Spatially Distributed Databases on Sensors *

Bo Xu
University of Illinois at Chicago
bxu@eecs.uic.edu

Ouri Wolfson
University of Illinois at Chicago
wolfson@eecs.uic.edu

Sam Chamberlain
Army Research Laboratory
wildman@arl.gov

ABSTRACT

We consider the problem of data dissemination in a sensor network. First we introduce a cost model for data dissemination in such networks. The model quantifies the tradeoff between the inconsistency of the data, and its transmission cost; the transmission cost is given in terms of energy and bandwidth. Then we introduce a novel broadcast policy, ABR, that utilizes our cost model. In the ABR policy, each sensor broadcasts updates only if the cost of sending is justified by (its view of) the current level of inconsistency. Data received from other sensors are rebroadcast using the same cost measures. Finally we compare the performance of ABR with that of several existing data dissemination policies. Via the comparison we discover an intricate interplay among the various parameters in choosing a dissemination policy.

Keywords

data replication, distributed databases, sensor networks

1. INTRODUCTION

In this paper we study the problems of broadcast and replication in a peer to peer architecture. This architecture is motivated by sensor and "smart dust" networks (see [13]), but our results are also applicable to other peer to peer broadcast networks. In sensor networks the nodes collaborate to assemble a complete database from local fragments. For instance, in the case of sensors that are parachuted or sprayed from an airplane, the database renders a global picture of an unknown terrain from local images collected by individual sensors.

We divide the area covered by the sensors into geographic cells. Ideally, the sensors are spread out such that a single sensor resides in a cell, but even if this condition is not satisfied our results still hold (the details are explained in subsection 2.1). For the purpose of data dissemination we use a "master" replication environment (see [2]), in which each node i "owns" the master copy of a data item D_i, i.e. it generates all the updates to D_i. For example, D_i may be the latest in a sequence of images taken periodically by the node i of its local surroundings; each new image updates D_i. The database of interest is $D = \{D_1,...,D_n\}$, where n is the number of nodes and also the number of items in the database. Since each sensor is very unreliable, a (possibly inconsistent) copy of D is stored at each sensor. [1]

In this paper we introduce a novel policy, Adaptive Broadcast Replication (ABR), for broadcasting new versions of the data items. In each broadcast a data item is associated with its version number, and a node that receives a broadcasted data item updates its local database if and only if the local version is older than the newly arrived version. The novel aspect of the ABR policy is that it takes into consideration the tradeoff between data consistency and communication cost. The intuition for the tradeoff is that if the new version of a data item is not much different than the previous one, then the communication cost of broadcasting it is not justified. The communication cost reflects the energy for message transmission, and the bandwidth consumption.

In order to quantify the tradeoff between consistency and communication we introduce the concept of the *inconsistency-cost* of a data item. This concept, in turn, is quantified via the notion of the cost difference between two versions of a data item D_i. In other words, the inconsistency cost of using an older version v rather than the latest version w is the distance between the two versions. If D_i is an image, an existing algorithm that quantifies the difference between a pair of images can be used (see for example [1]). If D_i is the quantity-on-hand of a widget, then the difference between the two versions is the difference between the quantities. Now, in order to quantify the tradeoff between inconsistency and communication one has to answer the question: what amount of bandwidth/energy/dollars am I willing to spend in order to reduce the inconsistency cost on a data item by one unit?

In order to motivate our proposed policy, ABR, consider first a naive policy that we call Full Broadcast Replication (FBR). In FBR, whenever D_i is updated, node i broadcasts its local copy of the whole database D, called $D(i)$. In other words, i broadcasts D_i, as well as its local version of each one of the other data items in the database. When a node j receives this broadcast, j updates its version of D_i, and j also updates its local copy of each other item D_k, for which

*This research was supported in part by Army Research Labs grant DAAL01-96-2-0003.

[1] D is inconsistent if some data items do not contain the most recent version.

the version number in $D(i)$ is more recent. Thus these indirect broadcasts of D_k (to j via i) are "gossip" messages that increase the consistency of each local database. Clearly, if every sensor in the system has the broadcasted version of D_k, then the extra communication cost expended to include D_k in a broadcast message is not justified.

The Adaptive Broadcast Replication (ABR) policy introduced in this paper optimizes the tradeoff between consistency and communication using a cost based approach. In the ABR policy, when node i receives an update to D_i it first determines whether the expected reduction in inconsistency justifies broadcasting a message. If so, then i "pads" the broadcast message that contains D_i with a set S of data items (that i does not own) from its local database, such as to optimize the total cost. One problem that we solve in this paper is how to determine the set S, i.e. how node i should select for each broadcast message which data items from the local database to piggyback on D_i. In order to do so, i estimates for each j and k the expected benefit (in terms of inconsistency reduction) to node k of including in the broadcast message its local version of D_j.

We compare by simulation FBR, ABR, and two other dissemination protocols for sensors networks, namely SPIN-BC, and Flooding (see [13]). Broadly speaking, in SPIN-BC, a node receiving a new version of a data item advertises it; if other nodes request it, then it broadcasts the new version. In flooding (FLD) a node receiving a new version of a data item simply broadcasts it. In comparing the four protocols we conclude that ABR is superior to FBR, and SPIN-BC is superior to FLD. In terms of comparison between ABR and SPIN-BC we discover an interesting and subtle interplay among three parameters: the message initiation cost, the cost of broadcasting a single data item, and the cost of a unit of inconsistency. For some parameter combinations ABR is superior (i.e. it has a lower cost), and for others SPIN-BC is superior.

