Paskin and C. E. Guestrin. Robust probabilistic inference in distributed systems. In *UAI*, 2004.

[22] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *HotNets-I*, 2002.

[23] S. Shenker. The data-centric revolution in networking. In *VLDB*, 2003. Keynote talk. `http://pier.cs.berkeley.edu/shenker-vldb.ppt`.

[24] C. R. Simpson Jr. and G. F. Riley. NETI@Home: A Distributed Approach to Collecting End-to-End Network Performance Measurements. In *Passive and Active Measurement Workshop (PAM)*, 2004.

[25] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *ACM SIGCOMM*, 2002.

[26] The DIMES Project. `http://www.netdimes.org/`, June 2004.

[27] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, , and D. Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI*, 2002.

[28] R. van Renesse, K. P. Birman, D. Dumitriu, and W. Vogel. Scalable management and data mining using astrolabe. In *IPTPS*, 2002.

[29] D. Wallach. A Survey of Peer-to-Peer Security Issues. In *Intl. Symp. on Software Security*, 2002.

[30] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, 2002.

[31] X. Yang. Nira: a new internet routing architecture. In *SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, 2003.

[32] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *SIGMETRICS*, 2003.

[33] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. Planetseer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, 2004.

Efficient Monitoring and Querying of Distributed, Dynamic Data via Approximate Replication

Christopher Olston
Carnegie Mellon University

Jennifer Widom
Stanford University

Abstract

It is increasingly common for an application's data to reside at multiple disparate locations, while the application requires centralized access to its data. A simple solution is to replicate *all relevant data at a central point, forwarding updates from master copies to replicas without any special processing or filtering along the way. This scheme maintains up-to-date centralized data, but incurs signficant communication overhead when the data is highly dynamic, because the volume of updates is large. If communication resources are precious, communication can be reduced by prioritizing and filtering updates inside the network, at or near the data sources. When updates are dropped, the replicas become* approximate *rather than exact. Fortunately, many real-world applications involving distributed, dynamic data can tolerate approximate data values to some extent, so* approximate replication *is an important technique for balancing replica precision against the communication resources to achieve it.*

This paper studies the problem of making efficient use of communication resources in approximate replication environments. After motivating and formalizing the problem, high-level descriptions of several complementary solutions are provided. The details of these solutions are found in previous papers by the authors, which are referenced here. This paper is intended to serve primarily as an introduction to and roadmap for the authors' prior work on approximate replication, as well as providing a significant bibliography of related work.

1 Introduction

In distributed environments that collect or monitor data, useful data may be spread across multiple distributed nodes, but users or applications may wish to access that data from a single location. One of the most common ways to facilitate centralized access to distributed data is to maintain copies of data objects of interest at central locations using *replication*. In a typical replication environment, illustrated abstractly in Figure 1, a central *data repository* maintains copies, or *replicas* of data objects whose *master copies* are spread across multiple remote and distributed *data sources*. (In general there may be multiple data repositories, but to simplify exposition we focus on a single repository.) Replicas are kept synchronized to some degree with remote master copies using communication links between the central repository and each source. In this way, querying and monitoring of distributed data can be performed indirectly by accessing replicas in the central repository.

While querying and monitoring procedures tend to become simpler and more efficient when reduced to centralized data access tasks, a significant challenge remains: that of performing data replication efficiently and effectively. Ideally, replicas of data objects at the central repository are kept exactly consistent, or synchronized, with the remote master copies at all times (modulo unavoidable communication latencies, of course). However,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

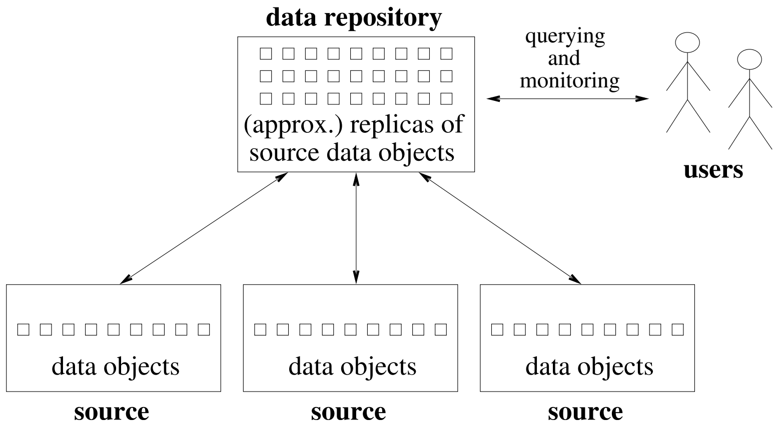


Figure 1: Abstract replication architecture.

propagating all master copy updates to remote replicas may be infeasible or prohibitively expensive: data collections may be large or frequently updated, and network or computational resources may be limited or only usable at a premium.

Situations where exact replica consistency is infeasible or only feasible at excessive expense can be found in many contexts. As one example, in video conferencing applications (*e.g.*, [6]), the viewer screen frame-buffer can be thought of as containing replicas of video data generated by remote cameras. Since streaming video data can be very large, it often becomes necessary to allow some staleness on parts of the screen. As another example, consider the important problem of monitoring computer networks in real-time to support traffic engineering, online billing, detection of malicious activity, etc. Effective monitoring often requires replicating at a central location a large number of measurements captured at remote and disparate points in the network. Attempting to maintain exact consistency can easily result in excessive burden being placed on the network infrastructure, thus defeating the purpose of monitoring [7], but fortunately network monitoring applications do not usually require exact consistency [17]. As a final example, consider the problem of indexing the World-Wide Web. Keeping an up-to-date Web index requires maintaining information about the latest version of every document. Currently, Web indexers are unable to maintain anything close to exact consistency due to an astronomical number of data sources and data that is constantly changing.

The infeasibility of exact replication in these environments and others is due in large part to the potentially high communication costs incurred. Communication cost tends to be of significant concern in many distributed environments, either because the bandwidth available on the network links is limited (relative to the size and update rate of the data collection), or because network resources can only be used at some premium. This premium for network usage may stem from the fact that increased congestion may cause service quality degradation for all applications that use the network. Alternatively, the premium may be manifest as a monetary cost, either in terms of direct payment to a service provider or as a loss of revenue due to an inability to sell consumed resources to others. As a result of communication resources being a valuable commodity, we have seen that in the applications described above (video conferencing, network monitoring, and Web indexing), maintaining replicas exactly synchronized with master copies cannot be achieved when data volumes or change rates are high relative to the cost or availability of bandwidth capacity.

1.1 Approximate Replication

In many applications such as the ones described above, exact consistency is not a requirement, and replicas that are not precisely synchronized with their master copies are still useful. For example, approximate readings from meteorological sensors often suffice when performing predictive modeling of weather conditions. In network security applications, "ball-park" estimates of current traffic levels can be used to detect potential denial-of-service attacks. Since exact data is often not a requirement in applications that rely on replication, it is common practice to use inexact replica consistency techniques such as periodic refreshing to conserve communication cost. We use the term *approximate replication* to refer collectively to all replication techniques that do not ensure exact consistency. Most existing approximate replication techniques for single-master environments can be classified into one of two broad categories based on the way they synchronize replicas[1]:

Periodic pushing: Sources propagate changes in master copies to the central data repository periodically, sometimes in large batches.

Periodic pulling: The central data repository accesses remote sources periodically to read master copies of data objects and update local replicas as necessary.

One motivation for performing periodic pushing is that one-way messages can be used in place of more expensive, round-trip ones. Also, sources can control the amount of their local resources devoted to replica synchronization. Periodic pulling, on the other hand, has the advantage that sources are not required to be active participants in the replica synchronization protocol. Instead, they need only respond to data read requests from the central repository, a standard operation in most environments. A principal feature shared by both these approaches is that the cost incurred for consumption of communication resources is bounded and controllable, which is not in general the case with exact synchronization methods. However, periodic pushing or pulling does not necessarily make good use of communication resources for the following two reasons:

1. Communication resources may be used wastefully while refreshing replicas of data objects whose master copy has undergone little or no change.

2. When the master copy of a data object undergoes a major change that could be propagated to the remote replica relatively cheaply, there may be a significant delay before the remote replica is refreshed to reflect the change.

1.2 Precision-Performance Tradeoff

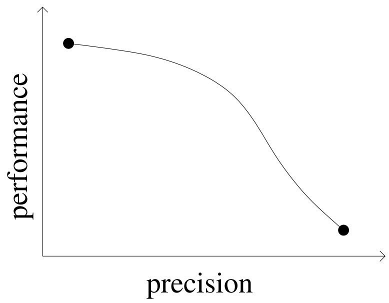
We study the problem of making better use of communication resources in approximate replication environments than approaches based on periodic pulling or pushing. Our work begins with the observation that a fundamental tradeoff exists between the communication cost incurred while keeping data replicas synchronized and the degree of synchronization achieved. We refer to this characteristic property as the *precision-performance tradeoff*, illustrated in Figure 2, where *precision* is a measure of the degree of synchronization between a data object's master copy and a remote replica, and *performance* refers to how sparingly communication resources are used (*i.e.*, the inverse of communication cost). When data changes rapidly, good performance can only be achieved by sacrificing replica precision and, conversely, obtaining high precision tends to degrade performance.

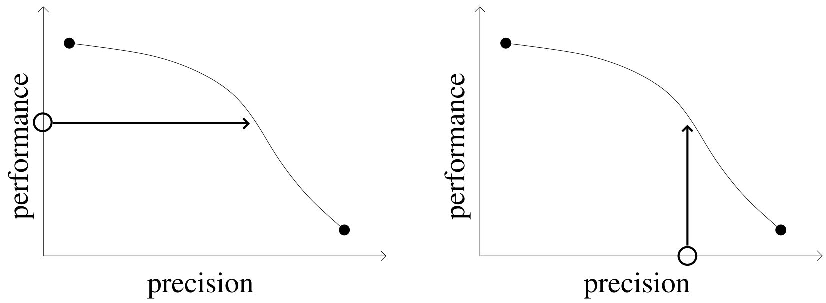
Since there appears to be no way to circumvent the fundamental precision-performance tradeoff, we propose the following two tactics for designing synchronization algorithms for replication systems:

(a) Push the precision-performance curve as far away from the origin as possible for a given environment.
(b) Offer convenient mechanisms for controlling the point of system operation on the tradeoff curve.

Tactic (a) leads us to focus primarily on push-based approaches to replica synchronization, because they offer the opportunity for the best precision-performance curves, *i.e.*, more efficient use of communication resources,

[1]Depending on the environment, it may not be practical or possible for sources to communicate with each other, so we assume that such communication is not allowed and synchronization of replicas is performed directly between each source and the central repository. We do not study environments amenable to efficient intersource communication in this paper.





Performance Fixed/Maximize Precision *Precision Fixed/Maximize Performance*

Figure 2: Precision-performance tradeoff. Figure 3: Strategies for managing the precision-performance tradeoff.

compared with pull-based approaches. (We verify this claim empirically in [14].) The reason for this advantage is that sources are better equipped than the central repository to make decisions about synchronization actions since they have access to the data itself and can obtain accurate precision measurements. Furthermore, our work uses metrics of replica precision that are based on content rather than solely on metadata. Metadata-based metrics have been used to drive synchronization policies in approximate replication environments (see Section 4 covering related work), and are usually intended to emulate an underlying metric based on content. For example, metadata metrics based on temporal staleness or number of unreported updates reflect an attempt to capture some notion of the degree of change to the content. If an appropriate content-based metric is used instead, precision can be measured directly, potentially leading to higher quality synchronization and precision-performance curves farther from the origin, as desired.

Tactic (b) leads us to study two ways to offer users or applications control over the position of system operation on the precision-performance tradeoff curve:

1. Users specify a minimum acceptable performance level (*i.e.*, fix a y-axis position in Figure 2), and the replication system attempts to maximize replica precision automatically while achieving the specified level of performance.

2. Users specify a minimum allowable precision level (*i.e.*, fix an x-axis position in Figure 2), and the replication system attempts to maximize performance while meeting the precision requirement provided.

In each of these complementary strategies, the user (or application) fixes the position of system operation along one dimension (precision or performance), and the system is expected to maximize the position along the other dimension. These converse strategies for establishing a point of system operation on the precision-performance curve are illustrated abstractly in Figure 3.

In our prior work [11, 12, 13, 14] we have studied the two strategies introduced above for controlling the operating point of a replication system in terms of precision and performance. In particular, we have proposed methods by which users or applications may constrain one dimension of the precision-performance space, and studied algorithms for maximizing the position along the other dimension. Our work forms the basis for offering a resource-efficient, user-controllable tradeoff between precision and performance while monitoring and querying distributed, dynamic data. In this paper we provide a high-level overview of this work (Sections 2 and 3), along with a summary of related work by others (Section 4).

2 Model, Assumptions, and Objectives

Recall that master copies of data objects are maintained at one or more distributed data sources (Figure 1). Consider a data object whose master source copy O undergoes updates over time. Let $R(O)$ represent the (possibly imprecise) replica of O at the central repository. Let $V(O,t)$ represent the value of O at time t. The value of O remains constant between updates. Let $V(R(O),t)$ represent the value of $R(O)$ at time t. Object O can be *refreshed* at time t_r, in which case a message is sent to the repository, and the replicated value is set to equal the current source value: $V(R(O),t_r) = V(O,t_r)$.

For simplicity, we assume that the communication latency between sources and the central repository is small enough to be neglected. However, the techniques we have developed can tolerate nonnegligible latencies, as discussed in [10]. We do assume there is adequate space at the central repository to store all replicas of interest. We also suppose that nodes are at all times connected to the network, and that the infrastructure is robust, *i.e.*, that node failures, network partitions, etc. are infrequent. Coping with failures or disconnections in the context of this work is not addressed in our work.

We now define the dual objectives of maximizing precision or performance in more detail.

2.1 Maximizing Precision

For strategy (1) in Section 1.2, the goal is to maximize replica precision. Precision is quantified using a simple and general metric called *divergence*: The divergence between a source object O and its replicated copy $R(O)$ at time t is given by a numerical function $D(O,t)$. When a refresh occurs at time t_r, the divergence value is zero: $D(O,t_r) = 0$. Between refreshes, the divergence value may become greater than zero, and the exact divergence value depends on how the master source copy relates to the replica. There are many different ways to measure divergence that are appropriate in different settings; see [14] for more discussion.

For the purpose of measuring overall divergence in the central repository, we associate with each data object O a numeric weight W_O that can be determined using a variety of criteria such as importance or frequency of access. The objective of strategy (1) is to minimize the weighted sum of the time-averaged divergence of each object, $\sum_O \left[W_O \cdot \int_t D(O,t)\, dt\right]$, under constraints on communication resources.

Communication resources may be limited at a number of points. First, the capacity of the link connecting the central data repository to the rest of the network, the *repository-side* bandwidth, may be constrained. Second, the capacity of the link connecting each source to the rest of the network, the *source-side bandwidth*, may also be constrained and may vary among sources. Moreover, all bandwidth capacities may fluctuate over time if resource limitations are related to traffic generated by other applications. We assume a standard underlying network model where any messages for which there is not enough capacity become enqueued for later transmission.

2.2 Maximizing Performance

The objective of strategy (2) in Section 1.2 is to maximize performance by minimizing the total *communication cost* incurred during a period of time in which replica precision is constrained. Each message sent between object O's source and the central repository (such as a refresh message) incurs a numeric cost $C_O \geq 0$, and costs are additive.

Constraints on precision arise with respect to *queries*, which are submitted by users or applications in order to access data. We consider situations in which aggregation queries over numeric data objects are submitted to the central repository. The repository is to provide answers to queries in the form of numeric intervals $[L,H]$ that are guaranteed to contain the precise answer V that could be obtained by accessing current master source copies, *i.e.*, $L \leq V \leq H$. As discussed later, guaranteed answer intervals can be produced by establishing bounds on replica divergence. Each query submitted at the central repository specifies a *precision constraint* that specifies the maximum acceptable width $(H-L)$ for the answer interval. Two modes of querying are considered in our work. *One-time queries* request the answer a single time, and do not persist once the (approximate) answer has been produced. By contrast, *continuous queries* are ongoing requests for continually updated answers that meet the specified precision constraint at all times.

3 Overview of Our Work on Approximate Replication

Maximizing Precision. In [14] we tackle the problem of maximizing replica precision when communication performance is limited due to constraints placed by users, applications, or the network infrastructure. The first step is to provide a general definition of precision, based on replica divergence, that can be specialized to a variety of data domains. We then present a replica synchronization technique whereby sources prioritize data objects that need to be synchronized based on precision considerations, and push synchronization messages to the central repository in priority order. The rate at which each source sends synchronization messages is regulated adaptively to ensure that the overall message rate conforms to the performance constraints. We evaluate our technique empirically and compare its effectiveness with that of a prior pull-based approach.

Maximizing Performance. In [11] we tackle the inverse problem: maximizing communication performance while maintaining acceptable levels of data precision as specified by users or applications at the granularity of queries over groups of objects. We focus on continuous queries, or CQ's for short, and propose a technique for performing push-based replication that meets the precision constraints of all continuous queries at all times. Without violating any query-level precision constraints, our technique continually adjusts the precision of individual replicas to maximize the overall communication performance. Results are provided from several experiments evaluating the performance of our technique in a simulated environment. In addition, we describe a testbed network traffic monitoring system we built to track usage patterns and flag potential security hazards using continuous queries with precision constraints. Experiments in this real-world application verify the effectiveness of our CQ-based approximate replication technique in achieving low communication cost while guaranteeing precision for a workload of multiple continuous queries.

Answering Unexpected Queries. Providing guaranteed precision for a workload of continuous queries does not handle an important class of queries that arises frequently in practice: one-time, unanticipated queries. Users or applications interacting with the data repository may at any time desire to obtain a one-time result of a certain query, which includes a precision constraint and may be different from the continuous queries currently being evaluated, or the same as a current CQ but with a more stringent precision constraint. Due to the ad-hoc nature of one-time queries, when one is issued the data replicas in the repository may not be of sufficient precision to meet the query's precision constraint. To obtain a query answer of adequate precision it may be necessary to access master copies of a subset of the queried data objects by contacting remote sources, incurring additional performance penalties. In [13] we study the problem of maximizing performance in the presence of unanticipated one-time queries, and devise efficient algorithms for minimizing accesses to remote master copies.

Managing Precision for One-Time Queries. A significant factor determining the cost to evaluate one-time queries with precision constraints at a central data repository is the precision of data replicas maintained in the repository. In [12] we study the problem of deciding what precision levels to use when replicating data objects not involved in continuous queries but subject to intermittent accesses by one-time queries. Interestingly, this problem generalizes a previously studied problem of deciding whether or not to perform exact replication of individual data objects. We propose an adaptive algorithm for setting replica precision with the goal of maximizing overall communication performance given a workload of one-time queries. In an empirical study we compare our algorithm with a prior algorithm that addresses the less general exact replication problem, and we show that our algorithm subsumes it in performance.

4 Overview of Related Work

We now cover previous work by others that is broadly related to our own. Other work not covered here is related to fairly specific aspects of our work; see [10] for additional coverage.

A number of replication strategies have been proposed based on abandoning strict transactional replication protocols that guarantee one-copy serializability, and performing asynchronous propagation of all database updates in a nontransactional fashion [5, 15], in order to reduce response time and improve availability. These approaches alleviate many of the problems associated with transactional protocols, but they do not focus on reducing communication cost except for some batching effects since all updates are propagated eventually.

The focus of our work is on conserving communication cost in nontransactional environments by performing approximate replication. The need for approximate replication is perhaps most obvious in the distributed World Wide Web environment. Partly this need arises because exact consistency is virtually impossible in the presence of the high degree of autonomy featured on the Web, but also because the volume of data is vast and aggregate data change rates are astronomical. On the Web, two forms of approximate replication are currently in heavy use: Web crawling and Web caching. Mechanisms for synchronizing document replicas in Web repositories via *incremental Web crawling*, *e.g.*, [18], are typically designed to work within a fixed communication budget, and generally aim to minimize some variant of the *temporal staleness* metric: the average amount of time during which Web document replicas in the repository differ in some way from the remote master copy. In Web caching environments, synchronization of a set of documents selected for caching is typically driven by constraints on the temporal staleness of cached replicas (the constraints are commonly referred to as time-to-live (TTL) restrictions), and the goal is to minimize communication, *e.g.*, [3].

Incremental Web crawling can be thought of as an instance of the *Performance Fixed/Maximize Precision* scenario, while Web caching represents an instance of the inverse *Precision Fixed/Maximize Performance* scenario. Due to the high degree of autonomy present in the Web environment, solutions to these problems almost always employ pull-oriented techniques for replica synchronization. In addition, owing to the wide variety of content found on the Web, synchronization techniques usually optimize for temporal staleness, a simple precision metric based solely on metadata. (Other metadata-based precision metrics have also been proposed, such as the number of updates not reflected in the remote replica, *e.g.*, [8].)

For replication environments in which greater cooperation among nodes is possible and more is known about the nature of the data and needs of the users, push-oriented synchronization based on richer content-based precision metrics tends to lead to more desirable results, *i.e.*, higher quality synchronization at lower cost, as discussed in Section 1.2. The CU-SeeMe video conferencing project [6] represents an interesting instance of a push-oriented synchronization approach using direct, content-based precision metrics, which focuses on the *Performance Fixed/Maximize Precision* scenario. In CU-SeeMe, refreshes to different regions of remotely replicated images are delayed and reordered at sources based on application-specific precision metrics that take into account pixel color differences. Another domain-specific approach has been proposed for moving object tracking [19], which focuses on the *Precision Fixed/Maximize Performance* scenario, or alternatively aims at maximizing an overall "information cost" metric that combines precision and performance.

Our goal is to establish generic push-oriented approximate replication strategies that exploit and expose the fundamental precision-performance tradeoff common to all environments, in a manner suitable to a wide variety of applications that rely on replication. Some initial steps toward this goal have been made by others in previous work. To our knowledge, the first proposal on this topic was by Alonso et al. [1], and recently others have extended that work, *e.g.*, [9, 16, 20, 21] ([21] studies pull-oriented techniques). All of this work falls into the *Precision Fixed/Maximize Performance* category, with precision constraints specified at the granularity of individual objects. One portion of our work [14] focuses on the inverse problem of *Performance Fixed/Maximize Precision*, which to our knowledge has not been studied in a general, application-independent setting with flexible precision metrics.

The portion of our work that addresses the *Precision Fixed/Maximize Performance* problem departs significantly from previous work by considering precision constraints at the granularity of entire queries rather than at the granularity of individual replicated objects. The rationale for this choice is twofold. First, we sought to align the granularity of precision constraint specification with the granularity of data access. (Since queries may be posed over individual data objects, our mechanism generalizes the previous approach.) Second, precision constraints at the per-query granularity leave open the possibility for optimizing performance by adjusting the allocation of precision requirements across individual objects involved in large queries. Indeed, much of our work [11, 12] focuses on realizing such optimizations using adaptive precision-setting techniques, which we show to enable significant improvements in synchronization efficiency. (The application of adaptive precision-setting techniques to hierarchical replication topologies has recently been studied in [4], with a particular focus on sensor network environments.)

Our work is also unique in focusing on user control over query answer precision, which may lead to unpredictable precision requirements. Specifically, to our knowledge it is the first to consider the problem of efficiently evaluating one-time queries with user-specified precision constraints that may exceed the precision of current replicas, thereby requiring access to some exact source copies [13]. (A version of this problem using a probabilistic model of precision was studied subsequently in [2].)

Acknowledgements. We thank the following individuals for their valuable input and assistance with this work: Michael Franklin, Hector Garcia-Molina, Jing Jiang, and Boon Thau Loo.

References

[1] R. Alonso, D. Barbara, H. Garcia-Molina, and S. Abad. Quasi-copies: Efficient data sharing for information retrieval systems. In *Proc. EDBT*, 1988.

[2] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. ACM SIGMOD*, 2003.

[3] E. Cohen and H. Kaplan. Refreshment policies for web content caches. In *Proc. IEEE INFOCOM*, 2001.

[4] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregatio with quality guarantees. In *Proc. EDBT*, 2004.

[5] L. Do, P. Ram, and P. Drew. The need for distributed asynchronous transactions. In *Proc. ACM SIGMOD*, 1999.

[6] T. Dorcey. CU-SeeMe desktop videoconferencing software. *Connexions*, 9(3), 1995.

[7] A. Householder, A. Manion, L. Pesante, and G. Weaver. Managing the threat of denial-of-service attacks. Technical report, CMU Software Engineering Institute CERT Coordination Center, Oct. 2001. `http://www.cert.org/archive/pdf/Managing_DoS.pdf`.

[8] Y. Huang, R. Sloan, and O. Wolfson. Divergence caching in client-server architectures. In *Proc. PDIS*, 1994.

[9] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *Proc. ACM SIGMOD*, 2004.

[10] C. Olston. Approximate replication. Doctoral dissertation, Stanford University, 2003.

[11] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. ACM SIGMOD*, 2003.

[12] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *Proc. ACM SIGMOD*, 2001.

[13] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. VLDB*, 2000.

[14] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proc. ACM SIGMOD*, 2002.

[15] Y. Saito and M. Shapiro. Replication: Optimistic approaches. Technical report, Hewlett-Packard Labs, 2002. HPL-2002-33.

[16] S. Shah, S. Dharmarajan, and K. Ramamritham. An efficient and resilient approach to filtering and disseminating streaming data. In *Proc. VLDB*, 2003.

[17] R. van Renesse and K. Birman. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. Technical report, Cornell University, 2001.

[18] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proc. WWW*, 2002.

[19] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. In *Proc. SSDBM*, 1998.

[20] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *Proc. OSDI*, 2000.

[21] S. Zhu and C. V. Ravishankar. Stochastic consistency, and scalable pull-based caching for erratic data stream sources. In *Proc. VLDB*, 2004.

Data Reduction Techniques in Sensor Networks

Antonios Deligiannakis
University of Maryland
adeli@cs.umd.edu

Yannis Kotidis
AT&T Labs-Research
kotidis@research.att.com

Abstract

Recent advances in microelectronics have made feasible the deployment of sensor networks for a variety of monitoring and surveillance tasks. The severe energy constraints met in such networks make imperative the design of energy efficient protocols for communication, which often constitutes the largest source of energy drain, of the collected information. In this paper, we describe several techniques that can be applied for the reduction of the transmitted information in different application scenarios.

1 Introduction

Recent advances in wireless technologies and microelectronics have made feasible, both from a technological as well as an economical point of view, the deployment of densely distributed sensor networks [11]. Although today's sensor nodes have relatively small processing and storage capabilities, driven by the economy of scale, it is already observed that both are increasing at a rate similar to Moore's law. In applications where sensors are powered by small batteries and replacing them is either too expensive or impossible (i.e., sensors thrown over a hostile environment), designing energy efficient protocols is essential to increase the lifetime of the sensor network. Since radio operation is by far the biggest factor of energy drain in sensor nodes [4], minimizing the number of transmissions is vital in data-centric applications. Even in the case when sensor nodes are attached to larger devices with ample power supply, reducing bandwidth consumption may still be important due to the wireless, multi-hop nature of communication and the short-range radios usually installed in the nodes.

Data-centric applications thus need to devise novel dissemination processes for minimizing the number of messages exchanged among the nodes. Nevertheless, in densely distributed sensor networks there is an abundance of information that can be collected. In order to minimize the volume of the transmitted data, we can apply two well known ideas: *aggregation* and *approximation*.

