Data Management in Interimistic Environments

Abheek Anand and Sudarshan S. Chawathe*
Computer Science Department
University of Maryland
College Park, MD 20742, USA
chaw@cs.umd.edu

Abstract

We address the problem of managing information in an interimistic environment following a disruptive event that results in widespread failures of computers, networks, and other resources. We motivate the problem and discuss the design requirements. As a concrete example of such information management, we present our work on data dissemination in an interimistic environment in which both network membership and data characteristics are subject to continual change. Our methods use low-overhead protocols to determine a suitable method for routing a stream of data. We also present a brief experimental evaluation of our methods.

Keywords: interimistic data management, data dissemination, multicast.

1 Introduction

We address the problem of *interimistic* data management in the chaotic environment in the minutes and hours following some catastrophic event, such as a natural disaster. A distinguishing feature of this work is that it focuses on a very small (by database standards) window of time, ranging from hours to days, but not weeks, months, or years. We may think of the problem as that of temporary data management, emphasizing the high premium on operating during the short time following a disaster and the relative irrelevance of longer term properties. This approach is in sharp contrast to the conventional bias of data management systems, which favor long-term guarantees (consistency, durability, etc.) over short-term ones. We envision scenarios in which our methods would be very useful for a few hours and then be rendered useless when the environment is back to normal.

As a motivating scenario that we use throughout this paper, consider the situation just after (or during) a devastating hurricane that has hit a typical suburban county. Such an event is likely to cause widespread power failures, road blockages, floods, and communication failures, among other problems. Although much of the information infrastructure may be damaged or disabled, it is also very likely that there are parts of it that function in a reasonable manner. For example, cellular and land-line phones may function in some locations and not others. Power and other utilities may have similar spotty availability. Now consider an ambulance responding to medical emergencies in this environment. It is likely that the crew is able to communicate with the central dispatch office using their reserved wireless frequencies. However, the office may not be able to provide the crew with the information it needs because of problems with its computers or those

*This work was supported by the National Science Foundation with grants IIS-9984296 (CAREER) and IIS-0081860 (ITR).

of other agencies on which it depends. For example, due to problems with traffic management systems, dispatch may not be able to route the ambulance around road closures and other hazards. Further, we must also consider the case in which the dispatch office suffers a catastrophic failure. Point-to-point communications between ambulance and other crews may still be possible over the air waves, but keeping track of a variety of information in an ad hoc manner over the radio will quickly become unmanageable. As a simple example of a better solution, consider methods that maintain soft state about recent positions of ambulances (using on-board Global Positioning System units and wireless communications) in a distributed manner. Simply being able to visualize the recent trails of other vehicles is likely to provide useful information about road conditions.

In the above scenario, it is quite likely that public buildings such as schoolhouses will be converted into emergency shelters and medical centers. Such a center has a wealth of important information of interest to others, such as, statistics on injuries, symptoms, and diagnoses that may help planners better judge the situation and quench any burgeoning outbreaks of disease. Conversely, such a center also has pressing information needs, such as the status of medical professionals and supplies in neighboring areas that could be called into service. Some of these needs can and are met using telephones and other ad hoc communications. However, consider the fact that the schoolhouse may have a perfectly working collection of computers, perhaps with reasonable network connectivity. The quality of information would be greatly improved if these systems, not designed for such uses, could be effectively pressed into service. Note that although one may expect emergency workers to carry dedicated and well designed information systems, there is a need for information beyond these systems. For example, although not life-saving, the ability to inform people of the status of their homes and friends is valuable.

In this paper, we focus on methods for selective dissemination of semistructured data in the interimistic environment described above. Briefly, we may describe this problem as the interimistic version of the well-studied problem of selective dissemination of information. While the most obvious features that distinguish this version of the problem from earlier work arise from the environment (e.g., lack of provisioning, high failure rates), there are additional distinguishing features as well. There are multiple sources of data and there is no qualitative difference between the hosts that serve as data sources and those that subscribe to data. Further, the arrival rate of data is likely to be bursty and unpredictable.