Let us now put this paper in the context of existing work on consistency in distributed systems. Our approach is new as far as we know. Although gossiping has been studied extensively in distributed systems and databases (see section 5), none of the existing works use an inconsistency-communication tradeoff cost function in order to determine what gossip messages to send. Furthermore, in the emerging resource constrained environments this tradeoff is crucial. Also our notion of consistency is appropriate for the types of novel applications discussed in this paper, and is different than the traditional notion of consistency in distributed systems discussed in the literature (e.g., [4]. Specifically, in contrast to the traditional approaches, our notion of consistency does not mean consistency of different copies of a data item at different nodes, and it does not mean mutual consistency of different data items at a node. In this paper a copy of a data item at a node is consistent if it has the latest version of the data item. Otherwise it is inconsistent, and the inconsistency cost is the distance between the local copy and the latest version of the data item. Inconsistency of a local database is simply the sum of the inconsistencies of all data items. We employ gossiping to reduce inconsistency, not to ensure consistency as in using vector clocks ([4]).

In summary, the key contributions of this paper are as follows.

- The introduction of a cost model that enables quantifying the tradeoffs between communication cost and inconsistency in sensor networks.
- The introduction and analysis of the Adaptive Broadcast Replication (ABR) policy, and its comparison with other existing policies.
- The discovery of an intricate interplay among the following three parameters in choosing a dissemination policy: the message initiation cost, the cost of broadcasting a single data item, and the cost of a unit of inconsistency.

The rest of the paper is organized as follows. In Section 2 we introduce the operational model and the cost model. In Section 3 we describe the ABR policy, and in Section 4 we compare the broadcast policies by simulation. In Section 5 we discuss relevant work, and in the last section we summarize the paper.

2. THE MODEL

In subsection 2.1 we precisely define the overall operational model, and in subsection 2.2 we define the cost model.

2.1 Operational model

The system consists of a set of n nodes that communicate by message broadcasting. Each node i $(1 < i < n)$ has a data item D_i associated with it. Node i is called D_i's owner. This data item may contain a single numeric value, or a complex data structure such as a motion trajectory, or an image of the local environment. Only i, and no other nodes, has the authorization to modify the state of D_i. A data item is updated at discrete time points. Each update creates a new version of the data item. In other words, the *kth version of* D_i, denoted $D_i(k)$, is generated by the kth update. We denote the latest version of D_i by $\overline{D_i}$. Furthermore, we use $v(D_i)$ to represent the version number of D_i, i.e. $v(D_i(k)) = k$.

For two versions $D_i(k)$ and $D_i(k')$, we say that $D_i(k)$ is *newer than* $D_i(k')$ if $k > k'$, and $D_i(k)$ is *older than* $D_i(k')$ if $k < k'$.

An owner i periodically broadcasts its data item D_i to the rest of the system. Each such broadcast includes the version number of D_i. Since the broadcast channel is unreliable, some broadcasts may be missed by some nodes, thus each node j has a version of each D_i which may be older than $\overline{D_i}$. The *local database* of node i at any given time is the set $< D_1^i, D_2^i, ..., D_n^i >$, where each D_j^i (for $1 \leq j \leq n$) is a version of D_j. Observe that since all the updates of D_i originate at i, then $D_i^i = \overline{D_i}$. Node i updates D_j^i $(j \neq i)$ in its local database when it receives a broadcast from j.

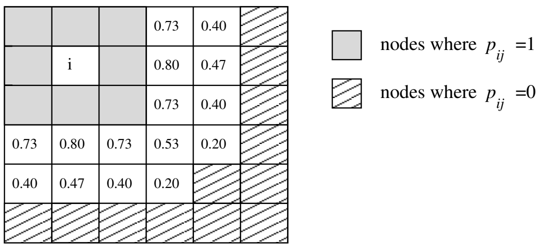


Figure 1: Reliability model of a wireless sensor network

We model the network reliability probabilistically. A broadcast message from a node i is received successfully and correctly by node j ($i \neq j$) with probability p_{ij}. We call p_{ij} the *probability of reception* between i and j and we call $(p_{ij})_{1 \le i \le n, 1 \le j \le n}$ the *reliability matrix*.

We divide a priori the geographic area into a grid of cells (see Figure 1). Ideally, the sensors are spread such that a single sensor will reside in each cell. Due to imperfection, a cell may contain more than a single sensor. In this case, all the sensors in the cell elect a leader which becomes the representative of the cell and participates in the data dissemination protocol (we assume that the probability of reception between any pair of sensors in a single cell is one). After the leader is elected, all the non-leader sensors shift into the inactive (sleep) mode and remain in that mode until the next leader election protocol is conducted. This next protocol is initiated by the leader before it runs out of power. The cell leaders use TDM (time division multiplexing) to share the broadcast channel. Each cell has a pre-allocated time slot, and different cells may share the same slot if they are far enough from each other such that their signals do not collide. Each sensor knows the cell to which it belongs using an embedded Global Positioning System (GPS), and thus knows the time slot it should use.

