Updates in Highly Unreliable, Replicated Peer-to-Peer Systems*

Anwitaman Datta, Manfred Hauswirth, Karl Aberer
Swiss National Institute of Technology (EPFL), Lausanne, Switzerland
{anwitaman.datta, manfred.hauswirth, karl.aberer}@epfl.ch

Abstract

This paper studies the problem of updates in decentralised and self-organising P2P systems in which peers have low online probabilities and only local knowledge. The update strategy we propose for this environment is based on a hybrid push/pull rumor spreading algorithm and provides a fully decentralised, efficient and robust communication scheme which offers probabilistic guarantees rather than ensuring strict consistency. We describe a generic analytical model to investigate the utility of our hybrid update propagation scheme from the perspective of communication overhead.

1. Introduction

In most peer-to-peer (P2P) systems data is assumed to be rather static and updates occur very infrequently. For application domains beyond mere file sharing, for example trust management [2] or peer commerce, such assumptions do not hold and updates in fact may occur frequently. Other typical applications where new data items are added, deleted, or updated frequently by multiple users are bulletin-board systems, shared calendars or address books, e-commerce catalogues, and project management information.

To improve fault-tolerance and response time data is heavily replicated in most P2P systems and the system must take into account that peers are autonomous and may be offline frequently and that no global knowledge on the system exists. This is especially relevant for upcoming mobile environments. Thus we are operating in a decentralised setting without global control. This is the setting we assume for updates in our P-Grid P2P system [1, 3] described in this paper. To meet the challenges imposed by a high replication factor, lack of global knowledge, and peers being online only with a very low probability, we exploit epidemic algorithms under the assumption that probabilistic guarantees instead of strict consistency is sufficient and such an approach can indeed be used in a decentralised and self-organising environment.

Our proposed update algorithm is based on rumor spreading. We modify existing message flooding algorithms to achieve lower communication overheads but provide similar probabilistic guarantees and low latency. Since we assume that peers are mostly offline, we propose a hybrid push/pull algorithm so that offline peers can inquire for updates that they had missed when they become online again. In the push phase the algorithm uses a new mechanism, apart from traditional feedback and probabilistic methods to propagate a rumor, to avoid many duplicate messages by propagating a partial list of peers to which a particular message has already been sent. It also employs this list in conjunction with the number of duplicate messages received at a particular node as a local metric to estimate the extent to which a message has spread globally, and thereby tunes the probabilistic parameters of the generic algorithm locally which is a novel contribution. As will be discussed later, the current taxonomy of epidemic algorithms does not exploit the advantages of speculation (feed-forward) for significant reduction of communication overhead.

Further, the algorithm deals with logical connectivity (knowledge), and is disentangled from the underlying network/physical connectivity. Consecutively, the propagation of messages in the physical network and thus the implementation of applications is an orthogonal issue and may employ any point-to-point/multicast/ad-hoc communication mechanism (for example, [4, 13]). Though this work was initially motivated by P-Grid, the algorithm is generic and can be applied in other systems too. Our modifications to existing message flooding algorithms and other results may as well be applied to other search/update algorithms or broadcast/multicast schemes which employ flooding.

Another significant contribution of this paper is an analytical model of the gossiping algorithm unlike most of the literature which relies on simulation results. Since our algorithm is generic as argued above the analytical model is valid for many of the other variants of flooding algorithms and so are the results of our analysis.

2. Motivation and problem statement

Various global storage systems have been proposed, for example, Freenet [7], OceanStore [24], Pastry [26], and Farsite [5]. Their main goal is to provide distributed storage that scales to very large numbers of users and data sets. Additionally, they may exhibit certain specialisations that stem from their intended application domains. For example, Freenet wants to support free speech and anonymity on the Internet, whereas OceanStore focuses on distributed archival storage, which requires special system support.

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

IEEE COMPUTER SOCIETY

From the viewpoint of data management these systems should address two critical areas:

1. Efficient, scalable data access which is provided more or less by all approaches, and
2. Updates to the data stored, especially with respect to replication and low online probabilities.