We consider a set H of heterogeneous hosts (computers, of varying capabilities). There is a subset $S \subseteq H$ of hosts that are sources of semistructured data streams in XML form. The XML stream is not necessarily segmented into documents or similar units and its structure is not known in advance. There is another subset $D \subseteq H$ of hosts that are interested in different parts of the streams emanating from hosts in S. (The sets S and D need not be disjoint.) The data interests are expressed using *XPath* queries [9] that are interpreted as subscriptions (standing queries). That is, there is a set of XPath queries $Q = \{Q_{mn} | m \in S, n \in D\}$, where Q_{mn} is the subscription of host $m \in D$ over the data emanating from host $n \in S$. At a point t in time, each host in $h \in H$ has limits $I_h(t)$ and $O_h(t)$ on its incoming and outgoing network bandwidths. Further, the available buffer space on host h at time t is limited by $B_h(t)$.

Our task is to devise a *strategy* for data dissemination that satisfies the resource constraints and provides each host in D with the requested data, over time. (More precisely, the actions required to follow the strategy at run time are required to be simple and efficient and exclude, for example, query replanning). Specifically, a data dissemination strategy in this context consists of a collection of rules that determine the manner in which hosts transmit data amongst themselves at any point in time. In addition to specifying the initial setup, such a strategy will indicate when and how the setup is modified (typically in response to changes in I_h, O_h, and B_h over time). We note that we are required to provide a *distributed* strategy *over time*, not simply a centralized solution to a snapshot of the problem parameters at some instant in time. In particular, we cannot

assume non-local knowledge of hosts, network conditions, and data repositories at a host. Of course, we may convert a static centralized solution to a dynamic distributed using periodic re-evaluation and one of the several standard methods for maintaining distributed state . However, such an approach would be very inefficient for even a small number of hosts.

Most existing work that addresses the problem of selective data dissemination is designed for a controlled, centralized environment. Siena [3] is an event-based publish subscribe system that uses a set of special broker nodes in the network to maintain state and forward data to clients. It uses techniques like query aggregation for efficient filtering, and is designed to work with multiple brokers to scale to a large set of subscribers. Continuous query systems like NiagaraCQ [8] use grouped subscription queries, coupled with change-based and timer-based events, to notify interested clients of changes in an available data source. Work on efficient filtering systems, such as XFilter [1] and YFilter [12], concentrates on developing fast algorithms for matching a stream of documents against a large set of queries. Our work is designed to distribute data in a completely decentralized, autonomous environment. While this assumption precludes us from using several of the techniques used in the above work, our methods are designed to be able to transparently use those existing techniques that only require local computation (e.g., the filtering component).

The remainder of the paper is organized as follows. Section 2 outlines our methods for data dissemination in an interimistic environment. Section 3 describes an experimental evaluation of our methods. We describe related work in Section 4 and conclude in Section 5.

2 Interimistic Data Dissemination

We now consider a following specific instance of the general problem described in the Section 1. We stress that our discussion below addresses only a part of the general problem, and makes some simplifying assumptions. For example, we assume that the data of interest is specified by indicating the source of a data stream along with an XPath expression. We do not address how the source itself is identified (perhaps using a replicated directory) or how application needs are mapped to XPath (a task that has been addressed elsewhere).

There is a single source s of XML documents that arrive in streaming form. Each host $h \in H$ sequentially subscribes to documents using an XPath filter expression. Nodes have a limit on the number of network connections (fan-out). We wish to determine a strategy for efficiently disseminating the documents according to the subscriptions while honoring the fan-out constraints. (In effect, we are using simplified variants of the bandwidth constraints introduced in Section 1.) Further, the stream of document is subject to frequent changes in characteristics (resulting in major changes to statistics). The main challenge is devising protocols that adapt to two kinds of changes at low overhead: (1) hosts joining and leaving the network and (2) changes in data characteristics.

A naive solution to this problem is to multicast the stream to each node, with query evaluation done at the individual nodes. However, this approach would entail a significant communication overhead, especially for participants with low-selectivity queries. We seek a method that organizes nodes into a network that takes advantage of the commonalities among subscriptions. However, our focus is on a low-overhead protocol that can respond rapidly to changes in membership, subscriptions, and data characteristics.