The probability of reception between each pair of cells is determined by the Euclidean distance between the centers of the cells. For example, the distance between cell i and the cell to its north is 1, and the distance between i and the cell on its southeast is $\sqrt{2}$. For each cell i, there are three types of cells. The first type are cells that receive messages from i with probability 1. The second are cells that receive messages with probability 0, i.e. they are too far and do not receive any messages from i. And the third are cells that receive them with probability p ($p < 1$) where p is proportional to inverse of their square distance from i ([10]). Consider Figure 1 as an example. For each neighbor j of i, the probability of reception between i and j is 1 (see the shadowed cells). For each node j out of a certain range, p_{ij} is 0 (see the striped cells). For each j in the rest of the area, p_{ij} is proportional to the inverse of the square distance between i and j (see the numbers in the white cells).

2.2 Cost Model

In this subsection we introduce a cost function that quantifies the tradeoff between consistency and communication. The function has two purposes. First, to enable determining the items that will be included in each broadcast of the ABR policy, and second, to enable comparing the various policies.

Inconsistency cost

Assume that the distance between any two versions of a data item can be quantified. For example, the distance between two data item versions may be taken to be the Euclidean distance between two locations of an object. If D_i is an image, one of the many existing distance functions between images (e.g. the cross-correlation distance ([1])) can be used. Formally, the *distance* between two versions $D_i(k)$ and $D_i(j)$, denoted $DIST(D_i(k), D_i(j))$, is a function whose domain is the nonnegative reals, and it has the property that the distance between two identical versions is 0. If the data item owned by each node consists of two or more types of logical objects, each with its own distance function, then the distance between the items should be taken to be the weighted averages of the pairwise distances.

We take the $DIST$ function to represent the cost, or the penalty, of using the older version rather than the newer one. More precisely, consider two consecutive updates on D_i, namely the kth update and the $(k+1)$st update. Assume that the kth update happened at time t_k and the $(k+1)$st update at time t_{k+1}. Intuitively, at time t_{k+1} each node j that did not receive the kth version $D_i(k)$ during the interval $[t_k, t_{k+1})$, pays a price which is equal to the distance between the latest version of D_i that j knows and $D_i(k)$. In other words, this price is the penalty that j pays for using an older version during the time in which j should have used $D_i(k)$. If j receives $D_i(k)$ sometime during the interval $[t_k, t_{k+1})$, then the price that j pays on D_i is zero. Formally, assume that at time t_{k+1} the latest version of D_i that j knows is v ($v \le k$). Then *j's inconsistency cost on version k of D_i* is $COST\_INCO_j(D_i(k)) = DIST(D_i(v), D_i(k))$.

The *inconsistency cost of the system on $D_i(k)$* is $COST\_INCO(D_i(k)) = \sum_{1 \le j \le n} COST\_INCO_j(D_i(k))$.

The *total inconsistency cost of the system on D_i up to* the *mth update of D_i*, denoted $COST\_INCO(i, m)$, is $\sum_{1 \le k < m} COST\_INCO(D_i(k))$.

The *total inconsistency cost for the system up to time t* is $COST\_INCO(t) = \sum_{1 \le i \le n} COST\_INCO(i, m_i)$, where m_i is the highest version number of D_i at time t.

Communication cost

The cost of a message depends on the length of the message. In particular, if there are m data items in a message, the cost of the message is $C_1 + m \cdot C_2$. [2] C_1 is called the *message initiation cost* and C_2 is called the *message unit cost.* C_1 represents the cost of energy consumed by the CPU to prepare and send the message. C_2 represents the incremental cost of adding a data item to a message. The values of C_1 and C_2 are given in inconsistency cost units. They are determined based the amount of resource that one is willing to spend in order to reduce the inconsistency cost on a version by one unit. For example, if $C_1 = C_2$ and one is willing to spend one message of one data item in order to reduce the inconsistency by at least 50, then $C_1 = C_2 = 1/100$.

The *total communication cost up to time t* is the sum of the costs of all the messages that have been broadcast from the beginning (time 0) until t.

System cost

The *system cost up to time t*, denoted $COST\_SYS(t)$, is the sum of the total inconsistency for the system up to t, and the total communication cost up to t. The system cost is the objective function optimized by the ABR policy. When comparing ABR with other broadcast policies, there are two additional costs (one of them being storage) which will come into play. We will explain the inclusion of these costs in the model in section 4.

3. THE ADAPTIVE BROADCAST REPLICATION POLICY

In this section we describe the Adaptive Broadcast Replication policy. Intuitively, a node i executing the policy behaves as follows. When it receives an update to D_i node i constructs a broadcast message by evaluating the benefit of

[2] Actually the cost of a message can be any non-decreasing function of the length of the message. In section 3.2 we will discuss how our approach can be extended to this more general case.

including in the message each one of the data items in its local database. [3] Specifically, the ABR policy executed by i consists of the following two steps.

(1) Benefit estimation: For each data item in the local database, estimate how much the inconsistency of the system could be reduced if that data item is included in the message.

(2) Message construction: Construct the message which is a subset of the local database so that the total estimated net benefit of the message is maximized (The net benefit is the difference between the inconsistency reduced by the message and the cost of the message). Observe that the set of data items to be broadcast may be empty. In other words, when D_i is updated, node i may estimate that the net benefit of broadcasting any data item is negative.

Each one of the above steps is executed by an algorithm which is described in one of the next two subsections.