Many of the access schemes are based on some mechanism that associates peers logically with a partition of the search space by means of a distributed, scalable index structure (P-Grid, OceanStore) and use replication to improve responsiveness and fault-tolerance. Some of the systems support updates. For example, OceanStore, uses classical schemes for updating replicas and assumes high availability of servers, whereas in the systems we envision the peers are fairly unreliable. Our assumptions are:

- Peers have low online probabilities and quorums cannot be assumed.
- Eventual consistency is sufficient.
- Since we do not target database systems update conflicts are rare and their resolution is not necessary in general.
- Probabilistic success guarantees for search are sufficient.
- Consecutive updates are distributed sparsely.
- The required communication overhead is the critical measure for the performance of the approach.
- The typical number of replicas is substantially higher than assumed normally for distributed databases but substantially lower than the total network size.
- The connectivity among replicas is high and the connectivity graph is random.

The two last points require further discussion: We may expect a few hundred to thousand replicas to be maintained. An intuitive explanation for such numbers is that if we need a 99.9% success guarantee for a search and only 10% of the replicas are online on average, then a serial search will need about 65 attempts (since $0.9^{65} \approx 0.001$). This does not account for load balancing among the available replicas which may account for another factor of 5 to 10 times of replication, particularly for "hot" items. Statistics from some of the music file sharing systems show that on average 200 to 250 replicas of same files are available, not counting the replicas that are not shared [30]. This requires devising a scheme which scales at least beyond the hundreds to thousands.

We assume that the replicas within a logical partition of the data space are connected among each other and each replica knows a minimal fraction of the complete set of replicas. If not enough replicas are known they can be efficiently obtained by randomized search. Additionally replicas get known through the update mechanism discussed in this paper. Even when peers potentially know a large fraction of the complete replica population the use of rumor spreading bears a number of advantages as compared to immediately contacting the complete neighborhoods for updates: better load balancing, reduced delay due to parallel propagation, improved robustness against changes in the peer network, and only partial knowledge of the neighborhood is required.

3. System model

In the analysis of the update algorithm we focus on the amount of communication required to achieve quasi-consistency and provide probabilistic guarantees for successful and appropriate results for queries. As observed in [22] we assume a very low rate of conflicts. Indeed, many applications, for example, music file sharing or news dissemination, have such a profile where, if data is altered, it may be treated as distinct and coexists as different versions. Similarly, deletions may use conventional tombstones and death certificates. These issues are relatively orthogonal to the communication mechanism used to convey the updates among the replicas. Further, in a decentralised system, such as P-Grid the "data" may indeed be knowledge regarding the system's topology, for example the routing tables used in P-Grid [1]. Most of these systems operate with a relatively high degree of imperfect knowledge, which is why probabilistic guarantee of information dissemination in such application scenarios is sufficient.

Our system assumes an infrastructure-less peer-to-peer system, i.e., all peers are equal and no specialised infrastructure, e.g., hierarchy, exists. No peer has a global view of the system but base their behaviour on local knowledge, i.e., its routing tables, replica list, etc. The peers can go offline at any time according to a random process that models the behaviour when peers are online. Physical connectivity and topology are ignored. We can assume that if two peers are online a communication channel may be established between them. This is not a serious relaxation of assumption, since if two peers may not communicate with each other, they will simply perceive each other to be offline. It is primarily the erratic behaviour of online availability and the complete lack of global knowledge, as well as the absence of any centralisation, and thus the need of self-organisation, which prompts us to call this environment unreliable. The limited resources, particularly bandwidth (and power in wireless/mobile environments), and the varying degree for tolerance of latency makes the environment even more challenging.

Our update propagation scheme has a push phase and a pull phase which are consecutive but may overlap in time. A new update is pushed by the initiator to a subset of responsible peers it knows, which in turn propagate it to responsible peers they know similar to a constrained flooding scheme. By "responsible" we denote peers that are affected by the update because they hold the original version of the data item. In our analysis of the push phase in the next section we assume a synchronous model which is a standard model for analysing epidemic algorithms [18].