Consider the following simple strategy: When a host arrives (or issues a subscription), it first requests a direct connection to s. In the likely event that such a connection is unavailable (due to the fan-out constraint), s forwards the request to one of its children, c. (We view the network as a multicast tree with s as the root.) The child chosen is one whose *effective subscription* has the largest overlap with the new subscription. Here, a node's effective subscription is the smallest subscription that contains (in a query-containment sense) the

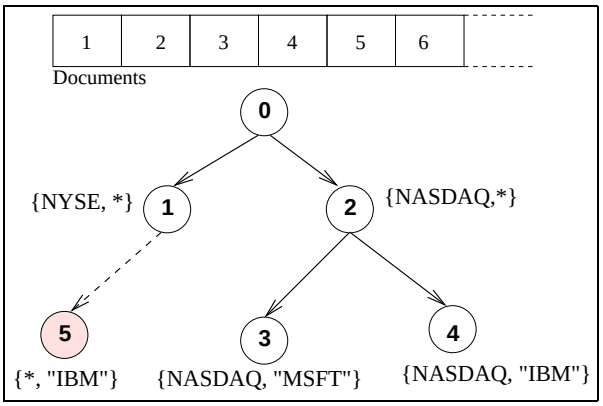

Figure 1: Example CoDD System

subscriptions of all hosts in the subtree rooted at that node. If c is able to accept s as a child (i.e., c's degree has not reached the limit), s connects to c. Otherwise, the procedure is repeated recursively. This *join protocol* is the static part of the strategy.

The dynamic part comes into play in terms of periodic reorganizations (triggered by events such as a node determining it's effective subscription being too large compared to its original subscription). The reorganization protocol itself is fairly straightforward, along the lines of the join protocol. The interesting difference is that instead of moving down the tree from s using *intensional overlap* of the subscriptions as a guide, we use *extensional overlap* of their recent results. The reason is two-fold. On the one hand, extensional overlaps (estimated by intersecting the document identifiers of recent results for the two subscriptions) are determined much more efficiently than intensional overlaps [6] (which require query-containment and common-subexpression computations). On the other, they may be effective at determining overlaps that do not exist based on the known constraints on data, but that recent documents exhibit.

We have implemented these ideas in CoDD, a system for data dissemination in a serverless environment. Unlike traditional systems that maintain a centralized infrastructure for query processing and data delivery, CoDD does not assume any server nodes. Instead, CoDD provides protocols that allow clients to organize themselves into a data-aware overlay network that is used to disseminate data based on subscriptions.

Each node in a CoDD system maintains simple statistics on the data that it forwards, and uses this information to make decisions for creating and maintaining the dissemination network. The protocols try to minimize the amount of extraneous data received by each node, which is data that the node must receive only to serve its downstream peers. The protocols do not rely on any centralized computation, and use reorganization to adapt to changes in the distribution of the queries and data stream. The following example illustrates some of the main ideas.

Example 1 Figure 1 suggests a distributed system for disseminating streaming data emerging from node 0. The data consists of a stream of attributed data items that contain some stock quote information. Each of them are in the form $\{\texttt{Exchange}, \texttt{Symbol}\}$, which describes the name of the exchange and the company corresponding to each quote. Each node subscribes to a subset of the data using a set of constraints on these attributes. For example, Node 2 has a query $\{\texttt{NASDAQ}, *\}$, which matches all quotes from the NASDAQ exchange. Similarly, Node 3 subscribes to documents containing Microsoft quotes from NASDAQ.

The dissemination protocol dynamically creates a tree-structured overlay network, with the data source as root and data flowing down the tree. Nodes join the system in increasing order of their IDs. Suppose

each node permits a maximum fanout of 2. (We pick a low value for illustration using a small example. In practice, values are likely to be in the range 4 through 10.) Every node filters documents needed by each of its subtrees before sending them downstream. Note that each of the queries for nodes 3 and 4 is contained in the query of node 2. Therefore, adding them as children of 2 entails no overhead in terms of the documents sent down that subtree.

