A Wakeup Call for Internet Monitoring Systems: The Case for Distributed Triggers

Ankur Jain[†,*], Joseph M. Hellerstein[‡,*], Sylvia Ratnasamy[*], David Wetherall[†]

[†]University of Washington
{ankur,djw}@cs.washington.edu

[‡]University of California Berkeley
jmh@cs.berkeley.edu

[*]Intel Research Berkeley
sylvia@intel-research.net

ABSTRACT

There have been a number of recent proposals for distributed monitoring infrastructures. We argue that a vital missing component from these proposals is a distributed triggering mechanism, by which nodes in the system can proactively work together to maintain invariants and constraints over the system-wide behavior. Without such a triggering component, existing proposals are doomed to poor performance, high detection latency, or both.

A key research challenge in building such a triggering mechanism lies in devising solutions whereby individual nodes can independently detect (and react to) violations of constraints that are defined in terms of the *aggregate* behavior of nodes in the system. We discuss our early efforts in designing such a triggering system and the research challenges that lie ahead.

1. INTRODUCTION

The growing scale and complexity of currently deployed distributed systems (such as PlanetLab, various overlay applications, and even the Internet itself) has given rise to the need for distributed tools that monitor the overall activity of the system. Recognizing this need, several recent research proposals have articulated a vision for large-scale monitoring systems that collect, store, aggregate and react to observations about network conditions [28, 15, 5]. At the core of these proposals is a distributed query engine whereby queries are shipped out to all the nodes in the system and responses are aggregated at either the source of the query [28] or in the network itself [15]. These systems thus adopt what is essentially a *query-driven* approach in which the monitoring nodes are largely passive, logging data locally and reacting only when a query is issued.

This approach is well suited to scenarios where the goal is to continually record system state (*e.g.*, to study traffic patterns in a network) or to obtain an instantaneous snapshot of system state (*e.g.*, "how many processes were running when we rebooted?"). Often however, our primary motivation in monitoring a system is merely to ensure that all is well. This is typically achieved by maintaining a well-defined set of logical predicates or invariants over the entire system. For example, one might want to ensure a distributed form of rate limiting so that a PlanetLab measurement experiment is not perceived to be a DDoS attack, *e.g.*, the aggregate traffic to any destination from a set of nodes running Scriptroute [25] should not exceed a pre-defined bandwidth threshold. Or, in a public DHT storage system such as OpenHash [17] one might want to take precautionary steps in resource allocation whenever the storage consumed by a client IP address exceeds a threshold number of bytes. Yet another example is a distributed IDS system that, like DShield [26] but in a distributed manner, uses correlated activity at different locations, including network telescopes or honeypots, to detect Internet worms and viruses.

In such cases, we do not want to record global system state at all times; instead, we would like to confine ourselves only to detecting and reacting in the event of occasional violations of these constraints. Ideally, this detection and response should be achieved in a manner that remains both timely and efficient at scale. Pure query-driven approaches, however, are ill suited to this task – to detect violations, existing systems are essentially reduced to periodic querying. Timely detection of violations requires frequent querying which is expensive while a lower rate of querying leads to long periods of undesirable behavior going undetected. Moreover, periodic querying leads to wasteful overhead in the common case where there are no violations.

In this initial paper, we argue that current monitoring systems should be augmented by a *distributed triggering* component in which the nodes being monitored proactively work to enforce a set of system-wide logical predicates. This notion of embedding triggers or predicates in query engines is not a new one – the topic has been extensively researched in the context of traditional (*i.e.*, centralized or modestly distributed) databases[29] and is now widely recognized as a useful and indeed a standard feature of modern databases [21]. Somewhat surprisingly then, triggering has yet to be identified as a key component in the recent focus on Internet query systems. We believe that if systems such as PIER [15] and Sophia [28] are to achieve true Internet-scale monitoring, it is crucial that they incorporate a distributed triggering component – without one, these systems appear destined to suffer from either excessive query overhead or high detection latency. With one, many system-wide conditions can be checked cheaply and in a manner that complements, not replaces, existing query-driven approaches.