In-network aggregation is more suitable for exploratory, continuous queries that need to obtain a live estimate of some (aggregated) quantity. For example, sensors deployed in a metropolitan area can be used to obtain estimates on the number of observed vehicles. Temperature sensors in a warehouse can be used to keep track of average and maximum temperature for each floor of the building. Often, aggregated readings over a large number of sensors nodes show little variance, providing a great opportunity for reducing the number of (re)transmissions by the nodes when individual measurements change only slightly (as in temperature readings) or changes in measurements of neighboring nodes effectively cancel out (as in vehicle tracking).

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Approximation techniques, in the form of lossy data compression are more suitable for the collection of historical data through long-term queries. As an example, consider sensors dispersed in a wild forest, collecting meteorological measurements (such as pressure, humidity, temperature) for the purpose of obtaining a long term historical record and building models on the observed eco-system [8]. Each sensor generates a multi-valued data feed and often substantial compression can be achieved by exploiting natural correlations among these feeds (such as in case of pressure and humidity measurements). In such cases, sensor nodes are mostly “silent” (thus preserving energy) and periodically process and transmit large batches of their measurements to the monitoring station for further processing and archiving.

While the preferred method of data reduction, either by aggregation or by approximation, of the underlying measurements can be decided based on the application needs, there is a lot of room for optimization at the network level too. Sensor networks are inherently redundant; a typical employment uses a lot of redundant sensor nodes to cope with node or link failures [1]. Thus, extracting measurements from all nodes in the network for the purpose of answering a posed query may be both extremely expensive and unnecessary. In a data-centric network, nodes can coordinate with their neighbors and elect a small set of *representative nodes* among themselves, using a localized, data-driven bidding process [5]. These representative nodes constitute a *network snapshot* that can, in turn, answer posed queries while reducing substantially the energy consumption in the network. These nodes are also used as an alternative means of answering a posed query when nodes and network links fail, thus providing unambiguous data access to the applications.

2 Characteristics of Sensor Nodes

Depending on the targeted application, sensor nodes with widely different characteristics may be used. Even though the processing and memory capabilities of sensor nodes are still limited, in recent years they have increased at a rate similar to Moore’s law. On the other hand, the amount of energy stored in the batteries used in such nodes has exhibited a mere 2-3% annual growth.[1] Since replacing the sensor batteries may be very expensive, and often impossible due to their unattended deployment, unless the sensors are attached to and powered by a larger unit, designing energy-efficient protocols is essential to increase the lifetime of the sensor network.

The actual energy consumption by each sensor node depends on its current state. In general, each sensor node can be in one of the following states: 1) *low-duty cycle*, where the sensor is in sleep mode and a minimal amount of energy is consumed; 2) *idle listening*, where the sensor node is listening for possible data intended for it; 3) *processing*, where the node performs computation based on its obtained measurements and its received data; and, 4) *receiving/transmitting*, where the node either receives or transmits data or control messages.

The cost of processing can be significant but is generally much lower than the cost of transmission. For example, in the Berkeley MICA nodes sending one bit of data costs as much energy as 1,000 CPU instructions [7]. For long-distance radios, the transmission cost dominates the receiving and idle listening costs. For short-range radios, these costs are comparable. For instance, in the Berkeley MICA2 motes the power consumption ratio of transmitting/receiving at 433MHz with RF signal power of 1mW is 1.41:1 [13], while this ratio can become even larger than 3:1 for the same type of sensor when the radio transmission power is increased [10]. To increase the lifetime of the network, some common goals of sensor network applications are (in order of importance) to maximize the time when a node is in a low-duty cycle, to reduce the amount of transmitted and received data, and to reduce the idle listening time. We note here that reducing the size of the transmitted data results in multiple benefits, since this also corresponds to a reduction of not only control messages, but also leads to fewer message collisions and retransmissions. Moreover, nodes that refrain from transmitting messages may switch to the low-duty cycle mode faster, therefore further reducing their energy drain.

[1] `http://nesl.ee.ucla.edu/courses/ee202a/2002f/lectures/L07.ppt`.

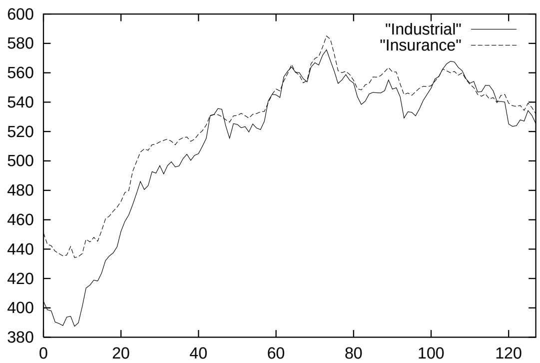


Figure 1: Example of two correlated signals (Stock Market)

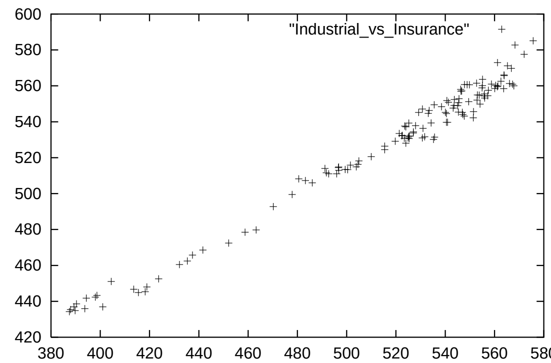


Figure 2: XY scatter plot of Industrial (X axis) vs Insurance (Y axis)

3 A Lossy Compression Framework for Historical Measurements

Many real signals observed by the sensors, such as temperature, dew-point, pressure etc. are naturally correlated. The same is often true in other domains. For instance, stock market indexes or foreign currencies often exhibit strong correlations. In Figure 1 we plot the average Industrial and Insurance indexes from the New York stock market for 128 consecutive days.[2] Both indexes show similar fluctuations, a clear sign of strong correlation. Figure 2 depicts a XY scatter plot of the same values. This plot is created by pairing values of the Industrial (X-coordinate) and Insurance (Y-coordinate) indexes of the same day and plotting these points in a two-dimensional plane. The strong correlation among these values makes most points lie on a straight line. This observation suggests the following compression scheme, inspired from regression theory. Assuming that the Industrial index (call it $\vec{X}$) is given to us in a time-series of 128 values, we can approximate the other time-series (Insurance: $\vec{Y}$) as $\vec{Y'} = a{*}\vec{X}{+}b$. The coefficients a and b are determined by the condition that the sum of the square residuals, or equivalently the L_2 error norm $||\vec{Y'} - \vec{Y}||_2$, is minimized. This is nothing more than standard linear regression. However, unlike previous methods, we will not attempt to approximate each time-series independently using regression. In Figure 1 we see that the series themselves are not linear, i.e., they would be poorly approximated with a linear model. Instead, we will use regression to approximate piece-wise correlations of each series to a base signal $\vec{X}$ that we will choose accordingly. In the example of Figure 2, the base signal can be the Industrial index ($\vec{X}$) and the approximation of the Insurance index will be just two values (a, b). In practice the base signal will be much smaller than the complete time series, since it only needs to capture the "important" trends of the target signal $\vec{Y}$. For instance, in case $\vec{Y}$ is periodic, a sample of the period would suffice.

The SBR framework. In the general case, each sensor monitors N distinct quantities $\vec{Y_i}$, $1 \leq i \leq N$. Without loss of generality we assume that measurements are sampled with the same rate. When enough data is collected (for instance, when the sensor memory buffers become full), the latest NxM values are processed and each row i (of length M) is approximated by a much smaller set of B_i values, i.e. $B_i \ll M$. The resulting "compressed" representation, of total size equal to $\sum_{i=1}^{N} B_i$, is then transmitted to the base station. The base station maintains the data in this compact representation by appending the latest "chunk" to a log file. A separate file exists for each sensor that is in contact with the base station. This process is illustrated in Figure 3. Each sensor allocates a small amount of memory of size M_{base} for what we call the *base signal*. This is a compact ordered collection of values of prominent features that we extract from the recorded values and are used as a base reference in the approximate representation that is transmitted to the base station. The data values that the sensor transmits to the base station are encoded using the in-memory values of the base signal at the time of the transmission. The base

[2]Data at `http://www.marketdata.nasdaq.com/mr4b.html`.

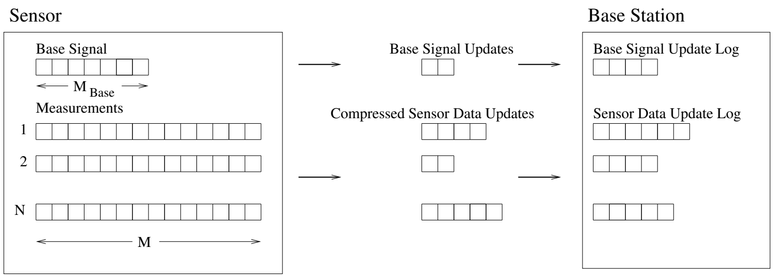


Figure 3: Transfer of approximate data values and of the base signal from each sensor to the base station

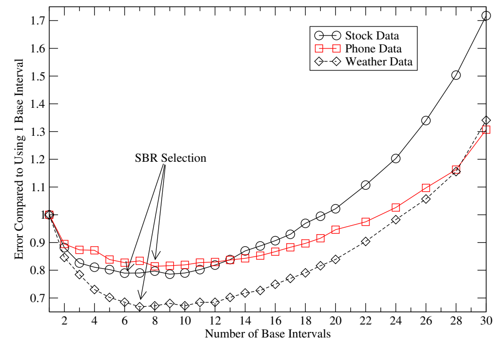


Figure 4: SSE error vs base signal size

signal may be updated at each transmission to ensure that it will be able to capture newly observed data features and that the obtained approximation will be of good quality. When such updates occur, they are transmitted along with the data values and appended in a special log file that is unique for each sensor.

The Self-Based Regression algorithm (SBR) breaks the data intervals $\vec{Y_i}$ into smaller data segments $I_i[k..l] = (Y_i[k], \ldots Y_i[l])$ and then pairs each one to an interval of the base signal of equal length. As discussed below, the base signal is simply the concatenation of several intervals of the same length W extracted from the data. The data interval I_i is shifted over the base signal and at each position s we compute the regression parameters for the approximation $\hat{I}_i[j] = a \times X[s + j - k] + b$, $k \leq j \leq l$ and retain the shift value $s = s^*$ that minimizes the sum-squared error of the approximation. The algorithm starts with a single data interval for each row of the collected data (Y_i). In each iteration, the interval with the largest error in the approximation is selected and divided in two halves. The compressed representation of a data interval $I_i[k..l]$ consists of four values: the shift position s^* that minimizes the error of the approximation, the two regression parameters a,b and the start of the data interval k in Y_i. The base station will sort the intervals based on their start position and, thus, there is no need to transmit their ending position. Given a target budget B (size of compressed data) we can use at most B/4 intervals using this representation.

Base Signal Construction. We can think of the base signal as a dictionary of features used to describe the data values. The richer the pool of features we store in the base signal the better the approximation. On the other hand, these features have to be (i) kept in the memory of the sensor to be used as a reference by the data-reduction algorithm and (ii) sent to the base station in order for it to be able to reconstruct the values. Thus, for a target bandwidth constraint B (number of values that can be transmitted), performing more insert and update operations on the base signal implies less bandwidth remaining for approximating the data values, and, thus, fewer data intervals that can be obtained from the recursive process described above.

We can avoid the need of transmitting the base signal by agreeing a-priori on a set of functions that will be used in the regression process. For instance, a set of cosine functions (as in the Distinct Cosine Transform) can be used for constructing a "virtual" base signal that does not need to be communicated. Similarly, using the identity function $X[i] = i$ reduces the compression algorithm to standard linear regression of each data interval. However, such an approach makes assumptions that may not hold for the data at hand. In [2] we have proposed a process for generating the base signal from the data values. The key idea is to break the measurements into intervals of the same length W. Each interval (termed *candidate base interval*) is assigned a score based on the reduction in the error of the approximation obtained by adding the interval to the base signal. Using a greedy algorithm we can select the top-few candidate intervals, up to the amount of available memory M_{base}. Then a binary-search process is used to eventually decide how many of those candidate intervals need to be retained.

The search space is illustrated in Figure 4 for three real data sets, discussed in [2]. The figure plots the error of only the initial transmission as the size of the base signal is varied, manually, from 1 to 30 intervals. We further show the selection of the binary-search process. For presentation purposes, the errors for each data set

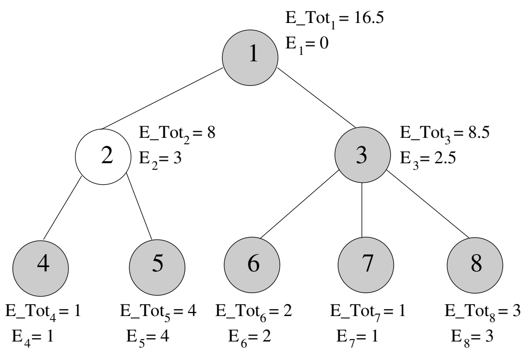


Figure 5: Error Filters on Aggregation Tree

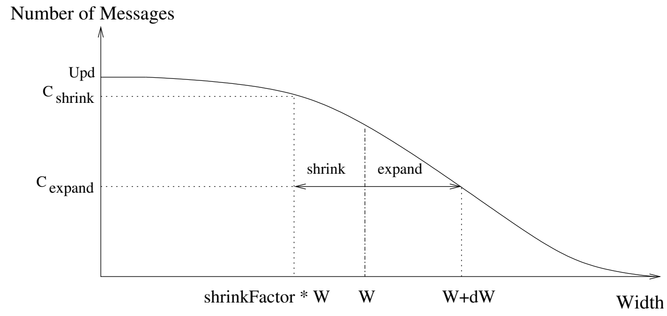


Figure 6: Potential Gain of a Node

have been divided by the error of the approximation when using just one interval. We notice that initially, by adding more candidate intervals to the base signal the error of the approximation is reduced. However, after a point, adding more intervals that need to be transmitted to the base station leaves insufficient budget for the recursive process that splits the data, and thus, the error of the approximation is eventually increased.

For a data set containing $n = N \times M$ measurements to approximate the complete SBR algorithm takes $O(n^{1.5})$ time and requires linear space, while its running time scales linearly to the size of both the transmitted data and the base signal. The algorithm is invoked periodically, when enough data has been collected. Moreover, our work [2] has demonstrated that, after the initial transmissions, the base signal is already of good quality and few intervals are inserted in it. This provides us with the choice not to update the base signal in many subsequent invocations, thus reducing the algorithm's running time, in these cases, to only a linear dependency on n.

4 Approximate in-Network Data Aggregation

In-Network Data Aggregation. The data aggregation process in sensor networks consists of several steps. First, the posed query is disseminated through the network, in search of nodes collecting data relevant to the query. Each sensor then selects one the nodes through which it received the announcement as its *parent node*. The resulting structure is often referred to as the *aggregation tree*. Non-leaf nodes of that tree aggregate the values of their children before transmitting the aggregate result to their parents. In [6], after the aggregation tree has been created, the nodes carefully schedule the periods when they transmit and receive data. The idea is for a parent node to be listening for values from its child nodes within specific intervals of each *epoch* and then transmit upwards a single partial aggregate for the whole subtree.

Framework Description. In order to limit the number of transmitted messages and, thus, the energy consumption in sensor networks during the computation of continuous aggregate queries, our algorithms install error filters on the nodes of the aggregation tree.[3] Each node N_i transmits the partial aggregate that it computes at each epoch for its subtree only if this value deviates by more than the maximum error E_i of the node's filter from the last transmitted partial aggregate. This method allows nodes to refrain from transmitting messages about small changes in the value of their partial aggregate. The E_i values determine the maximum deviation E_Tot_i of the reported from the true aggregate value at each node. For example, for the SUM aggregate, this deviation at the monitoring node is upper bounded (ignoring message losses) by: $\sum_i E_i$. A sample aggregation tree is depicted in Figure 5. As our work [3] has demonstrated, allowing a small error in the reported aggregate can lead to dramatic bandwidth (and thus energy) savings. The challenge is, of course, given the maximum error tolerated by the monitoring node, to calculate and periodically adjust the node filters in order to minimize the

[3]The use of error filters in error-tolerate applications in flat, non-hierarchical domains has been investigated in [9].

bandwidth consumption.

Algorithmic Challenges. When designing adaptive algorithms for in-network data aggregation in sensor networks, one has to keep in mind several challenges/goals. First, communicating *individual* node statistics is very expensive, since this information cannot be aggregated inside the tree, and may outweigh the benefits of approximate data aggregation, namely the reduction in the size of the transmitted data. Thus, our algorithm should not try to estimate the number of messages generated by each node's transmissions, since this depends on where this message is aggregated with messages from other nodes. Second, the error budget should be distributed to the nodes that are expected to reduce their bandwidth consumption the most by such a process. This benefit depends on neither the magnitude of the partial aggregate values of the node nor the node's number of transmissions over a period, but on the magnitude of the changes on the calculated partial aggregate. Isolating nodes with large variance on their partial aggregate and redistributing their error to other nodes is crucial for the effectiveness of our algorithm [3]. Finally, in the case of nodes where the transmitted differences from their children often result in small changes on the partial aggregate value, our algorithm should be able to identify this fact. We deal with this latter challenge by applying the error filters on the calculated partial aggregate values and not on each node's individual measurements. For the first two challenges, we collect a set of easily computed and composable statistics at each node. These statistics are used for the periodic adjustment of the error filters.

Algorithm Overview. Every Upd epochs all nodes shrink the widths of their filters by a shrinking factor $shrinkFactor$ ($0 < shrinkFactor < 1$). After this process, the monitoring node has an error budget of size $E\_Global \times (1 - shrinkFactor)$, where E_Global is the maximum error of the application, that it can redistribute recursively to the nodes of the network. Each node, between two consecutive update periods, calculates its *potential gain* as follows: At each epoch the node keeps track of the number of transmissions C_{shrink} that it would have performed with the default (smaller) filter at the next update period of width $shrinkFactor \times W_i$, where $W_i = 2 \times E_i$. The node also calculates the corresponding number of transmissions C_{expand} with a larger filter of width $W_i + dW$ and sets its potential gain to $Gain_i = C_{shrink} - C_{expand}$. This process is illustrated in Figure 6. The *cumulative gain* of a node's subtree is then calculated as the sum of the cumulative gains of the node's children and the node's potential gain. This requires only the transmission of the cumulative gains (a single value for each node) at the last epoch before the new update period. The available error budget is then distributed top-down proportionally, at each node, to each subtree's cumulative gain. In this process, nodes that exhibit large variance in their partial aggregate values will exhibit small potential gains and, thus, their error will gradually shrink and be redistributed to nodes that will benefit from an increase in their error filter.

We note here that the dual problem of minimizing the application maximum error given a bandwidth constraint can also be solved in a similar manner, using statistics collected at each node. The problem is more complicated, though, because the controlled quantity at each node (the width of its error filter) is different from the monitored quantity (the bandwidth consumption) and the bandwidth needs to be carefully computed, monitored and then disseminated amongst the sensor nodes.

5 Design of Data-Centric Sensor Networks

Sensor networks are inherently dynamic. Such networks must adapt to a wide variety of challenges imposed by the uncontrolled environment in which they operate. As nodes become cheaper to manufacture and operate, one way of addressing the challenges imposed on unattended networks is redundancy [1]. Redundant nodes ensure network coverage in regions with non-uniform communication density due to environmental dynamics. Redundancy further increases the amount of data that can be mined in large-scale networks.

Writing data-driven applications in such a dynamic environment can be daunting. Again, the major challenge is to design localized algorithms that will perform most of the processing in the network itself in order to reduce traffic and, thus, preserve energy. Instead of designing database applications that need to hassle with low-level networking details, we envision the use of data-centric networks that allow transparent access to the collected measurements in a unified way. For instance, when queried nodes fail, the network should self-configure to use redundant stand-by nodes as in [4], under the condition that the new nodes contain fairly similar measurements, where similarity needs to be quantified in an application-meaningful way [5]. This can be achieved using a localized mode of operation in which nodes can coordinate with their neighbors and elect a small set of *representative nodes* among themselves. Such a set of representatives, termed *network snapshot* [5], has many advantages. The location and measurements of the representative nodes provide a picture of the value distribution in the network. Furthermore, the network snapshot can be used for answering user queries in a more energy-efficient way, since significantly fewer nodes need to be involved in query processing. Finally, an elected representative node can take over for another node in its vicinity that may have failed or is temporarily out of reach.

6 Conclusions and Future Directions

In this paper we described several techniques for the reduction of the transmitted data in several sensor network applications, ranging from the communication of historical measurements to answering approximate aggregate continuous and snapshot queries. While these techniques aim to prolong the lifetime of the network, there are several issues that need to be additionally addressed. Little work has been done on the optimization of multiple concurrent continuous queries over sensor networks. The work of Olston et al. in [9] may provide some helpful solutions in this area. Moreover, in the presence of nodes with different transmission frequencies, as in the case of approximate aggregate query processing, several communication and synchronization algorithms may need to be revisited [12]. For example, the selection of the aggregation tree is often performed by assuming equal frequency of transmissions by all nodes. However, it might be more beneficial to prevent nodes that exhibit large variance in their measurements from appearing in lower levels of the tree, since such nodes often trigger transmissions on their ancestors as well. Such optimizations may lead to even larger energy savings.

References

[1] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In *INFOCOM*, 2002.

[2] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing Historical Information in Sensor Networks. In *SIGMOD*, 2004.

[3] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-Network Data Aggregation with Quality Guarantees. In *Proceedings of EDBT*, 2004.

[4] D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *MobiCOM*, 1999.

[5] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. In *Proceedings of ICDE*, 2005.

[6] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.

[7] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *ACM SIGMOD*, 2003.

[8] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *WSNA 2002*, pages 88–97, 2002.

[9] C. Olston, J. Jiang, and J. Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *SIGMOD*, 2003.

[10] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M.. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Sensys*, 2004.

[11] B. Warneke, M. Last, B. Liebowitz, and K. S.J. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. *IEEE Computer*, 34(1):44–51, 2001.

[12] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Netwo rks. *SIGMOD Record*, 31(3):9–18, 2002.

[13] W. Ye and J. Heidermann. Medium Access Control in Wireless Sensor Networks. Technical report, USC/ISI, 2003.

Robust Aggregation in Sensor Networks

George Kollios, John Byers, Jeffrey Considine, Marios Hadjieleftheriou, and Feifei Li
Computer Science Dept., Boston University
{gkollios, byers, jconsidi, marioh, lifeifei}@cs.bu.edu

Abstract

In the emerging area of sensor-based systems, a significant challenge is to develop scalable, fault-tolerant methods to extract useful information from the data the sensors collect. An approach to this data management problem is the use of sensor "database" systems, which allow users to perform aggregation queries on the readings of a sensor network. Due to power and range constraints, centralized approaches are generally impractical, so most systems use in-network aggregation to reduce network traffic. However, these aggregation strategies become bandwidth-intensive when combined with the fault-tolerant, multi-path routing methods often used in these environments. In order to avoid this expense, we investigate the use of approximate *in-network aggregation using small sketches and we survey robust and scalable methods for computing duplicate-sensitive aggregates.*

1 Introduction

As computation-enabled devices shrink in scale and proliferate in quantity, a relatively recent research direction has emerged to contemplate future applications of these devices and services to support them. A canonical example of such a device is a *sensor mote*, a device with measurement, communication, and computation capabilities, powered by a small battery [21]. Individually, these motes have limited capabilities, but when a large number of them are networked together into a *sensor network*, they become much more capable. Indeed, large-scale sensor networks are now being applied experimentally in a wide variety of areas — some sample applications include environmental monitoring, surveillance, and traffic monitoring.

In a typical sensor network, each sensor produces a stream of sensory observations across one or more sensing modalities. But for many applications and sensing modalities, such as reporting temperature readings, it is unnecessary for each sensor to report its entire data stream in full fidelity. Moreover, in a resource-constrained sensor network environment, each message transmission is a significant, energy-expending operation. For this reason, and because individual readings may be noisy or unavailable, it is natural to use data aggregation to summarize information collected by sensors. As a reflection of this, a database approach to managing data collected on sensor networks has been advocated [24, 29], with particular attention paid to efficient query processing for aggregation queries [24, 29, 31].

In the TAG system [24], users connect to the sensor network using a workstation or base station directly connected to a sensor designated as the sink. Aggregate queries over the sensor data are formulated using a simple SQL-like language, then distributed across the network. Aggregate results are sent back to the workstation over a spanning tree, with each sensor combining its own data with results received from its children. If there are no failures, this in-network aggregation technique is both effective and energy-efficient for distributive

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

and algebraic aggregates [19] such as MIN, MAX, COUNT and AVG. However, as shown in [7], this technique is much less effective in sensor network scenarios with moderate node and link failure rates. Node failure is inevitable when inexpensive, faulty components are placed in a variety of uncontrolled or even hostile environments. Similarly, link failures and packet losses are common across wireless channels because of environmental interference, packet collisions, and low signal-to-noise ratios [31]. For example, [31] reports on experiments in which more than 10% of the links suffered average loss rate greater than 50%.

When a spanning tree approach is used for aggregate queries, as in TAG, a single failure results in an entire subtree of values being lost. If this failure is close to the sink, the change in the resulting aggregate can be significant. Retransmission-based approaches are expensive in this environment, so solutions based upon multi-path routing were proposed in [24]. For aggregates such as MIN and MAX which are monotonic and exemplary, this provides a fault-tolerant solution. But for duplicate-sensitive aggregates such as COUNT or AVG that give incorrect results when the same value is counted multiple times, existing methods are not satisfactory.

In this paper, we survey robust and scalable methods for computing duplicate-sensitive aggregates across faulty sensor networks. Guaranteeing exact solutions in the face of losses is generally impractical, so we focus on approximate methods that are robust to both link and node failures, but are inexact. For example, we describe the application of well-known sketches to handle SUM and COUNT aggregates [14, 4] to this problem The methods we present combine duplicate insensitive sketches with multi-path routing techniques to produce accurate approximations with low communication and computation overhead.

2 Related Work

In-network Aggregate Query Processing: A simple approach to evaluate an aggregation query is to reliably route all sensed values to the base station and compute the aggregate there. Although this approach is simple, the number of messages and the power consumption can be large. A better approach is to leverage the computational power of the sensor devices and compute aggregates in-network. Aggregates that can be computed in-network include all decomposable functions [24]. Using decomposable functions, the value of the aggregate function can be computed for disjoint subsets, and these values can be used to compute the aggregate of the whole using an appropriate merging function. Our discussion is based on the Tiny Aggregation (TAG) framework used in TinyDB [24]. However, similar approaches are used to compute aggregates in other systems [29, 31, 22].

In TAG, the in-network query evaluation has two phases, the *distribution* phase and the *collection* phase. During the distribution phase, the query is flooded in the network and the nodes are organized into an *aggregation tree*. The base station broadcasting the query is the *root* of the tree. The query message has a counter that is incremented with each retransmission and counts the hop distance from the root. In this way, each node is assigned to a specific level equal to the node's hop distance from the root. Also, each sensor chooses one of its neighbors with a smaller hop distance from the root to be its parent in the aggregation tree.