Peers that have been disconnected (offline, disruption of communication) and get connected again, peers that do not receive updates for a long time (locally determined), or peers that receive a pull request, but are not sure to have the latest update, enter the pull phase to synchronise and reconcile. The pull scheme is similar to anti-entropy [9], in the sense that the pulling party tries to synchronise itself with the pulled party. Since the pulled party itself may be out of sync, it is preferable to contact multiple peers and choose the most up to date peer(s) among them.

Push phase of the update algorithm: When a peer p receives a update request (U, V, R_f, t), where U is the updated data item, V its version[1], t is a counter which counts the number of push rounds that have already been executed for the update, from a peer f, it also receives a partial (flooding) list R_f, to which the same update has been sent (not necessarily received by all peers in R_f). Then p chooses a random set R_p of its replicating peers and forwards the request $(U, V, R_f \cup R_p, t+1)$ with a probability $PF(t)$ to the set $R_p \backslash R_f$. $PF(t)$ can be any function, and is a self tuning parameter, determined locally by p. Another benefit of R_f is that p possibly discovers replicas unknown to her.

```
/* At replica 'p' (push phase) */
IF received(Push(U, V, R_f, t)) THEN
   /* process the update if not precessed it yet */
   IF ProcessedUpdate(U, V, R_f, t) == FALSE THEN
      Select a random subset Rp of replicas
         with  |Rp| = R * f_r;
      With probability PF(t):
         Push(U, V, Rf union Rp, t+1) to Rp \ Rf ;
      /* PF(t): deterministic or self
                  tuning function */
      ProcessedUpdate(U, V, R_f, t) = TRUE
```

Since any replica pushes the update at most once, the termination decision is trivial, and the number of push rounds gives the latency of propagating the update to all online replicas with high probability (arbitrarily close to 1).

Pull phase of the update algorithm: When a peer gets connected again because it was offline or suffered from a communication disruption, received no update for some time, or receives a pull request but is not sure whether it is in sync, then it enters the pull phase and inquires for missed updates.

```
/* At replica 'p' (pull phase) */
IF online_again OR no_updates_since(t) or
   (received_pull and not_confident) THEN
   /* not_confident is true:
         no update received within time T */
   Contact online replicas;
   Inquire for missed updates
           based on version vectors;
```

4. Analysis

4.1. Setup and notation for the analysis

The goal of our update algorithm is not to achieve complete consistency but rather to know what is the probability of a correct answer given certain model parameters. We assume that every peer knows a subset of all replicas that replicate the same data. We consider the replica network to be a small P2P network itself but with no internal structuring. It handles updates/requests for a partition of the data space.

In the analysis we start from a completely consistent state, analyse a single update request, and evaluate the number of messages and time (rounds of message exchange) required to reach a consistent state again. Since most of the replicas are offline most of the time, our notion of consistent state is more related to the online population R^{τ}_{on} at a given time τ rather than the whole set of replicas $\Re$. Though our analysis is generic, we evaluate the algorithm for realistic scenarios: availability of the peers to be a random process with expected value of being online between 10% to 30%. The replication factor is assumed to be between 100 to 1000 and though scalability is not a major issue for such small numbers larger replication factors too have been investigated. Table 1 shows the notation used in our analysis.

When an update U is initiated for a set $\Re$ of replicas with cardinality R, in general the online population in push round t will be $R_{on}(t) = R_{on}(t-1)*\sigma + [R - R_{on}(t-1)]*\beta$ where $\sigma = 1 - \epsilon_1$ and $\beta = \epsilon_2$, where $R_{on}(0) = R^{\tau}_{on}$ if the update starts at time τ. ϵ_1 is the probability of an online peer going offline in one push round and ϵ_2 is the probability of an offline peer coming online in a push round. These values are typically small and may vary in different push rounds. For the sake of simplification, we will initially ignore the effect of replicas coming online, and will further assume a constant σ, hence we have $R_{on}(t) = R_{on}(t-1) * \sigma$. Neglecting the effect of positive ϵ_2 is justified because peers coming online need to execute pull any way, and thus do not contribute to the push phase. Thus, even if some peers come online during a push phase, and receives update through push, it will not make any major difference to the whole system's behavior or the analysis of the same. The assumption of a very small ϵ_1 is justified because a single push round will take a very small time (network delay for a single message), and unless there is any kind of catastrophic failure, a very small number of peers will suddenly decide to go offline. Further, we choose a discrete time model for the rumor spreading algorithm, just like most other rumor algorithms. This in itself does not mean that we need synchronous rounds. It is indeed possible that because of variation in network latency, messages of different push rounds live in the network at the same instant of time. Thus, instead of treating t strictly as time, it needs to be interpreted as the round number, and the replicas which get infected by that round are effectively replicas that eventually gets updates from this round. Thus t does not in itself necessarily define an ordered chronology of receiving updates among all peers in the system.