A key challenge that arises in supporting triggering in a distributed system is that such triggers are naturally defined in terms of the *aggregate* behavior of a collection of nodes. For instance, in the above example of rate-limiting a PlanetLab measurement experiment, the constraint was defined in terms of the aggregate traffic generated by a set of nodes. Likewise, in the OpenHash scenario, the trigger was in terms of the sum, over all OpenHash nodes, of the storage allocated to a particular client. We call these *aggregate* triggers and distinguish them from *local* triggers where the constraint or invariant to be maintained can be defined entirely in terms of the state at, or behavior of, an individual node (*e.g.*, ensuring that no PlanetLab machine is over 60% utilized). While support for local triggers is likely to be fairly straightforward, aggregate triggers are more challenging. To the best of our knowledge they are a largely unexplored research direction. For example, there is much developed work [7, 9] on efficiently monitoring packets to, for example, identify "elephant" flows but these do not help us find the "distributed elephants". Similarly, pushback schemes [20] search for misbehaving aggregates locally but do not consider distributed searches.

In this workshop paper, we present initial ideas on supporting aggregate triggers in large distributed systems. We include an initial set of design goals and metrics (Section 2) some example solutions to illustrate possible approaches (Section 3), and some design considerations that arise from these examples (Section 4). We conclude by discussing a set of research directions in Section 5.

2. DESIGN GOALS AND METRICS

To motivate our triggering solutions and set the context for subsequent design discussions, we first identify key goals and metrics for a distributed triggering system.

Overhead: computing a distributed function involves wide-area communication. Ideally, this control traffic overhead should scale gracefully with increasing numbers of constraints and participant nodes. Moreover, such overhead should be low in the common case of "no suspect activity." This however is in direct conflict with our next goal.

Timeliness: this refers to the elapsed time between when a constraint was violated and when it was detected. Clearly, the earlier a violation is detected, the better. Note however that there is an inherent trade-off between timeliness and overhead. The more frequently an aggregate is computed, the faster a violation will be detected but the greater the communication cost and vice versa. The key insight that allows us to navigate this trade-off is that, often, *the value of timely detection depends on the extent to which a constraint is violated*. This leads directly to our next metric.

Penalty: this captures the "cost" of having violated a constraint and is a function of both the time-to-detection and the extent to which a constraint was violated. For example, in the rate-limited PlanetLab application introduced earlier, penalty might be measured as the *excess bytes* (*i.e.*, number of bytes over a target threshold) transmitted to a destination. Triggering solutions must aim to minimize penalties in the event that constraints are violated.

Accuracy: this refers to the accuracy of the computed value for the aggregate function. Clearly, this is important in order to reduce the likelihood of false positives or negatives. Accuracy might be impaired by a number of factors including stale input measures, packet loss, the failure of participant nodes, and incorrect inputs from misconfigured or malicious nodes.

Simplicity: a less quantifiable but equally important goal is that triggering algorithms should be both conceptually and operationally simple. This is particularly important if distributed triggering is to serve as a debugging and diagnostics tool. Distributed systems are sufficiently difficult to debug without having to debug the debugging tool!

With the above goals in mind, we now turn to designing solutions for a specific application – a rate-limited PlanetLab application.

3. ILLUSTRATIVE EXAMPLES

We illustrate two example approaches for distributed triggering using the problem of rate-limiting a PlanetLab application as our driving scenario. In Section 4, we extrapolate from these simple approaches to discuss design considerations for more complete approaches.

Consider a measurement experiment run on all PlanetLab nodes using the Scriptroute service [25] as a concrete application. Recall from our description in Section 1 that the goal of limiting this application is to ensure that the aggregate traffic emanating from a set of nodes to any destination does not exceed a specified rate limit. More formally, let n be the number of nodes, f_i^x be the flow rate (in bps) generated by node i to destination x and C be the global rate-limit threshold in bps. Then, we want to ensure that for every destination x:

$$\sum_{i=1}^{n} f_i^x \leq C \qquad (1)$$

For simplicity, we consider two key figures of merit for solutions: *communication overhead* in packets per second; and penalty as the *excess bytes* sent over the global limit C before a misbehaving aggregate is detected and can be halted.

A straightforward solution is to impose a hard *local* rate-limit of C/n at each node, so that the n nodes are simply unable to send at a rate greater than C. This is the mechanism currently used by Scriptroute. However, the per node limit of C/n either becomes too small as the system grows, so that it is overly restrictive on application behavior, or C must be made too large to detect all dangerous situations if it is to allow reasonable application behavior at individual nodes. Neither option is attractive and we seek a better alternative. We now present two example solutions using distributed triggers.