During the collection phase, each leaf node produces a single tuple and forwards this tuple to its parent. The non-leaf nodes receive the tuples of their children and combine these values. Then, they submit the new partial results to their own parents. This process runs continuously and after h steps, where h is the height of the aggregation tree, the total result will arrive at the root. In order to conserve energy, sensor nodes sleep as much as possible during each step where the processor and radio are idle. When a timer expires or an external event occurs, the device wakes up and starts the processing and communication phases. At this point, it receives the messages from its children and then submits the new value(s) to its parent. After that, if no more processing is needed for that step, it enters again into the sleeping mode [25].

Best-Effort Routing in Sensor Networks: Recent years have seen significant work on best-effort routing in sensor and other wireless networks. Due to high loss rates and power constraints, a common approach is to use dispersity multi-path routing, where more than one copy of a packet is sent to the destination over different paths. For example, directed diffusion [22] uses a flood to discover short paths which sensors would use to send back responses. Various positive and negative reinforcement mechanisms are used to improve path quality. Braided diffusion [15] builds on directed diffusion to use a set of intertwined paths for increased resilience. A slightly different approach is used by GRAB [30], where paths are not explicitly chosen in advance, but the width of the upstream broadcast is controlled. The techniques we describe are meant to complement and leverage any of these routing techniques [7]. We note that combining these methods with duplicate-insensitive in-network aggregation will allow some of the overhead of these techniques to be amortized and shared amongst data items from many different sensors.

Counting Sketches: A counting sketch for the purpose of quickly estimating the number of distinct items on a stream (the COUNT aggregate) in one pass while using only a small amount of space, was introduced by Flajolet and Martin (FM) in [14]. Since then, there has been much work developing and generalizing counting sketches (e.g., [1, 2, 8, 13, 16, 18, 5, 9]). The original FM sketches are particularly well-suited to sensor network applications, since they are very concise and accurate in practice. We describe them in more detail in Section 3, and show how to extend these sketches for computing SUM aggregates.

The Count-Min sketch, a counting sketch for computing the frequency of elements on a stream (per element or for ranges of elements), was introduced by Cormode and Muthukrishnan [9]. We show how the Count-Min sketch can be made duplicate-insensitive for exploiting multi-path routing in sensor networks. Even though other frequency counting techniques have been proposed in the past [5, 26, 17, 12, 11], the Count-Min sketch is robust, small in size, and provides error guarantees. A detailed analysis of the Count-Min sketch appears in Section 3.

3 Sketch Theory

One of the core ideas behind our work is that duplicate-insensitive sketches will allow us to leverage the redundancy typically associated with multi-path routing. We now present some of the theory behind such sketches and extend it to handle more interesting aggregates. First, we present details of the FM sketches of [14] along with necessary parts of the theory behind them. Then, we generalize these sketches to handle summations, and show that they have the same accuracy as FM sketches. Finally, we present the Count-Min sketch of [9] and show how it can be combined with FM sketches to produce a duplicate-insensitive frequency counting sketch that can provide frequency esimates for both a single element and ranges of elements on a stream.

3.1 FM Sketches

Given a multi-set of items $M = \{x_1, x_2, x_3, \ldots\}$, the *distinct counting* problem is to compute $n \equiv |\text{distinct}(M)|$. The FM sketch of M, denoted $S(M)$, is a bitmap of length k. The entries of $S(M)$, denoted $S(M)[0, \ldots, k-1]$, are initialized to zero and are set to one using a "random" binary hash function h applied to the elements of M. Given $x \in M$ and an integer i, then $h(x,i) = 1$ with probability 0.5 and $h(x,i) = 0$ otherwise (with the same probability). Formally,

$$S(M)[i] \equiv 1 \text{ iff } \exists x \in M \text{ s.t. } \min\{j \mid h(x,j) = 1\} = i.$$

By this definition, each item x is capable of setting a single bit in $S(M)$ to one – the minimum i for which $h(x,i) = 1$. This gives a simple serial implementation which is very fast in practice and requires two invocations of h per item on average. It has been shown that a single element can be inserted into an FM sketch in $O(1)$ expected time. We now describe some interesting properties of FM sketches observed in [14].

Property 1: The FM sketch of the union of two multi-sets is the bit-wise OR of their FM sketches. That is, $S(M_1 \cup M_2)[i] = (S(M_1)[i] \vee S(M_2)[i])$.

Property 2: $S(M)$ is determined only by the distinct items of M. Duplication and ordering do not affect $S(M)$.

Property 1 allows each sensor to compute a sketch of locally held items and send the small sketch for aggregation elsewhere. Since aggregation via union operations is cheap, it may be performed in the network without significant computational burden. Property 2 allows the use of multi-path routing of the sketches for robustness without affecting the accuracy of the estimates. We expand upon these ideas in Section 4.

The next lemma provides key insight into the behavior of FM sketches and will be the basis of efficient implementations of summation sketches later.

Lemma 3: For $i < \log_2 n - 2\log_2\log_2 n$, $S(M)[i] = 1$ with probability $1 - O(ne^{-\log_2^2 n})$. For $i \geq \frac{3}{2}\log_2 n + \delta$, with $\delta \geq 0$, $S(M)[i] = 0$ with probability $1 - O\left(\frac{2^{-\delta}}{\sqrt{n}}\right)$.

The lemma implies that given an FM sketch of n distinct items, one expects an initial prefix of all ones and a suffix of all zeros, while only the setting of the bits around $S(M)[\log_2 n]$ exhibit much variation. This gives a bound on the number of bits k required for $S(M)$ in general: $k = \frac{3}{2}\log_2 n$ bits suffice to represent $S(M)$ with high probability. It also suggests that just considering the length of the prefix of all ones in this sketch can produce an estimate of n. Formally, let $R_n \equiv \min\{i \mid S(M)[i] = 0\}$ when $S(M)$ is an FM sketch of n distinct items. That is, R_n is a random variable marking the location of the first zero in $S(M)$. In [14], the following relationship between R_n and n is proven: $\mathbf{E}[R_n] = \log_2(\varphi n) \pm 10^{-5} + o(1)$. Thus, after ignoring small terms, R_n can be used as an unbiased estimator of $\log_2 \varphi n$. The authors also prove that the variance of $R_n$is slightly more than one, which is a concern, as it implies that estimates of n will often be off by a factor of two or more in either direction. However, standard methods for reducing the variance exist, including those discussed in detail in [7]. Furthermore, using results from [16], FM sketches can give an (ϵ, δ)-approximation method for estimating the distinct items of a multi-set using $O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta})\log(n))$ space.

3.2 Summation FM Sketches

In order to make the application of FM sketches practical for in-network query processing in sensor networks, approximate counting sketches can be generalized to handle summations. Let M be a multi-set of the form $\{x_1, x_2, x_3, \ldots\}$ where $x_i = \langle k_i, c_i\rangle$ and c_i is a non-negative integer. The *distinct summation* problem is to calculate: $n \equiv \sum_{\text{distinct}(\langle k_i, c_i\rangle \in M)} c_i$.

For small values of c_i, it is practical to just count c_i different items based upon k_i and c_i, e.g., by considering the stream of *sub-items*: $\langle k_i, c_i, 1\rangle, \ldots, \langle k_i, c_i, c_i\rangle$. Since this is merely c_i invocations of the counting insertion routine, the analysis for FM sketches applies, and the running time of an insertion is $O(c_i)$ expected time. But for large values of c_i, this approach is impractical, leading us to consider more scalable alternatives. The basic idea is to set the bits in the summation *as if* we had performed c_i successive insertions into an FM sketch, but without actually performing them. Such an emulation of c_i insertions can be done by sampling uniformly at random (repeatably for a given $\langle k_i, c_i\rangle$) from the distribution of sketches produced when c_i keys are chosen uniformly at random and inserted into an empty sketch.

A key observation behind our faster sampling procedure is that most of the simulated insertions attempt to set the same low-order bits, and repeated attempts at setting those bits have no further effect. We observe that if a prefix of the first δ bits in the sketch is already set to 1, then the probability of an inserted sub-item modifying a bit (i.e., setting a zero bit to one) is at most $2^{-\delta}$. By sampling the Bernoulli distribution with the correct value of δ corresponding to the current state of the sketch, we can quickly emulate whether the insertion of a sub-item will change the sketch. If not, we do nothing else, otherwise we determine (again in O(1) expected time), precisely which bit is set. Also, by exploiting the memorylessness of the sampling procedure, we can quickly simulate "skipping ahead" to identify sub-items which actually change the sketch. With this approach, and using fast lookup tables to implement Walker's alias method for sampling an arbitrary probability density function in O(1) time and small space [28], insertion can be achieved in $O(\log c_i)$ expected time. Full details appear in [6].

3.3 Duplicate-Insensitive Count-Min Sketches

The CM sketch is a data structure that can be used to to estimate the frequency of individual elements or ranges of elements on a stream [9]. Let a multi-set $S = \{s_1, \ldots, s_N\}$ denote the stream. The CM structure consists of a $k \times m$ matrix CM of counters, and a set of k hash functions $h_i(\cdot) : \{1 \cdots |\text{distinct}(S)|\} \rightarrow \{1 \cdots m\}$. An insertion to the sketch evaluates all $h_i(\cdot)$ on every $s \in S$ and increases by one the counters with indices $CM[i, h_i(s)]$. In total, exactly k counters are associated with every element. If the hash functions are independent, then with high probability no two elements will hash to exactly the same set of counters. After the sketch is constructed, the estimated frequency of element s is given by $\widehat{f}(s) = \min_i (CM[i, h_i(s)]), 1 \leq i \leq k$. Essentially the counter with the smallest frequency estimate is the one that is the least affected by hash function collisions from other elements.

It can be easily shown that the CM sketch of the union of two multi-sets is the simple matrix addition of their individual CM sketches. Ideally, we would like the CM sketch to be duplicate-insensitive in terms of the union operation. Note that double-counting of CM sketches in a combined sketch, can yield unbounded errors in the estimated frequencies, subject to the total number of duplicate additions that occurred. Duplicate-insensitivity can be achieved by replacing every counter in the CM matrix with one Summation FM sketch, and taking advantage of the union property of FM sketches. Essentially, the Summation FM sketches are used to estimate the original magnitude of each counter. For this new construction, an element frequency can be estimated, in the worst case, as the minimum estimate of any of its associated FM sketches. More details can be found in [20].

4 Approximate Estimation of Duplicate-Sensitive Aggregates

We now discuss how to use duplicate-insensitive sketches to build a robust, loss-resilient framework for aggregation. The framework leverages two main observations. First, the wireless communication of sensor networks gives the ability to broadcast a single message to multiple neighbors simultaneously. Second, the duplicate-insensitive sketches discussed in Section 3 allow a sensor to combine all of its received sketches into a single message to be sent. Given proper synchronization, this will allow us to robustly aggregate data with each sensor sending just one broadcast.

We adapt the basic communication model of TAG [24] for continuous queries (one-shot queries simply terminate earlier). Given a new continuous query, the computation proceeds in two phases. In the first phase, the query is distributed across the sensor network, often using some form of flooding. During this phase, each node also computes its level (i.e., its hop distance from the root), and notes the level values of its immediate neighbors. The second phase is divided into a series of *epochs* specified by the query. The specified aggregate will be computed once for each epoch.

At the beginning of each epoch, each node constructs a sketch of its local values for the aggregate. The epoch is then sub-divided into a series of rounds, one for each level, starting with the highest level (farthest from the root). In each round, the nodes at the corresponding level broadcast their sketches, and the nodes at the next level receive these sketches and combine them with their sketches in progress. In the last round, the root receives the sketches of its neighbors, and combines them to produce the final aggregate. Duplicate-insensitivity of the sketching methods prevents double-counting from occurring. A similar communication model for computing aggregates, the rings overlay, was proposed by Nath et al. [27].

The tight synchronization described so far is not actually necessary. Our methods can also be applied using gossip-style communication - the main advantage of synchronization and rounds is that better scheduling is possible and power consumption can be reduced. However, if a node receives no acknowledgments of its broadcast, it may be reasonable in practice to retransmit. More generally, loosening the synchronization increases the robustness of the final aggregate as paths taking more hops are used to route around failures. (Similar ideas appear in the WILDFIRE protocol proposed by Bawa et al. [3].) This increased robustness comes at the cost of power consumption, since nodes broadcast and receive more often (due to values arriving later than expected) and increased time (and variability) to compute the final aggregate. As mentioned earlier, this general principle allows us to make use of any best-effort routing protocol (e.g., [22, 15]), with the main performance metric of interest being the delivery ratio. Other approaches are based on detecting frequent message losses and changes in network conditions, and adapting the topology hoping to reduce the loss rate [27]. With this protocol a node is allowed to jump to different levels of the tree, based on various heuristics according to the number of times that nodes from higher level have successfully included its sketch in the last few epochs, and the number of transmissions that the node can overhear from neighboring levels.

Alternatively, it may be possible to use CM sketches to approximate a consecutive sequence of values that each sensor generates, and then perform the computation only at the end of the sequence. In that case, each sensor computes a CM sketch of its last T values and then forwards this sketch to the upper levels. The advantage of this approach is the reduction in the number of messages (by a factor of T). However, at the same time the error in the approximation will increase. Another interesting direction is to combine sketch-based with tree-based methods in a *hybrid* approach. Furthermore, it will be interesting to combine robust sketch-based methods with model-based approaches proposed recently [10, 23].

5 Conclusions

We have presented new methods for approximately computing duplicate-sensitive aggregates across distributed datasets. Our immediate motivation comes from sensor networks, where energy consumption is a primary concern, faults occur frequently, and exact answers are not required or expected. An elegant building block which enables our techniques are the duplicate-insensitive sketches of Flajolet and Martin, which give us considerable freedom in our choices of how best to route data and where to compute partial aggregates. In particular, use of this duplicate-insensitive data structure allowed us to make use of dispersity routing methods to provide fault tolerance that would introduce inappropriate errors otherwise.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. *Proc. of RANDOM*, 2002.

[3] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The Price of Validity in Dynamic Networks. *Proc. of ACM SIGMOD*, pp. 515–526, 2004.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. of the ACM (CACM)*, 13(7):422–426, 1970.

[5] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

[6] J. Considine, Schedule-oblivious data management. Ph.D. Thesis, Boston University, December 2004.

[7] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. *Proc. of the International Conference on Data Engineering (ICDE)*, March 2004.

[8] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *Proc. of the VLDB*, 2002.

[9] G. Cormode and S. Muthukrishnan. Improved data stream summaries: The count-min sketch and its applications. *Journal of Algorithms*, 2004.

[10] A. Deshpande, C. Guestrin, S. Madden, J.M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. *Proc. of the VLDB*, pp. 588–599, 2004.

[11] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[12] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *ACM SIGCOMM Computer Communication Review*, 28(4):254–265, 1998.

[13] P. Flajolet. On adaptive sampling. *COMPUTG: Computing*, 43, 1990.

[14] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31, 1985.

[15] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. *ACM Mobile Computing and Communications Review*, 5(4), 2001.

[16] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. *Proc. of ACM SIGMOD*, 2003.

[17] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. *Proc. of ACM SIGMOD*, pp. 331–342, 1998.

[18] P.B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. *Proc. of ACM SPAA*, pp. 281–291, 2001.

[19] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[20] M. Hadjieleftheriou, J.W. Byers, and G. Kollios. Robust Sketching and Aggregation of Distributed Data Streams. *BU Technical Report*, March 2005.

[21] M. Horton, D. Culler, K. Pister, J. Hill, R. Szewczyk, and A. Woo. Mica, the commercialization of microsensor motes. *Sensors*, 19(4):40–48, April 2002.

[22] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *Proc. of MobiCOM*, 2000.

[23] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. *Proc. of IEEE ICDE*, 2005.

[24] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. *Proc. of OSDI*, 2002.

[25] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. *Proc. of ACM SIGMOD*, 2003.

[26] G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. *Proc. of VLDB*, pp. 346–357, 2002.

[27] S. Nath, P.B. Gibbons, S. Seshan, and Z. Anderson. Synopsis Diffusion for Robust Aggregation in Sensor Networks. *Proc. of ACM SenSys*, pp. 250–262, 2004.

[28] A.J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):253–256, 1977.

[29] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record 31(3)*, 2002.

[30] F. Ye, G. Zhong, S. Lu, and L. Zhang. GRAdient Broadcast: A Robust Data Delivery Protocol for Large Scale Sensor Networks. *ACM Wireless Networks (WINET)*, 11(2), 2005.

[31] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. *Proc. of SNPA*, 2003.

Efficient Strategies for Continuous Distributed Tracking Tasks

Graham Cormode
Bell Labs, Lucent Technologies
`cormode@bell-labs.com`

Minos Garofalakis
Bell Labs, Lucent Technologies
`minos@bell-labs.com`

Abstract

While traditional databases have focused on single query evaluation in a centralized setting, emerging applications require continuous *tracking of queries on data that is widely* distributed *and constantly updated. We describe such scenarios, and describe the challenges involved in designing communication-efficient protocols for the tracking tasks we define. We outline some solutions to these problems, by abstracting a model of the communication system, defining the tracking tasks of interest, and building query-tracking schemes based on three guiding principles of minimizing global information, using summaries to capture whole data streams, and seeking stability of the protocols.*

1 Introduction

Traditional data-management applications such as managing sales records, transactions, inventory, or facilities typically require database support for a variety of *one-shot queries*, including lookups, sophisticated slice-and-dice operations, data mining tasks, and so on. One-shot means the data processing is essentially done once, in response to the posed query. This has led to an enormously successful industry of database engines optimized for supporting complex, one-shot SQL queries over large amounts of data.

Recent years, however, have witnessed the emergence of a new class of *large-scale event monitoring* applications that pose novel data-management challenges. In one class of applications, monitoring a large-scale system is an operational aspect of maintaining and running the system. As an example, consider the Network Operations Center (NOC) for the IP-backbone network of a large ISP (such as Sprint or AT&T). The NOC has to continuously track patterns of usage levels in order to detect and react to hot spots and floods, failures of links or protocols, intrusions, and attacks. A similar example is that of data centers and web-content companies (such as Akamai) that have to monitor access to thousands of web-caching nodes and do sophisticated load balancing, not only for better performance but also to protect against failures. Similar issues arise for utility companies such as electricity suppliers that need to monitor the power grid and customer usage. A different class of applications is one in which monitoring is the goal in itself, such as wireless sensor networks which monitor the distribution of measurements for trend analysis, detecting moving objects, intrusions, or other adverse events.

Examining these monitoring applications in detail allows us to abstract a number of common elements. Primarily, monitoring is *continuous*, that is, we need real-time tracking of measurements or events, not merely one-shot responses to sporadically posed queries. Second, monitoring is inherently *distributed*, that is, the underlying infrastructure comprises several remote sites (each with its own local data source) that can exchange

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

information through a communication network. This also means that there typically are important *communication constraints* owing to network-capacity restrictions (e.g., in IP-network monitoring, where the collected utilization and traffic is very voluminous [6]) or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [14]). Furthermore, each remote site may see a *high-speed stream* of data and has its own local resource constraints, such as *storage-space* or *CPU-time* constraints. This is true for IP routers that cannot possibly store the log of all observed traffic due to the ultra-fast rates at which packets are forwarded. This is also true for the wireless sensor nodes, even though they may not observe large data volumes, since they typically have very small memory onboard.

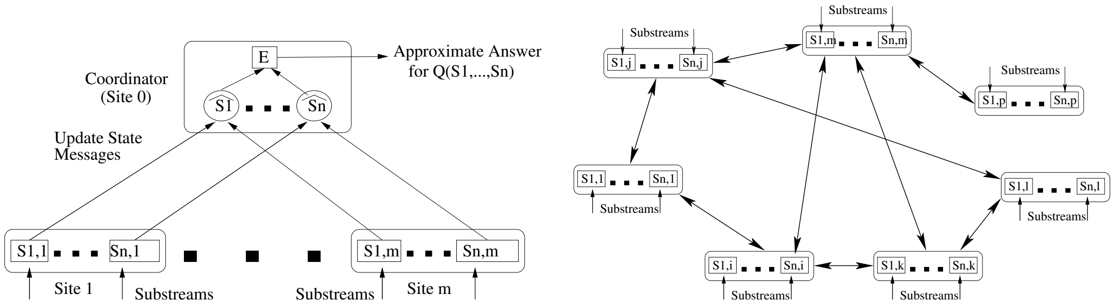
In addition, there are two key aspects of such large-scale monitoring problems. First, one needs a way to effectively track the *complete distribution of data* (e.g., IP traffic or sensor measurements) observed over the collection of remote sites, as well as *complex queries (e.g., joins)* that combine/correlate information across sites. The ability to have an accurate picture of the overall data distribution and effectively correlate information across remote sites is crucial in understanding system behavior and characteristics, tracking important trends and correlations, and making informed judgments about measurement or utilization patterns. In other words, while hardwired outlier detection methods can be of use for certain applications (e.g., network anomaly detection), data-distribution and correlation information gives us a much broader and more robust indicator of overall system behavior — such indicators are critical, for instance, in network-provisioning systems that try to provision routing paths with guaranteed Quality-of-Service parameters over an IP network (e.g., delay or jitter for a VoIP application). Second, answers that are precise to the last decimal are typically not needed when tracking statistical properties of large-scale systems; instead, *approximate estimates* (with reasonable guarantees on the approximation error) are often sufficient, since we are typically looking for indicators or patterns, rather than precisely-defined events. Obviously, this can work in our favor, allowing us to effectively tradeoff efficiency with approximation quality. To summarize, our focus is on large-scale monitoring problems that aim to continuously provide accurate summaries of the complete data distribution and approximate answers to complex queries over a collection of remote data streams. Solutions for such monitoring problems have to work in a distributed setting (i.e., over a communication network), be real-time or *continuous*, and be space and communication efficient; furthermore, approximate, yet accurate, answers suffice.

2 Challenges

In this section, we outline some of the key challenges to address in the area of approximate distributed data-stream tracking. The remainder of the paper further expands on these challenges, and discusses our technical approach and some of our recent results in this area.

• **To Develop New Models and System Architectures for Tracking Distributed Stream Queries.** New frameworks and query-processing architectures are required for the effective, approximate tracking of massive streams in a distributed setting, with guaranteed upper bounds on the approximation error. A suite of architectures may prove suitable, including *single-level hierarchies* (where a single central processing system directly communicates with and coordinates a collection of distributed monitor sites), to *multi-level hierarchies*, and the most general *fully-distributed architectures* (where no central coordinator exists and the goal is for all the distributed monitor sites to efficiently reach consensus on the approximate answer of a distributed stream query). The most applicable from the IP network monitoring point of view is the single-level hierarchy, although other distributed data streaming applications may find the other architectures most suitable.

• **To Identify Key Distributed Data Stream Monitoring Problems.** It is necessary to isolate the basic problems, drawing intuition from multiple application scenarios (such as IP traffic monitoring, mining message streams, and analysis of sensor measurement data), consultation with experts (such as network operators), and so on. Concrete examples that have already begun to be studied include: tracking top-k items [3]; set expression



(a) (b)

Figure 1: (a) Single-Level Hierarchical Stream-Processing Model. (b) Fully-Distributed Model.

cardinality [7]; histograms and quantiles [5]; and join query estimation [4].

• **To Develop New Algorithmic Techniques and Lower Bounds.** In designing novel, communication-efficient algorithms for tracking distributed-stream queries, a key step is to draw on known approximation methods in the centralized data-stream context that are time and space efficient (e.g., based on the linear-projection sketches of [1, 2, 9], or the hash-sketches of [10, 11]). The end goal is overall solutions that are simultaneously efficient in time, space, and communication, while trading off result accuracy. A complementary algorithmic challenge is to prove stronger lower bounds for these problems, which measure the required communication to *continuously* track quantities of interest and show hard theoretical limits in worst-case scenarios.

3 System Models

In this section, we outline a formulation of appropriate models for continuous distributed tracking tasks. Figure 1(a) depicts the *Single-level hierarchical architecture* for update-stream processing, with $m+1$ sites and n (distributed) update streams. Stream updates arrive continuously at *remote* sites $1,\ldots,m$, whereas site 0 is a special *coordinator* site that is responsible for generating answers to continuous user queries Q over the n distributed streams. In this simple hierarchical model, the m remote sites do not communicate with each other; instead, as illustrated in Figure 1(a), each remote site exchanges messages only with the coordinator, providing it with state information for (sub)streams observed locally at the site. Note that such a hierarchical processing model is, in fact, representative of a large class of applications, including network monitoring where a central Network Operations Center (NOC) is responsible for processing network traffic statistics (e.g., link bandwidth utilization, IP source-destination byte counts) collected at switches, routers, and/or Element Management Systems (EMSs) distributed across the network.

At each remote site j, the n update streams render n distinct multi-sets $S_{1,j},\ldots,S_{n,j}$ of elements, that are assumed (without loss of generality) to lie in the integer domain $[M]=\{0,\ldots,M-1\}$. Each stream update at remote site j is a triple of the form $< i, e, \pm v >$, where i identifies the multi-set $S_{i,j}$ being updated, $e \in [M]$ is the specific data element whose frequency changes, and $\pm v$ is the net change in the frequency of e in $S_{i,j}$, i.e., "$+v$" ("$-v$") denotes v insertions (resp., deletions) of element e. [1] For each $i=1,\ldots,n$, let $S_i=\cup_j S_{i,j}$. Thus, S_i reflects the global state of the i^{th} update stream, while each multi-set $S_{i,j}$ captures the local state of stream i at site j. In the development that follows, we will loosely refer to S_i and $S_{i,j}$ as streams even though the intended meaning is the current states of the underlying streams.

Multi-level hierarchical system architectures have individual distributed substream-monitoring sites at the

[1] Other relevant update models are to consider only transactions that are recent (sliding-window model), or to forbid deletions (insertions-only model).

leaves and internal nodes of a general *communication tree*, and the goal is to effectively track a continuous stream query $Q(S_1, \ldots, S_n)$ at the *root node* of the tree. The most general setting are *fully-distributed system architectures*, where individual monitor sites are connected through an arbitrary underlying *communication network* (Figure 1(b)); this is a distinctly different distributed system architecture since, unlike hierarchical systems, no centralized authority/coordination exists and the end goal is for all the distributed monitors to efficiently reach some form of *consensus* on the approximate answer of a distributed stream query. Such generalized distributed stream-processing architectures are representative of, e.g., wide-area networks and sensornet environments. Sensornets, for instance, typically process queries *in-network*, with the sensors performing local measurements and processing, and communicating partial results either over a dynamically-maintained *routing tree* towards a powerful root basestation [12, 13, 14], or through *fully-decentralized, broadcast-based* communication protocols.

It is also important to consider the characteristics of the underlying distributed computing infrastructure; for example, to distinguish between "fat" versus "thin" monitoring clients (e.g., NOCs for local-area networks versus switch Element Management Systems (EMSs), or heavy versus tiny sensornet motes).

4 Distributed Stream Processing Problems

Different distributed stream monitoring tasks rely on a variety of different queries to be evaluated. In this section, we describe some of the specific problems that we have identified.