Typically the parameters, such as f_r, σ, R, $R_{on}(0)$, may vary over time. But for the purpose of analysis we may assume that they remain constant throughout a single update push phase. In Section 6 we will give some indication of how the parameters can adapt over time to the varying network topologies.

The choice of two parameters P_F (probability of forwarding an update) and f_r (fraction of total replicas to which peers initially decide to forward an update) rather than defining only one parameter which couples both of them together is because we wanted to study the effects of both these factors in limited flooding algorithms. For example, a protocol like Gnutella [8] uses flooding with a fixed fanout, but uses no notion of P_F. Actually its use of TTL effectively means that P_F is 1 for TTL rounds, and 0 after that. Some other systems, for example one using gossip for

[1]This actually is a vector of version identifiers of the form $(VersionId_1, VersionId_2, \dots, VersionId_n)$. Version identifiers are universally unique identifiers computed locally by applying a a cryptographicly secure hash function to the concatenated values of the current date and time, the current IP address and a large random number.

1063-6927/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

R	Cardinality of the set of replicas $\Re$
t	Number of the push round for a particular update
U	The update message or its size (notation depends on the context)
$ML(t)$	Size of messages in round t
$L(t)$	Normalised size of the partial list of replicas which have the update in round t. This is equal to the number of entries in the list divided by R.
$R_{on}(t)$	Number of replicas online in round t
σ	Probability that a peer stays online in the next push round
f_r	Fraction of replicas to which peers initially decide to forward the update message
$newreplicas(t)$	Number of new replicas receiving update in round t
$msg(t)$	Number of messages in round t, including messages to offline replicas
$f_{\triangle aware}(t)$	Increment in fraction of online replicas which are aware of the update after round t
$f_{aware}(t)$	Total fraction of online replicas which are aware of the update at the beginning of round t.
$P_F(t)$	Probability that a peer pushes an update in round t if it received it in round $t-1$.
B	Size of data required to describe one replica (e.g., 10 bytes).

Table 1. Notation used in the analysis

ad-hoc routing [13], on the other hand uses probability of forwarding rumors as a design parameter. In order for our analysis to be general enough, such that all these variations of limited flooding can be reduced to special cases of our model, we included the notion of both fanout and probability of forwarding.

4.2. Analysis of the push phase

Round 0
The replica initiating the update propagation sends U to f_r fraction of replicas. Thus we obtain a total number of messages, $msg(0) = R * f_r$ (including messages to offline replicas). The number of new replicas which receive the update is $newreplicas(0) = R_{on}(0)f_r$. The number of online replicas without update is $R_{on}(0)(1-f_r)$. The message length in this round is $ML(0) = U + R * B * f_r$

Round 1
Assuming message flooding, where every replica which received an update message decides with probability $P_F(1)$ to forward it to $R * f_r$ replicas, we have:

$$msg(1) = R_{on}(0)\sigma P_F(1) R f_r^2 (1 - f_r)$$

The expression may be explained as follows. $R_{on}(0)f_r$ of the online population received the update in the previous round, a fraction σ of these replicas continue to stay online in the present round, a $P_F(1)$ fraction of these replicas decide to forward the message. Each of the $R_{on}(0)f_r\sigma P_F(1)$ peers decide to push the update, forwarding it to $R(f_r - f_r^2)$ replicas, since it knows that the update has already been sent to f_r^2 of the f_r fraction of randomly chosen replicas. Actually, in case a replica receives update information from more than one replica, it can use the list of 'updated replicas' in each of those messages, and hence the number of messages can be further trimmed, at an additional computational cost.