3.1 Rate-limiting PlanetLab: Take 1

Our first solution leverages a distributed hash table (DHT) to spread the responsibility for maintaining aggregates among the participating nodes. Each node i maintains a counter b_i^x of the number of bytes transmitted to every destination x. Each node also maintains a threshold number of bytes that is the fair share rate for node i (i.e., C/n) sent to destination x over an averaging interval of τ seconds, an epoch. Then, every time b_i^x exceeds the threshold, node i sends an update to the DHT node corresponding to HASH(x) and resets its byte counter for x. These updates enable the DHT node at HASH(x) to maintain an estimate of the aggregate flow to destination x. It can then take corrective action if the aggregate is estimated to exceed C. The following analysis illustrates the desirable tradeoff between overhead and penalty achieved by this scheme.

Analysis: Consider the scenario in which there is a destination x to which n' nodes send at a rate $f_i^x = f$ such that $n'f = F > C$. The threshold is $\frac{C\tau}{n}$ bytes per destination. Thus the number of packets seen by the corresponding DHT node is $nF/C > n$ every epoch of τ seconds. Averaging rates, no more than n packets are expected in an epoch if the aggregate is C. Thus a violation can be estimated after $C\tau/F$ seconds, in which case the number of excess bytes is $(C/F).(F-C)\tau$. This is very low for F slightly greater then C and bounded from above by $C\tau$ for large F, i.e., the excess is bounded by a constant no matter whether the violation is small or large. The overhead is also a fraction of the aggregate value (determined by the threshold), and in particular is small in the expected case that the aggregate is much less than C.

3.2 Rate-limiting PlanetLab: Take 2

In our second solution, every node i monitors its outgoing traffic to every destination x as before, averaging its local rates over τ seconds (an epoch) to smoothen out bursty traffic. Let the average rate be f_i^x. Then, with probability $p = \frac{f_i^x}{C}$ for $f_i^x \geq C/n$ (with $p = 1$ for $f_i^x > C$) or $p = 0$ for $f_i^x < C/n$, node i triggers and polls the entire set of participant nodes for their local rates. The cutout of $p = 0$ is to avoid overhead for the safe situation of all nodes sending below their fair rate of C/n. Once the triggering node has accumulated all the responses, it can trivially compute the aggregate to destination x, $\sum_{i=1}^{n} f_i^x$, and determine whether it exceeds C. If so, the triggering nodes disseminates an alarm to all participants.

Analysis: As before, consider n' nodes that send at a rate $f_i^x = f$ such that $n'f = F > C$. This implies that $f > C/n$ and so $p = f/C$. Hence at the end of every epoch, $n'p = F/C$ nodes are expected to trigger and initiate aggregation. To detect the violation we require that one or more nodes trigger, and assume that multiple triggers can be combined into a single aggregation operation. Each aggregation requires n to-and-fro messages. Thus, the expected communication overhead when there is a violation is n packets per epoch. When there is no violation it is substantially below this level, as nodes with rates less than the fair share contribute no overhead.

To calculate the expected time-to-detection and hence excess bytes, observe that the probability that no node triggers aggregation until time $(k-1)\tau$ is $(1-p)^{n'(k-1)}$. The probability that at least one node triggers in the next epoch is $1-(1-p)^{n'}$. Hence the probability that it triggers for the first time in epoch k is $[(1-p)^{n'(k-1)}]\cdot[1-(1-p)^{n'}]$. This gives the expected time after which aggregation is triggered as $\sum_{k=1}^{\infty} k\tau \cdot (1-p)^{n'(k-1)} \cdot [1-(1-p)^{n'}] = \frac{\tau}{1-(1-p)^{n'}}$. Thus, the expected number of excess bytes across all nodes is $\frac{(F-C)\tau}{1-(1-p)^{n'}}$. Since p and n' are related by $n' > 1/p$, the denominator is usually close to 1 (the worst-case lower-bound being $(1-1/e)$). The excess is then close to $(F-C)\tau$. The result is that the scheme has an excess that is somewhat larger than the previous one (but still competitive with periodic querying) in exchange for its decrease in overhead.

3.3 Relation to Periodic Querying

The above analyses highlight some interesting properties of the two schemes, particularly as they relate to the alternative of periodic queries. With periodic queries, n messages are exchanged every epoch whether or not there is any suspect activity in the system. This is in contrast to both our solutions which, in the common case where flow aggregates are below C, send little or no traffic, and more generally have overhead related to activity levels. Yet in the (hopefully rare) event where the aggregate constraint is violated the excess bytes in both our solutions is close to what they would be in the case of pure periodic querying, *i.e.*$(F-C)\tau$.

