An Incremental Approach for Maintaining Up-to-Date Global Aggregates

Nabeel Ahmed, David Hadaller and Srinivasan Keshav
University of Waterloo
200 University Avenue West
Waterloo, Ontario N2L 3G1
{n3ahmed, dthadaller, keshav}@cs.uwaterloo.ca

Abstract

In massively distributed systems, having local access to global information is a key component for many applications. Examples include applications that aggregate sensor values, search in Peer-to-Peer systems, or perform Top-K queries in stream-oriented databases. Efficient computation of such aggregates is difficult due to massive scale and system dynamics and has led to the proposal of several approximate techniques such as randomized gossip-based algorithms. The focus of our work – $maintenance of freshness$ *of such aggregates – has not been adequately addressed in the literature. We motivate the need to adopt an incremental algorithm for maintaining these aggregates. In doing so, we make three broad contributions to this field. First, we identify the key components required for an incremental update algorithm. Second, we present solutions for each component by using: 1) Incremental routing techniques such as incremental random walk and incremental gossip, and 2) A variant of the well-known FM aggregation scheme that significantly reduces protocol overhead. Finally, we present a detailed performance evaluation of our techniques, and find that we can achieve a reduction of as much as 60% in computation time compared to existing methods.*

1. Introduction

Large decentralized and self-organizing networks are increasingly becoming more prevalent. In such systems, there is a need to have local access to global information, such as a global aggregate [14]. For example:

- In a sensor network, one may want to compute the average sensor values; extremal sensor value, such as min or max; or quantile values, such as the median.
- In Peer-to-Peer (P2P) systems, it is often necessary to make local decisions based on global information. Examples include: choosing how to search for a document, choosing when to replicate a document in a replicated file system, and performing ranking in a full-text search system. PlanetP [4] is an example of the latter, which uses a gossip-based protocol to provide local access to vital ranking aggregates such as term frequencies.
- In a distributed database that records the number of hits to an item in a content distribution network, the set of K documents that have the most hits (top-k query) may be of interest. Having local access to this global aggregate allows for load balancing of popular documents.

The differing dynamics in each of these systems present different challenges for maintaining global aggregates. To bring some structure to the problem, we categorize systems based on their rate of change (or *network churn*).

1. **Stable:** No data or membership changes occur, therefore a global aggregate, once computed, does not need updating.
2. **Semi-stable:** Data changes and nodes joining and leaving occurs relatively infrequently. In this case, incremental updates are more efficient than performing a complete re-computation.
3. **Chaotic:** Data and node changes are frequent, thus incrementally updating a global aggregate may be less efficient than re-computing the aggregate on demand.

The value of a global aggregate changes not at all in stable systems, and too often to be incrementally maintained in chaotic systems. Thus, our focus is on $incremental$ algorithms to maintain global aggregates in semi-stable networks. This is a challenging problem due to the dynamic nature of the system and the need for cost-efficiency.

A key contribution of our work stems from the following intuition. Suppose some global aggregate has been computed in a decentralized fashion and then a change occurs at some node in the system.

1. If the change in the system is insignificant one can, perhaps, get away with doing nothing.
2. If the change is significant but less than some threshold β, then an incremental *update algorithm* should be executed.
3. If the change is extreme (i.e. greater than β), a recomputation over the entire network should be performed.

This intuition allows us to decide if an incremental update is needed at all. Once we identify the need for an incremental update, it turns out that it is more efficient to simply modify the existing global aggregate, rather than recompute the global aggregate. Moreover, it is also much more efficient to use a modified routing protocol to disseminate the incremental updates rather than use a standard message routing protocol. The three main contributions of our work are, therefore, (a) the use of a threshold to initiate incremental updates (b) the design of an incremental version of a global aggregate and (c) the design of incremental approaches to disseminating these incremental updates.

We elaborate these ideas as follows. We first identify the elements required for an incremental update algorithm: update threshold, aggregation mechanism, and routing protocol (Section 2). We then describe our incremental techniques in Section 3. We show the correctness of our approach by means of a formal analysis in Section 3.4. Finally, we perform a detailed evaluation of the performance gains achieved using our incremental update algorithm, and explore a four-way tradeoff between accuracy, convergence time, cost, and robustness in Section 4.

2. Background and Related Work

In this section, we briefly discuss the background related to the techniques we present in this paper. In Section 2.1, we present the formal model that we use later to describe our approach. In Section 2.2, we present previous work that has motivated the need for maintaining global aggregates. Section 2.3 presents routing techniques that we explore for our incremental protocols. Finally, section 2.4 describes the aggregation mechanism that we enhance in our work and section 2.5 formally describes it in light of the model described in section 2.1.

2.1. Model

In this section, we briefly outline the theoretical framework that we adopt in order to describe our aggregates. We also present a proof of how it is theoretically impossible to compute *exact* global aggregates in a dynamic system, thus motivating the need to compute approximate global aggregates in our algorithms.

Consider a distributed system with N nodes where, at time t, the i^{th} node has local information s_i^t and knows the IDs of its neighbors (i.e. nbr_i^t). In many problems of interest, N is very large and nodes arrive and depart over time. Moreover, communications between nodes may be lost.

Our goal is to have the nodes self-organize to compute a function $f(S^t)$ where S^t is a *global multi-set* defined as $S^t = \{s_1^t, s_2^t, ..., s_i^t\}$; s_i^t is a local set at node i that constitutes a part of the global multi-set S^t. f can be any aggregation query. Using this framework, in later sections, we describe the problem of computing and maintaining global aggregates for this multi-set.

Theorem 1. *Due to unreliable nodes and links, although $f(S^t)$ is well-defined, it may not be computable.*

Proof. Consider a two-node system where node 1 and node 2 collaborate to compute some global aggregate. Without loss of generality, suppose node 1 is chosen to act as a coordinator and wishes to compute the global aggregate at time t. If node 2's value changes just before this time, the communication of this changed value from node 2 to node 1 at time t is lost, and immediately after communication, node 2 also dies, then there is no way for $f(S^t)$ to be computed. Incidentally, if any one of these three conditions does not hold, then f can be correctly computed. This simple result shows that, in general, f is not computable. □

When N is large, it may be possible to compute some functions f over a sufficiently large fraction of nodes such that the result computed on this fraction approximates its true value. In the rest of this paper, we will only consider such approximate computations of global aggregates.