$$\begin{aligned} newreplicas(1) &= R_{on}(0)\sigma(1-f_r) * \\ &\quad [1-(1-f_r)^{R_{on}(0)f_r\sigma P_F(1)}] \end{aligned}$$

The expression may be explained as follows: Of the $R_{on}(0)\sigma(1 - f_r)$ uninformed online peers, a fraction $(1 - f_r)^{R_{on}(0)f_r\sigma P_F(1)}$ peers continue to stay uninformed when each of the $R_{on}(0)f_r\sigma P_F(1)$ informed peers forward (push) to f_r fraction of random peers. The others receive the update after this round. For the message length we have:

$$\begin{aligned} ML(1) &= U + R * B * (f_r + f_r(1-f_r)) \\ &= U + R * B * (1-(1-f_r)^2) \end{aligned}$$

Round $t \geq 2$
The results may be generalised as follows:

$$\begin{aligned} newreplicas(t) &= R_{on}(t-1)(1-f_{aware}(t-1))\sigma * \\ &\quad (1-(1-f_r)^{R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)}) \end{aligned}$$

Thus we obtain the fraction $f_{\triangle aware}(t)$ and $f_{aware}(t)$ as:

$$\begin{aligned} f_{\triangle aware}(t) &= (1-f_{aware}(t)) * \\ &\quad (1-(1-f_r)^{R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)}) \end{aligned}$$

Then,

$$\begin{aligned} f_{aware}(t) &= f_{aware}(t-1) + f_{\triangle aware}(t-1) \\ &= 1-(1-f_{aware}(t-1)) * \\ &\quad (1-f_r)^{R_{on}(t-2)f_{\triangle aware}(t-2)\sigma P_F(t-1)} \end{aligned}$$

Note that this is a recursive relationship and f_{aware} rapidly grows to 1. The expression for f_{aware} may exceed the value of 1, but that will have no physical relevance, and thus the function needs to be determined using a ceiling function and $f_{\triangle aware}$ too needs to be reevaluated accordingly in the final push round. Also note that $P_F(t)$ can be any arbitrary

1063-6927/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

function of t, which individual nodes can define in an ad-hoc manner, and we will see that this will be a self-tuning parameter for our push phase algorithm.

It is subtle to determine the number of messages and length of these messages. If the partial list of replicas, to which the update has already been transmitted along with the update information U, is ignored, we have

$$msg(t) = R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)Rf_r$$

since each of $R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)$ replicas (these replicas received the update in the previous round, and continued to stay online, and decided to forward the same) forward the update to Rf_r replicas. If the partial list of replicas is accounted for, then the number of messages decrease to

$$\begin{aligned} msg(t) &= R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t) * \\ &\quad Rf_r(1-f_r)^t \end{aligned}$$

and the length of each message in round t is given as

$$ML(t) = U + R * B * (1-(1-f_r)^{t+1})$$

We now prove the two equations above by induction. Let the normalised length of the partial list of replicas in a message be denoted by $L(t)$. The normalised length of the partial list is the fraction of the total replicas that the partial list contains. Then $ML(t) = U + R * B * L(t)$.

Induction hypothesis: $L(t) = 1-(1-f_r)^{t+1}$

Now $L(t+1) = f_r + L(t) - f_r L(t)$, since the Rf_r replicas chosen randomly are independent of the replicas in the partial list. Now, if our hypothesis is true then,

$$\begin{aligned} L(t+1) &= f_r + 1 - (1-f_r)^{t+1} - f_r(1-(1-f_r)^{t+1}) \\ &= 1-(1-f_r)^{t+1} + f_r(1-f_r)^{t+1} \\ &= 1-(1-f_r)^{t+2} \end{aligned}$$

Thus the hypothesis is consistent. Since the hypothesis is true for $t = 0, 1$, using induction, we conclude that $L(t) = 1-(1-f_r)^{t+1}$ where $ML(t) = U + R * B * L(t)$. Thus,

$$\begin{aligned} msg(t) &= R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)R * \\ &\quad f_r(1-L(t-1)) \\ &= R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)Rf_r(1-f_r)^t \end{aligned}$$

As may be observed, $L(t)$ increases with round number t and a legitimate question to ask is its effect on the resource (Memory/CPU/Bandwidth/Power) available at each of the replicas. A way to deal with increasing $L(t)$ may be to chose a normalised threshold length $L_{max}(t)$ such that $L(t) = min(L_{max}(t), L(t)^*)$ where $L(t)^* = L(t-1) + f_r - L(t-1)f_r$. This can be achieved by discarding either random entries or the head or tail of the partial list. In this case, $msg(t+1) = R_{on}(t)f_{\triangle aware}(t)\sigma P_F(t+1)Rf_r(1 - L_{max}(t))$.