Tracking Set-Expression Cardinality. The problem of *estimating the result cardinalities of set expressions* over the distributed streams is, given a set expression E over the streams $S_1, \ldots, S_n$ (with the standard set operators $\cup, \cap$, and $-$ as connectives), to estimate the answer to the query $Q = |E|$, i.e., the number of distinct elements in E. For example, the problem of counting the number of distinct IP source addresses described in the context of Distributed Denial of Service (DDoS) monitoring, is a special case of the more general set-expression cardinality estimation problem. As another example of a DDoS-monitoring scenario, we may want to employ the set difference cardinality query $|S_{curr} - S_{prev}|$ to detect significant traffic deviations in real time – here, S_{curr} is the set of source-IP addresses for the sliding window spanning this past week (until now), and S_{prev} denotes the set of source-IP addresses from the week prior to that (e.g., two weeks ago). Set-expression cardinality queries can also provide valuable information to an ISP for network traffic engineering. For instance, suppose that S and T are the sets of IP destination addresses observed in the flows from two of the ISP's peers. Then, the set intersection query $|S \cap T|$ yields the overlap between the destination sets for IP traffic that the peers transmit over the ISP's network. In order to increase network efficiency, it makes sense to locate the traffic exchange points with peers for which this overlap is large, at the same set of border routers (that are optimally situated to carry traffic for the common destinations). A solution based on a dynamic-charging scheme with costs charged to each member of a set, and global broadcast when these charges are recalibrated, which shows significant cost savings over sending every update is given in [7].

Tracking Distributed Stream Summaries. Another important distributed stream task is that of continuously *summarizing the frequency distribution* of a distributed stream of updates S_i, comprising one or multiple data attributes. *Quantiles, Histograms* and *wavelet-coefficient synopses* are natural, useful methods for summarizing data distributions for data visualization and analysis that have been extensively studied in the database literature. The basic goal of all these earlier efforts has been, given a multi-attribute, disk-resident data set and a fixed amount of space, construct the "best" histogram or wavelet synopsis for the given space budget; that is, build the synopsis that minimizes an overall error metric in the approximation of the data distribution, for some suitably-defined notion of error. More recent work has also addressed the problems of quantile finding, histogram and wavelet construction over a single, continuous stream of data in a centralized context.

Tracking Distributed Stream Joins. SQL *join queries* represent another fundamental way of correlating information from two or more distinct data sets (or, streams); thus, effective support for such queries is very important for most of our target distributed stream processing application domains, including network management and sensornet query processing. This class of queries takes the general form Q = "`SELECT AGG FROM` $S_1, S_2, \ldots, S_n$ `WHERE` $\mathcal{E}$", where `AGG` is an arbitrary aggregate operator (e.g., `COUNT`, `SUM`, or `AVERAGE`) and $\mathcal{E}$ represents the conjunction of of equi-join constraints of the form $S_i.A_j = S_k.A_l$ ($S_i.A_j$ denotes the j^{th} attribute of the streaming relation S_i).

5 A General Query-Tracking Framework

We now outline some of our recent work and results on developing general query tracking solutions in the single-level hierarchical model. We have applied this approach to both the quantile and join tracking problems [4, 5].

The goal of our tracking algorithms is to ensure strong error guarantees for approximate answers to queries at the coordinator over the collection of global streams S_i while minimizing communication with the remote sites. We can also identify other important design desiderata that helped guide our overall approach: (1) *Minimal global information exchanges* — schemes in which the coordinator distributes information on the *global* streams to remote sites would typically need to re-broadcast up-to-date global information to sites (periodically or during some "global resolution" stage [3, 7]) to ensure correctness; instead, our solutions are designed to explicitly avoid such expensive "global synchronization" steps; (2) *Summary-based information exchange* — rather than shipping complete update streams $S_{i,j}$ to the coordinator, remote sites only communicate concise summary information (e.g., sketches, quantiles) on their locally-observed updates; and, (3) *Stability* — intuitively, stability means that, provided the behavior of the local streams at remote sites remains reasonably stable (or, *predictable*), there is no need for communication between the remote sites and the coordinator.

Our schemes avoid global information exchange entirely by each individual remote site j continuously monitoring only properties of their own, *local* update streams $S_{i,j}$. When a certain amount of change is observed locally, then a site may send a concise *state-update* message in order to update the coordinator with more recent information about its local update stream, and then resumes monitoring its local updates (Figure 1(a)). Such state-update messages typically comprise a small summary of the offending local stream(s) (along with, possibly, additional trend information), to allow the coordinator to continuously maintain accurate approximate answers to user queries. The tracking scheme depends on two parameters ϵ and θ, where: ϵ captures the error of the local summaries communicated to the coordinator; and θ captures (an upper bound on) the deviation of the local-stream properties at each remote site involved in the query since the last communication with the coordinator. The overall error guarantee provided at the coordinator is given by a function $g(\epsilon, \theta)$, depending on the specific form of the query being tracked. Intuitively, larger θ values allow for larger local deviations since the last communication and, so, imply fewer communications to the coordinator. But, for a given error tolerance, the size of the ϵ-approximate summaries sent during each communication is larger (since $g(\epsilon, \theta)$ is always increasing in both parameters). This tradeoff can be analyzed under worst-case assumptions, and an optimal setting of ϵ and θ can be given, for an overall error bound and a fixed $g(\epsilon, \theta)$.

A local summary communicated to the coordinator gives an (ϵ-approximate) snapshot of the $S_{i,j}$ stream at time t.[2] To achieve *stability*, a crucial component of our solutions are concise *summary-prediction models* that may be communicated from remote sites to the coordinator (along with the local stream summaries) in an attempt to accurately capture the anticipated behavior of local streams. The key idea here is to enable each site j and the coordinator to share a prediction of how the stream $S_{i,j}$ evolves over time. The coordinator employs this prediction to answer user queries, while the remote site checks that the prediction is close (within θ bounds) to the actual observed distribution $S_{i,j}$. As long as the prediction accurately captures the local update behavior at the remote site, no communication is needed. To keep communication costs low, we need to reflect the predicted

[2] To simplify the exposition, we assume that communications with the coordinator are instantaneous. In the case of non-trivial delays in the underlying communication network, techniques based on time-stamping and message serialization can be employed to ensure correctness, as in [15].

distribution with a predicted summary, by taking advantage of linearity or other properties of the summaries. For example, the random sketches of [1, 2] are linear projections of the distribution, so scalings and combinations of the distribution can be predicted by scaling sketches; heavy-hitter counts can be predicted by scaling previous counts; quantiles may be assumed to remain the same, or change slowly; and so on. Thus, our predictions are also based solely on concise summary information that can be efficiently exchanged between remote site and coordinator when the model is changed. The key insight from our results is that, as long as local constraints are satisfied, the combined predicted summaries at the coordinator are basically equivalent to $g(\epsilon, \theta)$-*approximate summaries* of the global data streams. We now describe how we have mapped this general outline onto two distinct problems, and the different results that follow from fleshing out the required details of our framework.

Quantile Tracking. For quantile tracking, the summaries used are the ϵ-quantiles of each remote stream [5]. It can be shown (Theorem 3.1 of [5]) that, so long as the deviation between the predicted ranks of the most recently recorded set of ϵ-quantiles observed locally and their true ranks is less than $\theta|S_{i,j}|$, the coordinator can accurately answer quantile queries with bounded error. The error function, $g(\epsilon, \theta)$ is given by $\epsilon + \theta$. In order to achieve stability, a variety of prediction models can be applied. The simplest is to assume that ranks are fixed; the effect of this is to cause a communication when the number of updates (in an insert-only model) exceeds a θ-fraction of the size of the stream at the last update, $|S_i|$. Analyzing this shows that the communication cost is optimized by setting $\theta = \epsilon$, and the overall cost is $O(\frac{1}{\epsilon^2} \ln |S_{i,j}|)$ (Lemma 3.4 of [5]). A more sophisticated prediction tracks the rate of change of each local stream, and models this with a linear "velocity" component. Although not easily amenable to analysis, this model shows significant communication savings over the naive solution of sending every update to the coordinator [5].

Join Size Estimation (Sketch Tracking). To track binary and multi-way join aggregates between streams (and a variety of other problems) [4], the summaries used are variations of the sketches defined by Alon et al. [1]. If the coordinator holds a sufficiently accurate sketch of each local stream, then it can be shown (Theorem 3.1 of [4]) that the coordinator can build a sketch that is $g(\epsilon, \theta)$-accurate for answering join-size queries. Formally, the answer has additive error $g(\epsilon, \theta) = \epsilon + (1 + \epsilon)^2((1 + \theta)^2 - 1) \approx \epsilon + 2\theta$ times the product of the L_2 norms of the frequency vectors (which is of comparable magnitude to the error due to using approximate sketches). A prediction model is applied to predict the distribution at time t, but, because of the linearity properties of the sketches, the effect is to produce a *predicted sketch*. In order to give this guarantee, each remote site ensures that the deviation between the sketch of the true distribution and its predicted sketch is bounded by $\theta\|S_{i,j}\|_2$, a θ-factor times the L_2 norm of the stream. Again, simple prediction models based on the assumption that the distributions generated by the streams remain static are reasonably successful at reducing the amount of communication between remote sites and the coordinator. More sophisticated prediction models capture the dynamics of streams by modeling first- and second-order effects: the prediction is made up by adding a "velocity" and "acceleration" component to the last sketch sent to the coordinator. This is often very successful at capturing the movement of the streams, hence further reducing communication cost, but requires a careful balancing act: since the coordinator bases its approximate answers on the current prediction, details of the model must be sent to the coordinator. Thus, a more complex model also requires more communication, and the correct balance between complexity of the model and accuracy with which it captures the stream distributions needs to be struck.

The overall result of these approaches are conceptually simple but effective schemes for tracking a variety of aggregates of interest. In practical evaluations [4, 5] these can be seen to use an amount of communication that is significantly smaller (tens or even hundreds of times less) than the cost of shipping every update to the central coordinator for processing, while giving the overall $g(\epsilon, \theta)$ accuracy guarantees for query answering.

Generalizing the above results gives solutions for other query types. The quantile structure naturally extends to finding the "heavy hitters" (frequent items), by applying similar models and bounds to the local heavy-hitter items, and combining them appropriately at the coordinator [5]. More generally, our sketch-tracking solution gives a $g(\epsilon, \theta)$-approximate sketch at the coordinator; this can then be used to answer a variety of data-analysis problems which make use of sketches: self-join size approximation, building multi-dimensional histograms and wavelets, answering point and range queries, and so on [4].

Our techniques also extend naturally to the multi-level hierarchical setting, since they effectively give a $g(\epsilon, \theta)$-approximate summary (quantiles or sketch) at the coordinator. Therefore, if the coordinator runs the same protocol with its parent in the tree, the result at the next level is a $g(g(\epsilon, \theta), \theta)$-approximate summary. This approach can be continued up the hierarchy, increasing the error bound at each step [5]. However, many challenges remain unanswered, including (a) how to better take advantage of correlations or anti-correlations across remote sites (the schemes described in this section treat each remote site entirely independently, which gives simple to manage solutions, and no global information, but could benefit from utilizing greater knowledge about the behavior of the streams); and, (b) how to extend to the more general, fully-distributed model. There is much scope for work that builds very different models of the update streams—a completely different notion of models is used in [8] for choosing which sensors (sites) to query in a one-shot query evaluation setting.

6 Concluding Remarks

We have introduced a class of problems based on continuous, distributed tracking tasks. We have described three key challenges: to develop new models and architectures for such queries; to identify key problems in this area; and to provide new algorithms and theoretical lower bounds for them. We have also outlined some of our recent and ongoing work to address these three challenges.

Acknowledgments. We thank S. Muthukrishnan for several useful discussions related to this work.

References

[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". In *ACM PODS*, 1999.

[2] N. Alon, Y. Matias, and M. Szegedy. "The Space Complexity of Approximating the Frequency Moments". In *ACM STOC*, 1996.

[3] B. Babcock and C. Olston. "Distributed Top-K Monitoring". In *ACM SIGMOD*, 2003.

[4] G. Cormode and M. Garofalakis. "Sketching Streams Through the Net: Distributed Approximate Query Tracking". Bell Labs Tech. Memorandum, 2005.

[5] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. "Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles". In *ACM SIGMOD*, 2005.

[6] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications". In *ACM SIGMOD*, 2003.

[7] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. "Distributed Set-Expression Cardinality Estimation". In *VLDB*, 2004.

[8] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. "Model-Driven Data Acquisition in Sensor Networks". In *VLDB*, 2004.

[9] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. "Processing Complex Aggregate Queries over Data Streams". In *ACM SIGMOD*, 2002.

[10] P. Flajolet and G. N. Martin. "Probabilistic Counting Algorithms for Data Base Applications". *Journal of Computer and Systems Sciences*, 31:182–209, 1985.

[11] S. Ganguly, M. Garofalakis, and R. Rastogi. "Processing Set Expressions over Continuous Update Streams". In *ACM SIGMOD*, 2003.

[12] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. "Beyond Average: Towards Sophisticated Sensing with Queries". In *IPSN*, 2003.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks". In *OSDI*, 2002.

[14] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "The Design of an Acquisitional Query Processor for Sensor Networks". In *ACM SIGMOD*, 2003.

[15] C. Olston, J. Jiang, and J. Widom. "Adaptive Filters for Continuous Queries over Distributed Data Streams". In *ACM SIGMOD*, 2003.

Resource-Aware Wireless Sensor-Actuator Networks

Amol Deshpande
University of Maryland
amol@cs.umd.edu

Carlos Guestrin
CMU
guestrin@cs.cmu.edu

Samuel R. Madden
MIT
madden@csail.mit.edu

Abstract

Innovations in wireless sensor networks (WSNs) have dramatically expanded the applicability of control technology *in day-to-day life, by enabling the cost-effective deployment of large scale sensor-actuator systems. In this paper, we discuss the issues and challenges involved in deploying control-oriented applications over unreliable, resource-constrained WSNs, and describe the design of our planned* Sensor Control System (*SCS*) *that can enable the rapid development and deployment of such applications.*

1 Introduction

Embedded sensing and control systems are used on a daily basis by nearly all consumers and businesses in the industrialized world. These systems are critical in manufacturing, automobiles, aviation, building heating and air-conditioning, power distribution, and a huge array of other domains. With dramatically dropping hardware prices, wireless sensor networks (WSNs) are finally becoming a reality; *low-power, wireless embedded control* systems have the potential to significantly alter and expand the applicability of control technology. In industrial control settings, for example, wireless networks can be installed for a fraction of the cost of wired devices (one study estimates that *each* physical wire in a commercial workplace costs $800.00 to install [22]), and can provide unprecedented flexibility, with high density sensing and deployments in unsafe areas that may be impossible to instrument with standard wired approaches (such as inside waterways, or in high-temperature oil refineries).

Realizing this potential, however, requires the computer science community to develop novel software technologies that can enable rapid development and deployment of wireless sensing and control systems. Though there has been much interest in developing such software for deploying and maintaining pure data collection systems, developments in integrating battery-powered, wireless technologies into closed-loop control settings have been limited. In this paper, we describe the design of the *Sensor Control System* (*SCS*) that we are currently building. *SCS* consists of an application infrastructure and a suite of embedded systems software that is *designed to allow rapid deployment of control-oriented wireless applications*.

As a simple example of a control-oriented wireless sensor network deployment, we consider systems for *building control.* Modern buildings typically feature highly sophisticated heating, ventilation and air conditioning (HVAC) systems that enable fine-grained monitoring and control over the HVAC settings. As with many embedded control systems, these systems typically have four major components: (1) A set of *preferred temperature levels* that the building users specify; (2) *sensors* that monitor the temperature, humidity, etc. in various

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

parts of the building; (3) *actuators* that control the heating/cooling devices such as air conditioning vents and furnaces; and (4) a *controller* which implements a *control law* that takes the current sensor values and selects a setting to the actuators that attempts to fulfill the users' preferences. In addition to monitoring temperatures, the sensor network can monitor occupancy in the building to optimize HVAC and lighting settings to the needs of the current occupants. Due to ever increasing energy costs, better monitoring and control of such systems can significantly reduce the building's operation expenses.

HVAC systems are one example of a larger set of control systems in offices and homes that can benefit from wireless sensor networks. A modern home contains a large number of appliances and equipment that can be both remotely monitored and controlled. Besides the heating and cooling system, such systems include lights, digital media equipment, and major appliances like refrigerators and ovens. The possibilities for automation using sensing devices are numerous – for example, a user might express a preference such as "keep the floor clean", and "do not run the noisy robotic vacuum cleaner when I am listening to music". This is a simple control system that uses sensors that detect dirt and the presence of a user, and actuates the vacuum cleaner as needed.

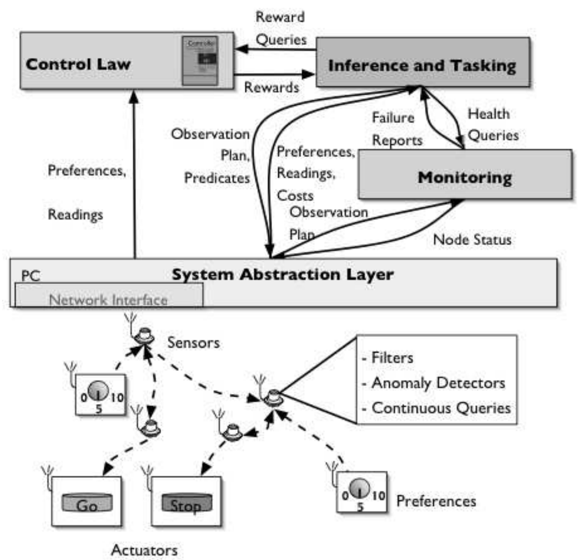


Figure 1: The basic architecture of the *SCS* platform.

Unfortunately, low-cost wireless networks are often *resource constrained*, with limited power, computation and communication capabilities, and are *unreliable*, sensors can fail due to manufacturing defects, power overuse, or environmental impact. These two real-world limitations of sensor networks suggest opposite solution approaches: resource constraints suggest that we should we reduce sensing and communication to a minimum, while unreliability suggests that we add sensing redundancy and increase data collection rates. To address these conflicting needs, the approach we advocate uses statistical models and control theory to design robust control laws and tailor sensing requirements depending on the precision needed by the control law at each moment in time. Our resource-aware approach builds on this theory by choosing the sensor readings to capture and actuators to adjust that will incur the minimum energy cost to the system subject to the constraint that users' preferences regarding the state of the system are approximately satisfied.

1.1 Challenges

There are significant challenges that need to be addressed to enable the effective deployment of control systems with sensor networks.

Resource limitations of WSNs: Large networks of wireless sensors and actuators pose a number of problems that are not present in smaller networks of wired devices. Although current generation devices, such as the Mica motes, have limited processors (motes have an 8-bit, 7 MHz processor) and memory (motes have 4 KB of RAM and 512 KB of Flash), we expect that in a few years these limitations will be much less severe. However, there are a number of other limitations we anticipate will still be significant for the foreseeable future [16].

First, the *network channel is lossy and has variable latency*, which means that control applications have to be tolerant of missing and delayed data. Loss rates can be 50% or greater when multiple nodes are transmitting simultaneously; retransmissions can mitigate this, at the expense of increased and unpredictable latency as well as additional energy utilization [28, 29]. Second, *sensor readings are noisy and subject to drift over time*, even when carefully calibrated [26]. Third, *devices fail*, as batteries run out or harsh environmental conditions damage hardware, meaning that the control system needs to be prepared to detect those failures and work around them to satisfy the user-specified control law as best as possible [26]. Finally, the battery-powered nature of most WSNs means that *power is of utmost importance, and must be aggressively conserved* [21, 11].

Ease of use and deployment: The second major class of challenges relates to the difficulty associated with deploying, configuring, and learning how to use automated control systems, particularly in home automation settings [19]. To reap the full benefits of the technology trends in sensing and actuating technology, we must make this deployment process much less difficult than it is today.

There are many aspects to the user-system interaction in this setting. First, *deploying a sensor network should not require knowledge of the low-level functioning of sensor networks*, or the wireless technology. Similarly, addition and removal of sensors should be made very easy. Second, the specification of the *control law* that specifies the optimization goal should be intuitive and should not require detailed knowledge of control theory. Finally, the system should be able to *hide the peculiarities of the sensor networks* such as dynamic topologies, missing data and/or lossy sensors from the user.

1.2 Prior Work

Though there has been much work in developing and deploying embedded control systems that use *wired* sensors and actuators [2, 20], using low-power wireless sensor-actuator networks fundamentally changes the nature of the problem because of the bandwidth and power limitations of these networks (Section 2). Wireless sensor networks itself has been a very active area of research in recent years [1, 30], but most of this work has focused on the *sensing* aspect of WSNs, and not as much on *actuation*. Due to lack of space, we omit a detailed discussion of the prior work, and refer the reader to the full version of the paper [7].

1.3 Planned System Architecture

Figure 1 illustrates the design of our planned sensor control system, *SCS*. Main components of the system are:

1. *Inference and tasking component:* This component is responsible for deciding which sensors and preferences to observe, for inferring which readings will be most valuable and which actuators will best address the user's needs.
2. *Control law component:* This component encapsulates the relationship between the sensor readings, users preferences, and state of the actuators.
3. *Monitoring component:* This component is responsible for determining the status of each of the nodes of the network (*e.g.,*, whether it is online, whether its sensors are functioning properly, etc.).
4. *System abstraction layer:* The system abstraction layer is responsible for collecting preferences and sensor values from the network and for disseminating actuator values throughout the network while simultaneously managing the wake/sleep schedule of the nodes, synchronizing their clocks, and managing their interface to low level actuators and sensors.

In the rest of the paper, we briefly discuss issues in applying control theory in wireless sensor networks, and elaborate upon the components of *SCS*.

2 Applying Control Theory in Power-Constrained Sensor Networks

The control and monitoring systems described in Section 1 share three common requirements: observing the state of the system, obtaining user preferences, and disseminating the action choice, or actuator setting. In settings that combine wireless networks with embedded control, we must fulfill these requirements while simultaneously addressing resource constraints, such as limited power, low network bandwidth, changing link qualities, and sensor noise and faults. In this section, we formalize the requirements for a general platform for embedded control using sensor networks.

2.1 Sensor Network Abstraction for Embedded Control

Embedded control systems typically use a *control law* to decide appropriate settings for various actuators given the current state of the system. More formally, a *closed-loop control law* $\pi(\mathbf{x})$ provides a mapping from states $\mathbf{x}$ of sensors and user preferences to actions $\mathbf{a}$ [4, 2, 20] taken on the actuators. For example, in HVAC, the control law will describe whether air conditioning vents need to be opened or closed, given the current temperatures at every location. More generally, if there is uncertainty about the system state, *e.g.,* due to unreliable sensors, the control law maps a probability distribution $\boldsymbol{b}(\mathbf{x})$ over the state of the system (also called a belief state) to an action choice [25, 12, 15]. Again in HVAC, if the sensors are making *noisy* observations of the temperature values, instead of having perfect knowledge of the state, we might only have a probability distribution over the possible states of the system. The action choice then depends on the distribution, *e.g.,* on its mean and variance.

In *SCS*, the sensor network provides the control law with a probability distribution over the possible states of the system $\boldsymbol{b}$, and with the current setting for the user preferences. These are used by the control law to specify the current action choice, which is then disseminated to the specific actuators in the network. In order to specify the belief state $\boldsymbol{b}(\mathbf{x})$, we need a probabilistic model that relates the noisy observations made by the sensor network with the true state of the system [13, 23]. To do this, we need: (1) a *sensor model* that specifies a probability distribution over possible sensor measurements $\mathbf{o}$ given the current state of the system $\mathbf{x}$, and (2) a *prior distribution* that specifies our initial belief about the system state. Consider an HVAC system with a single temperature sensor i with Gaussian noise. In that case, the sensor model $P(o_i \mid x_i)$ would have Gaussian distribution, centered around the *true* temperature. The prior $P(x_i)$ could also be a high-variance Gaussian that represents the usual distribution of temperatures during the current time of year. Given these, we can obtain our belief state $\boldsymbol{b}(x_i \mid o_i)$ as the distribution over possible states after the sensor value is observed (also called the *posterior distribution*) using Bayes rule:

$$\boldsymbol{b}(x_i \mid o_i) = P(x_i \mid o_i) \propto P(o_i \mid x_i)P(x_i).$$

In a more general setting, we may have observed a subset $\mathcal{O}$ of a total of n sensors, and would like to compute the posterior distribution over the state of the whole system $\mathbf{x}$ given the observed value $\mathbf{o}$. Note that in general *hybrid* systems the state $\mathbf{x}$ contains both discrete and continuous variables [24, 3]. We again use Bayes rule:

$$\boldsymbol{b}(\mathbf{x} \mid \mathbf{o}) = P(\mathbf{x} \mid \mathbf{o}) \propto P(\mathbf{o} \mid \mathbf{x})P(\mathbf{x}). \tag{1}$$

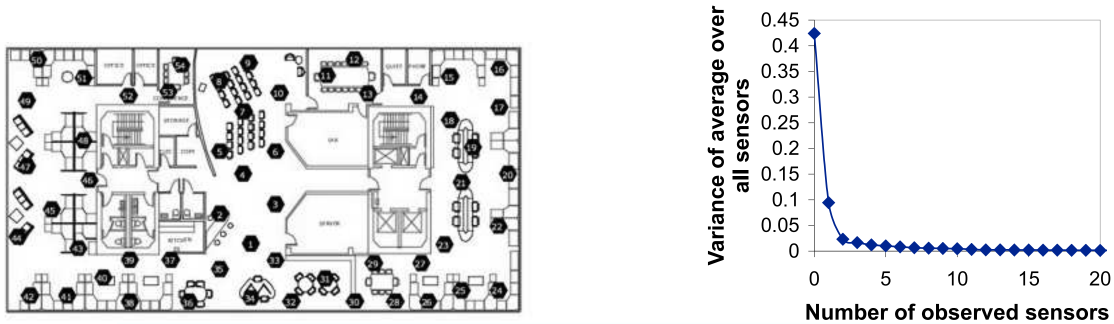
In HVAC, this belief state is the posterior probability of possible temperatures at all locations, given the observed sensor values at a subset of the locations. The prior distribution $P(\mathbf{x})$ and the sensor model $P(\mathbf{o} \mid \mathbf{x})$ can be learned from historical data using standard machine learning algorithms [18, 8].

Control systems usually deal with processes that evolve over time. The evolution of real systems is usually uncertain, and should also be modeled in terms of probability distributions by a process called *filtering* [13, 23]. We omit the details of this process due to lack of space.

2.2 Limited Sensing, Preference Acquisition and Action Dissemination

Collecting sensor values and user preferences, and disseminating action choices as dictated by the control law, requires significant energy expenditure in wireless sensor networks. Figure 2 (i) shows a wireless network we deployed at the Intel Lab in Berkeley. In this deployment, if we were to continuously collect data, the network would only survive for a few days due to battery capacity constraints. Thus, we strive to minimize the amount of data we must capture.

Although a few control systems take such sensing costs into account [23], most do not have such capabilities. Furthermore, in wireless sensor networks, the cost of sensing is non-local: since we need to traverse the network to collect sensor values, the cost of data gathering is non-linear in the number of data points. Though the cost of making a temperature measurement is independent of the sensor chosen, the cost of collecting the data can vary



(i) (ii)

Figure 2: (i) Layout of sensor network deployment with 54 nodes; (ii) Variance of the *average* temperature decreases rapidly in the beginning as observations are made sequentially.

widely with the actual subset collected. For the lab deployment above, the cost of collecting data from a sensor varies from a minimum of 5 mJ to a maximum of 191 mJ depending on the sensor chosen. Additionally, control systems rarely consider the cost of collecting user preferences and disseminating action choices which, again, are non-linear and possibly very high.