2.2. Global Aggregate Maintenance

Computing global aggregates is very similar to maintaining global consistency in a traditional distributed system where protocols such as the Chandy-Lamport distributed snapshot protocol are employed [2]. However, such protocols are limited in applicability to small-scale distributed environments and were not designed to massively scale as required on the Internet today nor to handle challenges found in other fields such as sensor databases. As a result, gossip-based protocols have emerged as a lightweight and robust mechanism for computing such aggregates.

Gossip-based (or epidemic) protocols perform exceptionally well for computing global aggregates [12, 3, 10, 9]. They work as follows: At every time step, each node selects one or a few nodes with which to exchange data. The dynamics of how information spreads through the network resembles the spread of an epidemic [1], which provides increased fault-tolerance [5] and resilience to failures. However, due to randomness, only probabilistic bounds on convergence are obtained.

More recently, gossip-based protocols have been shown to provide time complexities of the order of $O(\log n)$, where *n* is the number of participants in the network. Recent efforts by [12, 3, 10] have also developed sophisticated techniques that reduce the state space to within an $O(\log n)$ bound of the network size. These protocols are, therefore, extremely scalable and robust and a large body of theoretical work provably confirms these properties [10, 7]. There has also been an interest in studying the effect of system dynamics on the performance of these protocols. Although some theoretical work provides encouraging results, they do not consider incremental updates. Jelasity et al [9] employ an automatic restart mechanism where they restart the protocol on a periodic basis by dropping the current estimate of the global aggregate and rerunning the protocol from scratch. Such drastic measures are not necessary if the changes in the global aggregate are minimal. This motivates the need to build more efficient mechanisms to disseminate updates. Our work provides a first step in this direction.

In order to address the problem of global state maintenance, it is important to understand the components which make up a gossip-based protocol. We adopt the logical decoupling of routing (or $Routing$ mechanisms and aggregation proposed by Nath et al. [12], which we discuss next.

2.3. Routing Mechanisms

Here we discuss different routing mechanisms used in our work. We remind the reader that routing mechanisms for gossip-based protocols typically function in the form of rounds. In each round, one or more node(s) communicate with one or more other node(s) in the system. In this way, information eventually reaches all participant nodes in the system.

- **Flooding**: Flooding involves sending data in an epidemic fashion, where the source node sends a message to all of its neighbours, who then forward previously unseen messages to all of their neighbours and so on. This process repeats until either all nodes in the network have received the data or a certain distance from the initiating point (flood depth) is reached. Flood provides near optimal convergence properties but however comes at the cost of excessive message transmission overhead.
- **Uniform Gossip**: In uniform gossip, during each round, each node selects one neighbour with whom to exchange information. This process repeats and information eventually propagates through the entire network. Uniform Gossip protocols have many useful theoretical properties and have been shown to provide exponentially fast convergence with low message transmission overhead [10].
- **Random Walk**: A random walk is initiated by a node sending its data to a random neighbour. During each round, when a random walk (or package) arrives at a node, that node adds its own data to the package and passes it on to another random neighbour. This continues until all data is sent throughout the network. Multiple random walks can be used simultaneously to improve performance. Random walk provides a greater deal of flexibility than uniform gossip by allowing a tuning mechanism between convergence time and message transmission overhead, as we show later.

2.4. Aggregation Mechanisms

Aggregation refers to a technique where participants (or nodes) in a distributed system combine their local values (collected via a routing mechanism) into a global aggregate of the system, e.g. the Top-K documents in a database. A specific problem when computing aggregates is *double-counting*, where nodes may contribute to an aggregate more than once, causing inaccuracy in the final result. In order to avoid such problems, Order and Duplicate Insensitive (ODI) mechanisms can be used, such as was proposed by Nath et al. [12]. Order and Duplicate Sensitive (ODS) approaches can also be employed but require additional state to be maintained in order to prevent double counting.

In [12], order and duplicate insensitivity is achieved through the use of the approximate FM counting algorithm, originally pioneered by Flajolet and Martin [6]. Other ODI techniques include a push synopsis approach proposed by [10]. However, due to the compact nature of FM, we restrict our attention to this mechanism for the remainder of this paper.

In FM, each node maintains the aggregate of the entire network of *n* nodes in a bit vector of approximately $\log n$ size. It should be noted, however, that the FM counting algorithm has loose error bounds. We discuss in Section 2.6 how FM may be improved by slightly increasing the state maintained at each of the nodes.

2.5. FM Aggregates

We now formally describe FM aggregates. We discuss how to extend this framework in order to support updates to these aggregates in Section 3.1.

Consider a multi-set $S^t = \{s_1^t, s_2^t, ..., s_x^t\}$ with x elements as described in Section 2.1. We would like to compute $f(S^t)$, where f is the *count* function. In FM, the multi-set S^t is represented as a bit vector of some length k (typically $k = \frac{3}{2}\log_2(n)$). Initially the bit vector (denoted A) is set to zero. Then, with the help of a random binary function $cointoss$, $count$ is computed as follows:

Algorithm 1 ItemInsert(A)

```
1: k = 0;
2: while cointoss() = 0 do
3:     k = k + 1;
4: end while
5: A[k] = 1;
```

Algorithm 2 FM Counting Algorithm (S)

```
1: $R = 0$;
2: for $i = 1$ to $x$ do
3:     ItemInsert(A[i], $s_i$)
4:     $R = R \bigvee A[i]$;
5: end for
```

Once the coin tossing experiments are complete, the resulting bit vector is used to compute the actual value of $count$. Functions other than count can also be computed using the basic FM counting algorithm described above and are discussed in more detail in [12, 3]. The FM counting algorithm described above has some interesting properties that allows the algorithm to be deployed in a distributed fashion. In particular, FM has the following interesting features:

- **Constant Insertion time**: Each element (or contribution) of the multi-set can be inserted into the vector in $O(1)$ time.
- **Set Unions**: The union of FM bit vectors representing two multi-sets S_1 and S_2 is simply the bitwise OR of their respective bit vectors.
- **Duplicate Insensitive**: FM bit vectors are duplicate insensitive. Therefore, applying any duplicate values to the FM bit vectors thus not affect the correctness of the aggregate.