$f_{\triangle aware}$ and f_{aware} stay unchanged, since the extra messages generated by reducing the $L(t)$ are all duplicate messages. Thus the nodes which push the update in the next round pay the penalty of forwarding extra messages.

Note that the case where $L_{max}(t)$ is zero for all replicas corresponds to the case where no list is propagated, and will enhance the number of duplicate messages, without any improvement in coverage of unreached replicas.

4.3. Analysis of the pull phase

If a replica p comes online at a random time (after the push phase is over), then it will (very likely) find the update information from any of its online replicas. The underlying assumption for such an optimism is that any replica that came online in the meantime must have pulled the update information by the time the concerned peer p came online. This justifies the eagerness of the Update Pull algorithm.

What is more interesting is what happens if p comes online while a push of an update is underway. If f_{aware} fraction of the replicas R_{on} are already aware of the update, the probability of a replica p getting the update in a attempts is

$$1 - [1 - (R_{on}f_{aware}/R)]^a$$

which implies that a constant number of pull attempts should give the update information with high probability. Since updates are propagating by push as well, the above term gives a worst case estimate. Indeed if $f_{\triangle aware}$ (refer to push phase analysis) fraction of online replicas received updates in the previous push round $(t-1)$, then (if they continue pushing) the probability of getting a push is

$$1 - (1 - f_r(1-L(t)))^{R_{on}(t-1)f_{\triangle aware}(t-1)\sigma P_F(t)}$$

4.4. Query (request)

Servicing requests under (possibly relatively frequent) updates is similar to the Pull phase of updates. For simple servicing of requests, we may indeed use the same analysis as in the Pull section. Since requests are more sensitive (updates can be lazy, and strong consistency is not our goal, however we intend to return correct and most recent result for any query) we may define some majority logic, or use a version scheme for identifying latest updates, or a hybrid of the two.

5. Analytical results

Based on the analytical model developed in the previous section we investigated for various environmental parameters the performance of the push phase of the propagation of a single update. For the evaluation of the recursive analytical functions a C-program has been developed.

Our performance criterion for this analysis is primarily the number of messages that are generated as part of a single update, compared to the extent to which the update propagates among the online population. As a simplifying (and for fixed networks, realistic) assumption we ignore message size, as single messages can accommodate the messages of maximal size that can occur in our setting.

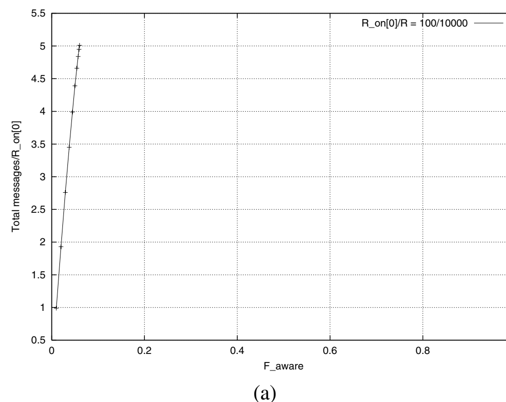
In the following result plots (e.g., Fig. 1) we will show on the y-axis the number of messages generated per member of the initial online population. As assumed in the previous section peers coming online are not participating in