3.1 Benefit Estimation

Intuitively, the benefit to the system of including a data item D_j in a message that node i broadcasts is in terms of inconsistency reduction. This reduction depends on the nodes that receive the broadcast, and on the latest version of D_j at each one of these nodes. Node i maintains data structures that enable it to estimate the latest version of D_j at each node. Then the benefit of including a data item D_j in a message that i broadcasts is simply the sum of the expected inconsistency reductions at all the nodes.

In computing the inconsistency reduction for a node k we attempt to be as accurate as possible, and we do so as follows. Node i maintains a "knowledge matrix" which stores in entry (k, j) the last version number of D_j that node i received from node k (this version is called $v(D_j^k)$), and the time when it was received. Additionally, i saves in the "real history" for each D_j all the versions of D_j that i has "heard" from other nodes, the times at which it has done so, and from which node they were received[4]. The reason for maintaining all this information is that now, in estimating which version of D_j node k has, node i can take into consideration two factors: (1) the last version of D_j that i received from k at time, say t, and (2) the fact that since time t node k may have received updates of D_j by "third party" messages that were transmitted after time t, and "heard" by both, k and i. Node i also saves with each version v of D_j that it "heard", the distance (i.e. the inconsistency caused by the version difference) between v and the last version of D_j that i knows; this difference is the parameter necessary in order to compute the inconsistency cost reduction that is obtained if node i broadcasts its latest version of D_j.

In subsection *3.1.1* we describe the data structures that are used by a node i in benefit estimation. In subsection *3.1.2* we present i's benefit estimation method.

3.1.1 Data Structures

(1) The Knowledge matrix: For each data item D_j $(j \neq i)$, denote by $v(D_j^k)$ the latest version number of D_j that i received from k, and denote by $t(D_j^k)$ the last time when D_j^k was received at i. The *knowledge matrix* at node i is:

[3]In this paper ABR is executed by node i when an update occurs on D_i. But this is not strictly necessary. The same policy can be executed periodically regardless of updates.

[4]There is a potential storage problem here, which we address, but we postpone the discussion for now.

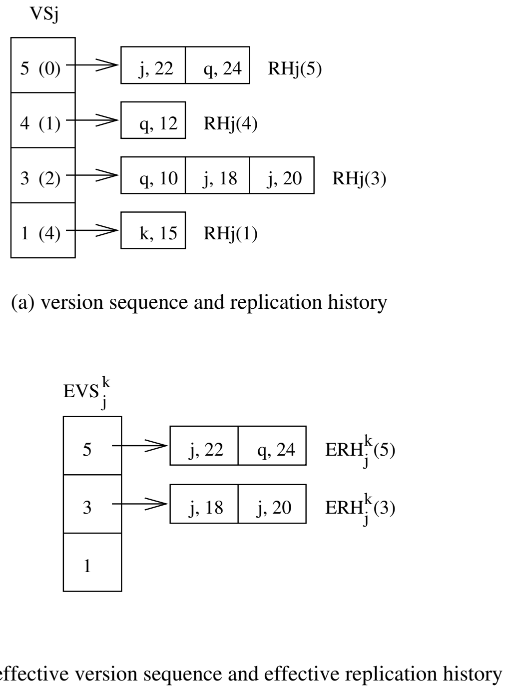


Figure 2: Data structures in benefit estimation

$$M_i = \begin{pmatrix} (t(D_1^1), v(D_1^1)) & (t(D_2^1), v(D_2^1)) & \dots & (t(D_n^1), v(D_n^1)) \\ (t(D_1^2), v(D_1^2)) & (t(D_2^2), v(D_2^2)) & \dots & (t(D_n^2), v(D_n^2)) \\ \vdots & \vdots & \ddots & \vdots \\ (t(D_1^n), v(D_1^n)) & (t(D_2^n), v(D_2^n)) & \dots & (t(D_n^n), v(D_n^n)) \end{pmatrix}$$

Node i updates the matrix whenever it receives a message. Specifically, when i receives a message from k that includes D_j, i updates the entry (k,j) of the matrix. In addition, if the version of D_j received is newer than the version in i's local database, then the newer version updates D_j in the local database.

(2) Version sequence: A version sequence records all the version numbers that i has ever known about a data item. Due to unreliability, it is possible that i has not received all the versions of a data item. In particular, the *version sequence of* D_j is $VS_j =< v_1, v_2, ..., v_h >$ where $v_1 < v_2 < ... < v_h$ are all the version numbers that i has ever known about D_j. For each $v \in VS_j$, i saves in the distance between $D_j(v)$ and $D_j(v_h)$. Figure 2(a) illustrates an example of a version sequence. The number in parenthesis besides a version number is the distance between that version of D_j and the last version of D_j which is 5. Thus, in this example $DIST(D_j(v), D_j(v')) = |v - v'|$.

(3) Replication history: For each version number v in each VS_j, i maintains a *replication history* $RH_j(v)$. In this history there is a record for every time point at which i received $D_j(v)$ from a node, along with the identification of that node. $RH_j(v)$ also contains a record for every time point at which i broadcast $D_j(v)$. Figure 2(a) gives an example of a version sequence and its replication histories. Figure 2(a) shows that node i received version 1 of D_j from k at time 15, and it received version 3 from q at time 10 and from j

at times 18 and 20.