Using these three features, we can compute an aggregate in a distributed fashion by having each node maintain an individual set that is part of the *global* multi-set. Local sets are also maintained as bit vectors and these bit vectors are propagated in the network to other nodes that union them with their local bit vectors (a.k.a fusing). Using this approach, and given that the bit vectors are duplicate insensitive, we can employ any randomized routing mechanism (described in Section 2.3) in order to disseminate local bit vectors and eventually compute the global aggregate of the entire network.

2.6. Improving Accuracy of FM Aggregates

As originally proposed in [6], the accuracy of FM can be improved by maintaining a larger bit vector representation. This is achieved by maintaining multiple bitmaps. Considering the original multi-set S, each item s_i can be inserted in multiple bit vectors to produce k values that can then be averaged to produce the resulting aggregate. More specifically, the aggregate f is computed as follows:

$$f = (1/m)(1/0.77351)\Sigma_i 2^{s_i}$$

where,
s_i= Bit position of rightmost zero in i^{th} vector

The resulting aggregate is more accurate, and the standard error decreases by $O(1/\sqrt{m})$, where m is the number of vectors. However, this technique requires more insertion time since m insertions are required in order to add an item into the aggregate. This can be reduced by using a more efficient PCSA technique that is discussed in more detail in [6]. With PCSA, the standard error is approximately $0.78/\sqrt{m}$. In our study, since we are primarily interested in comparing the performance of our incremental protocols with the drop and recompute schemes, in order to limit the impact of PCSA[1] on our results, we employ the m insertions technique discussed earlier.

3. Incremental Algorithms for Global Aggregate Maintenance

In this section, we present our incremental algorithms for maintaining up-to-date global aggregates. We first outline some basic requirements that the algorithms must satisfy. We then present update enhancements to FM counting to allow efficient aggregation and present incremental routing protocols to support such updates. Ideally, incremental algorithms should have:

- **Fast Convergence** The algorithm must provide convergence times comparable to or better than existing methods (e.g. $O(logN)$ for gossip-based protocols).
- **Cost-Effectiveness** The algorithm must provide significant cost advantages (e.g. computation time, communication time) over existing methods.
- **Fault-Tolerance** The algorithm must, in the least, exhibit fault-tolerance capabilities of existing solutions.

1 The bias of PCSA is significant till about 32 vectors and needs to be corrected for smaller numbers of vectors

- **Scalability** The algorithm must provide the same guarantees on both small as well as large-scale systems.
- **Accuracy** The algorithm must provide an acceptable amount of accuracy compared to existing methods.

3.1. Incremental FM Aggregates

In this section, we present our first contribution by discussing how changes may be applied to a previously computed FM aggregate. We assume that our function f is the *sum* over the multi-set S. To compute the function f, we follow the approach discussed in [3], where for any given item s_i in S whose value is given by v, we perform v coin toss experiments and union the bits to produce the resulting bit vector for that value. We do the same for all other items in S and obtain the aggregate vector (called $sum(S)$) that represents the global sum of the items in S. As one may expect, this approach does not scale for very large values of v and can be improved in which case we can adopt the scalable version discussed in [3]. Using this method, we are able to efficiently compute the sum over the multi-set S.

Now suppose that a value of any given item s_i in the multi-set changes from v to $\acute{v}$. How do we incorporate the new value into the already computed $sum(S)$? Suppose an item s_i's initial value v sets the first 3 bits and it unions them to the bit vector representing the aggregate $sum(S)$. Later, when its value changes to $\acute{v}$, it may not be able determine what bits it contributed to the vector. Moreover, even if it did, since the first 3 bits may possibly have been set by one or more other items in S as well, these cannot be legitimately reset by s_i. Therefore, intuitively, it seems that we may need to drop and recompute the entire sum again. However, as we discuss further, this may not be needed.

Two cases need to be considered for the new value $\acute{v}$, i.e. $\acute{v} > v$ and $\acute{v} < v$. For $\acute{v} > v$, we can simply do $(\acute{v} - v)$ more coin toss experiments in order to incorporate the new larger value $\acute{v}$ in the aggregate $sum(S)$. For $\acute{v} < v$, suppose instead that we construct another multi-set D that is defined as follows:

$$\begin{aligned} D = \{d | d = v_i - \acute{v_i}, &\text{ where } v_i = \text{old value} \\ &\acute{v_i} = \text{new value at node } i\} \end{aligned}$$

Using this multi-set D, we can now maintain the difference $v - \acute{v}$ for multiple updated items in S. Further, we can apply the same sum function for summing up the values of all the items in D to generate the aggregate $sum(D)$. Once both $sum(S)$ and $sum(D)$ are computed, we can apply the FM evaluation function discussed in [6] in order to obtain the sum values for both sets S and D. Using these two values, we then compute the new aggregate by subtracting the sum value of S from the sum value of D to produce the new sum of the network, after the updates were applied. Therefore, this technique can be used in order to support updates to FM aggregates.

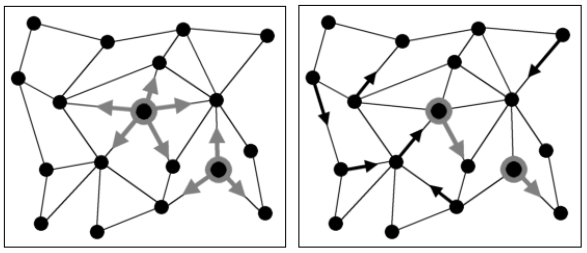

Figure 1. The left diagram illustrates our IRWP protocol while the right diagram illustrates the standard random walk protocol.