Now consider the addition of node 5. Node 5 subscribes to documents with quotes for IBM, irrespective of the stock exchange. Since there are no containment relations for node 5 with any of the existing nodes, its addition entails some extraneous data on the network. Further, lack of knowledge of future document distributions and node arrivals and departures makes it difficult to predict the optimal position for node 5. For example, the system might determine that the best place to add it is as a child of node 1. This has the nice properties that the fanout constraint is not violated, and adding it as a child of 1 (instead, say, as a child of 3) results in low data latencies.

Note however, that in general there might be little overlap between the queries for 1 and 5, and that this is not clear by comparing their respective subscription queries. For example, it might be the case that the only data items that satisfy the query of 5 are from the NASDAQ exchange. The system should be able to determine this extensional commonality as soon as possible, and might decide to reorganize the topology to better exploit this overlap (for example, by moving node 5 as a child of node 4). ■

3 Experimental Evaluation

The current implementation of CoDD consists of a set of Java classes, which can be deployed in a distributed setting. We have also developed a simulator for nodes, which allows us to test a large number of configurations locally. We conducted our experiments using the Sun Java SDK version 1.4.1_01 running on a PC-class machine with a 1.6 GHz Pentium IV processor and 256 MB of main memory, running the Redhat 9 distribution of GNU/Linux (kernel 2.4.22). We simulate network data transfers using standard interprocess communication mechanisms, such as local pipes for the simulator and sockets for the distributed deployment of CoDD. Each node runs an instance of a query engine corresponding to the data and query language in use. This engine evaluates each child's filter query on each data item, and enqueues the item to be sent to each of the children whose filter query it satisfies.

The primary metric we study is the network overhead. Consider the transfer of a data item over a network link. If the item does not satisfy the destination's subscription, the transfer is termed *extraneous*. (The item may be useful to one of the node's descendants in the multicast tree.) A node's overhead is computed by dividing the number of extraneous transfers sent to the node by the total number of transfers sent to it. Similarly, the system overhead is computed by dividing the number of extraneous transfers (occurring over all links in the network) by the total number of transfers. In some cases we are interested in measuring system average over the set of nodes that potentially receive extraneous items (i.e., the non-leaf nodes), and we refer to this as the interior node overhead.

Our test dataset is an abstraction of the mapping between data and subscriptions. We assign nodes to one or more of k category buckets. Intuitively, each bucket represents an interest class. We use $k = 20$ by default. Subscription queries are modeled by mapping each data item to every member of a fixed number of these buckets. The selectivity (s) of the query set for each experiment is defined as the fraction of the total number of documents a query is mapped to, averaged over the total number of queries. We model changes in data distribution by performing a random shuffle on the buckets, at a rate determined by the change-frequency parameter cf. Each shuffle operation consists of splitting c buckets into two sub-buckets and merging uniformly randomly chosen pairs of these sub-buckets to form new mappings. We use $c = k/5$

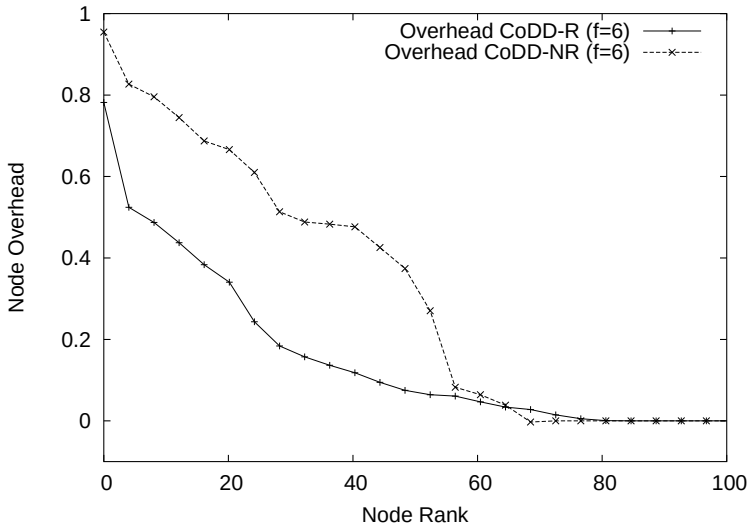

Figure 2: Overhead versus Reorganization Frequency

by default. The default synthetic dataset we used had a selectivity (s) of 0.2, with 400 nodes (n), and 10000 documents (d) being published in the system. The document distribution was made to change every 200 documents (cf), and the default fanout (f) was 6. The reorganization was initiated after every 200 documents were published in the system, by nodes that experienced an overhead greater than a threshold ($rthresh$) of 0.2.