This, in essence, is the behavior desired of a distributed trigger mechanism – it should incur low overhead in the common case when the system behaves well; in the rare cases when violations do occur, they should be detected with low penalty.

4. DESIGN CONSIDERATIONS

An aggregate triggering mechanism rests on the local logic for triggering, and the global logic for testing the aggregate condition in the rule. The previous section explored two example points in the trigger design space. We now make an effort to taxonomize this design space. We begin with the local component of triggering. We then turn our focus to the global checking of the aggregate condition, which hinges on distributed coordination in both space and time.

4.1 Deterministic vs. Probabilistic Triggering

Notice that in Take 2, the choice of whether to compute an aggregate was a probabilistic one while Take 1 used a deterministic approach. Both, however, use what we call a *proportional* approach in which the *frequency* at which global checking is invoked is proportional to a locally available measure that is indicative of the aggregate value. More importantly, a proportional approach yields the property that, taken together, the *collective* outcome of the decisions by individual nodes is a reliable indicator of the likelihood that a global condition is being violated.

Hence, a proportional approach allows us to relax on timeliness and achieve low overhead in precisely those cases where violations are unlikely or at worst of low penalty. Such flexibility is not possible with a strict periodic approach. Also note that the above holds without assumptions on the time-varying behavior of local rates based on past observations, which might fail to hold in the case of a malicious adversary.

4.2 Coordination of Condition-Checking

Aggregate condition-checking is a distributed computation, which widens the design space considerably. In order to compute an aggregate, all the items in a set must be brought together and distilled into aggregates, either piecewise or *en masse*. This implies that items in the set have to be acquired, and rendezvous in the network in both space and time. We discuss each in turn.

Space: We use our two rate-limiting examples to illustrate several issues.

Participants: static vs. dynamic. In Take 2, we assumed that the set of nodes was well-known (and static), and a request for data could be sent to each. By contrast, in Take 1 we made no assumption about global knowledge of participants; in fact, using a DHT as a substrate is expressly intended to account for dynamic sets of participants.

Spatial Rendezvous: trigger vs. canonical. In Take 2, the aggregating node accumulated all the data for aggregation; the choice of accumulator could change each time the trigger was fired. This design is natural given that the triggering node is already responsible for contacting all the other nodes in the system. By contrast, in Take 1 there was a canonical accumulator address for measurements about a particular destination IP. It is also natural in the DHT scenario to implement hierarchical aggregation, which brings the data together piecewise, aggregating it along the way.

Dissemination: yes or no. In both of our rate-limiting schemes, only a single accumulator node learned the true aggregate. An alternative goal might be to disseminate the aggregate result to multiple nodes; these nodes could then make better-informed local decisions about checking triggers in future. In order to achieve broader dissemination, one can perform aggregation followed by dissemination, or integrate the two, e.g. via a gossip-style algorithm [18].

Time: In order to check an aggregate, the right data must come together at the same place *at the same time*. We have seen two alternative mechanisms for achieving that.

Coupled Pull: In Take 2, the node that fires the trigger also kicks off a protocol that requests ("pulls") readings from all the participants at once. Achieving this kind of synchronous pull requires coordination among the nodes, which can be difficult to scale.

Decoupled Push: In Take 1, each node sends ("pushes") data toward the accumulating address autonomously, whenever its local trigger fires. This decoupling of nodes in time is achieved by the use of storage at the accumulator: the accumulator is responsible for maintaining the state of a moving aggregate that it updates appropriately when new data arrives.

4.3 Roundup

In sum, our simple scenario highlighted four distinct variables in the design space, though presumably there are more. We note that the combinations of the variables are not entirely arbitrary. For example, the coupled-pull approach in Take 2 makes rendezvous at the triggering site quite natural, where as the decoupled-push of Take 1 requires some canonical location to host the data. A more detailed exploration of this design space should yield a large family of techniques that work in different settings. There is also the possibility of hybrid techniques that use different combinations of variable-settings at different scales.

5. MOVING FORWARD

Distributed rate limiting is an important, practical example of our agenda and we intend to explore it more deeply. However, we believe this is only the tip of the iceberg in the broader area of distributed triggering. In this section we explore other challenges that emerge as we vary assumptions made in previous sections. We stress that this is just a "tasting", and by no means an attempt to exhaustively cover the possible topics of interest.