Although the update techniques described above allow additions/deletions to already computed aggregates, note that FM can only count in steps of powers of two. As the count gets large, FM can only represent coarse-grained information, as for every extra bit that is set, the count doubles. Moreover, when a large amount of contributions have been made to the bit vectors, over time the vectors may become *polluted*. In this situation, we advocate dropping and recomputing the aggregate. The decision of when to drop and recompute could be based on an update-specific threshold or through the use of some other mechanism to identify anomalies in the stored aggregate. For example, when computing count, if the *delete* vector becomes larger than the *add* vector, a negative aggregate results and this can signal a re-computation of the aggregate. However, this problem is an open area of research.

3.2. Incremental Routing Protocols

In this section, we discuss extensions to protocols described in Section 2.3 for our incremental algorithm. We propose the following incremental routing techniques.

- **Incremental Flooding Protocol (IFP)**: Traditional flooding works by having every node flood its data to the entire network. We propose an incremental version, where, if an update occurs at a node, only that node floods an update message throughout the network.
- **Incremental Gossip Protocol(IGP)**: In our incremental approach, nodes gossip with each other only if they have received the update. Nodes receiving the update become *active* and begin communicating with their neighbours until the update protocol terminates. This approach reduces wasteful communications between

Algorithm 3 Update Algorithm: Phase One

```
1: for each updated node do
2:     Update local aggregate $G_t$
3:     $\Delta G = G_t - G_{t-1}$
4:     if $\beta > \Delta G >= \alpha$ then
5:         Send update to neighbour node(s)
6:     else if $\Delta G > \beta$ then
7:         Drop local aggregate and send re-computation command to all nodes
8:     else
9:         Do nothing
10:    end if
11: end for
```

Algorithm 4 Update Algorithm: Phase Two

```
1: if received update $A$ at node $n$ then
2:     Union update $A$ with local aggregate $G_t$.
3:     $\Delta G = G_t - G_{t-1}$
4:     if $\Delta G >= \alpha$ then
5:         Send update to neighbour node(s)
6:     else
7:         Do nothing
8:     end if
9: end if
```

neighbours that do not have updates to contribute to the aggregate being maintained at the nodes.

- **Incremental Random Walk Protocol (IRWP)**: In this protocol, when an update occurs, instead of starting random walks at random nodes in the network, all the random walks are initiated from the update point, and contain update-specific contents (as shown in Figure 1. This can be viewed as a node "sending out" packages to its neighbours containing its update. Nodes receiving the package update their local aggregate and if needed, the aggregate contained in the package. They then perform the standard random walk protocol as described in Section 2.3, until the network has converged. For multiple updates, multiple random walks will be initiated from different points in the network. Deciding how to allocate random walks to each update in a decentralized fashion is a hard problem.

  Population control (first proposed in [15]) provides intuitions on how to control the assignment of random walks (RWs) to each update. Such a mechanism operates in a completely decentralized and self-adaptive manner. Using this technique, each node increases/decreases the RWs it assigns to its updates based on feedback that it receives over time. Moreover, what mechanisms to exactly use to achieve this is an important and open problem.

3.3. Incremental Update Algorithm

The update algorithm we propose combines the aggregation mechanism described in Section 3.1 with the incremental routing protocols presented in Section 3.2. The protocol proceeds in the form of *rounds* where in each round, a single communication process takes place between pairs of nodes. The time to converge to the correct global aggregate is referred to as an *epoch*.

The update algorithm proceeds in two phases. In the initiation phase (Algorithm 3), at the beginning of an epoch (at time t), when an update occurs at a particular node n, n uses the updatable FM approach discussed in Section 3.1 in order to generate the new aggregate that incorporates the update. Using this newly generated aggregate G_t, n computes the change in the locally maintained aggregate by computing $\Delta G = |G_t - G_{t-1}|$. If ΔG is greater than or equal to a *change threshold* α, n propagates the change to its neighbors using an incremental routing protocol. However, if ΔG is greater than a *recompute threshold* β, the aggregate is dropped and a re-computation is initiated on all of the nodes (e.g. by flooding a restart message). If ΔG is strictly less than α, then n does nothing as the change is not significant.

In the processing phase (Algorithm 4), the algorithm propagates updates throughout the network. When an update is received at a node, the node unions it with the local aggregate and propagates the result to its neighbours. Due to the nature of FM, propagated aggregates may contain multiple updates as they proceed through the network. This allows faster dissemination in the network and such update propagations can be viewed as *co-operative*.

The algorithm terminates when there is no update that causes ΔG at any of the nodes to be greater than the change threshold α. As discussed next, this is a coarse-grained way of deciding when to terminate the protocol.

3.3.1. Update Thresholds The update thresholds function as tuning knobs allowing the system designer to trade off the accuracy and cost of the incremental update algorithm. Thresholds should be chosen based on global significance which is defined as the significance of a $local$ change to the global aggregate.

Significance should also depend on the query type. For example, if we are computing the sum across a very large network (e.g. 100k nodes), updates that cause very large changes in the sum (e.g. updates on 50K nodes) should signal a re-computation of the aggregate. In this case, β could be equal to x, where x is the percentage of error tolerated for the aggregate. On the other hand, if we are computing max, then β may be set to infinity since no change in the system will be significant enough to require re-computation. Updates need only be sent for changes to the maximum value. Therefore, the selection of thresholds depends on the query type. Determining the global significance of a local change as a function of the query type is difficult. For queries such as sum and max, the choice is relatively straight forward. However, the general problem is hard.

3.4. Analysis

In this section, we present a proof of correctness of our update algorithm. We gain additional insight through simulations in Section 4.2.

3.4.1. Correctness of Update-Algorithm We now present a simple proof to support the methods used by our update algorithm.

Lemma 1. *Once updated to a new value, a node cannot revert back to an old value.*

Proof. By construction. □

Theorem 2. *Given that an update is applied to any node in the network and the network does not experience a partition, all nodes eventually receive the update.*

Proof. We construct an inductive argument to prove the theorem. Starting at the updated node, since it is the node that initiates the update, clearly it has the new state. Further, every message from this node to any of its randomly chosen neighbours in all subsequent rounds causes the neighbours to receive and apply the update. In the same way, the neighbours also propagate the update to their randomly chosen neighbours. Since, by Lemma 1, an updated node does not revert, eventually all nodes in the network receive the update. □

One way to visualize update propagation is to think of all nodes being assigned a starting color of blue. A node that is updated turns from blue to green. Further, its message to a blue node causes that node to change its color to green. Since a green node can never turn to blue, as long as all nodes are touched, eventually every blue node will turn green.