Now we discuss how we limit the amount of storage used. Observe that the lengths of each version sequence VS_j and replication history $RH_j(v)$ increases unboundedly as i receives more broadcasts. This presents a storage problem. A straight-forward solution to this problem is to limit the length of each version sequence to α and the length of each replication history to β. We call this variant of the ABR policy ABR(α, β). The drawback of ABR(α, β) is that when the length of a replication history $RH_j(v)$ is smaller than β, since each replication history is limited to β, other replication histories can not make use of the free storage of $RH_j(v)$. A better solution, which we adopt in this paper, is to limit the sum of the lengths of each replication history in each version sequence. In particular, we use ABR-s to denote the ABR policy in which $\sum_{1 \leq j \leq n} \sum_{v \in VS_j} |RH_j(v)|$ is limited to s [5]. s must be at least n.

3.1.2 The Benefit Estimation Method

When an update on D_i occurs, node i estimates the benefit of including its latest version of D_j in the broadcast message, for each D_j in the local database. Intuitively, i does so using the following procedure. For each node k compute the set of versions of D_j that k can have, i.e. the set of versions that were received at i after D_j^k was received. Assume that there are m such versions. Then, compute the set of broadcasts from which k could have learned each one of these versions. Based on this set compute the probabilities $p_1, p_2, ..., p_m$ that k has each one of the possible versions $v_1, v_2, ..., v_m$. Finally, compute the expected benefit to k as the sum $p_1 \cdot DIST(v(D_j), v_1) + p_2 \cdot DIST(v(D_j), v_2) + ... + p_m \cdot DIST(v(D_j), v_m)$.

Formally, node i performs the benefit estimation in five steps:

(1) Construct an *effective version sequence* (EVS) of D_j^k which is a subsequence of VS_j:

$$EVS_j^k = \{\ v | v \in VS_j \text{ and } v \geq v(D_j^k) \text{ and there exists } (n_l, t_l) \in RH_j(v) \text{ such that } t_l \geq t(D_j^k)\ \} \quad (1)$$

Intuitively, EVS_j^k is the set of versions of D_j that k can have, as far as i knows. In other words, EVS_j^k contains each version v that satisfies the following two properties: (i) v is higher than or equal to the latest version of D_j that i has received from k (i.e. $v(D_j^k)$), and (ii) i has received at least one broadcast which includes $D_j(v)$, and that broadcast arrived later than D_j^k. For example, Figure 2(b) illustrates EVS_j^k for the example in Figure 2(a). Observe that the version of D_j^k is 1, and it was received at time 15. Notice that EVS_j^k is not necessarily a consecutive subsequence of VS_j^k. For example, version 4 is not in EVS_j^k because $(q, 12)$ was broadcast at time 12, i.e. before D_j^k. This means that k has not received this broadcast, and thus, as far as i is concerned, 4 is not a possible current version number of D_j in k's local database.

(2) For each v in EVS_j^k construct an *effective replication history* which is a subset of $RH_j(v)$:

$$ERH_j^k(v) = \{\ (n_l, t_l) | (n_l, t_l) \in RH_j(v) \text{ and } t_l > t(D_j^k)\ \} \quad (2)$$

[5] $|A|$ denotes the size of the set A.

Intuitively, $ERH_j^k(v)$ contains all the broadcasts from which k could have learned $D_j(v)$, based on i's knowledge. Figure 2(b) illustrates each $ERH_j^k(v)$ which is generated from the example in Figure 2(a). Notice that $(q, 10)$ is not contained in $ERH_j^k(3)$ because $(q, 10)$ was broadcast before D_j^k (which was broadcast at time 15), and thus k could not have received that broadcast (otherwise it would have broadcast a higher version number at time 15).

(3) For each v in EVS_j^k, compute η_v which is the probability that the version number of D_j in k's local database is v:

$$\eta_v = \Big(1 - \prod_{(n_l, t_l) \in ERH_j^k(v)} (1 - p_{n_l k})\Big) \prod_{v' \in EVS_j^k \text{ and } v' > v} \ \prod_{(n_l, t_l) \in ERH_j^k(v')} (1 - p_{n_l k}) \quad (3)$$

(4) If the version number of D_j in k's local database is v, then the estimated benefit to k of including D_j^i in the broadcast message is taken to be the distance between $D_j(v)$ and D_j^i (i.e. $DIST(D_j(v), D_j^i)$). Denote this benefit $B(D_j^i, k, v)$.

(5) The estimated benefit to k of including D_j^k in the broadcast message is taken to be $p_{ik} \cdot \sum_{v \in EVS_j^k} \eta_v \cdot B(D_j^i, k, v)$. Denote this benefit by $B(D_j^i, k)$. Then the estimated benefit $B(D_j^i)$ of including D_j^i in the broadcast message is:

$$B(D_j^i) = \sum_{1 \leq k \leq n \text{ and } k \neq i,j} B(D_j^i, k) \quad (4)$$

3.2 Message Construction Step

The objective of this step is for node i to select a subset S of data items from the local database for inclusion in the broadcast message. The set S is chosen such that the expected net benefit of the message (i.e. the total expected inconsistency-reduction benefit minus the cost of the message) is maximized.