Fortunately, sensor measurements are highly correlated. Figure 2 (ii) illustrates this on real temperature data from our lab deployment, plotting the variance of the value of average temperature over the sensor network as we observe the values of the sensors sequentially. As we can see, though the initial variance in our estimate of the average temperature is quite high, it decreases very rapidly as observations are made, thus allowing us to estimate the correct average with very high accuracy with only a few observations.

We have recently developed a framework for exploiting such correlations between sensor values to significantly reduce the amount of communication and sensing required in the network [6]. By collecting only a subset of observations, the energy savings can be very significant. However, our estimate of the belief state, $\boldsymbol{b}^t$, at time t, may become less certain than when all sensor values are incorporated. We must be careful, as this additional uncertainty could potentially cause a significant decrease in the effectiveness of the control law, and bound the effect of approximate estimates of the belief state on the control law [2].

In addition to limiting the observations of sensor values, *SCS* also considers the potential to costs of disseminating action choices and collecting user preferences using models and uncertainty. Some research has been done in the context of uncertainty over user preferences, considering probability distributions over possible preferences [5]. We can build on this framework, using techniques analogous to those in [6] to minimize the amount of preference information communicated.

Similarly, the actuation setting may not change significantly from one time step to the next. In such cases, we can save energy and bandwidth by not updating the actuation values at some nodes. In particular, if the new actions do not change the reward function significantly from the optimal value, they should not be disseminated. Here, again, we can use control theory bounds to estimate the loss in rewards from these local perturbations [2].

2.3 Resource Management and Optimization

Given the setting outlined thus far, we can formalize a very significant challenge that we plan to address: **resource optimization in sensor networks for control systems**. In particular, our goal is the selection of the subset of sensor values and preferences to retrieve, and new action settings to disseminate that will utilize minimum power resources, while still provably maintaining bounds on the quality of the control law. This selection process includes three important challenges that we address in *SCS* :

1. **Cost estimation:** Once we select a set of nodes (actuators to disseminate actions, and sensors to collect data), we must determine the best protocol to visit these nodes, and the expected cost of running this

protocol on them. Though we have done some initial work in this direction [17], designing a efficient data collection protocol that can function in presence of arbitrary failures is one of our research goals. In Section 3, we present an active network monitoring component on which this protocol will be based.

2. **Decrease in control quality due to limited data collection and action dissemination:** Here, we are faced with a very interesting theoretical challenge. We have three sources of loss:
   - **Sensing loss:** By observing only a subset of the nodes, our estimate of $\boldsymbol{b}^t$ may be more uncertain. We can bound the effect of such uncertainty using the techniques in [2].
   - **Preference loss:** User preferences can change over time, if these changes are not monitored continuously, we may have uncertainty over user preferences. This issue can be tackled by a combination of probabilistic models of preferences [5] and sensitivity analysis techniques for control laws [27].
   - **Action dissemination loss:** If actuators are not informed of changes in the desired action, the quality of the control law may decrease. An initial solution for this problem can be based on the sensitivity analysis [2], using the difference between the optimal action setting and the action setting that the nodes are currently using. A novel, potentially more effective approach that we are developing as part of *SCS*, is to perform a constrained optimization of the action selection mechanism. Given that we are going to visit a subset of the nodes, we should compute the best action choice for this subset, assuming that the other nodes' action choices remain constant. This constrained optimization approach can potentially lead to significantly lower loss in the quality of the control strategy.

3. **Optimization of subset of nodes:** Finally, given the cost function and the estimate for the quality loss of the control law, we must solve the computationally challenging problem of *optimizing the subset of visited nodes* given a maximum allowable decrease in the quality of the control law. In our previous work [6], we have proposed a simple, but highly effective, heuristic for the (narrower) problem of selecting a subset of nodes to visit to most efficiently increase the confidence in the estimate of the global system state. Even though the general problem of selecting which nodes to visit, which attributes to acquire, and which actuations to perform at every time step seems extremely challenging at first, we believe that it is possible to develop efficient approximation algorithms for this problem. In addition, for many practical cases, we believe that the use of machine learning techniques can allow us to obtain provably near-optimal solutions.

3 Active Network Monitoring as a Resource Optimization Problem

Effective monitoring of the sensor network system itself is a central part *SCS*, since our algorithms depend on up-to-date topology and correlation information to function most efficiently. Hence, a core piece of *SCS* is a tool that monitors the network topology (including link quality and network dynamics), the status of individual nodes (*e.g.*,, remaining battery life, whether they are failed or producing noisy readings, etc.), as well as the overall quality of the probabilistic models used to drive the control system. This last condition is particularly important as it allows us to decide when our models need to be re-learned, *e.g.*,, from a new set of historical data.

In addition to enabling our optimization algorithms, and allowing us to perform maintenance decisions, a monitoring tool will also enable us to address another interesting challenge: making decisions about where to place sensors. The basic idea is as follows: if our state estimates $\boldsymbol{b}^t$ are too uncertain to satisfy user-specified certainty goals, our monitoring system should also allow us to determine the positions of new sensors that will most likely decrease uncertainty. Similarly, if communication quality is too low, our monitoring tool should recommend placements that will help improve the communication topology.

The network topology, the status of a sensor, or the quality of a model are all descriptions of the current state of the system analogous to $\boldsymbol{b}(\mathbf{x})$ in Section 2.1. We can thus view the monitoring problem in the same resource management light described in Section 2. However, there are some interesting research challenges specific to the monitoring problem:

1. **Failure models:** Models for outlier detection and failures modes are essential for effective monitoring. Our goal is to scale existing machine learning techniques for this purpose (*e.g.*,, [8, 14]) to work with the limited resources and distributed, dynamic regime of wireless networks.
2. **Model selection and evaluation:** Again, building on statistical techniques (*e.g.*,, [10]), we can evaluate the quality of the fit of our control and fault models to the observed set of sensor readings and actuator responses. This evaluation step allows us to detect inappropriate models and retrain the system as needed.
3. **Resource management:** The optimization algorithms in Section 2.1 also need to be extended to incorporate these outlier detection and fault models. A general, unified framework of this sort can significantly decrease the resource load on the sensor network by allowing data used for outlier detection to also be used in our control-oriented planning algorithms.
4. **In-network component:** A sensor node, or nearby nodes, should be able to detect failures more rapidly locally than is possible from a central base station. On the other hand, the base station has a "global" view that can detect widespread correlated failures. Thus, we are developing an approach that integrates an in-network detection component with our centralized model-based tool to balance this tradeoff appropriately.

Although learning failure models and model selection are relatively well-studied tasks in the machine learning literature, the resource management and in-network components are novel, and require significant new research. Others in the systems community have looked at tools for sensor network deployment and management [16, 9]. We believe our approach, however, can detect failures and recommend placements for new nodes in ways that other tools cannot by exploiting statistical correlations and models of network behavior.

4 A Low-Level System Interface for Sensornet Control

To hide the low-level functioning of the sensors and actuators from the higher level control and monitoring facilities, we utilize a *system abstraction layer* that provides the following basic API:

1. *Estimate the current network topology and link qualities*: from networking data collected over time.
2. *Observe one or more sensors or preferences:* Using the current network topology, or a user specified path, collect readings from a specified set of sensors or preferences and notify the user as these arrive.
3. *Set the states of one or more actuators*: Using the current network topology, route action choices to a subset of the actuators.
4. *Continuously monitor the state of a subset of the nodes*: Using a user-specified predicate, continuously monitor those devices for sensor readings or new preferences that satisfy the predicate.

Though this API may appear to be missing many of the features one would expect to find in a "general purpose" interface to WSNs, we believe that this API will address a wide range of applications, and its simplicity will allow us to provide effective response times and power efficiency. One of the long-term research challenges is to refine this API such that it strikes the appropriate balance between simplicity, expressiveness and efficiency.

5 Conclusions

In this paper, we described the design of our planned sensor control system, *SCS*, that is aimed towards enabling rapid development of software control and monitoring systems using low-power wireless networks. The design of our system is motivated by both the needs of real-world monitoring and control systems, such as heating, ventilation, and air conditioning problems (HVAC), process control systems (PCS), and health monitoring of industrial equipment (HM), and the peculiarities of power-constrained wireless sensor networks not observed in more prevalent wired sensor networks. Our platform works by combining techniques from machine learning and statistics with high-level abstractions inspired by work in software systems and databases, allowing for mathematically-sound decision making despite the loss and uncertainty that is inherent in these networks. *SCS* also insulates users from the low-level sensor network details by allowing them to express control system application requirements at the granularity of the entire network, and enabling them to focus on the novel requirements of their deployments.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38, 2002.

[2] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

[3] H. A. P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33(8):780–783, August 1988.

[4] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Blaisdell, New York, 1969.

[5] U. Chajewska and D. Koller. Utilities as random variables: Density estimation and structure discovery. In *UAI*, 2000.

[6] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[7] Amol Deshpande, Carlos Guestrin, and Samuel Madden. Resource-aware wireless sensor-actuator networks. 2005.

[8] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, New York, 2001.

[9] L. Girod, T. Stathopoulos abd N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *SenSys*, 2004.

[10] D. Heckerman, D. Geiger, and M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, Redmond, Washington, 1994.

[11] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, and David Cullerand Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of ASPLOS*, November 2000.

[12] R. A. Howard and J. E. Matheson. Influence diagrams. In *Readings on the Principles and Applications of Decision Analysis*. 1984.

[13] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960.

[14] U. Lerner, B. Moses, M. Scott, S. McIlraith, and D. Koller. Monitoring a complex physical system using a hybrid dynamic bayes net. In *UAI*, 2002.

[15] M. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, 1996.

[16] S. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. PhD thesis, UC Berkeley, 2003.

[17] A. Meliou, C. Guestrin, and J. Hellerstein. Routing queries in sensor networks. Tech. report, Intel Research, 2004.

[18] Tom Mitchell. *Machine Learning*. McGraw-Hill, Boston, Massachusetts, 1997.

[19] M. C. Mozer. Lessons from an adaptive house. *Smart environments: Technologies, protocols, and applications*, 2004.

[20] K. Ogata. *Modern Control Engineering*. Prentice Hall, Upper Saddle River, NJ, fourth edition, 2001.

[21] Greg Pottie and William Kaiser. Wireless integrated network sensors. *Comm. of the ACM*, 43(5), May 2000.

[22] C. Rios. WLANs and power over ethernet, 1999. `www.ieee802.org/3/power_study/public/july99/rios_1_0799.pdf`.

[23] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[24] R. H. Shumway and D. S. Stoffer. Dynamic linear models with switching. *Journal of the American Statistical Association*, 86(415):763–769, September 1991.

[25] R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[26] R. Szewczyk, A. Mainwaring, J. Polastre, D. Culler. An analysis of a large scale habitat monitoring application. In *SenSys*, 2004.

[27] C. C. White and H. K. Eldeib. Markov decision processes with imprecise transition probabilities. *Operations Research*, 42(4):739–749, 1994.

[28] A. Woo, T. Tong, D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys*, 2003.

[29] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys*, 2003.

[30] F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, 2004.

Efficient Queries in Peer-to-Peer Systems

Prasanna Ganesan Hector Garcia-Molina
Stanford University
$\{$*prasannag, hector*$\}$*@cs.stanford.edu*

Abstract

A peer-to-peer (P2P) system consists of a large, dynamic set of machines, distributed over a wide-area network, collaboratively storing and managing data in a fully decentralized fashion. The scale, dynamism and network-wide distribution of P2P systems provide new twists to age-old data management problems, requiring the development of novel techniques for data storage and retrieval. In this article, we describe solutions developed by the Stanford P2P group for enabling efficient queries in P2P systems. We focus particularly on how to optimize data storage to support various kinds of queries, including range queries, multi-dimensional queries and similarity search.

1 Introduction

The peer-to-peer (P2P) model of distributed computation first rose to prominence thanks to the popularity of internet-scale file-sharing applications such as Napster and Kazaa. Evolving from those humble beginnings, P2P data management has now emerged as a major new area of computer science research with applications to large-scale networking infrastructure, wide-area monitoring, collaborative data sharing, data archival and, in some cases, even to traditional parallel and distributed databases. In this article, we discuss some of the research undertaken at Stanford in the quest for the holy grail of peer-to-peer data management — a system that offers powerful, database-like declarative queries while being implemented over a peer-to-peer substrate[1].

What is a P2P system? The difference between P2P systems and traditional parallel and distributed systems may be captured by the following three "laws" of P2P:

I. **All peers are born equal.** P2P systems are completely decentralized, with the constituent peers being functionally identical; there are no central servers in the system, and the system load is divided equally across all peers.

II. **The set of peers is dynamic.** Peers may join and leave the system at will, and the system size may itself vary dramatically over time. For example, the Kazaa network contains twice as many peers during the day as it does at night.

III. **Peers are distributed over a wide-area network.** The participant peers may potentially be spread out across the internet. Consequently, communication characteristics can be very heterogeneous; a peer may experience very low-latency communication with some other peers that are geographically close to it, but may also experience higher latencies when communicating with other peers that are further away.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] A holier grail to aspire for would be a system that additionally offers highly reliable storage and strong transactional semantics, two areas of active research that we do not consider in this paper.

Each of the above three laws present significant challenges to efficient query processing in a P2P system. The first law forces decentralized design and the enforcement of load-balance guarantees in the system. The second law requires highly adaptive designs for data storage and query execution to deal with peers frequently joining and leaving the system. The third law requires us to deal with widely distributed peers by localizing communication as much as possible, thereby reducing query latency and optimizing the usage of network bandwidth. The problem of tackling all the above challenges and ensuring efficient queries can be broken down into two partially independent sub-problems: (1) **Storage:** Determining how to partition and store data across peers in an efficient fashion; and, (2) **Retrieval:** Determining how to retrieve the relevant data from across the network when a query is posed.

In the following sections, we will discuss how both these problems may be solved to efficiently support various classes of queries, by combining appropriate storage solutions with overlay networks that enable efficient routing of queries and consequent retrieval of relevant data. Due to space limitations, our description will focus on the storage methods and provide a far more cursory overview of the associated retrieval algorithms and routing networks.

2 Key Lookups and Distributed Hash Tables

Distributed Hash Tables (DHTs) implement a simple dictionary abstraction, storing *(key,value)* pairs and allowing retrieval of all values that correspond to a given key (a selection query with an equality predicate) as well as insertion and removal of *(key, value)* pairs. While a large number of solutions have been proposed for implementing this abstraction [1, 2, 3, 4], all of them are founded on the same basic principles – principles that we illustrate with one example system, Chord [2].

Storage. Chord uses a technique known as consistent hashing [5] to partition data across nodes (peers). In consistent hashing, data keys are hashed into a large (circular) N-bit identifier space, and each node is assigned a unique *ID*, drawn at random from this same N-bit space. At any time, the set of node IDs induces a partitioning of the hash space, and each node stores data falling in the hash bucket ranging from its predecessor's ID to its own. Note that the assignment of hash buckets changes in the obvious fashion when nodes join and leave. The fact that IDs are chosen at random suffices to guarantee that each node manages a hash bucket of roughly the same size (within a logarithmic factor), which provides load balance as required by the first law.

Retrieval. When a lookup query is posed at some node X, it needs to be sent along to the node with the relevant data – the node Y whose ID is the closest successor of the query key's hash. Chord constructs an *overlay network* in which each of the n nodes maintains logical links to $O(\log n)$ neighbors, enabling X to send the query across to Y through a sequence of $O(\log n)$ links. The low network degree ensures that overlay maintenance is cheap as new nodes join or existing nodes leave. The low network diameter ensures that queries can be routed, and data retrieved, efficiently.

It is also necessary to optimize the overlay network *for the underlying physical network*, in accordance with the third law, to ensure that the routing path between any pair of nodes has low latency. While many different solutions exist for this problem, one approach that we have pursued [6] is to exploit the hierarchical structure of the physical network and ensure, among other things, that overlay routing between two nodes within some subtree of the hierarchy never involves any intermediate nodes that are outside that subtree. In addition, we are able to couple such overlay networks with a hierarchical caching scheme to further reduce query latency [6].

3 Enabling Range Queries with Online Load Balancing

Distributed hash tables are effective for enabling efficient queries for individual keys. However, they are problematic if we want to extend the querying power of our system to support range queries. For example, if we wanted to find all tuples (*(key,value)* pairs) with a key that begins with the letter "a", hashing offers no locality

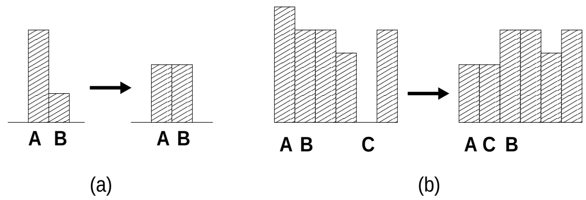


Figure 1: (a) NBRADJUST involving A and B and (b) REORDER involving A and C. The height of a bar represents the load of the corresponding node.

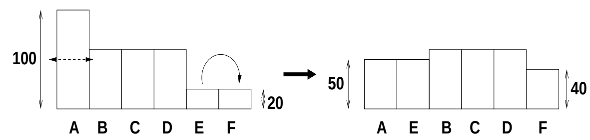


Figure 2: The cost of load balancing using REORDER is 70 while using successive NBRADJUST operations costs 250.

of reference and forces us to probe every single node for answers to the query, which is expensive. Our solution to the above problem is to do away with hashing in the storage solution, and instead distribute data across nodes by range partitioning. (The retrieval solution involving overlay networks and routing remains almost the same as that for DHTs as we discuss later.)

More precisely, data is divided into n range partitions on the basis of a key attribute, with partition boundaries at $R_0 \leq R_1 \leq \ldots \leq R_n$, with node N_i managing the range $[R_{i-1}, R_i)$ for all $0 < i \leq n$. We desire two properties from this range partitioning: (a) the partitions must be well-balanced with each node storing an equal amount of data; (b) it should be possible to dynamically change n, the number of partitions.

Let us first consider property (a) which requires load balance. If the data distribution were fixed ahead of time, we could carefully construct our partitions to satisfy this property. However, as the system evolves over time, data distributions change, and partitions need to be constantly reorganized to maintain load balance, introducing not only a data migration cost but also a management headache: when and how should data be repartitioned? We tackle both these problems with a fully automated *online load balancing* scheme that guarantees load balance at all times by continuously and automatically migrating data in the background using algorithms that have a provably small overhead. In fact, the scheme generalizes naturally to also support property (b) which requires us to also enable nodes to join and leave seamlessly without affecting load-balance properties.

Load Balancing Operations. What operations can be used to perform load balancing? An intuitive operation is as follows: when a node becomes responsible for too much data, it can move a portion of its data to its neighbor and thus attempt to balance out the load. We call such an operation NBRADJUST (illustrated in Figure 1(a) and defined below).

NBRADJUST *A pair of neighboring nodes N_i and N_{i+1} may alter the boundary R_i between their ranges by transferring data from one node to the other.*

We could devise load-balancing algorithms based on just this operation, but such algorithms are provably expensive, requiring massive amounts of data migration [7]. The key to efficient load balancing lies in a second operation, REORDER, illustrated in Figure 1(b) and defined below.

REORDER *A node N_i with an empty range $[R_i, R_i)$ changes its position and splits the range $[R_j, R_{j+1})$ managed by a node N_j: N_j now manages range $[R_j, X)$ while N_i takes over $[X, R_{j+1})$ for some value of X, $R_j \leq X \leq R_{j+1}$. The nodes are re-labeled appropriately.*

Example 1: Consider the scenario shown in Figure 2, where node A has 100 tuples, the next three nodes (B, C, D) have 60 tuples each, while the last two (E, F) have 20 tuples each. The least expensive scheme to improve load balance while preserving key ordering is to transfer all 20 tuples from E to F, and then use REORDER to split the load of A between A and E. The cost of such a scheme is 70 tuple movements; in contrast, a NBRADJUST-based balancing would require 250 tuple movements.

Load-Balancing Algorithm. Our load-balancing algorithm makes judicious use of the above two operations in order to provide a guarantee on load balance while also ensuring that the total amount of data moved across nodes in the process is very small. The key idea is to create an infinite geometric sequence of *load thresholds*, e.g., all powers of two, and let each node attempt load balancing every time its own load crosses a threshold. We do not discuss our algorithm's details here, and instead summarize its performance with the following theorem.

Theorem 1: Using a geometric threshold sequence with ratio δ, for any $\delta \geq (\sqrt{5}+1)/2$, we may obtain the following properties: (1) All node loads are within a factor δ^3 of each other; and, (2) The (amortized) number of tuples moved per tuple insertion or deletion is constant.

The above theorem states that it is possible to maintain load balance within any constant factor larger than the cube of the golden ratio ($\simeq 4.24$). Moreover, the total cost of maintaining load balance is linear in the number of updates, meaning that each insertion or deletion induces a constant number of tuples to migrate on average. We may interpret this cost as a constant-factor slowdown of updates in the system, since each tuple migration may be viewed as an extra deletion followed by an insertion. In fact, even experiments with highly adversarial workloads designed to stress the system forced only 2 tuple migrations per insert/delete, which could be viewed as a slowdown of updates by a factor of 5 [7]. In reality, the slowdown will be much smaller, both because the tuple migrations are batched in much larger chunks and because the overhead is an order of magnitude smaller with realistic, non-adversarial workloads.

From Load Balancing to Range Queries. So far, we have described a method for maintaining range partitions over a set of nodes while ensuring load balance. In fact, this can be useful in the context of plain-old parallel databases [7], rather than just P2P systems. The next step involves combining this range-partitioning scheme with a P2P routing network for efficient retrieval. We use overlay networks known as *skip graphs* [10, 11], with properties almost identical to that of standard DHT overlay networks such as Chord, in order to obtain efficient routing properties. In consequence, we are left with a true generalization of the Distributed Hash Table: a range-partitioned P2P system that matches the asymptotic efficiency of DHTs for all the standard operations – insertion and deletion of tuples, arrival and departure of nodes, and simple lookup queries – while being able to answer range queries far more efficiently, in an asymptotically optimal fashion.

4 Multi-dimensional Range Queries

We next discuss how a P2P system can support *multi-dimensional* range queries, i.e., conjunctive queries containing range predicates on two or more attributes of a relation. One simple solution is to perform range-partitioning on one of the attributes, use the solution from the previous section to evaluate the predicate on that attribute, and evaluate the remaining predicates as a filter. However, this solution is likely to prove very expensive in the context of motivating applications such as P2P photo-sharing and massively multi-player games, where data is inherently spatial and tagged with multiple "coordinates" [8]. Applying a selection predicate on just one coordinate would still retrieve far too much irrelevant data.

What is needed, therefore, is a data storage and retrieval mechanism specialized for multi-dimensional queries. In the world of databases, there are many solutions available for multi-dimensional queries, including space-filling curves, quadtrees and kd-trees. We adapt these techniques to develop efficient storage methods in the P2P context, and combine them with new overlay networks and retrieval algorithms to architect our overall solution for supporting multi-dimensional queries. Here, we illustrate the storage methods underlying our two solutions: SCRAP and MURK. Both solutions are designed to ensure data *locality* – minimizing the number of nodes that need to process the query – since the dominant cost of answering queries in a P2P system is often simply the cost of transmitting and initiating the query at each node that needs to process it. For multi-dimensional range queries, data locality can be obtained by ensuring that tuples nearby in the data space are stored at the same node for the most part. Our two solutions attempt to achieve locality in two different ways.

SCRAP: Space-filling Curves with Range Partitioning. Our first approach to supporting multi-dimensional queries, *SCRAP*, uses a two-step solution to partition the data: (a) Data is first mapped down into a single dimension using a space-filling curve; (b) this single-dimensional data is then range-partitioned across a dynamic set of participant nodes.

For the first step, we may map multi-dimensional data down to a single dimension using a space-filling curve such as *z-ordering* [12] or the *Hilbert* curve (e.g., [13]). For example, say we have two-dimensional data that consists of two 4-bit attributes, X and Y. A two-dimensional data point $\langle x, y\rangle$ can be reduced to a single 8-bit value z, by inter-leaving the bits in x and y. Thus, the two-dimensional point $\langle 0100, 0101\rangle$ would be mapped to the single-dimensional value 00110001. (This mapping corresponds to the use of z-ordering as the space-filling curve.) Note that this mapping is *bijective*, i.e., every two-dimensional point maps to a unique single-dimensional value and vice versa. In the second step, once data has been reduced to one dimension using the space-filling curve, we simply use range partitioning to split the data across the available nodes, just as in our solution of Section 3.

Locality is achieved since the space-filling curve attempts to ensure that nearby data points in the multi-dimensional space are also adjacent in the single dimension. However, as the number of dimensions increases, locality becomes worse since space-filling curves are afflicted by the curse of dimensionality. Load balance across nodes is ensured by using online load balancing of range partitions, as discussed earlier.

MURK: Multi-dimensional Rectangulation with kd-trees. Our second approach, MURK, partitions the data *in situ* in the high-dimensional space, breaking up the data space into "rectangles" (hypercuboids in high dimensions), with each node managing one rectangle. One way to achieve this partitioning is to use kd-trees, in which each leaf corresponds to a rectangle being stored by a node.

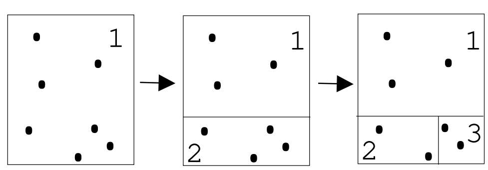


Figure 3: Evolution of partitions as nodes join the network. Each partition corresponds to a network node.

We illustrate this partitioning in Figure 3. Imagine there is initially one node in the system that manages the entire 2-d space, corresponding to a single-node kd-tree. When a second node arrives, the space is split along the first dimension into two parts of equal load, with one node managing each part; this corresponds to splitting the root node of the kd-tree to create two children. As more nodes arrive, each node splits the partition managed by an existing node, i.e., an existing leaf in the kd-tree. The dimensions are used cyclically in splitting, to ensure that locality is preserved in all dimensions.

When a node leaves, its space needs to be taken over by existing nodes. The simple case is when the node's sibling in the tree is also a leaf, e.g., node 3 in the figure; in this case, the sibling node 2 takes over 3's space. The more complex case arises when the node's sibling is an internal node, e.g., node 1 in the figure. When node 1 leaves, a lowest-level leaf node in its sibling subtree, say node 2, hands over data to its sibling node 3, and takes over the position of node 1.

The MURK solution offers good data locality, since it partitions the data space into exactly as many rectangles as there are nodes. As the number of dimensions increase beyond two, MURK offers much better data locality than SCRAP and outperforms SCRAP in our experimental comparisons [8]. However, load balancing across partitions is trickier with MURK and, in case the data distribution itself is dynamic, we cannot provide any theoretical guarantees on the cost of load balancing. In contrast, we always obtain strong load balance guarantees with SCRAP. An interesting open question is whether it is possible to obtain both load balance and locality with some data-partitioning scheme.

5 High-dimensional Similarity Search

We next consider how to support similarity-search queries in P2P systems to find high-dimensional objects most similar to a given object. For example, objects could be text documents, with the query trying to find documents in the system that are most similar to a given document. In an end-user P2P system, objects could even be peers themselves; users may wish to find what other users are similar to themselves on the basis of the content they share. In either case, we would like an efficient method for finding similar objects in a P2P system. Similarity search is a non-trivial problem even in a centralized context and, in fact, our solution turns out to be as applicable in a centralized environment as it is in the P2P context.