4. Simulations

In this section, we provide detailed simulations to analyze the performance of our protocols. In Section 4.1, we outline the simulation setup and the performance metrics. We then discuss the simulation results in Section 4.2.

4.1. Simulation Setup

We have implemented a compact simulator in C. Since we are interested only in analyzing the behavior of our incremental techniques, we set the α threshold to 0 and β threshold to ∞. We use the sum query for our simulations and only model data changes in the system. All participants in the system maintain identical configurations and are equal in their capabilities. Until otherwise indicated, the topology considered for our simulations is a clique.

Component	Accuracy	Protocol Cost	Delay	Robustness
Incremental Routing	–	✓	✓	✓
Incremental Aggregation	✓	✓	–	–
Update Threshold	✓	✓	✓	–

Table 1. Effect of protocol components on performance metrics

4.1.1. Assumptions Here we briefly list the assumptions made in our simulations:

- **Non-malicious Peers:** We assume that peers cooperate with each other and do not behave maliciously. Existing research has already addressed such issues [8].
- **Uniform Data Distribution:** We assume data changes are uniformly distributed across the entire system. We do not consider *hotspot* changes within the system.
- **Batched Data Changes:** In our study, data changes and the running of the incremental update algorithm occur sequentially. We use this model for ease of simulation.

All nodes are synchronized and our protocol proceeds in the form of *rounds*. This assumption is made for ease of simulation and does not affect the correctness of our incremental update algorithm.

4.1.2. Simulation Metrics Here we briefly list the performance metrics that we use in our simulations. Each of our three contributions affect multiple metrics as shown in Table 1. They are briefly discussed further:

Accuracy. We use the mean error in the local estimate of the global aggregate to measure the accuracy of each of the schemes. The mean error is defined as $e_t = \frac{1}{N}\frac{1}{m_t}\sum|\hat{m_t} - m_t|$, where $\hat{m_t}$ is the local estimate of the aggregate, m_t is the true value of the aggregate, and N is the total number of nodes in the network.

Protocol Cost. The cost of a protocol has three components: computation time, communication time, and the state maintenance overhead. In this study, we only consider the communication time (that includes message transmission overhead and message size) and the state maintenance overhead. Moreover, since message size and state maintenance overhead are fixed for our incremental FM aggregates, we analyze only the message transmission overhead.

Delay. The delay metric is defined as the time taken to converge to the approximate global aggregate from the time the update occurred. Therefore, the equation for delay (d) is $d = t_{det} + t_{conv}$, where t_{det} is the time to detect the update and t_{conv} is the time to converge to the approximate global aggregate. We assume that t_{det} is zero and only consider t_{conv} in our study. Convergence time is measured in terms of the number of rounds, where each round represents a fixed time interval[2].

Robustness. The robustness of each of the schemes is characterized by the protocol's resilience to node/link failures and in the system. Since our change model only considers changes to the data at individual nodes, we defer simulating such changes as part of future work.

In our simulations, we report the average measure over 500 epochs. We begin our simulations with a *stable* network, i.e. where the global aggregate has converged. During the update process, we introduce updates uniformly at random in the network. Unless otherwise indicated, we experiment with 10,000 nodes in order to analyze the scalability of our protocols; for the random walk and IRWP protocols, we utilize as many random walks as there are nodes in the system. We show in Section 4.2.4 that this does not significantly increase the overhead of the IRWP protocol.

4.2. Simulation Results

First, we present results comparing our incremental update algorithm with existing re-computation schemes. Second, we provide a comparison between our incremental routing protocols, followed by an analysis of the effect of increasing the number of random walks (RWs) on IRWP. Third, we present results quantifying the accuracy of incremental FM aggregation and finally end with a sensitivity analysis of our incremental routing protocols.

4.2.1. Incremental Updates vs. Re-computation

- *Convergence time* Figure 2 compares the performance of our IGP protocol with the standard uniform gossip protocol. We observe that convergence rates overlap for varying numbers of updates. We observe this because when an update occurs, the updated nodes delete vectors need to be propagated to the entire network and this would have the same convergence rates as a re-computation of the aggregate. Therefore, IGP does not provide significant improvements in convergence time. Figure 4 compares the convergence times of IRWP and standard random walk. IRWP reduces convergence time by as much as 60% for small number of updates. We conjecture that since IRWP minimizes the number of wasteful messages per round that do not propagate updates, it is able to converge significantly faster. Due to space limitations, we do not present results for flood and incremental flood. However, we indicate that flood and IFP perform identically.

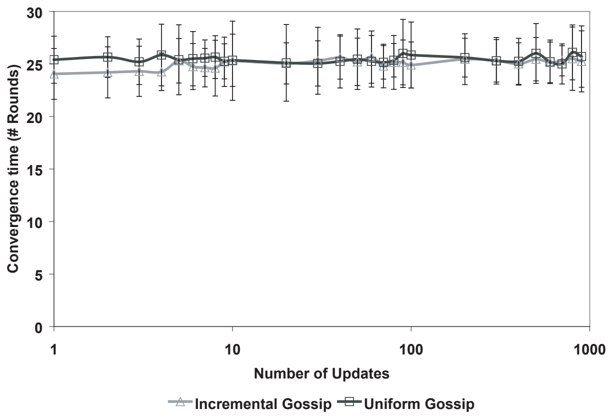

Figure 2. **A Comparison of convergence times of IGP and Uniform Gossip protocols.**

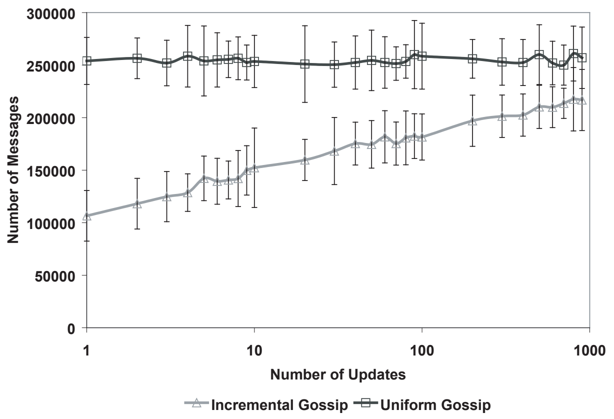

Figure 3. **A Comparison of message transmission overhead of IGP and Uniform Gossip protocols.**