Figure 2 summarizes the results of our experiments studying overhead with a changing document distribution for the default synthetic dataset described above, using a fanout of 6. The figure displays the overhead of the top 100 interior CoDD nodes with and without reorganization (CoDD-R and CoDD-NR, respectively), ranked according to their overhead. Overhead for leaf nodes is always zero, since they do not need to accept items for the purpose of transferring them to their descendants. Reorganization is seen to significantly lower overhead: the average overhead decreases by over 50% as a result of initiating reorganization, indicating that reorganization allows CoDD to adapt well to changing data distributions.

We also compared our system with Siena [3], an event-based publish-subscribe system used to selectively disseminate data to a large set of subscribers. Siena uses a set of coordinated brokers, with each client being mapped to one broker. Brokers communicate with each other depending on the requirements of the clients they serve. This design is significantly different from that of CoDD, and is based on a centrally controlled setup of the server environment. In contrast, CoDD assumes no central coordination. Nevertheless, Siena is the implementation that is closest to CoDD. For purposes of comparison, we emulate CoDD's server-less environment using Siena as follows. The network is built incrementally, with a fraction of randomly selected nodes designated as brokers. These nodes are connected in a balanced tree topology, and clients connect to a uniformly randomly selected server in this network. We use f clients per server, where f models the fanout constraint in the corresponding CoDD environment.

Figure 3 depicts the results of experiments comparing the overheads of Siena and CoDD using the default dataset. This experiment favors Siena because the overhead of Siena is only due to the designated broker nodes. We do not model the overhead incurred by protocols used to set up the brokers. (Recall that Siena assumes some centralized control.) Nevertheless, we note that CoDD perform well because it generates