Our solution [16] is based on the influential locality-sensitive hashing (LSH) scheme of Indyk and Motwani [15]. The basic idea behind (centralized) LSH is the following: objects are hashed using special locality-sensitive hash functions (designed for the particular similarity metric of interest), such that "similar" objects are much more likely to collide (hash to the same bucket) than dissimilar objects[2]. At query time, the query is also hashed to a bucket; objects which hash to the same bucket are retrieved as "candidate" answers, their actual similarity to the query is computed, and the most similar among them are returned as answers to the query. The results returned by LSH are guaranteed to have a similarity within a small factor ϵ (for any fixed $\epsilon > 0$) of the optimal.

While the LSH scheme provides many nice properties, and could be implemented in a distributed fashion with DHTs, it suffers from two problems: (a) various parameters of the hash function need to be carefully tuned for the particular data domain of interest, and may have to be changed periodically if data characteristics change; and (b) the scheme requires the data distribution to be "uniform" in order to work well while being space-efficient.

We develop a new solution called LSH Forest, which uses locality-sensitive hashing while avoiding both the above problems, eliminating the need for tuning and offering improved accuracy with lower space utilization. The core idea is to construct a tree structure with the objects forming the leaf nodes in the tree; the similarity between two objects is approximately captured by how close they are in the tree. In the P2P context, our implementation of LSH Forest once again exploits range partitioning to efficiently partition the leaves of the tree across the participant nodes. Query execution requires various primitives for navigating the tree, which can be implemented on top of standard P2P overlay networks [16].

Interestingly, in the special case where the objects being searched are the peers themselves, we can embed the entire data structure in the *overlay network* itself! The very fact of which peers connect to which others contains information about which peers are similar to which others, eliminating the need for any explicit data structures for similarity search. We refer the interested reader to [16] for details.

6 Related Work

Early P2P systems such as Gnutella were built as unstructured networks and supported only keyword searches, with no guarantees of completeness on the search results. Distributed Hash Tables [2, 1, 3, 4] established the notion of structured overlays and showed how to support simple lookup queries in a clean fashion. Recent research has considered how to support more powerful queries by using alternatives to hashing.

Concurrent with our work on range partitioning, Karger and Ruhl also proposed the use of range partitions for supporting range queries. Their solution was based on a slightly different online load-balancing scheme with weaker guarantees on load balance, albeit under a different analytical model. Reference [7] discusses the similarities and differences between our solution and that of [9] in greater detail. Ratnasamy et al. [14] propose using tries for enabling range queries. Such an approach is arguably simpler and can be implemented on top of standard DHTs; on the other hand, it introduces data-dependent query costs and does not support the powerful forms of load balancing possible with range partitions [7].

The PIER project [17] aims to build a P2P system with support for arbitrary SQL queries. Research in the project so far has relied on hashed data storage, and focused on building a SQL query engine with efficient operator implementations. It is conceivable that the storage solutions described herein could be incorporated

[2]Many similarity measures have corresponding LSH functions that provide this property. Examples of such similarity measures include Jaccard coefficient, the Hamming metric and the l_1 and l_2 norms [15].

into PIER to expand the space of design possibilities available for data storage. The Astrolabe [18] system also provides a SQL-like interface on top of a P2P system, but is specialized to handle only aggregate queries on a hierarchical structure.

7 Conclusions

We have described the key challenges involved in supporting efficient queries over P2P systems. We presented different storage solutions that help extend the querying power of P2P systems to support range queries, multi-dimensional queries and similarity search. In conjunction with suitably designed overlay networks, these solutions enable the efficient execution of powerful queries over a P2P system. Many questions remain, however. For example, we would like to integrate these techniques into a P2P database system that can support arbitrary SQL queries, optimize these queries, and efficiently execute them in a distributed fashion across the network. Another big open issue is to define clear semantics for the results of queries in a P2P system; when computing query results takes up a non-trivial amount of time, it is not always clear what the results of a query actually mean, since peers may be joining and leaving while the query is in a partial state of execution. Given the number of open questions remaining unanswered, it appears safe to suggest that the holy grail of P2P data management is still at least a few years away from being realized.

References

[1] S. Ratnasamy, P. Francis, M. Handley and R. M. Karp. A Scalable Content Addressable Network. In *Proc. SIGCOMM*, 2001.

[2] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. SIGCOMM*,2001.

[3] A.I.T. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Proc. Middleware, 2001.

[4] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph and J.D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications 22(1)*, 2004.

[5] D.R. Karger, E. Lehman, F.T. Leighton, M.S. Levine, D. Lewin and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proc. STOC*, 1997.

[6] P. Ganesan, K. Gummadi and H. Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *Proc. 24th Intl. Conf. on Distributed Computing Systems (ICDCS)*, 2004.

[7] P. Ganesan, M. Bawa and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *Proc. VLDB*, 2004.

[8] P. Ganesan, B. Yang and H. Garcia-Molina. One Torus to Rule Them All: Multi-dimensional Queries in P2P Systems. In *Proc. WebDB*, 2004.

[9] D.R. Karger and M. Ruhl. Simple Efficient Load-Balancing Algorithms for Peer-to-Peer Systems. In *Proc. SPAA*, 2004.

[10] J. Aspnes and G. Shah. Skip Graphs. In *Proc. SODA*, 2003.

[11] N.J.A. Harvey, M. Jones, S. Saroiu, M. Theimer and A. Wolman. Skipnet: A Scalable Overlay Network with Practical Locality Properties. In *Proc. USITS*, 2003.

[12] J. Orenstein and T. Merrett. A Class of Data Structures for Associative Searching. In *Proc. PODS*, 1984.

[13] H. Jagadish. Linear Clustering of Objects with Multiple Attributes. In *Proc. SIGMOD*, 1990.

[14] S.Ratnasamy, J.M.Hellerstein and S.Shenker. Range Queries over DHTs. *Intel Technical Report IRB-TR-03-009*, 2003.

[15] P. Indyk and R. Motwani. Approximate Nearest Neighbor - Towards Removing the Curse of Dimensionality. In *Proc. STOC*, 1998.

[16] M. Bawa, T. Condie and P. Ganesan. LSH Forest: Self-Tuning Indexes for Similarity Search. To appear in *Proc. WWW*, 2005.

[17] R. Huebsch, B. Chun, J.M. Hellerstein, B.T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, A.R. Yumerefendi. The Architecture of PIER: an Internet-Scale Query Processor. In *Proc. CIDR*, 2005.

[18] R. Van Renesse, K.P. Birman and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management and data mining. *ACM Transactions on Computer Systems (TOCS), 21(2)*, 2003.

Structured Peer-to-Peer Networks: Faster, Closer, Smarter

P. Felber [1], K.W. Ross [2], E.W. Biersack [3], L. Garcés-Erice [4], G. Urvoy-Keller [3]
[1] University of Neuchâtel, Switzerland [2] Polytechnic University, NY, USA
[3] Institut EURECOM, Sophia Antipolis, France [4] IBM Research, Zürich, Switzerland

Abstract

Peer-to-peer (P2P) distributed hash tables (DHTs) are structured networks with decentralized lookup capabilities. Each node is responsible for a given set of keys (identifiers) and lookup of a key is achieved by routing a request through the network toward the current peer responsible for the desired key. DHT designs are usually compared in terms of degree (number of neighbors) and diameter (length of lookup paths). In this paper, we focus three other desirable properties of DHT-based systems: We first present a topology-aware DHT that routes lookup requests to their destination along a path that mimics the router-level shortest-path, thereby providing a small "stretch." We then show how we can take advantage of the topological properties of the DHT to cache information in the proximity of the requesters and reduce the lookup distance. Finally, we briefly discuss techniques that allow users to look up resources stored in a DHT, even if they only have partial information for identifying these resources.

1 Introduction

Several important proposals have recently been put forth for providing distributed peer-to-peer (P2P) lookup services based on distributed hash tables (DHTs). A DHT maps keys (data identifiers) to the nodes of an overlay network and provides facilities for locating the current peer node responsible for a given key. DHT designs differ mostly by the way they maintain the network and perform lookups: there is a fundamental trade-off between the degree of the network (the number of neighbors per node, i.e., the size of the routing tables) and its diameter (the average number of hops required per lookup) [12]. There are, however, other important aspects that should be taken into consideration when designing a DHT.

In this paper, we present our research on three facets of DHT-based systems. We first explore in Section 2 the design of a P2P lookup service for which topological considerations take precedence so as to provide *faster* lookup. We propose a P2P network design with a new lookup service, TOPLUS (Topology-Centric LookUp Service). In TOPLUS, peers that are topologically close are organized into groups. In turn, groups that are topologically close are organized into supergroups, close supergroups into hypergroups, etc. The groups within each level of the hierarchy can be heterogeneous in size and in fan-out. Groups can be derived directly from the network prefixes contained in BGP tables or from other sources. TOPLUS has many strengths, including a small "stretch" (the ratio of the latency between two hosts through the overlay to the latency through layer-3 IP routing), efficient forwarding mechanisms, and a fully symmetric design.

We also elaborate on the natural caching capabilities of TOPLUS and show how it can be leveraged to replicate content *closer* to the requesters. Distributed caches can be deployed in a straightforward manner at any level of the TOPLUS hierarchy in order to reduce transfer delays and limit the external traffic of a given network (campus, ISP, etc.).

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Finally, we present in Section 3 distributed indexing techniques that allow users to locate data using incomplete information, i.e., partial keys, and hence provide a *smarter* lookup. These techniques have been designed for indexing data stored in arbitrary DHT networks and discovering the resources that match a given user query. Our system creates multiple indexes, organized hierarchically, which permit users to locate data even using scarce information, although at the price of a higher lookup cost.

Due to space limitations, we shall refer the reader to other publications [5, 4, 3] for a more detailed description of our research on DHTs, including additional technical information, experimental evaluation, and exhaustive comparison with related work.

2 Topology Awareness

Early DHT designs did not take into account network topology. In particular, small steps in the logical address space usually resulted in large geographical jumps, which was observed to be a major bottleneck. As a result, some researchers have modified their DHTs to take topology into special consideration (e.g., [10, 1, 6]), but as an add-on rather as a foundation of the P2P system. For instance, in Pastry [1], a message takes small topological steps initially and big steps at the end of the route so as not to wander too far off the source, but the message is unlikely to proceed physically *toward* the destination.

In contrast, our TOPLUS lookup service was designed from the ground up with topological considerations in mind. It strives to route messages along a path that mimics the router-level shortest-path distance, i.e., by taking steps that systematically reduce the distance remaining to the destination. Informally, routing proceeds as follows: Given a message with key k under the responsibility of a peer p_d in the system, (1) source peer p_s sends the message (through IP-level routers) to a first-hop peer p_1 that is "topologically close" to p_d; (2) after arriving at p_1, the message remains topologically close to p_d as it is routed closer and closer to p_d through the subsequent intermediate peers. Clearly, if the lookup service satisfies these two properties, the stretch should be very close to 1.

We now formally describe TOPLUS in the context of IPv4. Let I be the set of all 32-bit IP addresses.[1] Let $\mathcal{G}$ be a collection of sets such that $G \subseteq I$ for each $G \in \mathcal{G}$. Thus, each set $G \in \mathcal{G}$ is a set of IP addresses. We refer to each such set G as a *group*. Any group $G \in \mathcal{G}$ that does not contain another group in $\mathcal{G}$ is said to be an *inner group*. We say that the collection $\mathcal{G}$ is a *proper nesting* if it satisfies all the following properties: (1) $I \in \mathcal{G}$; (2) for any pair of groups in $\mathcal{G}$, the two groups are either disjoint, or one group is a proper subset of the other; (3) For each $G \in \mathcal{G}$, if G is not an inner group, then G is the union of a finite number of sets in $\mathcal{G}$; and (4) each $G \in \mathcal{G}$ consists of a set of contiguous IP addresses that can be represented by an IP prefix of the form $w.x.y.z/n$ (for example, $123.13.78.0/23$).

As shown in [5], the collection of sets $\mathcal{G}$ can be created by collecting the IP prefix networks from BGP (border gateway protocol) tables and/or other sources [7, 11]. An autonomous system (AS) uses BGP to know which ASes it is connected to, and more important, which other ASes can be reached through each of them. A network is represented by an IP prefix and prefix aggregation allows to hide the complexity of routing inside the AS while offering other ASes enough information to route IP packets.

In TOPLUS, many of the sets $\mathcal{G}$ would correspond to ASes, other sets would be subnets in ASes, and yet other sets would be aggregations of ASes. This approach of defining $\mathcal{G}$ from BGP tables requires that a proper nesting is created. In order to reduce the size of the nodal routing tables, groups may be aggregated and artificial tiers may be introduced. Note that the groups differ in size, and in the number of subgroups (the fanout).

If $\mathcal{G}$ is a proper nesting, then the relation $G \subset G'$ defines a partial ordering over the sets in $\mathcal{G}$, generating a partial-order tree with multiple tiers. The set I is at tier-0, the highest tier. A group G belongs to tier-1 if

[1] For simplicity, we assume that all IP addresses are permitted. Of course, some blocks of IP addressed are private and other blocks have not been defined. TOPLUS can be refined accordingly.

there does not exist a G' (other than I) such that $G \subset G'$. We define the remaining tiers recursively in the same manner (see Figure 1).

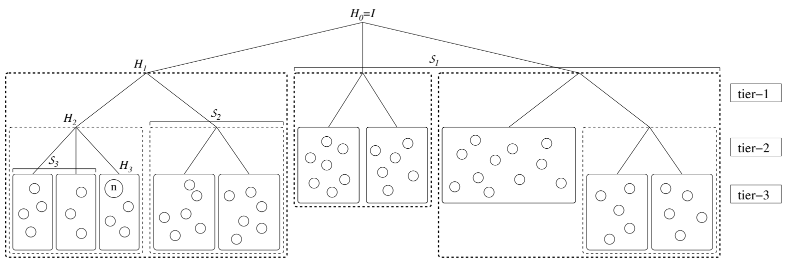


Figure 1: A sample TOPLUS hierarchy (plain boxes are inner groups)

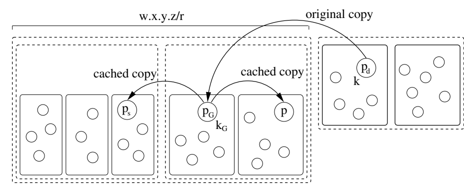


Figure 2: Data caching in TOPLUS.

Peer state. Let L denote the number of tiers in the TOPLUS tree, let U be the set of all current up peers and consider a peer $p \in U$. Peer p is contained in a collection of telescoping sets in $\mathcal{G}$; denote these sets by $H_i(p), H_{i-1}(p), \cdots, H_0(p) = I$, where $H_i(p) \subset H_{i-1}(p) \subset \cdots \subset H_0(p)$ and $i \leq L$ is the tier depth of p's inner group. Except for $H_0(p)$, each of these telescoping sets has one or more siblings in the partial-order tree (see Figure 1). Let $\mathcal{S}_i(p)$ be the set of siblings groups of $H_i(p)$ at tier-i. Finally, let $\mathcal{S}(p)$ be the union of the sibling sets $\mathcal{S}_1(p), \cdots, \mathcal{S}_i(p)$.

For each group $G \in \mathcal{S}(p)$, peer p should know the IP address of at least one peer in G and of all the other peers in p's inner group. We refer to the collection of these two sets of IP addresses as peer p's *routing table*, which constitutes peer p's state. The total number of IP addresses in the peer's routing table in tier i is $|H_i(p)| + |\mathcal{S}(p)|$.

The XOR metric for DHTs. Each key k' is required to be an element of I', where I' is the set of all b-bit binary strings ($b \geq 32$ is fixed). A key can be drawn uniformly randomly from I', or it can be biased (e.g., for balancing the keys among the groups). For a given key $k' \in I'$, denote k for the 32-bit suffix of k' (thus $k \in I$ and $k = k_{31}k_{30}\ldots k_1k_0$). Throughout the discussion below, we shall refer to k rather than to the original k'.

The XOR metric defines the distance between two IDs j and k as $d(j,k) = \sum_{\nu=0}^{31} |j_\nu - k_\nu| \cdot 2^\nu$. Let $c(j,k)$ be the number of bits in the common prefix of j and k. The metric $d(j,k)$ has the following properties: if $d(i,k) = d(j,k)$ for any k, then $i = j$; if $d(i,k) \leq d(j,k)$, then $c(i,k) \geq c(j,k)$; and $d(j,k) \leq 2^{32-c(j,k)} - 1$. The XOR metric is a refinement of longest-prefix matching. If j is the unique longest-prefix match with k, then j is the closest to k in terms of the metric. Further, if two peers share the longest matching prefix, the metric will break the tie. The Kademlia DHT [8] also uses the XOR metric. The peer p' that minimizes $d(k,p)$, $p \in U$ is "responsible" for key k.

The lookup algorithm. Suppose peer p_s wants to look up key k. Peer p_s determines the peer in its routing table that is closest to k in terms of the XOR metric, say p_j. Then p_s forwards the message to p_j. The process continues, until the message with key k reaches a peer p_d such that the closest peer to k in p_d's routing table is p_d itself. p_d is trivially the peer responsible for k.

If the set of groups form a proper nesting, then it is straightforward to show that the number of hops in a lookup is at most $L + 1$, where L is the depth of the partial-order tree. In the first hop the message will be sent to a peer p_1 that is in the same group, say G, as p_d. The message remains in G until it arrives at p_d. Each peer in TOPLUS mimics a router in the sense that it routes messages based on a generalization of longest-prefix matching of IP addresses using highly-optimized algorithms.

Overlay maintenance. When a new peer p joins the system, p asks an arbitrary existing peer to determine (using TOPLUS) the closest peer to p (using p's IP address as the key), denoted by p'. p initializes its routing table with p''s routing table. Peer p's routing table should then be modified to satisfy a "diversity" property: for each peer p_i in the routing table, p asks p_i for a random peer in p_i's group. This way, for every two peers in a group G, their respective sets of delegates for another group G' will be disjoint (with high probability). This guarantees that, in case one delegate fails, it is possible to use another peer's delegate. Finally, all peers in p's inner group must update their inner group tables.

Note that groups, which are virtual, do not fail (individual peers fail) but they can be partitioned or aggregated at runtime when needed.

Data caching. In DHT storage systems, caching can help reduce access times by creating local copies of popular data so as to avoid fetching the original data from the responsible peer. The caching mechanisms that have been proposed in major DHT systems consist in replicating the data along the lookup path of a query after a successful lookup. This approach suffers from two major drawbacks: First, it assumes that the lookup paths of two requests for a given key converge quickly, which is not always the case (especially when the number of peers grows). Second, the data is typically not cached close to the clients: even if a local copy does exist near the client, that copy is unlikely to be on the lookup path.

In addition to improving lookup performance, a major objective of data caching is to save on bandwidth costs by caching content within a given network (company, ISP, AS, etc.). Because of its hierarchical organization derived from IP prefixes, TOPLUS can straightforwardly fulfill this goal and provide a powerful caching service for network infrastructures.

Suppose that a peer p_s wants to obtain the file f associated with key $k \in I$, located at some peer p_d (see Figure 2). It would be preferable if p_s could obtain a cached copy of file f from a topologically close peer. To this end, suppose that some group $G \in \mathcal{G}$, with network prefix $w.x.y.z/r$, at any tier, wants to provide a caching service to the peers in G. Further suppose all pairs of peers in G can send files to each other relatively quickly (high-speed LAN) or at least quicker than to any other peer outside of G. Peer $p_s \in G$ creates a new key, k_G, which is equal to k but with the first r bits of k replaced with the first r bits of $w.x.y.z/r$. Peer p_s then looks up the peer p_G responsible for k_G. This peer is obviously inside group G and the lookup message will not leave the group.

If p_G has a copy of f (cache hit), then it will serve the file to p_s at a relatively high rate. Otherwise (cache miss), p_G will use key k to obtain f from the global lookup service. After downloading the file, p_G will send f to p_s and create a local cache copy for other users of G. Another peer $p \in G$ will obtain the file faster by directly downloading it from p_G (Figure 2). It is obviously possible to cache data at multiple levels of the hierarchy (e.g., company, ISP, AS) and hence implement a hierarchical Internet cache.

Performance. We have measured the efficiency of TOPLUS with $1,000$ nodes using IP network prefixes obtained from several BGP tables and routing registries.[2] When constructing a TOPLUS hierarchy from a direct mapping of IP prefixes to groups, we obtain an average stretch of 1.17, that is, a query in TOPLUS takes on average only 17% more time to reach its destination than using direct IP routing. Unfortunately, this hierarchy has many tier-1 groups, which leads to large routing tables. We can reduce their size by "aggregating" small groups that have a long prefix into larger groups not present in our IP prefix sources. Even after aggressive aggregation of large sets of tier-1 groups into coarse 8-bit prefixes, we have measured a remarkably small stretch of 1.28. Complete evaluation results are available in [5].

Limitations. In designing TOPLUS to optimize topological locality, we had to sacrifice some other desirable properties of P2P lookup services. In particular, TOPLUS suffers from limitations related to the non-uniform population of the ID space (peers in sparse regions may be overloaded if keys are uniformly distributed over the ID space), the lack of support for virtual peers, and the vulnerability to correlated peer failures (a whole inner group being unavailable, e.g., due to a network partition). Some solutions to these problems are discussed in [5].

[2]BGP tables were provided by the University of Oregon and by the University of Michigan and Merit Network. Routing registries were provided by Castify Networks and RIPE.

3 Content Indexing

A major limitation of DHT lookup services is that they only support exact-match lookups: one needs to know the exact key (identifier) of a data item to locate the peer responsible for that item. In practice, however, P2P users often have only partial information for identifying these items and tend to submit broad queries (e.g., all the articles written by "John Smith"). In this section, we propose to augment DHT-based P2P systems with mechanisms for locating data using incomplete information. We do not aim at answering complex database-like queries, but rather at providing practical techniques for searching data within a DHT. Our mechanisms rely on indexes, stored and distributed across the peers of the network, that maintain useful information about user queries. Given a broad query, a user can obtain additional information about the data items that match her original query; the DHT lookup service can be recursively queried until the user finds the desired data items. Indexes can be organized hierarchically to reduce space and bandwidth requirements, and to facilitate interactive searches. They integrate an adaptive distributed cache to speed up accesses to popular content. Our indexing techniques can be layered on top of an arbitrary DHT lookup service, including TOPLUS, and thus benefit from any advanced features implemented in the DHT (e.g., replication, load-balancing).

P2P storage system. We assume an underlying DHT-based P2P data storage system in which each data item is mapped to one or several peers. Example of such systems are Chord/DHash/CFS [2] and Pastry/PAST [9]. We shall use the example of a bibliographic database system that stores scientific articles. Files are identified by *descriptors*, which are textual, human-readable descriptions of the file's content (in their simplest form, they can be file names). Let $h(descriptor)$ be a hash function that maps identifiers to a large set of numeric keys. The peer responsible for storing a file f is determined by transforming the file's descriptor d into a numeric key $k = h(d)$. This numeric key is used by the DHT lookup service to determine the peer responsible for f. In order to find f, a peer p has to know the numeric key or the complete descriptor.

```
<article>
  <author>
    <first>John</first>
    <last>Smith</last>
  </author>
  <title>TCP</title>
  <conf>SIGCOMM</conf>
  <year>1989</year>
  <size>315635</size>
</article>
```

d_1

```
<article>
  <author>
    <first>John</first>
    <last>Smith</last>
  </author>
  <title>IPv6</title>
  <conf>INFOCOM</conf>
  <year>1996</year>
  <size>312352</size>
</article>
```

d_2

```
<article>
  <author>
    <first>Alan</first>
    <last>Doe</last>
  </author>
  <title>Wavelets</title>
  <conf>INFOCOM</conf>
  <year>1996</year>
  <size>259827</size>
</article>
```

d_3

Figure 3: Sample File Descriptors.

```
q1 = /article[author[first/John][last/Smith]] ···
     [title/TCP][conf/SIGCOMM][year/1989][size/315635]
q2 = /article[author[first/John][last/Smith]] ···
     [conf/INFOCOM]
q3 = /article/author[first/John][last/Smith]
q4 = /article/title/TCP
q5 = /article/conf/INFOCOM
q6 = /article/author/last/Smith
```

Figure 4: Sample File Queries.

Data descriptors and queries. In the rest of this section, we shall assume that descriptors are semi-structured XML data, as used by many publicly-accessible databases (e.g., DBLP). Examples of descriptors for bibliographic data are given in Figure 3. These descriptors have fields useful for searching files (e.g., author, title), as well as fields useful for administering the database (e.g., size).

To search for data stored in the peer-to-peer network, we need to specify broad queries that can match multiple file descriptors. For this purpose, we shall use a subset of the XPath XML addressing language, which offers a good compromise between expressiveness and simplicity. XPath treats XML documents as a tree of nodes and offers an expressive way to specify and select parts of this tree using various types of predicates and wildcards. An XML document (i.e., a file descriptor) *matches* an XPath expression when the evaluation of the expression on the document yields a non-null object.

For a given descriptor d, we can easily construct an XPath expression (or query) q that tests the presence of all the elements and values in d.[3] We call this expression the *most specific query* for d or, by extension, the *most specific descriptor*. Conversely, given q, one can easily construct d, compute $k = h(d)$, and find the file. For instance, query q_1 in Figure 4 is the most specific query for descriptor d_1 in Figure 3.

[3]In fact, we can create several equivalent XPath expressions for the same query. We assume that equivalent expressions are transformed into a unique normalized format.

Given two queries q and q', we say that q' *covers* q (or q is covered by q'), denoted by $q' \sqsupseteq q$, if any descriptor d that matches q also matches q'. Abusing the notation, we often use d instead of q when q is the most specific query for d and we consider them as equivalent ($q \equiv d$); in particular, we say that q' covers d when $q' \sqsupseteq q$ and q is the most specific query for d. Note that the covering relation introduces a partial ordering on the queries.

Indexing algorithm. When the most specific query for the descriptor d of a file f is known, finding the location of f is straightforward using the key-to-peer (and hence key-to-data) underlying DHT lookup service. The goal of our architecture is to also offer access to f using less specific queries that cover d.

The principle underlying our technique is to generate multiple keys for a given descriptor, and to store these keys in indexes maintained by the DHT in the P2P system. Indexes do not contain key-to-data mappings; instead, they provide a key-to-key service, or more precisely a query-to-query service. For a given query q, the index service returns a (possibly empty) list of more specific queries, covered by q. If q is the most specific query of a file, then the P2P storage system returns the file (or indicates the peer responsible for that file). By iteratively querying the index service, a user can traverse upward the partial order graph of the queries and discover all the indexed files that match her broad query.

In order to manage indexes, the underlying P2P storage system must be slightly extended. Each peer should maintain an index, which essentially consists of query-to-query mappings. The "$insert(q, q_i)$" function, with $q \sqsupseteq q_i$, adds a mapping $(q; q_i)$ to the index of the peer responsible for key q. The "$lookup(q)$" function, with q not being the most specific query of a file, returns a list of all the queries q_i such that there is a mapping $(q; q_i)$ in the index of the peer responsible for key q.

Roughly speaking, we store files and construct indexes as follows: Given a file f and its descriptor d, with a corresponding most specific query q, we first store f at the peer responsible for the key $k = h(q)$. We generate a set of queries $q = \{q_1, q_2, \ldots, q_l\}$ likely to be asked by users (to be discussed shortly), and such that each $q_i \sqsupseteq q$. We then compute the numeric key $k_i = h(q_i)$ for each of the queries, and we store a mapping $(q_i; q)$ in the index of the peer responsible for each k_i in the P2P network. We iterate the process shown for q to every q_i, and we continue recursively until all the desired index entries have been created.