- *Protocol Cost* Figure 3 compares the message transmission overhead of IGP with uniform gossip. For a small number of updates, we see an almost 60% decrease in the number of messages required for convergence. Since IGP on average requires significantly fewer nodes to gossip in the first few rounds, the total message overhead drops substantially. However, as the number of updates are increased, more nodes communicate in earlier rounds causing an increase in the number of messages. If we compare IRWP with standard random walk in Figure 5, we see similar trends that indicate the benefits of propagating walks from the up-

2 The actual length of the round depends on the underlying transmission medium and is not explicitly considered in our simulations

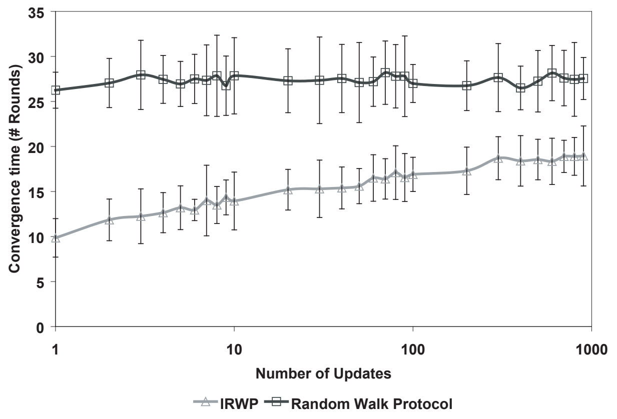

Figure 4. **A Comparison of convergence times of IRWP and Random Walk protocols.**

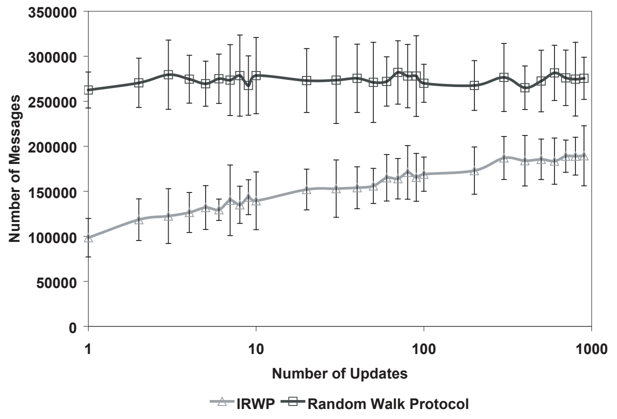

Figure 5. **A Comparison of message transmission overhead of IRWP and Random Walk protocols.**

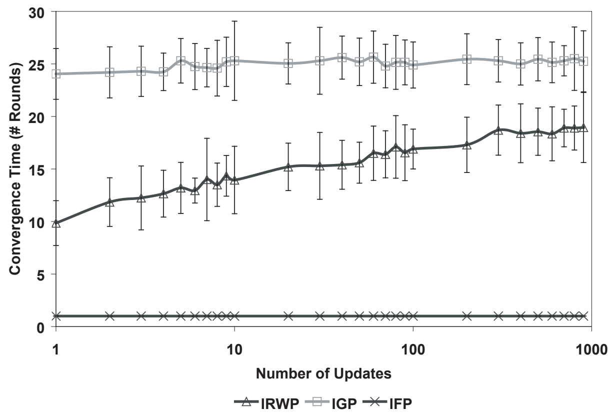

Figure 6. **A Comparison of convergence times of incremental routing protocols.**

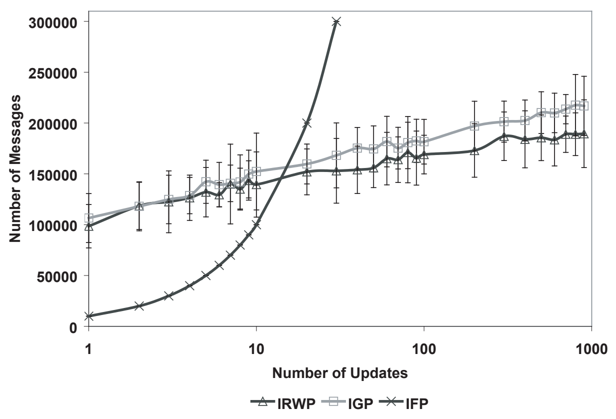

Figure 7. **A Comparison of message transmission overhead of incremental routing protocols.**

dated node(s), thus minimizing the amount of wasted messages. Although not presented here, the IFP and traditional flood variation is much more marked for small numbers of updates but exponentially decreases with larger numbers of updates.

4.2.2. Comparison of Incremental Routing Protocols

- *Convergence time* Figure 6 compares the convergence times of our incremental routing protocols. We see that IFP performs the best since it simply floods the update in the first round to all nodes in the clique. IRWP converges faster than IGP since IRWP performs more useful transmissions per round (based on the number of walks originating from the updated node(s)) than IGP.

- *Protocol Cost* Figure 7 compares the message transmission overhead of each of the incremental routing protocols. Clearly, incremental flood performs the worst since update messages increase exponentially with an increasing number of updates. Once again, IRWP is, on average, more cost-effective than IGP since it utilizes its communication resources (i.e. walks) more effectively than IGP. Since IGP causes more wasteful communications in later rounds, the benefits of only a few nodes gossiping in initial rounds is off-set by an increase in gossiping later. Therefore, due to the effectiveness of IRWP in both convergence time and cost dimensions, we analyze this protocol more closely in section 4.2.4.