5.1 A Variety of Statistical Methods

Our discussion so far focused on checking constraints over simple statistical summaries of the data – *e.g.*, functions like SUM and COUNT. Here, we consider a variety of additional statistical properties that may be useful to compute and test.

More than Sum: Complex Aggregates. Network aggregation infrastructures can support a large class of functions for merging data incrementally in a single pass up a tree. Up to now we have focused on SUM (which also subsumes counting or voting). Standard database languages offer other aggregates including AVERAGE, STDEV, MAX and MIN.

Given a constraint on one of these global aggregates (*e.g.* "*ensure that the STDEV of latency is* ≤ 1 *second*"), it is not immediately clear what local "event" should trigger global constraint checks.

These open questions apply to standard SQL aggregates, which are frankly quite rudimentary. It is possible to go much further with user-defined aggregate functions that summarize a distribution – these might include aggregates to compute histograms, wavelets, samples, and the like [2]. Constraints on distributions are not unnatural to specify: for example, one could ask to be notified when today's distribution of some attribute differs significantly (*e.g.*, as measured by relative entropy) from a baseline distribution. Efficient distributed triggering schemes for constraints over complex aggregates are an interesting open problem.

Aggregation with Redundant Routing: Recent work has focused on aggregation schemes that can compute correct (typically approximate) answers in the face of duplicate delivery of measurements. In some cases this is motivated by robustness – aggregation on a tree topology is very sensitive to loss, so it's better to replicate data along multiple paths (a DAG) to the final collecting node [19, 6, 23]. There may be value in pursuing these approaches in triggering, particularly to ensure delivery of critical alerts and/or to prevent malicious parties from having too much control over loss or error in the aggregate communication path.

Other work on gossip-based algorithms assumes that there is no collecting node, and that the goal is for all nodes to eventually know the value of an aggregate [18]. This approach is intriguing in its robustness and simplicity, and its ability to disseminate the aggregate value (which can enable high-accuracy local triggers). Gossip does have significant drawbacks in bandwidth and latency, so it may more appropriate in a hybrid scheme – *e.g.*, gossip locally to decide whether to trigger a more structured global aggregate check.

Statistical Methods Beyond Aggregation: In many cases, the desired trigger cannot be specified as a simple constraint of the form *"aggregate-fn(x) op constant"*. Instead, one wants to know properties of data with respect to a distribution. Outlier detection is a good example: the definition of an outlier is that it is significantly different from most of the other data *currently being observed*. Put succinctly, we would sometimes want to do statistical data mining rather than explicit querying.

5.2 Database-Like Approaches

To this point we have focused largely on triggers based on statistical properties. This is fairly natural in large-scale network monitoring tasks where the gross behavior is often of significant interest. But it is not the traditional focus of the triggering schemes explored in database research. Here we highlight some research topics that are analogous to themes in database research, but mapped into the massively distributed domain.

Traditional ECA Rules Standard SQL database triggers are based on a decade or so of research into so-called "active databases", which are database systems augmented with *Event-Condition-Action (ECA)* rules [29]. These rules specify a simple triggering event *E* that fires the rule (*e.g.*, insert tuple into table *T*"), followed by a potentially complex condition *C* (an arbitrary SQL query returning a non-empty answer set), which if true leads to a database action *A* (insertion or deletion of data, transaction rollback, invocation of a stored procedure, etc.) Database triggers are a recasting of the Production Rule ("Expert") Systems popular in the AI community in the 1970's and 1980's [16] – these were essentially "condition-action" rule systems on small (main-memory) databases.[1]

Database triggers and production rules tend to focus on matching (joining) events, rather than on aggregates. For example, a simple network-monitoring rule might say "*on arrival of a packet from host S at honeypot H, if another packet from S arrived in the last five minutes at any participating host N, send an alarm to N*". These kinds of rules are not unlike pattern-based intrusion detection systems, but distributed across all hosts. There is a rich literature on *discrimination networks*, which are actually single-site data structures that can efficiently handle a large set of such triggers [11, 22, 14]. Mapping those ideas to a massively distributed domain is an open challenge.