First, node i sorts the estimated benefits of the data items in descending order. Thus we have the benefit sequence $B(D_{k_1}^i) \geq B(D_{k_2}^i) \geq ... \geq B(D_{k_n}^i)$. Then i constructs the message as follows. If there is no number t between 1 and n such that the sum of the first t members in the sequence is bigger than $(C_1 + t \cdot C_2)$, then i will not broadcast a message.[6] Else, i finds the shortest prefix of the benefit sequence such that the sum of all the members in the prefix is greater than $(C_1 + m \cdot C_2)$, where m is the length of the prefix. i places the data items corresponding to the prefix in the broadcast message. Then i considers each member j that succeeds the prefix. If $B(D_j^i)$ is greater than or equal to C_2, then i puts D_j^i in the message. [7]

This concludes the description of the ABR-s policy, which consists of the benefit estimation and message construction steps. It is easy to see that the time complexity of the policy is $O(n \cdot s)$.

[6] Remember that the cost of a message containing m data items is $(C_1 + m \cdot C_2)$.

[7] For the general case where the cost of a message is a non-decreasing function of the length of the message, i computes the net benefit of the first t members in the benefit sequence for each $1 \leq t \leq n$. If for all the values of t the net benefit is not greater than zero, then i will not broadcast a message. Else, i finds the t such that the net benefit is maximized and includes the first t data items the message.

4. COMPARISON OF THE POLICIES BY SIMULATION

To study the effectiveness of the ABR policy, we conducted extensive simulations. We compared ABR with FBR and with two other broadcast policies, namely the classical flooding (FLD) ([12]) and SPIN-BC ([3]). In this section we first briefly describe how FLD and SPIN-BC work. Then we discuss the simulation method followed by the simulation results.

4.1 FLD and SPIN-BC

In the FLD policy, a node that receives or generates a new version of a data item, rebroadcasts a copy of the data item to all the other nodes. In contrast to ABR and FBR, in FLD a node i broadcasts the new version even if it is for a data item D_j other than D_i. In this case the new version must have been received from a message that provided a new value for D_j in i's local database. In the SPIN-BC policy, upon receiving or generating a new version of a data item, each node negotiates with other nodes before it decides whether to rebroadcast the new version. Basically, when a node i receives a new version of a data item, it broadcasts an advertise message which includes the name of the data item and its version number. If the version is newer than the one in a node j's local database, j broadcasts a request message. Then i sends a message which contains the actual data. In our experiments the cost of advertise and request messages is the message initiation cost C_1.

4.2 Simulation Method

4.2.1 Inconsistency Cost Functions

First we discuss the two distance functions used for measuring inconsistency. One is version-based, namely the inconsistency between two versions $D_i(k)$ and $D_i(k')$ is the difference between the version numbers, i.e. $DIST(D_i(k), D_i(k')) = |k-k'|$. We call this the *version-based* distance function. The other distance function is value-based, namely the distance between two versions $D_i(k)$ and $D_i(k')$ is the difference between the values (we assume D_i contains a single numeric value), i.e. $DIST(D_i(k), D_i(k')) = |D_i(k) - D_i(k')|$. We call this the *value-based* distance function. Each time a data item is updated, we randomly select a real number between 0 and 100 as the value for the new version of that data item. This way the two distance functions represent two extreme patterns in which data changes over time: the version-based distance function represents a very regular pattern in which data is increased by one unit on each update, whereas the value-based distance function represents a very chaotic pattern in which data changes randomly. For space considerations the plots in our figures refer only to the version based distance function. However, the results for the value based function are qualitatively similar.

4.2.2 Extra Resource Cost Incurred by the ABR Policy

Observe that ABR performs additional computation for each message that it broadcasts (to determine the set of data items that will be broadcast) compared with the other policies. It also uses additional storage for the data structures that it maintains.

The additional computation incurred by ABR is modeled by the *CPU factor* denoted C_3. The CPU factor is a fraction added to the message initiation cost C_1 of ABR. For example, if in a simulation run $C_1 = 10$ and $C_3 = 0.2$, then ABR's message initiation cost is 12, rather than 10 for the other policies.

Parameter	Symbol	Value	
Number of nodes	n	36	
Maximum data item value		100	
Update intensity of node i	λi	Randomly selected from [0.00001,0.1] and fixed for all the simulations	
Policy		FBR, ABR, FLD, SPIN-BC	
CPU factor	$C3$	0.1	
Storage unit cost	$C4$	0.0001	
Information cost model		Version-based	Value-based
Message initiation cost	$C1$	1 - 20	1 - 1000
Message unit cost	$C2$	0.01 - 10	0.01 - 500

Table 1: Parameter settings

A node using the ABR policy also pays additional storage cost compared to the other policies. This depends on both, the size of the data structure it maintains and the length of time for which the data structure is kept. We use C_4 to denote the cost of a unit of storage occupied for one time unit, and call it the *storage unit cost.* C_4 can be determined by the number of storage units that one is willing to maintain for one time unit, in order to reduce the inconsistency cost on a version by one unit. Formally, for ABR-s (recall that s is the size of data structures) the extra storage cost of i up to time t is $C_4 \cdot s \cdot t$.

Note that C_3 and C_4 are introduced for the sole purpose of comparing the system cost of ABR with that of the other policies. In contrast to the communication cost and the inconsistency cost, they have no impact on the execution of ABR. The reason for this is that the storage and extra computation expended by ABR is fixed, independently of the set of items that are actually included in the broadcast message. Another thing needs to be mentioned is that C_3 and C_4 are always charged regardless of whether the node decides to broadcast a message, whereas C_1 is charged only when a message is sent.