4.2.3. Accuracy of Incremental FM Figure 8 compares the accuracy of incremental FM aggregation with standard FM aggregation, where the aggregate is dropped and recomputed each time an update occurs. Incremental FM is able to consistently maintain better accuracy for small numbers

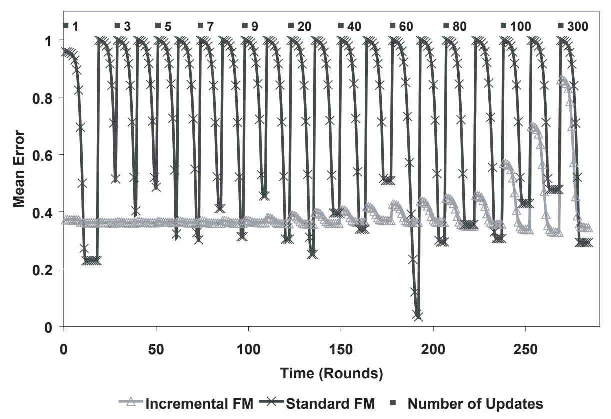

Figure 8. Accuracy of standard FM and incremental FM as updates are made in the system; the points at the top of the graph indicate the number of updates introduced into the system at that point in time. Standard FM drops and recomputes the aggregate when updates are introduced.

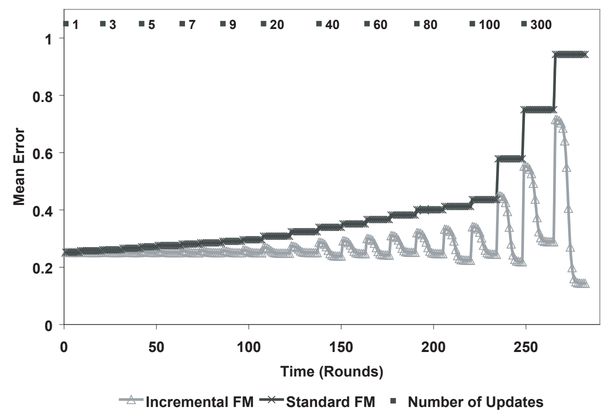

Figure 9. Accuracy of standard FM and incremental FM as updates are made in the system; standard FM does not drop the aggregate in this case.

of updates. On the other hand, standard FM's accuracy continually changes and thus on average is higher than incremental FM. Upon further analysis, Figure 9 presents a similar graph where standard FM does not drop the aggregate but maintains it over time as updates are applied to the system. The noticeable gap between incremental FM and standard FM is due to the added accuracy achieved with the use of delete vectors[3]. Incremental FM is able to make updates to the aggregate and minimize the error where as for

3 Delete vectors are able to maintain a higher resolution since when combined with the add vector, they can represent values that are not representable by using just add vectors themselves

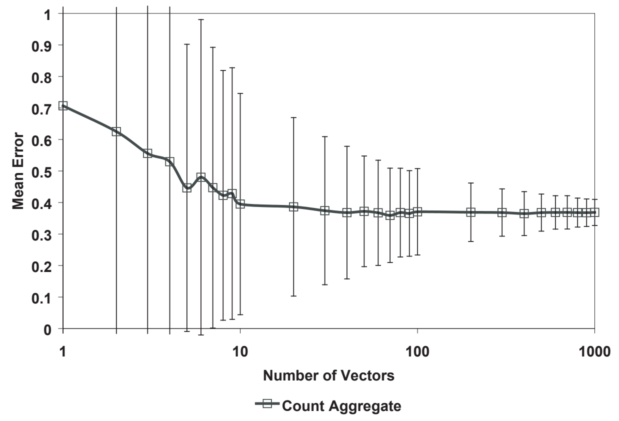

Figure 10. Accuracy of standard FM when computing the count in a 10,000 node network. Increasing the number of vectors corresponds to increasing the message size.

standard FM, the error compounds for an increasing number of updates. Therefore, higher accuracy is achieved with the use of delete vectors. Moreover, Figure 10 shows that for increasing numbers of bit vectors, a higher amount of accuracy can be maintained as was discussed in Section 2.6.

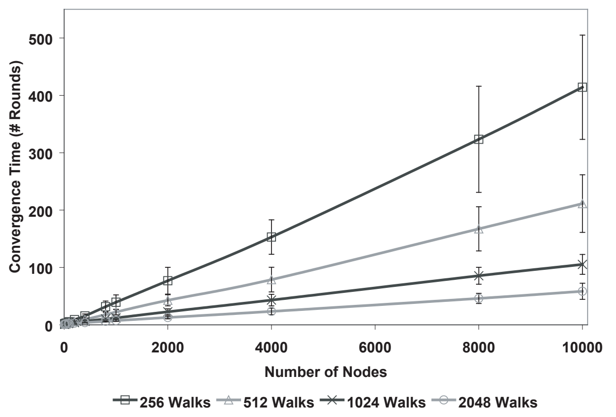

Figure 11. Analysis of differing number of walks on IRWP convergence time.

4.2.4. Effect of Multiple Random Walks

- *Convergence time* Figure 11 illustrates the effect of increasing the number of random walks for IRWP on convergence time. We observe that in almost all cases, for a small number of walks (e.g. 32), doubling the number of walks results in approximately a 50% decrease in the convergence time. This decrease eventually diminishes as the number of walks are increased,

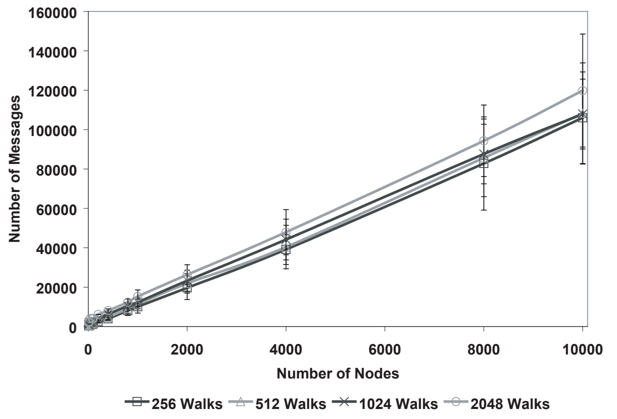

Figure 12. Analysis of differing number of walks on IRWP message transmission overhead.

since the network eventually becomes saturated. Nevertheless, the initial improvements in convergence time are encouraging.