Operator Composition: A recurring theme in database system design is to focus on libraries (algebras) of simple operators that can be functionally composed in arbitrary ways to form complex queries. In our triggering discussion here, we have focused largely on testing predicates over simple single-aggregate expressions. It would be attractive to extend this work to handle composition of operators in a seamless and efficient way. The following is an example of a constraint with one aggregate computed over another: "*compute the sum of all flows that originate from nodes with a greater-than-average number of distinct recipients; if this is greater than X sound an alarm.*" One could also compare two separate aggregate values (*e.g.*over different subsets of nodes), or test aggregates over joins of events, etc. Approximating these complex queries is a hard problem on its own [12, 13]; coming up with efficient condition-testing protocols seems strictly harder.

A note on relation to Continuous Queries We distinguish our general approach here from related work on continuous queries in databases. Rather than set up a distributed trigger, one could register a similar continuous query with a distributed stream query system [19, 4, 24]. However, existing

[1] Expert systems were hyped to the point where they shouldered partial blame for the onset of "AI winter" in the late 1980's. Database triggers are largely considered useful, in part because they are typically used sparingly, often in a manner that avoids cascading execution in which one rule's actions trigger another's conditions, etc.

distributed continuous query approaches do not employ the kind of local triggering that we rely upon here; instead they take new incoming data and proactively push it through a distributed dataflow – either upon arrival, or on a periodic schedule. Moreover, distributed query systems tend to focus on propagating query results, which may be approximate to a single point in the network; by contrast we are interested in maintaining global invariants without reference to a specific query node.

5.3 Security and Privacy Issues

There are a host of security issues in delivering triggering schemes. One main theme is to ensure that the triggers are tamper-proof, and will reliably "sound the alarm" iff the desired conditions are truly met. Another is to prevent the triggering infrastructure from being fooled into becoming a source of undesirable traffic itself. This is frankly a wide open research area. We forgo the opportunity to speculate about these issues in detail here, though we can point the interested reader to a relevant survey of security issues in p2p systems [27].

On the privacy front, there is a growing body of useful work on privacy-preserving aggregation (*e.g.* [3]), join (*e.g.* [1]) and data mining (*e.g.* [10]). The aggregation and join work both rely on homomorphic cryptography schemes (*e.g.* [8]) that enable arithmetic or comparison operators to be executed in the encrypted domain. The data mining work uses both cryptographic and randomization-based approaches to hiding information. It would be interesting to see how the use of these schemes might change the considerations for triggering.

6. REFERENCES

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM SIGMOD*, June 2003.

[2] D. Barbará, *et al*. The New Jersey data reduction report. *IEEE Data Engineering Bulletin*, 20(3), 1997.

[3] J. Canny. Collaborative filtering with privacy. In *IEEE Conf. on Security and Privacy*, May 2002.

[4] M. Cherniack, *et al*. Scalable distributed stream processing. In *CIDR*, 2003.

[5] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A knowledge plane for the internet. In *SIGCOMM*, 2003.

[6] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *ICDE*, 2004.

[7] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *SIGCOMM*, 2003.

[8] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4), July 1985.

[9] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM*, 2002.

[10] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy-preserving mining of association rules. *Information Systems*, 29(4), June 2004.

[11] C. Forgy. RETE: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19, 1982.

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

[13] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In *ACM SIGMOD*, June 1999.

[14] E. N. Hanson, S. Bodagala, and U. Chadaga. Trigger condition testing and view maintenance using optimized discrimination networks. *IEEE Trans. Knowl. Data Eng.*, 14(2), 2002.

[15] R. Huebsch, *et al*. Querying the internet with PIER. In *VLDB*, 2003.

[16] P. Jackson. *Introduction to Expert Systems*. Addison-Wesley Publishing Company, 1986.

[17] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Adoption of DHTs with OpenHash, a Public DHT Service. In *IPTPS*, Feb. 2004.

[18] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS*, 2003.

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

[20] R. Mahajan, *et al*. Controlling high bandwidth aggregates in the network. *SIGCOMM CCR*, 32(3), 2002.

[21] J. Melton. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann, 2 edition, 2000.

[22] D. P. Miranker. TREAT - a better match algoritm for AI production systems. In *AAAI*, July 1987.

[23] S. Nath, P. Gibbons, Z. Anderson, and S. Seshan. Synopsis Diffusion for Robust Aggregation in Sensor Networks. In *ACM SenSys*, Nov. 2004.

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

[25] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility, 2002.

[26] J. B. Ullrich. Dshield, http://www.dshield.org.

[27] D. Wallach. A Survey of Peer-to-Peer Security Issues. In *Int'l. Symposium on Software Security*, 2002.

[28] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *HotNets-II*, Nov. 2003.

[29] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.