Indexing example. To best illustrate the principles of our indexing techniques, consider a P2P bibliographic database that stores the three files associated to the descriptors of Figure 3. We want to be able to look up publications using various combinations of the author's name, the title, the conference, and the year of publication. A possible hierarchical indexing scheme is shown in Figure 5. Each box corresponds to a distributed index, and indexing keys are indicated inside the boxes. The index at the origin of an arrow stores mappings between its indexing key and the indexing key of the target. For instance, the *Last name* index stores the full names of all authors that have a given last name; the *Author* index maintains information about all articles published by a given author; the *Article* index stores the most specific descriptors (MSDs) of all publications with a matching title and author name. After applying this indexing scheme to the three files of the bibliographic database, we obtain the distributed indexes shown in Figure 6 (left). The top-level *Publication* index corresponds to the raw entries stored in the underlying P2P storage system: complete keys provide direct access to the associated files. The other indexes hold query-to-query mappings that enable the user to iteratively search the database and locate the desired files. Each entry of an index is potentially stored on a different peer in the P2P network, as illustrated for the *Proceedings* index. One can observe that some index entries associate a query to multiple queries (e.g., in the *Author* index).

Figure 6 (right) details the individual query mappings stored in the indexes of Figure 6 (left). Each arrow corresponds to a query-to-query mapping, e.g., $(q_6; q_3)$. The files corresponding to descriptors d_1, d_2, and d_3 can be located by following any valid path in the partial order tree. For instance, given q_6, a user will first obtain q_3; the user will query the system again using q_3 and obtain two new queries that link to d_1 and d_2; the user can finally retrieve the two files matching her query using d_1 and d_2.

Lookups. We can now describe the lookup process more formally. When looking up a file f using a query q_0, a user first contacts the peer p responsible for $h(q_0)$. That peer may return f if q_0 is the most specific query for f, or a list of queries $\{q_1, q_2, \ldots, q_n\}$ such that the mappings $(q_0; q_i)$, with $q_0 \sqsupseteq q_i$, are stored at p. The user can then choose one or several of the q_i and repeat this process recursively until the desired files have been found. The user effectively follows an "index path" that leads from q_0 to f ("guided tour"). The lookup process can be interactive, i.e., the user directs the search and restricts her query at each step, or automated, i.e., the system recursively explores the indexes and returns all the file descriptors that match the original query.

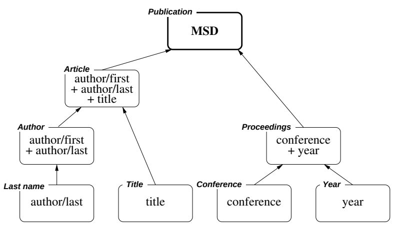


Figure 5: Sample indexing scheme for a bibliographic database.

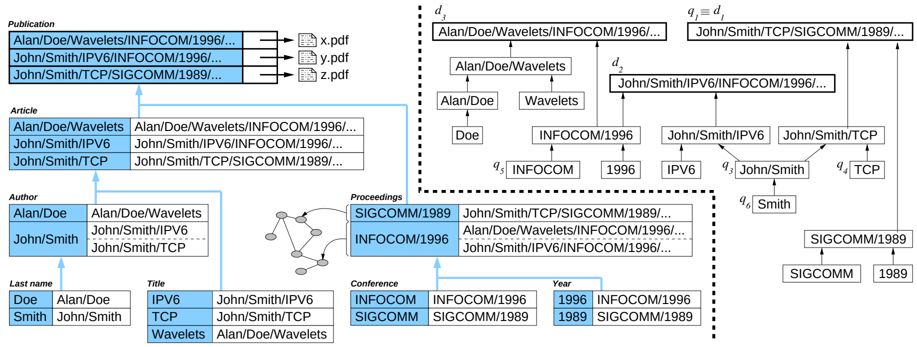


Figure 6: Sample distributed indexes (left) and query mappings (right) for the three documents of Figure 3 and the indexing scheme of Figure 5.

Lookups may require several iterations when the most specific query for a given file is not known. Deeper index hierarchies usually necessitate more iterations to locate a file, but are also generally more space-efficient, as each index factorizes in a compact manner the queries of other indexes. In particular, the size of the lists (result sets) returned by the index service may be prohibitively long when using a flat indexing scheme (consider, for example, the list of all articles written by the persons whose last name is "Smith"). There is therefore a trade-off between space requirements, size of result sets, and lookup time.

When a user wants to look up a file f using a query q_0, it may happen that q_0 is not present in any index, although f does exist in the peer-to-peer system and q_0 is a valid query for f. It is still possible to locate f, by (automatically) looking for a query q_i such that $q_i \sqsupseteq q_0$ and q_i is on some index path that leads to f. For instance, given the distributed indexes of Figure 6, it appears that query q_2 in Figure 4 is not present in any index (q_2 =`/article[author[first/John][last/Smith]][conf/INFOCOM]`). We can however find q_3, such that $q_3 \sqsupseteq q_2$ and there exists an index path from q_3 to d_1. Therefore, the file associated to d_1 can be located using this generalization/specialization approach, although at the price of a higher lookup cost. We believe that it is natural for lookups performed with less information to require more effort.

Building and maintaining indexes. When a file is inserted in the system for the first time, it has to be indexed. The choice of the queries under which a file is indexed is arbitrary, as long as the covering relation holds. As files are discovered using the index entries, a file is more likely to be located rapidly if it is indexed "enough" times, under "likely" names. The quantity and likelihood of index queries are hard to quantify and are often application-dependent. For instance, in a bibliographic database, indexing a file by its size is useless for users, as they are unlikely to know the size beforehand. However, indexing the files under the author, title, and/or conference are appropriate choices.

Index entries can also be created dynamically to adapt to the query patterns of the users. For instance, a user who tries to locate a file f using a non-indexed query q_0, and eventually finds it using the query generalization/specialization approach discussed above, can add an index entry to facilitate subsequent lookups from other users. More interestingly, one can easily build an adaptive cache in the P2P system to speed up accesses to popular files. Assume that each peer allocates a limited number of index entries for caching purposes. After a successful lookup, a peer can create "shortcuts" entries (i.e., direct mappings between generic queries and the descriptor of the target file) in the caches of the indexes traversed during the lookup process. Another user looking for the same file via the same path will be able to “jump” directly to the file by following the shortcuts stored in the caches. With a least-recently used (LRU) cache replacement policy (i.e., the least used entries in the cache are replaced by the new ones once there is no cache space left), we can guarantee that the most popular files are well represented in the caches and are accessible in few hops. The caching mechanism therefore adapts automatically to the query patterns and file popularity.

An in-depth discussion of the properties of our indexing techniques, such as space-efficiency, scalability, loose coupling, flexibility and adaptability, can be found in [4], together with a detailed experimental evaluation.

4 Conclusion

DHTs suffer from a number of limitations over unstructured P2P networks due to the rigidness of their logical organization. We have discussed some of these limitations and presented several techniques to help alleviate them. TOPLUS integrates network topology in its design, by organizing peers in a group hierarchy defined by IP prefixes that map directly to the underlying topology. The TOPLUS architecture also allows for the straightforward deployment of distributed caches in order to reduce transfer delays and limit the external traffic of a given network. Finally, we have studied the problem of looking up information in a DHT using incomplete information—a major improvement over the exact-match lookup of DHTs. Our distributed indexing techniques can be used to locate data in a DHT matching a broad query. We hope that these various improvements can contribute to the large-scale deployment of DHTs in the Internet and bring them on par with unstructured P2P networks in terms of popularity.

References

[1] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Topology-aware routing in structure peer-to-peer overlay network. In *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 103–107, June 2002.

[2] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18^{th} ACM Symposium on Operating System Principles (SOSP)*, pages 202–215, October 2001.

[3] L. Garcés-Erice, E.W. Biersack, P. Felber, K.W. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. *Parallel Processing Letters*, 13(4):643–657, 2003.

[4] L. Garcés-Erice, P. Felber, E.W. Biersack, K.W. Ross, and G. Urvoy-Keller. Data indexing in DHT peer-to-peer networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, pages 200–208, March 2004.

[5] L. Garcés-Erice, K.W. Ross, E.W. Biersack, P. Felber, and G. Urvoy-Keller. TOpology-centric Look-Up Service. In *Proceedings of COST264/ACM 5^{th} International Workshop on Networked Group Communications (NGC)*, volume 2816 of *LNCS*, pages 58–69. Springer, September 2003. Best paper award.

[6] A.D. Joseph, B.Y. Zhao, Y. Duan, L. Huang, and J.D. Kubiatowicz. Brocade: Landmark routing on overlay networks. In *Proceedings of the 1^{st} International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *LNCS*, pages 34–44. Springer, March 2002.

[7] B. Krisnamurthy and J. Wang. On network-aware clustering of Web sites. In *Proceedings of SIGCOMM*, pages 97–110, August 2000.

[8] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer informatic system based on the XOR metric. In *Proceedings of the 1^{st} International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *LNCS*, pages 53–65. Springer, March 2002.

[9] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18^{th} ACM Symposium on Operating System Principles (SOSP)*, pages 188–201, October 2001.

[10] S. Shenker, S. Ratnasamy, M. Handley, and R. Karp. Topologically-aware overlay construction and server selection. In *Proceedings of INFOCOM*, pages 1190–1199, June 2002.

[11] J. Wang. *Network Aware Client Clustering and Applications*. PhD thesis, Cornell University, May 2001.

[12] J. Xu, A. Kumar, and X. Yu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. *IEEE Journal on Selected Areas in Communications*, 22(1):151–163, January 2004.

Network Awareness in Internet-Scale Stream Processing*

Yanif Ahmad, Uğur Çetintemel
John Jannotti, Alexander Zgolinski, Stan Zdonik
{yna,ugur,jj,amz,sbz}@cs.brown.edu
Dept. of Computer Science, Brown University, Providence, RI 02912

Abstract

Efficient query processing across a wide-area network requires network awareness*, i.e., tracking and leveraging knowledge of network characteristics when making optimization decisions. This paper summarizes our work on network-aware query processing techniques for widely-distributed, large-scale stream-processing applications. We first discuss the* operator placement *problem (i.e., deciding where to execute the operators of a query plan) and present results, based on a prototype deployment on the PlanetLab network testbed, that quantify the benefits of network awareness. We then present a summary of our present focus on the* operator distribution *problem, which involves parallelizing the evaluation of a single operator in a networked setting.*

1 Introduction

As applications involving large numbers of geographically distributed data sources proliferate, network efficiency and scalability are emerging as key design goals for future data processing systems (e.g., [6, 4, 7, 9, 11]). Central to achieving these goals is *network awareness*, i.e., the capability to track and exploit information about specific network characteristics such as inter-site latencies, network topologies and link bandwidths. Traditional query optimization models are commonly designed for CPU- and disk-bound optimization and thus need to be rethought for network-bound processing.

To this end, the SAND project at Brown is currently developing a network-aware distributed query optimization framework in the context of Internet-scale stream processing applications and the Borealis project [1], which strives to develop a full-fledged distributed stream processing engine. In this paper, we provide an overview of SAND, focusing on two key problems that arise in networked stream processing:

- *Operator placement* — deciding the network locations where the operators of a query plan should be placed and executed.
- *Operator distribution* — deciding how a single operator, such as a join, should be distributed across nodes in a networked setting.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work has been supported in part by the National Science Foundation under the ITR grant IIS-0325838.

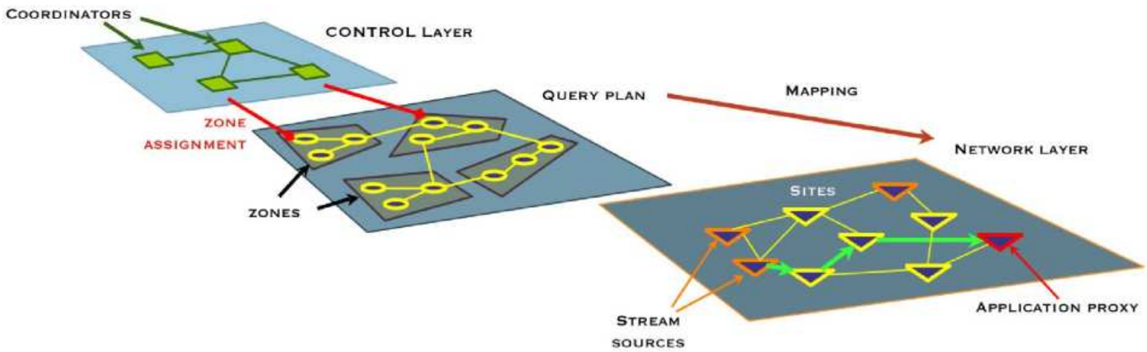


Figure 1: A query plan and the corresponding control and network layers.

Even though both problems have been extensively studied in the context of traditional distributed and parallel databases (e.g., [10, 5]), existing solutions typically assume a small-scale, static and homogeneous networking environment and rely on centralized algorithms. These solutions, thus, fall short of addressing the unique requirements of Internet-scale query processing where network awareness and distributed algorithms are crucial for achieving efficiency and scalability. Our work strives to eliminate these assumptions and limitations of prior work.

We first summarize SAND's basic system model. We then present various operator placement algorithms that differ primarily in what nodes they consider as candidates for operator placement and how they take network knowledge into account. We follow by presenting experimental results that characterize the relative performance of the approaches in a wide-area setting, through deployment on the PlanetLab network testbed on top of the Borealis stream processing engine. Finally, we briefly discuss the operator distribution problem and highlight the key ideas that we are currently exploring towards an efficient and scalable solution.

2 Networked Operator Placement

2.1 Basic System Model

We target a widely-distributed query processing environment with geographically dispersed data sources that produce high-volume data streams that are to be processed for purposes of filtering, fusion, aggregation, and correlation. We assume an underlying distributed stream processing system (such as Borealis) that provides core stream processing functionality and mechanisms for dynamic operator migration across nodes.

We abstractly represent our query plan as a tree of operators, called *a processing tree*, whose edges represent outputs from one operator that are used as inputs by another operator. The query's data sources are represented as leaves of this tree. We use a cost function combining (1) the bandwidth used while transporting data between operators and (2) the network latency between the nodes selected to host the operators. The bandwidth usage here is obtained by considering input rates to the operator and the operator's selectivity. Two connected operators incur a cost of zero if they are located at the same node, capturing the property that no network cost is incurred if sequential operators are evaluated locally.

We use three heuristics-based algorithms to construct a placement minimizing this cost function: (1) *Edge*, a network-agnostic algorithm placing operators at the data sources alone, (2) *Edge+*, an extended version of Edge that factors in network latencies and topology during placement, and (3) *In-Network*, an algorithm capable of placing operators at arbitrary locations within the network.

2.2 Control Model

For improved scalability and parallelism, we use a distributed control model. For each processing tree, we create a corresponding *control tree* of coordinator nodes by logically partitioning the processing tree into a number of subtrees (not necessarily extending to the leaves), called *zones*. Each zone is then assigned a coordinator node that is responsible for the placement (and periodic dynamic re-placement) of its operators. The *application proxy*, the node that delivers the output of processing to the clients, is always assigned as the root coordinator and decides how many additional zones to create.

Figure 1 provides a high-level view of this model, illustrating a processing tree as well as the corresponding control and processing networks overlayed on top of the IP network. The processing tree is partitioned into three zones, each assigned to a coordinator node. Each coordinator maps all operators in its zone using one of the aforementioned heuristics. An advantage of this distributed optimization model is that zones can be optimized locally, concurrently and asynchronously.

Note that a coordinator is dependent upon other coordinators whose zones contain descendant operators. Our operator tree is, thus, mapped in a bottom-up fashion, starting at coordinators responsible for leaves, and triggering placement of parent operators as each individual operator is mapped. Coordinators communicate with each other when one coordinator has mapped its zone, sending optimization metadata to the coordinator responsible for mapping parents. The contents of this metadata depend on the heuristic in use and will be described along with the heuristics. In our distributed algorithm, a coordinator may also backtrack on the placements prior to its round of control. This approach helps to overcome locally optimal solutions that are not necessarily globally optimal, given operators are initially mapped without knowledge of their siblings residing in other zones.

2.3 Placement Algorithms

The algorithms in this section determine an operator's placement, given the placement of the operator's immediate children. The heuristics also determine if backtracking is necessary, and provide updated positions for the operator's children.

- The *Edge* heuristic considers placing an operator at the locations its children are placed, and any *common* locations. We define common locations as follows. Each data source (i.e., the leaves of our tree) is assumed to reside at a fixed location in the network. An operator may potentially be placed at the locations of any data sources providing its inputs. A common location is a node present in the set of potential locations of each child. As previously mentioned, Edge is a network-agnostic heuristic that uses a cost function of bandwidth alone, and does not consider network latencies between nodes. Edge simply places the operator at the location minimizing the total cost of each child's subtree, and any edges between the operator and its children. If the selected location is a common location, rather than some child's existing location, each child's location is updated to the common location. When using this heuristic, coordinators must send the set of potential locations for the root of their workload, as well as the cost of placing this root at each potential location. The recipient coordinator may then map a parent, attempting to determine an optimal solution by reconsidering each child's placement in light of their respective placements.

- The *Edge+* heuristic incorporates network latencies between nodes when making its placement decision. Edge+ considers placing operators at their child and common locations, in an identical manner to Edge. However, rather than considering cost as the bandwidth utilized by parent-child edges alone, Edge+ calculates a cost based on the product of edge bandwidth and latencies. The latency here is that between a potential placement of the operator, and the placement of a child. Edge+ searches over all potential placements of the both the operator and its children, selecting a configuration with the minimal bandwidth-latency product. In addition to an expanded cost function, Edge+ also incorporates a pruning technique not found in Edge. Specifically, Edge+ prunes potential placements of an operator based on a latency criterion. This criterion eliminates locations if the total latency between the location under consideration and the locations of all children exceeds the total latency between child locations alone. In a similar manner to Edge, coordinators exchange optimization metadata comprising of placement costs at potential locations.

- The *In-Network* heuristic extends Edge+ in the set of potential locations considered for placement, and in the pruning heuristic used to eliminate locations. In-Network searches over a set of candidate locations obtained as a shortest-path tree between the application proxy, and all data sources. We remark that candidate node selection remains an open issue and are investigating several other heuristics (such as candidates chosen by random walks and by directed flooding in the neighborhood of our data sources). In addition to pruning based on a distance criterion as in Edge+, In-Network computes a ranking based on the total latency between potential operator locations and child locations, and selects the top k such locations to actually compare costs (k is a configuration parameter). Consequently, coordinators only forward optimization metadata containing placement costs at this subset of potential locations.

We refer the reader to [2] for a detailed description of the placement algorithms, as well as extensions that can meet user-specified per-query latency bounds.

3 System Architecture and Deployment

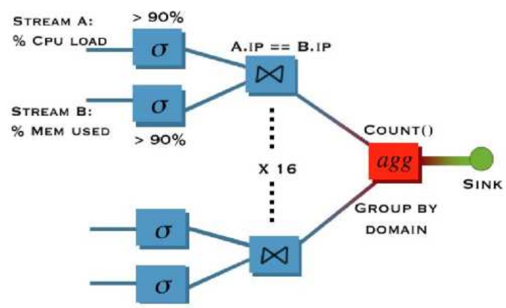
We have implemented a SAND prototype using the OCaml language, and deployed our implementation across the PlanetLab testbed. Our key design principles included retaining a flexible optimization and network protocol framework, allowing modular extensibility to experiment with various operator placement heuristics. To support this extensibility, our prototype includes a loosely coupled protocol stack and optimization engine. Lower layers of our protocol stack include a DHT-driven lookup protocol (using the OpenHash implementation) and latency and bandwidth probing protocols.

The SAND prototype interfaces with the Borealis stream processing engine, and uses this engine to perform widely distributed query processing. Borealis provides an XML-RPC interface enabling SAND to both collect selectivity statistics from a running query, and subsequently perform operator placement by moving Borealis boxes. Presently the two systems are loosely coupled, with SAND capable of mapping a Borealis query network onto a physical network in an offline manner. Our recent efforts have been to deploy the Borealis prototype onto PlanetLab, implementing a wide-area network monitoring tool as an example application. This application generates input data for Borealis through the use of a data wrapper interacting with the Ganglia, Slicestat and Comon "sensors" collecting statistics from each PlanetLab site, and periodically pushing tuples into the Borealis engine running at that site. We expect to integrate the SAND and Borealis prototypes to collect online selectivity estimates in the immediate future.

4 Experimental Results

We briefly present two sets of experiments, where SAND first maps abstract query plans containing a variety of edge rates between operators, and secondly maps a Borealis query network prior to measuring actual edge rates on the running query. Both sets of experiments were run on PlanetLab.

Our experimental evaluation methodology compares the bandwidth consumption ratio and stretch factor of the mapping obtained to a "warehousing" scenario, where data sources push tuples to a single site in the network and the query is evaluated entirely at this site. The bandwidth consumption ratio is defined as the total sum of our bandwidth-latency product of all transfers between operators across our network, as a ratio of the equivalent metric in the centralized scenario. The stretch factor is a ratio of the maximum end-to-end latency resulting from our mapping compared to the centralized scenario.



i)

ii)

Figure 2: Experiment setup and deployment across PlanetLab.

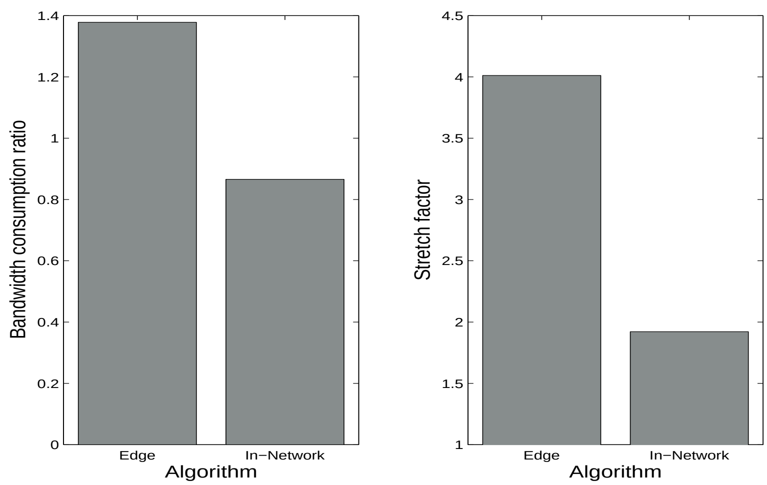


Figure 3: i) Bandwidth consumption ratio and ii) Stretch factor of abstract query plan mappings.

Figure 2 shows the Borealis query plan we deployed across 20 PlanetLab sites chosen on the East and West coasts of North America and Europe. Each data source provides statistics on CPU and memory utilization. The query identifies the sites with very high resource utilization and outputs an aggregate count of those, grouped by their domain identifiers. In the final placement, a filter and join box are placed at each source site, and the aggregate box is mapped to the MIT site. We collect the query's results at Brown.

Our abstract representations of query plans were binary trees of depth four, and fanout two. Our results in Figures 3i) and 3ii) confirm those witnessed in our emulation [2], namely the dominance of the In-Network heuristic over the network-agnostic Edge heuristic both in terms of the bandwidth consumption ratio, and the stretch factor. Indeed the incorporation of network awareness is key: results comparing Edge and Edge+ (omitted) exhibit similar gains in the Edge+ heuristic.

Figure 4 shows the bandwidth consumption ratio and stretch factor measured for the query network in Figure 2, for two different selectivity values. To achieve these selectivity values, we used a dummy CPU and memory utilization sensor generating values between 0-100%, and subsequently adjusted the threshold of our filter's predicate. The results here again demonstrate the benefits of in-network processing, where in this scenario, the join operators are performed at the source sites, and the aggregate operator at a centroid-like location (in this case MIT) with respect to all sources. The differences in the absolute values of the bandwidth

	Selectivity									
	0.1		0.2		0.3		0.4		0.5	
# sources	BWCR	SF	BWCR	SF	BWCR	SF	BWCR	SF	BWCR	SF
8	0.2154	1.2589	0.2510	1.2723	0.2940	1.2749	0.3382	1.2510	0.3890	1.2642
16	0.2365	1.3469	0.2710	1.4031	0.3227	1.3945	0.3686	1.4052	0.4146	1.4073

(BWCR = bandwidth consumption ratio, SF = stretch factor)

Figure 4: Results for the Borealis processing network from Fig. 2.

consumption ratio between Figure 3 and Figure 4 arise due the higher selectivity values used in the former set of experiments.

5 Operator Distribution

As part of ongoing work in the SAND project, we are investigating mechanisms to further the distribution of a query plan across a heterogeneous network. Our current focus is on the parallelization of query operators, motivated by the vast divide between the scale of the abstract processing graph representation of a query, and the abstract graph representation of the network. To this end, we are pursuing techniques enabling an individual operator to utilize multiple sites during evaluation, while considering the necessary interaction between these sites to provide an equivalent operational semantics as would result from a centralized evaluation.

Our distribution mechanisms revolve around *replicating* and *partitioning* operators. To effectively support distribution, we focus on exploiting two properties: *network locality* and *data locality*. Network locality refers to the proximity of the data sources' network locations. Data locality refers to the similarity between data sources in terms of the input values produced, and the frequency at which these values are produced. Data locality also captures temporal properties of the inputs, such as the synchronicities of the input values.

We now discuss these mechanisms in the context of evaluating a join operator in a networked environment, with the goal of improving the average end-to-end latency of processing tuples (more details and relevance to previous work can be found in [3]):

- *Replication*: Our replication mechanism constructs a *join tree* of replicas (i.e., join instances), with each replica capable of performing a partial evaluation of the join operator based on the subset of all data sources it receives inputs from. Following partial evaluation, an operator replica may immediately deliver its output to any interested parties, creating a "short-circuit" route between sources, processing site, and sinks. We rely on a routing and evaluation policy between replicas to ensure correctness in our evaluation. First, operators in our tree evaluate the join predicate on tuples only if they arrive on differing input branches. Second, operator replicas forward input tuples received from their children towards the root of the tree. The first property ensures that our replica tree does not produce duplicate results, allowing input tuples to only join at their sources' lowest common ancestor in the tree. The second property ensures completeness of outputs, since all tuples will be forwarded to the root. This replication mechanism will prove beneficial in terms of average end-to-end latency provided the majority of the join's output occurs at the lower levels of the replica tree and thus a decreasing join selectivity at operator replicas closer to the root. We note that PIER recently considered a technique that performs a similar hierarchical join but in a rather opportunistic manner [8].

- *Partitioning*: While replication targets exploiting network locality amongst data sources by enabling nearby sources to compute their join results as early as possible, a second mechanism, operator partitioning, focuses on improving the networked deployment of a join operator by exploiting data locality amongst sources. Operator partitioning requires sources to introspectively maintain a probability distribution over the input values they produce. Sources subsequently exchange these distributions, using customizable approximate representations such as wavelet-based histograms, to compute join output probability distributions. A distributed partitioning algorithm identifies trends in these distributions across pair-wise combinations of data sources, placing partition boundaries on the join-key attribute domain according to a gradient-based heuristic applied to the output probability distribution. In this way, partitions are chosen to highly differentiate the set of sources that produce output values for each input value contained within each partition.