- *Protocol Cost* In Figure 12, which illustrates the effect of increasing the number of random walks on message transmission overhead for IRWP, we surprisingly see that the message transmission overhead is effectively independent of the number random walks. We attribute this to the *co-operative* nature of the updates. Since each of the walks is able to carry information for the others, increasing the number of random walks increases the number of messages per round but decreases the overall number of rounds (as seen in the convergence time). Therefore, as a result the message transmission overhead does not change.

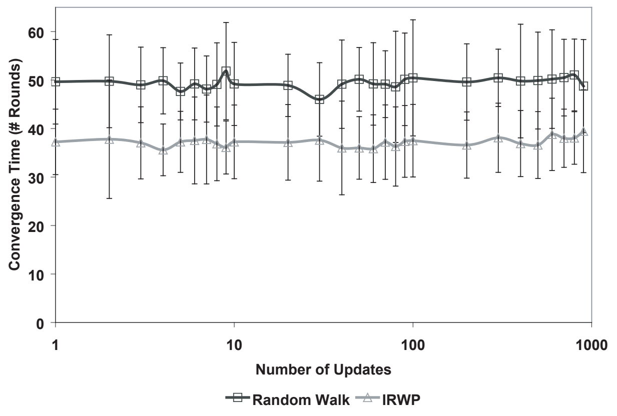

Figure 13. IRWP and Random Walk Convergence times on PLRG topology.

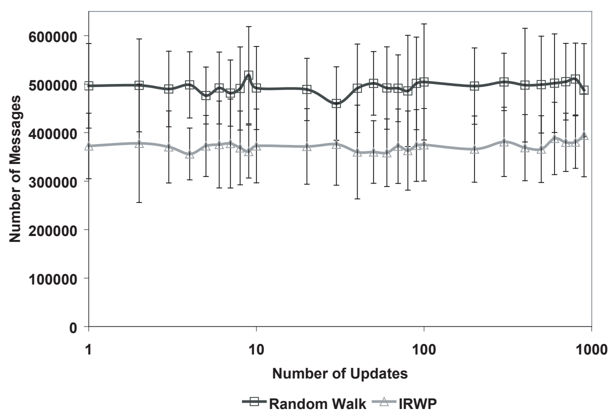

Figure 14. IRWP and Random Walk message transmission overhead on PLRG topology.

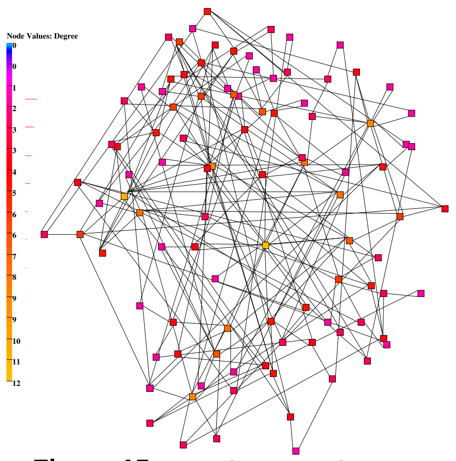

Figure 15. Sample 100 node PLRG topology, generated using BRITE. The line graph on the left represents the frequency of node degrees.

4.2.5. Effect of Complex Topologies We now provide a sensitivity analysis of our incremental routing protocols by presenting results on more complex topologies. We make use of a BRITE-based [11] power-law random graph (PLRG). An example of a PLRG for a network of size 100 is shown in Figure 15. Due to space limitations, we only present results comparing random walk and IRWP protocols. Results for other protocols follow similar trends.

Figures 13 and 14 present a comparison of the convergence times and message transmission overheads for IRWP and standard random walk on a 10,000 node BRITE-based PLRG topology. As can be seen, IRWP still performs significantly better both in terms of convergence time and message overhead than standard random walk. This indicates that the protocol is insensitive to the underlying topology.

5. Conclusions and Future Work

Computing global aggregates is an important problem that has been well studied in traditional distributed database literature [13]. Incrementally maintaining such aggregates has also been well studied in this context, particularly for incremental view maintenance of materialized views in distributed databases. However, this work relies on the stability and reliability found in traditional distributed systems (often relying on a central server) [13], which precludes its use on today's large-scale decentralized networks.

Our work takes a first step in addressing the problem of maintaining global aggregates in dynamic decentralized networks, an area with many open problems. Our contributions are: 1) We introduce new incremental routing protocols such incremental random walk and incremental gossip and show through simulations that these protocols outperform existing protocols by as much as 60%, 2) We present an incremental version of FM that we show maintains higher amounts of accuracy than standard FM aggregation and 3) We highlight the important elements required for an incremental algorithm that allows maintenance of aggregates. To the best of our knowledge, this is the first serious treatment of maintaining up-to-date aggregates in dynamic decentralized systems. Our ongoing efforts include evaluating other aspects of the problem space including a more elaborate change model that incorporates node and link failures, an examination of protocol performance using more sophisticated aggregate queries, and a detailed theoretical framework for our methods.

References

[1] N. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Hafner Press, 1975.

[2] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 449, Washington, DC, USA, 2004. IEEE Computer Society.

[4] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, pages 236–246. IEEE Press, June 2003.

[5] A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *SIGOPS Oper. Syst. Rev.*, 22(1):8–32, 1988.

[6] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[7] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of INFOCOMM 2004*, March 2004.

[8] M. Jelasity, A. Montresor, and O. Babaoglu. Towards secure epidemics: Detection and removal of malicious peers in epidemic-style protocols. Technical Report UBLCS-2003-14, University of Bologna, Nov. 2003.

[9] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Journal*, 15(5), November 2004.

[10] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 482, Washington, DC, USA, 2003. IEEE Computer Society.

[11] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: an approach to universal toplogy generation. In *Proceedings of MASCOTS*, Aug. 2001.

[12] S. Nath, P. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of SenSys '04*, November 2004.

[13] M. T. Ozsu and P. Valduriez. *Principles of distributed database systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[14] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[15] E. H. Spafford. The internet worm program: An analysis. Technical Report Purdue Technical Report CSD-TR-823, West Lafayette, IN 47907-2004, 1988.