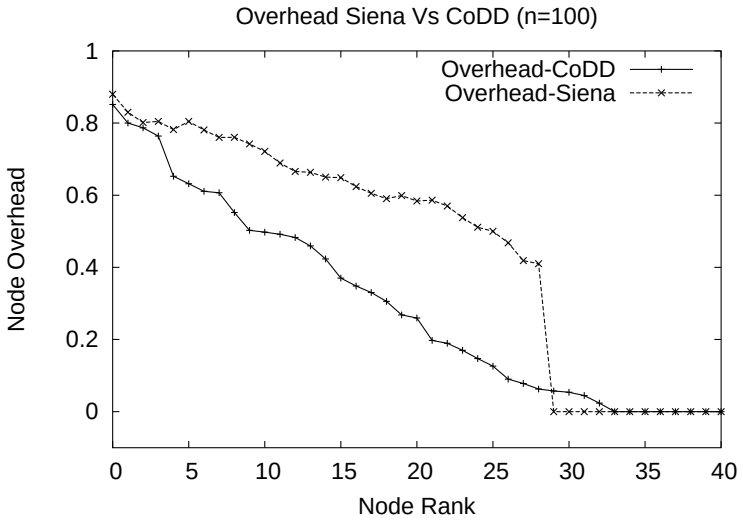

Figure 3: Overhead of Siena versus CoDD

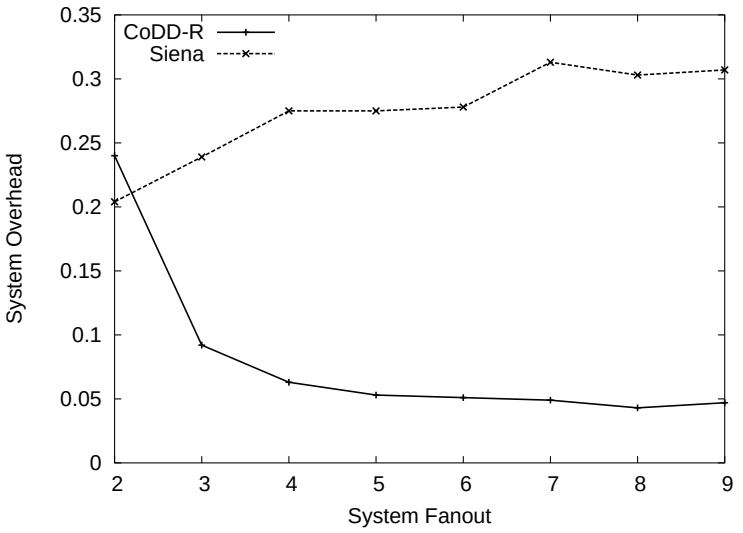

Figure 4: Average Overhead versus Fanout

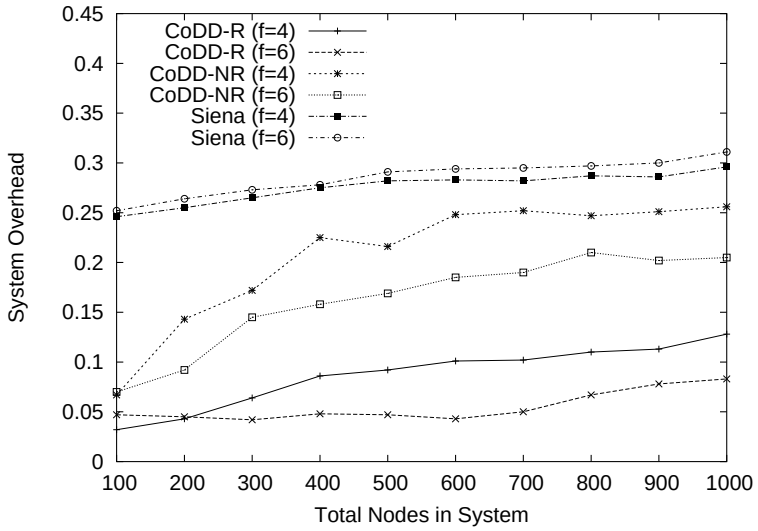

Figure 5: Average Overhead in System

topologies that are data-aware. The overhead of Siena dips below that of CoDD at $x = 60$ because only interior nodes incur any overhead in either system, and the number of such nodes is fewer in Siena because of the balanced tree that it creates.

In Figure 4, we summarize our study of the effect of fanout on overhead. We observe that the overhead in CoDD drops rapidly as fanout is increased from 2 to 4, with further increases in fanout providing only minor improvements. This result suggests that it is not necessary to use high fanouts, which have the effect of increasing the processing and network load on each node. In contrast, Siena's overhead gradually increases as fanout is increased. This result may be explained by noting that increasing the fanout increases the number of children for each interior node in the tree. These nodes are usually going to have little overlap with their new children, and thus their overhead is likely to increase significantly.

Figure 5 summarizes the results of experiments varying the number of nodes in the system, with two fanouts (4 and 6), using Siena and CoDD, the latter both with (CoDD-R) and without (CoDD-NR) reorganization. The figure suggests that the overhead is substantially lowered by reorganization. Further, the rate at which overhead increases with increasing network size is lower for the reorganization case.

4 Related Work

4.1 Publish-subscribe systems

Siena [3] is a closely related event-based publish-subscribe system. Clients in Siena subscribe to messages by registering subscription queries. Data messages are routed using cooperating brokers that share subscription information. The primary data model used in Siena represents events as a set of typed attributes. Queries express constraints over a subset of these attributes. Subscriptions are aggregated using internal data structures that describe a partial order on the covering relationships of these queries, and that can be efficiently traversed to match incoming notifications. The brokers comprise a centralized infrastructure that the protocol maintains, and scalability is achieved by adding brokers to the system. SIFT [18] is a keyword-

based text filtering and dissemination system for Internet News articles. It supports two profile models, one that does exact boolean predicate matches, and one that does a fuzzy match using a similarity value assigned to every (document, profile) pair. The Gryphon [14] system implements data transformation and dissemination using a similar architecture. An information flow graph is used to describe the flow of events in the system, and each node in the graph implements operators that merge, transform, filter and interpret data items. This graph is then optimized and deployed on a network of brokers, which are used for data delivery. CoDD differs from these systems by using protocols that are decentralized, and that work well with loosely-coupled subscriber nodes without centrally controlled servers. CoDD also enables resource-poor nodes to participate because nodes can manage their level of cooperation using capacity constraints and reorganization. Another difference is the use of extensional overlaps in CoDD. While this feature precludes methods that take advantage of specific features of the query language, it permits easy application of our protocols to a variety of data models and query languages.