4.2.3 Simulation Testbed

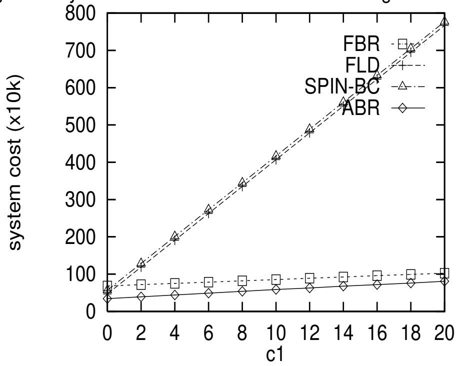
We use the 36-node grid shown in Figure 1 (see section 2.1). For each node i, the updates at i are generated by a Poisson process with intensity λ_i. This means that on average i generates λ_i updates per time unit. Each λ_i is randomly selected from the interval $[0.00001, 0.1]$. We select λ_i only once and keep it fixed for all the simulations. For each set of parameters we execute a simulation run for 10000 logical time units, which on average introduces 500 updates to each node. All the parameters and their value ranges are summarized in Table 1.

Each simulation run is executed as follows. In each time unit, updates are generated and all the nodes are processed in sequence. When a new update occurs on D_i, the total inconsistency on D_i is increased for each node that did not receive the previous update. Node i uses the policy of the simulation run to construct a message. i "broadcasts" the message and each other node j "receives" it with probability p_{ij} and updates its local database accordingly. A message sent by a node in time slot i of a round is accounted for in the same round by all the nodes with higher slots. The nodes with lower slots will account for this message in the next round. The total resource cost is increased by the cost of this message. In the SPIN-BC and FLD policies, node i may generate multiple messages in a single time unit as responses to the messages it received during that time unit. In such a case i can broadcast only one message in each time unit, and the other messages are queued for broadcasting in future time units.

4.3 Simulation Results

In this subsection we present the results of the comparison among the four policies FBR, FLD, SPIN-BC and ABR. [8] We compare them in terms of system cost, inconsistency cost and resource cost. The resource cost of ABR is the sum of the communication cost (which includes the extra CPU cost) and the storage cost. The resource cost of FBR, FLD and SPIN-BC is simply the communication cost. The system cost is the sum of the inconsistency cost and the resource cost. The experiments considered a large number of combinations of the parameters. Some of the simulation results are given below. We conducted many more simulation runs, but the results are omitted for space consideration. The omitted results confirm the conclusions given below.

Figure 3: System cost as a function of the message initiation cost (c2=1)

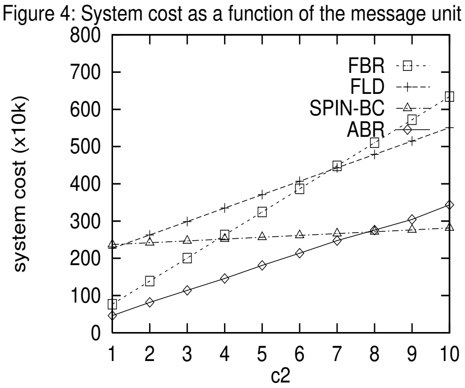


4.3.1 *Impact of the message initiation cost C_1*

Figure 3 plots the system cost of the four policies as a function of the message initiation cost C_1 (ranging from 0 to 20), with $C_2 = 1$. It can be observed that the cost curves of FBR and ABR are very similar to each other, and so are the cost curves of FLD and SPIN-BC. Overall, the system costs of FLD and SPIN-BC are much higher than those of FBR and ABR. Furthermore, the difference between them increases as the message initiation cost C_1 increases. The reason for this is that the number of messages used by FLD and SPIN-BC is higher than that used FBR and ABR. Actually the number of messages sent in FLD and SPIN-BC is about 20 times higher than that of FBR and ABR.

[8]In the experiments with ABR, the total area of the replication histories is limited to 36x36x2.

Figure 4: System cost as a function of the message unit cost (c1=5)



4.3.2 *Impact of the message unit cost C_2*

Figure 4 plots the performance of the four policies as a function of the message unit cost C_2 (ranging from 1 to 10), with $C_1 = 5$. First notice that ABR is better than SPIN-BC when C_2 is small ($C_2 < 8$), but worse than SPIN-BC when C_2 is high ($C_2 > 8$). One reason for this is that SPIN-BC uses control messages in order to reduce the number of data messages, whereas ABR uses only data messages. The second reason for this is that each one of ABR's data messages includes multiple data items, thus its cost is multiple times C_2.

4.3.3 *Impact of the CPU factor C_3 and storage unit cost C_4*

Clearly the system cost of ABR increases as the CPU factor C_3 and the storage unit cost C_4 increase. And given all the parameters excluding C_3 (or C_4), for each other protocol, there exists a value of C_3 (or C_4) such that above this value ABR has higher system cost than that protocol.

5. RELEVANT WORK

Heinzelman et al. ([13],[3]) proposed a family of protocols, called SPIN, which disseminate information among sensors in an energy-constrained wireless sensor network. Two of them, namely SPIN-BC and SPIN-RL, are designed for broadcast networks. The differences between our work and the SPIN protocols are as follows. First, the SPIN protocols use control messages to reduce the number of data messages and each data message contains a single data item. In contrast, ABR uses only data messages and each data message includes multiple data items. Second, we quantify the amount of inconsistency that can be reduced by a message, and they do not. Actually they do not consider updates to data items. Third, the two methods model the communication cost differently. In their work, the amount of resource consumed by a message is proportional to the length of the message. In other words, they do not consider the amount resource needed to initiate a message.