Note that the replication and partitioning mechanisms are orthogonal in their applicability, and thus may be composed: we can create a replica tree for each partition instantiated, first applying our distributed partitioning algorithm before constructing the replica tree using a hierarchical clustering algorithm.

5.1 Implementation Plans

Our next phase of implementation involves integrating these proposed algorithms into the SAND framework, and investigating the benefits these algorithms would bring to both a massively multiplayer online game, while performing "area-of-interest"-based data dissemination, and our wide-area network monitoring tool deployed on PlanetLab on top of the Borealis stream processing system.

References

[1] D. 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 *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Jan. 2005.

[2] Y. Ahmad and U. Cetintemel. Network-Aware Query Processing for Distributed Stream-Based Applications. In *Proc. of the 30th International Conference on Very Large Data Bases (VLDB'04)*, 2004.

[3] Y. Ahmad, U. Cetintemel, J. Jannotti, and A. Zgolinski. Locality Aware Networked Join Evaluation. In *Proc. of the 1st International Workshop on Networking Meets Databases (NetDB'05)*, 2005.

[4] S. Chandrasekaran, A. Deshpande, M. Franklin, and J. Hellerstein. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. of the First Biennial Conference on Innovative Data Systems Research (CIDR'03)*, Jan. 2003.

[5] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsaio, and R. Rasmussen. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.

[6] M. Franklin, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design Considerations for High Fan-in Systems: The HiFi Approach. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Jan. 2005.

[7] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4):22–33, Oct. 2003.

[8] R. Huebsch, B. Chun, J. 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. of the Second Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Jan. 2005.

[9] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proc. of the 29th International Conference on Very Large Data Bases (VLDB'03)*, Sept. 2003.

[10] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[11] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: An Acqusitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems*, 2005.

Implementing a Sensor Database System using a Generic Data Dissemination Mechanism

Omprakash Gnawali[‡], Ramesh Govindan[‡], and John Heidemann[§]

[‡]USC/Department of Computer Science, [§]USC/Information Sciences Institute
gnawali@usc.edu, ramesh@usc.edu, johnh@isi.edu

Abstract

The vision of a sensor network as a database has been reasonably well explored in the recent literature. Several sensor database systems have been prototyped [1, 11], and some have even been deployed [2]. However, these systems have largely integrated the query processing and the routing mechanisms. In this paper, we explore the design and implementation of a sensor database system (specifically, TinyDB [11]) on top of a generic sensor network data dissemination mechanism (Directed Diffusion [8]). Such a decoupled design is desirable, since it allows us to significantly re-use functionality and promotes overall system robustness. In conducting this exercise, we found that TinyDB influenced the re-design of Diffusion in several important ways.

1 Introduction

Prior work has proposed the use of database-like interfaces to program sensor networks. Enabling technology that uses SQL [3] to program sensor networks makes large-scale and fine-grained sensing accessible to scientists and researchers in other disciplines. Several such sensor database systems have been prototyped [1, 11], and some have even been deployed [2]. TinyDB [11] is one such system. TinyDB allows users to query a sensor network using SQL queries. In response to the SQL queries, query processors that run in each node in the network process and aggregate streams of sensor values much like how streams are processed in a database.

TinyDB developers have crafted a networking mechanism tailored specially for routing and topology maintenance in a TinyDB network. When queries are propagated to the network, a tree is formed which is used to route data back to the base station. While this might be adequate for specific platforms in which TinyDB has been developed, to make systems like TinyDB more flexible to changes in networking technologies and mechanisms in the future, we argue for building TinyDB on top of a standard networking substrate for sensor networks. The benefits of doing this are two-fold: it isolates networking from the application logic so that TinyDB can be ported to different networks with minimal change, and TinyDB or TinyDB-like system developers can reuse the networking layer services without having to roll out their own and spend a lot of effort debugging the networking layer. At a higher-level, our work can be said to be a concrete step towards examining how tuples in sensor databases can be routed using generic (as opposed to hand-crafted) routing mechanisms.

To demonstrate these benefits and to make a case for using a previously implemented and well-tested framework, we modified TinyDB to make it run in the filter framework [7] and over the Directed Diffusion [8] routing

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

protocol. Many database-like aggregation operations can be naturally expressed using these filters. Directed Diffusion provides a mechanism for naming the data in a generic fashion and also the routing protocols to efficiently search the network for relevant nodes and route the data back to the *sink* node.

Our system, called TinyDBD (TinyDB on Diffusion), has the same front-end as TinyDB. However, queries are injected into the network running Diffusion using *interest* messages. An interest message consists of a set of attributes that defines the query. Replies to the queries are routed to the querying node using *data* messages. A data message is also represented as a collection of attributes describing the data. The query processor is implemented as a filter on the nodes. Directed Diffusion allows for multiple nodes to inject interest messages at the same time. So, a TinyDBD system can have multiple active queries from different nodes at the same time.

Our implementation of TinyDBD runs on a testbed of PC104 nodes. It does not include any networking code and relies on Directed Diffusion to properly route queries and replies. In addition to supporting the TinyDB features, it is able to support multiple nodes issuing simultaneous queries.

In summary, the contribution of this paper is the design of, and experiences with, a framework for in-network processing that can be used to implement a SQL-like query system in sensor networks. Furthermore, we note the opportunity to reap the benefits of using a well-designed and well-tested framework for aggregation in a sensor network. Such benefits include isolation of aggregation and routing logic, assurance of functionality of the routing system, and potential to evolve the system to new platforms and new routing algorithms with minimal software changes.

2 Related Work

Directed Diffusion [8] is a data-centric networking substrate that provides mechanisms for naming, routing, and in-network processing in sensor networks. The filter framework [7] in Directed Diffusion allows custom code to run on the sensor nodes. Thus, filters can be used to process the information (aggregation) in the network before data reaches the querying node. TinyDBD is an attempt to use this architecture to implement an SQL-like query processing system. Prior work [5, 1] has suggested the feasibility of abstracting a sensor network as a database to increase the accessibility of the sensor data. TinyDB [11, 10] is an attempt to port database technology to sensor networks. TinyDB looks similar to desktop or server database systems at the interface layer. It comes with its own network stack. We integrated the TinyDB front-end with our Diffusion-based backend to demonstrate that applications like TinyDB can be built using the filter architecture in Diffusion and also to demonstrate the benefits of using an existing networking substrate to build such applications. Cougar [13] is another sensor database system that uses a query plan to determine the role of the nodes in in-network query processing. The Cougar project is also studying the interaction of characteristics of queries and the underlying routing layer.

3 Design

TinyDB on Diffusion (TinyDBD) exports an SQL interface to the end users of a sensor network. TinyDBD uses the filter architecture of Directed Diffusion to implement query processing mechanisms in the sensor nodes. It consists of three major components: (a) The base station, (b) Filter based query processing, and (c) Sensors.

The base station. The base station is a PC node that allows a user to formulate a query using a GUI. The station injects the query into the network using the transceiver node. TinyDBD supports multiple base stations injecting a query to the network at the same time. Each base station runs its own instance of the front end and the glue code. In a TinyDBD base station there are two main components: the (1) TinyDB front-end, which communicates with (2) Diffusion-based base station code. Figure 1 shows the architecture of the base station software.

We use the TinyDB front-end in TinyDBD. It is written in Java and presents an interface in which users can

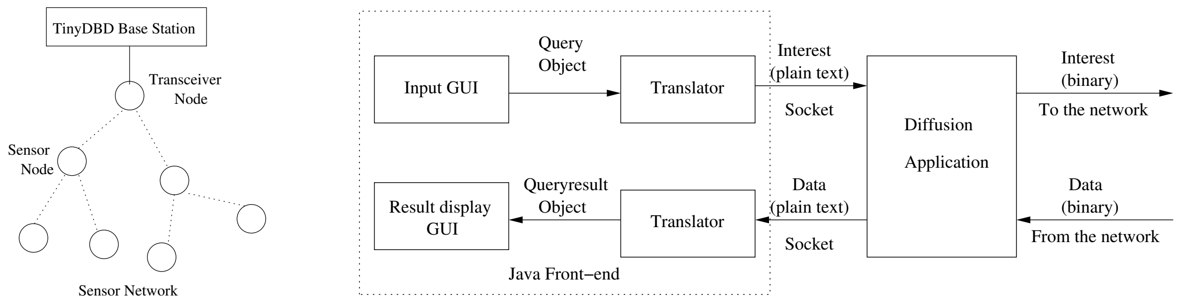


Figure 1: TinyDBD and Sensor Network (left) and architecture of the base station software (right)

```
SELECT AVG(light), temp
    FROM sensors
    GROUP BY temp
    SAMPLE PERIOD 2048
```

Figure 2: SQL Query

```
CLASS IS INTEREST_CLASS
PROTOCOL IS ONE_PHASE_PULL
TARGET IS DB
QID EQ 0
OP IS AVG
EPOCH IS 2048
LIGHT IS -1
FIELD IS TEMP
```

Figure 3: TinyDBD Query

```
CLASS IS DATA_CLASS
TARGET IS DB
QID IS 0
EPOCH IS 3
LIGHT IS 25
TEMP IS 15
```

Figure 4: TinyDBD Response

manipulate GUI controls to compose an SQL query. As in TinyDB, one sampling interval is called an *epoch* and is specified in the SQL query as a *sample period*. A sample query is shown in Figure 2. This query describes the result set consisting of average light values corresponding to each distinct temperature value in the network. This query specifies a sampling interval of two seconds.

In TinyDB, the glue code between the front-end and the transceiver node used serial communication and was designed for sending messages to a sensor network in TinyDB message format. We removed this code and inserted our code directly underneath the front end to generate messages in a format suitable for a network running Diffusion.

We parse the query object created in the TinyDB Java environment and compose the corresponding Diffusion messages. The idea is to translate the queries into a set of attributes that describe the orginal query. The glue code translates the query (Figure 2) to a Diffusion message shown in Figure 3. We serve the translated query in plain text using sockets to the Diffusion-based base station software. The base station, upon receiving this "interest" message, translates the text message into Diffusion-style attribute-value pairs. Table 1 presents the attributes used in TinyDBD Diffusion messages. The transceiver node injects this message into the network.

When the query results stream back to the transceiver node, it translates diffusion messages into text messages (Figure 4) and serves them to the Java based TinyDB front-end. The Java-based front-end translates the plain-text message into data structures expected by the TinyDB GUI.

Query processing in the nodes. Queries are distributed to the nodes in the network as Diffusion *interest* messages. Each node in the network has a filter, DBFilter, that listens to new queries in the network and sets up appropriate state to generate, aggregate, and forward results back to the querying node as data messages.

Diffusion provides a framework for in-network query processing using filters. TinyDBD uses the filter framework to do in-network processing (aggregation) of query results. DBFilter specifies the attribute `TARGET EQ DB` during the startup. This instructs Diffusion to forward to DBFilter all the messages that include the attribute `TARGET IS DB`.

When a node receives a query (an interest message), DBFilter sets up appropriate buffer and state for the query and floods the query to the neighboring nodes. This hop-by-hop flooding eventually distributes the query to the entire network. DBFilter also sets up a timer to wake up query processing module every epoch to process

Attribute	Description
CLASS	INTEREST_CLASS for queries, DATA_CLASS for results.
PROTOCOL	For TinyDBD, either ONE_PHASE_PULL or TWO_PHASE_PULL
TARGET	All the messages in TinyDBD is tagged with TARGET IS DB attribute
QID	Query ID. This is formed by concatenating HostID and a locally unique counter.
OP	Aggregation Operator such as AVG, MIN, MAX. If non aggregate query, OP IS SEL.
EPOCH	Duration of epoch in milliseconds.
LIGHT	For each field in the query, an attribute with that name is created (its value is ignored) and included in the interest message. In DATA message, the value of this attribute is the value of this field in the query.
FIELD	Group by field.

Table 1: Attributes used in TinyDBD messages.

the results. When the query results (data message) are forwarded by the neighbors, the receiving node buffers it until the begining of the next epoch. The query processing module then aggregates the buffered results and forwards it to the next hop towards the sink. If accumulated results are for a non-aggregate query, the module forwards all the messages and flushes the buffer.

DBFilter is concerned only with appropriately processing the query results, and is the application-specific component running on each node. DBFilter itself is not involved in forming or discovering the paths to be used for forwarding the query replies. This is done by the Gradient filter, which is the generic routing component usable by other applications as well. The Gradient filter runs in the nodes at the highest priority, and it intercepts any incoming and outgoing messages to maintain and provide the routing information. While queries are flooded to the network, the Gradient filter keeps track of the incident edge for an interest. When the results stream back for that query, the Gradient filter looks up the interest cache for the next hop for the matching attributes in the data message (query result), and forwards the response to the neighbor. This is one possible configuration for Gradient filter. For discussion on other configurations, please see the discussion in Section 5.

Generating data (Sensor Application). Sensor nodes run an application to drive the sensors within the framework provided by the Diffusion application API. The Diffusion application API consists of two calls: *Subscribe* and *Publish*. *Subscribe* describes to the Diffusion software the type of messages that an application is interested in. Diffusion will forward messages to the application only if the attributes in a given message match the attributes specified by the application during its startup. *Publish* is used to push data to the network.

On startup, the sensor application subscribes for queries in the network. Then, it stays idle until it receives a query. Upon receiving a query, it stores the parameters for the query and schedules the sensing module to wake up once every epoch. The sensing module samples the sensors and makes *Publish* calls to disseminate fresh data. Alternatively, the sensor application could start publishing data to the network immediately after the startup rather than idling till the first query is received. The effect of initial idling is conservation of energy while there is no query running in the network. One can put the application in idle state after the query has expired to avoid spending energy to run and sample the sensors when there is no query. Either way, the Gradient filter running in each node ensures that the sensor data propagates to the network only when there are active and unexpired queries.

4 Implementation

TinyDBD was implemented using Diffusion 3.1.2 libraries in C++. The sensors and the front end make *Publish* and *Subscribe* calls. The database logic is written in a filter called DBFilter. We have tested TinyDBD on a 10-node PC104 network. Our limited experiments show that it is possible to implement a database-like system on Diffusion. During the implementation of TinyDBD, we decided to use a TCP socket interface between the TinyDB front end and Diffusion-based base station software rather than integrating these two pieces of software.

This enabled us to use previously available software as much as possible. It is possible to implement the database logic as an application on the nodes, but we used filters because they provide a more natural framework for in-network processing of packets.

We exposed a subtle bug in the design of Diffusion 3.1.2 during our implementation of TinyDBD. In TinyDBD, at the end of each epoch, DBFilter generates a new message aggregating all the messages collected during the epoch. However, the Gradient filter expects packet IDs to be preserved even when messages are aggregated; this is an unclean design since, ideally, the Gradient filter should have been matching on attributes as the semantics of Diffusion matching dictates. This bug has since been fixed in Diffusion 3.2.

5 Design Choices

Our goal was to demonstrate the feasibility of our approach for implementing in-network query processing in sensor networks. Yet, many interesting design issues came up during the project. In this section, we list some of the interesting design issues that arose in TinyDBD and explain their implications for sensor database design.

Schema. TinyDB has an online schema system called TinySchema. One can define new attributes and push them to the network. In TinyDBD, we did not implement this dynamic schema definition mechanism. Our sensor ID's are hard coded in the application code and base station software. We think that attributes will rarely need to be redefined in a running network. When attributes change and they need to be propagated to the nodes, we believe that generic code distribution mechanisms [9] can be employed for this purpose.

Disjunctive queries. TinyDBD relies on the Gradient filter to match the attributes and find the path back to the sink. Attribute matching works by matching *all* the attributes which effectively computes a conjunctive query. A query with predicate such as $light < 50$ *or* $sound > 10$ translates to the following set of attributes in Diffusion: `LIGHT LT 50, SOUND GT 10`. The default attribute matching mechanism will forward a data message to the next hop if both $light < 50$ and $sound > 10$ are true. This makes disjunctive queries nontrivial to implement in Diffusion. This is an example where our investigation revealed a shortcoming of Diffusion.

However, it is possible to support a disjunctive query by supporting a new attribute matching mechanism, where a match is found if at least one attribute corresponding to disjunctive fields matches. Alternatively, all the disjunctive clauses can be concatenated into a single attribute with an *is* operator to exclude it from the list of attributes to be matched by the Gradient filter: `PRED IS LIGHT LT 50 OR SOUND GT 10`. Note that this attribute has an *is* operator so it will match all the gradients at the routing layer. This shifts the responsibility of disjunctive query attribute matching to DBFilter.

Aggregating over the same epoch. The current implementation schedules in-network processing without any synchrony to the schedules of the children nodes in the aggregation tree. Even though the aggregation uses a single value from each child per epoch, data from different epochs can processed (compared, aggregated) together because the epochs are not synchronized. In the worst case, the aggregate data will have sensor values sampled at n different epochs where n is the depth of the forwarding tree.

We chose the above approach for simplicity in implementation and it is not a reflection of limitations of our platform. Synchronized schedules (where the parent wakes up after its children) ensure that the information from the bottom of the tree can propagate all the way to the root of the tree within one epoch. This is the approach taken by TinyDB. Tagging data with epoch ID's, maintaining buffer size equal to the depth of the tree, and aggregating data only with matching tags would be another possible solution.

Choice of routing algorithm. The only requirement that TinyDBD imposes on the underlying routing system is that forwarding paths get setup and forwarding uses attribute matching. Because of this minimal assumption, it is possible to run TinyDBD using a variety of routing protocols without any change in the software.

Gradient, which is a filter responsible for implementing the routing protocol, uses a One Phase Pull protocol by default, however it can be made to use other algorithms [6]. If Gradient is configured to use Two Phase Pull, that will change the way forwarding paths are set up. However, DBFilter will still work without any change. The requirement that a path is setup towards the sink is fulfilled in either case. DBFilter does not work with another variant of Diffusion called Push, because that would require computing and disseminating replies to all the possible queries before a sink injects a query to the network. One or Two phase pull implementations set up paths along the nodes that are able to reach the source nodes with the least latency. In a different study, we have modified One Phase Pull to use routing metrics other than latency [4]. This will enable forming query-reply forwarding paths along the edges that have different levels of resources or reliability depending on the routing metric. This mechanism could be transparently used by DBFilter.

Choice of Platform. Our implementation was done on PC-104 and PC based platforms running linux. After this project, a stable version of Diffusion called TinyDiffusion [12] has become available for the motes. It is now possible to implement TinyDBD on the mote platform but one needs to be careful to design less verbose communication protocols in this resource-constrained platform. Even though the basic functionality will be similar, performance in terms of message reliability is likely to be different in different platforms. Some platforms might also make available CPU cycles to perform sophisticated aggregation in the intermediate nodes.

6 Conclusion

We have demonstrated that Diffusion can be used to build a database-like, in-network query processing system for sensor networks. We also wanted to investigate how proper networking infrastructure can aid in the development of applications that require in-network aggregation. We note that a proper networking infrastructure with a framework for in-network processing (e.g., Diffusion) helps isolate application logic from routing logic. This makes the system easy to understand and maintain. Unlike TinyDB, we were able to leverage previous work in routing in sensor networks and build a query system that is more portable.

References

[1] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards Sensor Database Systems. In *MDM*, 2001.

[2] P. Buonadonna, D. Gay, J. Hellerstein, W. Hong, and S. Madden. TASK: Sensor Network in a Box. In *EWSN*, 2005.

[3] C.J. Date and H. Darwen. *A Guide of the SQL Standard*. Addison Wesley, third edition, 1994.

[4] O. Gnawali, M. Yarvis, J. Heidemann, and R. Govindan. Interaction of Retransmission, Blacklisting, and Routing Metrics for Reliability in Sensor Network Routing. In *IEEE SECON 2004*.

[5] R. Govindan, J. M. Hellerstein, W. Hong, S. Madden, M. Franklin, and Scott Shenker. The sensor network as a database. Tech. Report 02-771, USC/CS, 2002.

[6] J. Heidemann, F. Silva, and D. Estrin. Matching Data Dissemination Algorithms to Application Requirements. In *SenSys*, 2003.

[7] J. Heidemann, F. Silva, Y. Yu, D. Estrin, and P. Haldar. Diffusion Filters as a Flexible Architecture for Event Notification in Wireless Sensor Networks. Tech. Report ISI-TR-556, USC/ISI, 2002.

[8] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Mobile Computing and Networking*, 2000.

[9] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A Self Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *NSDI*, 2004.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *ACM SIGMOD*, 2003.

[11] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[12] E. Osterweil, M. Mysore, M. Rahimi, and A. Wu. The Extensible Sensing System, 2003. Center for Embedded Networked Sensing (CENS) Poster.

[13] Y. Yao and J. Gehrke. The Cougar Approach to In-network Query Processing in Sensor Networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.

ICDE 2005 TOKYO

International Conference on Data Engineering

The 21st International Conference on

Data Engineering (ICDE 2005)

April 3-9, 2005

National Center of Sciences, Tokyo, Japan

Sponsored by

The IEEE Computer Society

The Database Society of Japan (DBSJ)

Information Processing Society of Japan (IPSJ)

The Institute of Electronics, Information and Communication Engineers (IEICE)

http://icde2005.is.tsukuba.ac.jp/ http://icde2005.naist.jp/ (mirror)

Welcome to ICDE 2005 Tokyo

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The 21st International Conference on Data Engineering (ICDE 2005) provides a premier forum for:

- sharing research solutions to problems of today's information society
- exposing practicing engineers to evolving research, tools, and practices and providing them with an early opportunity to evaluate these
- raising awareness in the research community of the problems of practical applications of data engineering
- promoting the exchange of data engineering technologies and experience among researchers and practicing engineers
- identifying new issues and directions for future research and development work.

The conference will be held in Tokyo, the capital city of Japan. Tokyo is the most populous metropolitan area in the world. Yet despite its complex urban landscape and impressive architecture, this city abounds with parks and gardens. Particularly in the spring season, blooming *cherry blossoms* will welcome you to Tokyo.

Advanced Technology Seminars

- XQuery Midflight: Emerging Database-Oriented Paradigms and a Classification of Research Advances (I. Manolescu and Y. Papakonstantinou)
- Data Stream Query Processing (N. Koudas and D. Srivastava)
- Web Service Coordination and Emerging Standards (F. Casati and G. Alonso)
- Online Mining Data Streams: Problems, Applications, Techniques and Progress (H. Wang, J. Pei, and P. Yu)
- Rank-Aware Query Processing and Optimization (I. Ilyas and W. Aref)
- Data Mining Techniques for Microarray Datasets (L. Liu, J. Yang, and A. Tung)

Workshops

- International Workshop on Ubiquitous Data Management (UDM 2005), Apr. 4, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Biomedical Data Engineering (BMDE 2005), Apr. 3-4, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Challenges in Web Information Retrieval and Integration (WIRI 2005), Apr. 8-9, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Privacy Data Management (PDM 2005), Apr. 8-9, 2005. Deadline: Nov. 17, 2004.
- International Workshop on Autonomic Database Systems, Apr. 8-9, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Realworld Multimedia Corpora in Mobile Environment (RWCinME 2005), Apr. 8-9, 2005. Deadline: Nov. 15, 2004.
- 2nd International Workshop on XML Schema and Data Management (XSDM 2005), Apr. 8-9, 2005. Abstract Deadline: Oct. 15, 2004. Paper Deadline: Oct. 22, 2004.
- International Workshop on Data Engineering Issues in E-Commerce (DEEC 2005), Apr. 9, 2005. Deadline: Nov. 22, 2004.
- International Workshop on Managing Data for Emerging Multimedia Applications (EMMA 2005), Apr. 8-9, 2005. Deadline: Nov. 15, 2004.

Keynote Speakers (other speaker to be announced)

- Pat Selinger (IBM, USA)
- Mike Stonebraker (MIT, USA)

Conference Officers

Honorary Chair:	The Late Yahiko Kambayashi (Kyoto University, Japan)
Organizing Committee Chair:	Yoshifumi Masunaga (Ochanomizu University, Japan)
General Chairs:	Rakesh Agrawal (IBM Almaden Research Center, USA)
	Masaru Kitsuregawa (University of Tokyo, Japan)
Program Chairs:	Karl Aberer (EPF Lausanne, Switzerland)
	Michael Franklin (UC Berkeley, USA)
	Shojiro Nishio (Osaka University, Japan)
PC Area Chairs:	Anastassia Ailamaki (Carnegie Mellon University, USA)
	Gustavo Alonso (ETH Zurich, Switzerland)
	Phillip Gibbons (Intel Research, USA)
	Takahiro Hara (Osaka University, Japan)
	Jayant R. Haritsa (IISc Bangalore, India)
	Alfons Kemper (University of Passau, Germany)
	Sharad Mehrotra (UC Irvine, USA)
	Wolfgang Nejdl (University of Hannover, Germany)
	Jignesh M. Patel (University of Michigan, USA)
	Evaggelia Pitoura (University of Ioannina, Greece)
	Jayavel Shanmugasundaram (Cornell University, USA)
	Kyuseok Shim (Seoul National University, Korea)
	Kian-Lee Tan (National University of Singapore, Singapore)
Executive Committee Chair:	Jun Adachi (National Institute of Informatics, Japan)
Panel Chairs:	Umeshwar Dayal (HP Labs, USA)
	Hongjun Lu (HKUST, China)
	Hans-Jörg Schek (UMIT, Austria/ETH Zurich, Switzerland)
Seminar Chairs:	Michael J. Carey (BEA Systems, USA)
	Stefano Ceri (Politecnico di Milano, Italy)
	Kyu-Young Whang (KAIST, Korea)
Demo Chairs:	Daniel Keim (University of Konstanz, Germany)
	Ling Liu (Georgia Institute of Technology, USA)
	Xiaofang Zhou (University of Queensland, Australia)
Industrial Chairs:	Anand Deshpande (Persistent Systems, India)
	Anant Jhingran (IBM Silicon Valley Lab, USA)
	Yasushi Kiyoki (Keio University, Japan)
	Eric Simon (Medience, France)
Local Arrangement Chair:	Haruo Yokota (Tokyo Institute of Technology, Japan)
Workshop Chair:	Masatoshi Yoshikawa (Nagoya University, Japan)
Proceedings Chair:	Motomichi Toyama (Keio University, Japan)
Publicity Chair:	Hiroyuki Kitagawa (University of Tsukuba, Japan)
DBSJ Liaison:	Hiroshi Ishikawa (Tokyo Metropolitan University, Japan)
CCF-DBS Liaison:	Xiaofeng Meng (Renmin University, China)
KISS SIGDB Liaison:	Chin-Wan Chung (KAIST, Korea)
DBIndia Liaison:	Krithi Ramamritham (IIT Bombay, India)
Australian DB Liaison:	James Bailey (University of Melbourne, Australia)
Treasurer:	Miyuki Nakano (University of Tokyo, Japan)

Related Conferences and Workshops

- 15th International Workshop on Research Issues on Data Engineering: Stream Data Mining and Applications (RIDE-SDMA), Apr. 3-4, Tokyo.
- The Seventh Asia Pacific Web Conference (APWeb 2005), Mar. 29-Apr. 1, 2005, Shanghai, China. http://apweb05.csm.vu.edu.au/
- 10th International Conference on Database Systems for Advanced Applications (DASFAA 2005), Apr. 18-20, Beijing, China. http://dasfaa05.cs.tsinghua.edu.cn/
- 4th International Workshop on Databases in Networked Information Systems (DNIS 2005), Mar. 28-30, Aizu, Japan. http://www.u-aizu.ac.jp/labs/sw-db/DNIS2005.html

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903