Xlyeme [17] is a system used in conjunction with publish-subscribe systems that tackles the problem of finding the events associated with monitoring queries that are satisfied by incoming documents. It uses a hierarchy of hash tables to index sets of atomic events that compose more complex events. Le Subscribe [13] proposes a predicate model to specify subscriptions. To match incoming events (i.e., a set of attribute value pairs) with predicates efficiently, all predicates are indexed, and all subscriptions are clustered by their common conjunctive predicates. A cost model and algorithms are developed to find good clustering structure and to dynamically optimize it. A common feature of these systems, in contrast with CoDD, is the use of restricted profile languages and data structures tailored to the complexity of the languages in order to achieve high throughput.

4.2 Multicast in Networking literature

Network layer multicast [11] is an efficient mechanism for packet delivery in one-to-many data transfer applications. It uses protocols operating in the interior of the network that create a connected tree, with the clients at the leaves of the trees, such that the length of the path from the root to each client along the tree is close to the underlying network path. End system multicast [15, 16] pushes the functionality needed to create this tree to edge nodes on the network, and the data is transmitted over an overlay architecture. The tradeoff for using this end-to-end approach is a sub-optimal performance in terms of network paths and number of duplicate packets on each link. The hierarchical approach used by CoDD is similar to the approach taken in NICE [2], an end system multicast protocol that organizes nodes in a layered tree hierarchy, and is designed to have low control overheads and scale well with the size of the participant set. The fanout constraint is captured in NICE using a parameter that determines the number of nodes at each layers, and bounds the number of outgoing connections for each node. CoDD works in a more general environment, where the data is disseminated selectively to participating clients, and the topology is constructed according to implicit constraints imposed by each clients requirements and the data distribution.

Content-based multicast (CBM) in ad-hoc networks [19] describes a similar problem in wireless sensor networks, where the data needs to be multicast to a subset of the group based on its content. Sensors push data to neighboring nodes, and clients pull data from these nodes. Geographical regions are divided into blocks, and an election mechanism creates leader nodes for each block that are responsible for coordinating the local dissemination. In a CBM system, mobile node express interest in data that is a d distance away, or that will be available after time t (assuming a relative-velocity of the node towards the source). In contrast, CoDD uses general-purpose query languages that allow nodes to express more precise subscription interests. Further, CBM concentrates on wireless node mobility, and uses leader nodes to simulate a centralized infrastructure, while CoDD assumes a decentralized, connected environment that allows it to make more

general-purpose optimizations (like reorganization).

4.3 Filtering Systems

Filtering systems are designed to efficiently match data items with a set of predicates expressed in a query language specific to the data model. XFilter [1] uses an XML data model and encodes XPath queries as FSMs by mapping location steps in the expression to machine states. Arriving XML documents are parsed with an event-based (SAX) parser, and the events raised during parsing are used to drive the FSMs through their various transitions. A query is determined to match a document if during parsing an accepting state is reached in the corresponding FSM. YFilter [12] extends this idea for multiple queries, by merging common prefixes into a single NFA, which is processed once for multiple queries. This approach makes it possible to share processing costs between multiple queries, and reduces the number of machine states that need to be maintained. There has also been related work on maintaining index data structures for efficiently filtering data against a set of queries. In contrast with XFilter, XTrie [7] indexes on input substrings that contain parent-child operators rather than element names. This allows it to share the processing of common substrings among queries using this index, and allows it to decrease the number of index probes necessary and avoid redundant matchings. In our description of CoDD, we present policies to create topology structures that are independent of the data model being used. The filtering component in the system can be replaced by any of these techniques to enable fast filtering of data streams without any change to the topology management protocols.