Lazy replication by gossiping has been extensively investigated in the past (see for example [9]). Epidemic algorithms such as the one used in Grapevine ([6]) also propagate updates by gossiping. However, there are two major differences between our work and the existing works. First, none of these works considered the cost of communication; this cost is important in the types of novel applications considered in this paper. Second, we consider the tradeoff between communication and inconsistency, whereas the existing works do not. Alonso, Barbara, and Garcia-Molina ([8]) studied the tradeoff between the gains in query response time obtained from quasi-caching, and the cost of checking coherency conditions. However, they assumed point to point communication and a centralized (rather than a distributed) environment.

Achieving global consistency by gossip messages is also used in distributed systems research (see [4]). A typical mechanism is vector clocks, used for example by Birman ([4]) in order to implement causal delivery. As explained in the introduction, the main difference between these works and the present paper is the consistency model. This leads to other differences. First, the communication cost is not considered in vector clock works. Second, in such works a node is not selective about what clocks are piggybacked in a message (similarly to the FBR policy, all clocks are piggybacked). Third, the piggybacked information is meta data (timestamps), whereas in our present work the piggybacked information are data items.

The data broadcasting problem studied in [11] is how to organize the broadcast and the cache in order to reduce the response time. The above works assume a centralized system with a single server and multiple clients communicating over a reliable network with large bandwidth. In contrast, in our environment these assumptions about the network do not always hold, and the environment is totally distributed and each node is both a client and a server.

Pagani et al. ([7]) proposed a reliable broadcast protocol which provides an exactly once message delivery semantics and tolerates host mobility and communication failures. Birman et al. ([5]) proposed three multicast protocols for transmitting a message reliably from a sender process to some set of destination processes. Unlike these works, we consider a "best effort" reliability model and allow copies to diverge.

6. CONCLUSION AND FUTURE WORK

In this paper we studied data broadcasting in peer to peer networks. Each node i "owns" the master copy of a data item D_i, i.e. it generates all the updates to D_i (see [2]). Each update is broadcast to the other nodes. The database of interest is $D = D_1,...,D_n$, where n is the number of nodes. A version of this database is stored at each node. First we introduced a cost model for quantifying the tradeoff between inconsistency and communication. The inconsistency cost is captured via the notion of the difference in cost between two versions of a data item D_i. The communication cost is captured via the notion of a message cost which is proportional to the length of the message.

Then we introduced the Adaptive Broadcast Replication (ABR) policy. In this policy, when a node i receives an update to D_i, it first estimates the expected benefit to the system of including in the broadcast message each data item D_j in i's local database. Then i constructs and broadcasts a message which consists of a subset of i's local database. The subset is chosen such that the "net" benefit of the message is maximized, i.e. the inconsistency-cost-reduction minus the message-cost is maximized. It can be shown (although this result is omitted for space considerations) that ABR is optimal for the level of knowledge that each node has about the distributed system. Optimality is in the sense that if ABR were to broadcast a different subset of data items, then the expected cost would be higher.

We compared the ABR policy with FBR, a naive broadcast disseminating policy, where a node broadcasts the entire local database when an update occurs. We also compared ABR with two other broadcast protocols, namely flooding (FLD) and SPIN-BC ([3]). The comparisons are done by simulation for a large number of parameters combinations. In comparing the four protocols we conclude that ABR is superior to FBR, and SPIN-BC is superior to FLD.

Concerning future work, observe that broadcast dissemination involves two problems, namely what to broadcast and when to broadcast. In this paper we concentrated on the first problem and adopted a naive solution for the second one, i.e. i broadcasts only when an update to D_i occurs. In the future, we plan to examine the problem of when to broadcast. For example, i is triggered to execute the ABR policy when an update on D_i occurs or when i receives a message that includes a new version of a data item (as in flooding).

7. REFERENCES

[1] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, December 1992.

[2] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. ACM SIGMOD 96*, pages 173–182. Montreal, Canada, 1996.

[3] W. R. Heinzelman J. Kulik and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor network. *submitted to ACM Wireless Networks*, 2000.

[4] P. Stephenson K. Birman, A. Schiper. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3), August 1991.

[5] T. A. Joseph K. Birman. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1), Feb. 1987.

[6] R. M. Needham M. D. Schroeder, A. D. Birrell. Experience with grapevine: the growth of a distributed system. *ACM Transactions on Computer Systems*, 2(1):3–23, Feb. 1984.

[7] E. Pagani and G. P. Rossi. Reliable broadcast in mobile multihop packet networks. In *Proc. ACM MOBICOM'97*, pages 34–42. Budapest, Hungary, 1997.

[8] D. Barbara R. Alonso and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems*, 15(3), Sept. 1990.

[9] S. Ghemawat R. Ladin, B. Liskov. Providing high availability using lazy replication. *ACM Transactions on Computer Systems*, 10(4), Nov. 1992.

[10] W. Stallings. *Data and Computer Communications.* Prentice Hall, 1997.

[11] B. Badrinath T. Imielinski. Mobile wireless computing: challenges in data management. *CACM*, 37(10), October 1994.

[12] A. S. Tanenbaum. *Computer networks.* Prentice Hall, 1996.

[13] J. Kulik W. R. Heinzelman and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proc. of MOBICOM99.* Seattle, WA, August 1996.