Work on *Query Merging* seeks to reduce the cost of answering a set of subscription queries by grouping similar queries, such that the aggregated queries match a superset of the documents matched by each input query. Given a set Q of queries and a cost model for answering these queries, the task is that of determining a collection M of subsets of Q such that evaluating queries as suggested by M minimizes cost. Previous work has shown QM to be NP-Hard in the general case $(|Q| > 2)$, by reducing it to the set cover problem [10]. This work also quantifies the effect of merging for various cost models, and presents a set of heuristics for approximating the QM problem. While CoDD uses similar techniques to group queries, it does so without assuming a-priori knowledge of all nodes (and their respective queries) that will participate in the dissemination. Instead of relying on heuristics to try and optimize the query merging, it uses adaptive reorganization combined with data-independent statistics on query matches to maintain low overhead topologies.

4.4 Similar approaches in Pervasive Computing:

The decentralized services provided by CoDD have several characteristics that are similar in nature to the resource discovery problem in pervasive computing environments. For example, in the VIA system [4], domains organize into clusters based on their resources. Resources are described using a list of attribute-value pairs. Queries are specified as a set of constraints on a subset of these attributes. Clusters are built using upper-bound queries that are created by replacing some constraints by wildcard constraints. These upper-bound queries are then used to discover commonalities in metadata and to build a topology corresponding to these commonalities in a bottom-up fashion. VIA* [5] extends these techniques to allow queries to be generalized based on an impedance parameter. The query impedance is the average number of times a data attribute does not match a query attribute, and describes the relevant importance of that attribute in contributing to query matches. The extensional method for grouping nodes used by CoDD may be viewed as a generalization of this idea. The emphasis in CoDD is low-overhead protocols for dynamic environments that emphasize node autonomy. In VIA, groups of nodes are managed by an administrative domain controller. It would be interesting to explore combinations of these methods.

5 Conclusion

In this paper, we motivated the need for data management in interimistic environments. We discussed the characteristics of such environments and the design guidelines resulting from them. We presented our work on the problem of disseminating data in such environments using a server-less network of computers that is in a state of continual change. Our methods are based on simple, low-overhead network-formation and reorganization protocols. We presented a brief experimental study that suggests that our protocols cope well with changes to both the network (as host join and leave) and data.

In continuing work, we are conducting a more thorough experimental evaluation of our protocols. We also hope to extend our protocols to adapt to a wider range of changes. Further, we are exploring different formulations of the general problem of data management in an interimistic environment.

References

[1] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 53–64, 2000.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, Aug. 2002.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, Aug. 2001.

[4] P. Castro, B. Greenstein, R. Muntz, P. Kermani, C. Bisdikian, and M. Papadopouli. Locating application data across service discovery domains. In *Proceedings of the International Conference on Mobile Computing and Networking (MOBICOM)*, pages 28–42, 2001.

[5] P. Castro and R. Muntz. An adaptive approach to indexing pervasive data. In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*, 2001.

[6] C. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi. Tree pattern aggregation for scalable XML data dissemination. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Aug. 2002.

[7] C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 235–244, Feb. 2002.

[8] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 379–390, May 2000.

[9] J. Clark and S. DeRose. XML path language (XPath) version 1.0. W3C Recommendation. `http://www.w3.org/`, Nov. 1999.

[10] A. Crespo, O. Buyukkokten, and H. Garcia-Molina. Efficient query subscription processing in a multicast environment. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 83–83, Mar. 2000.

[11] S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems (TOCS)*, 8(2):85–110, 1990.

[12] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Transactions on Database Systems (TODS)*, 28(4):467–516, 2003.

[13] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2001.

[14] The Gryphon messaging system. `http://www.research.ibm.com/gryphon/`.

[15] Y. hua Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 1–12, Santa Clara, California, June 2000.

[16] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr.. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Symposium on Operating System Design and Implementation*, pages 197–212, Oct. 2000.

[17] B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the web. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 437–448. ACM Press, 2001.

[18] T. W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems (TODS)*, 25(4):529–565, 1999.

[19] H. Zhou and S. Singh. Content based multicast (CBM) in ad hoc networks. In *Proceedings of the 1st ACM International Symposium on Mobile ad hoc Networking and Computing*, pages 51–60. IEEE Press, Aug. 2000.