IEEE COMPUTER SOCIETY

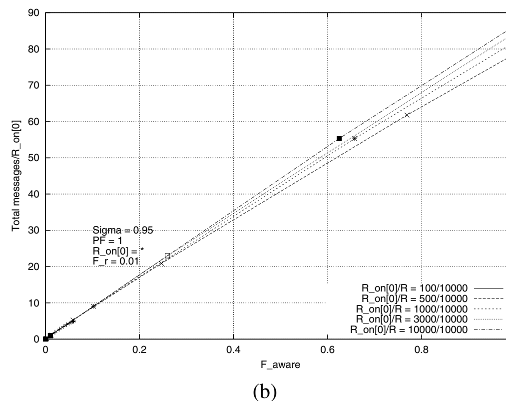
the propagation. Ignoring the fact that peers may go offline throughout the push phase makes the analysis more pessimistic. On the other hand since other approaches do not account for peers going offline, we chose the simple metric of comparing to the initial population size, in order to enable comparisons to related approaches. On the x-axis we will give the percentage of the online peers that have become aware of the update. Since the analysis is made in rounds the plot is discrete, and the marks (points) on the curves indicate the discrete steps. From the number of points on the curves it can be seen how fast the rumor spreads (latency), but our main interest is the communication cost involved in updating all online peers.

5.1. Result 1: Impact of the initial online population size

In this analysis we studied the impact of varying the initial online population for the plain flooding scheme. If the initial population size is too small as compared to the total population, the probability that a peer to which a message is sent is available is too low, and the rumor will not spread. Varying initial online replicas $R_{on}(0)$ between 1 to 100% it is observed in Fig. 1(a) that without a significant initial online population($< 5\%$), it is difficult to make all online peers aware of an update. In case there is a significant initial online population, the message overhead is relatively independent of the online population, as seen in the Fig. 1(b) for a variation from 5-30% of total population. However, message overhead is very high for this plain flooding scheme, around 80 messages per online peer.



(a)



(b)

Figure 1. Varying initial online replicas $R_{on}(0)$ between 1 to 100%

5.2. Result 2: Impact of Varying f_r

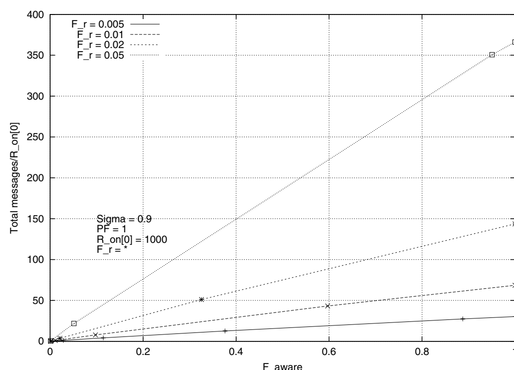


Figure 2. Varying f_r

Since flooding is exponential in nature, a limited fanout is sufficient to spread the update to a complete population. A large fanout will cause unnecessary duplicate messages. Varying f_r it is inferred in Fig. 2 that the intuitive expectations are true, and it is not necessary to push to too many replicas, since it does not significantly enhance the update propagation, however creates eight to ten times more duplicate messages. Thus it is sufficient, and indeed desirable to have a small fanout.

5.3. Result 3: Impact of σ

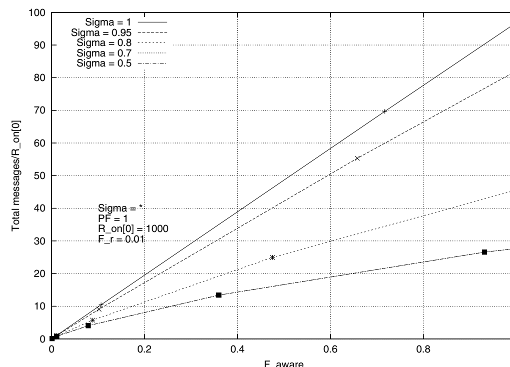


Figure 3. Varying sigma (σ)

Even if the environmental parameter σ (probability of online peers staying online in consecutive push rounds) varies, and is quite low, Fig. 3 demonstrates that the algorithm is quite robust to replicas going offline (without forwarding the update) after receiving the update. Indeed, typically σ will be larger than 0.95. We investigated lower values of σ, because curiously the message overhead decreases significantly if several replicas 'fail' to forward the update.

1063-6927/03 $17.00 © 2003 IEEE

This was an additional reason that prompted us to introduce $P_F(t)$ in our analysis, and is discussed next.

5.4. Result 4: Impact of $P_F(t)$

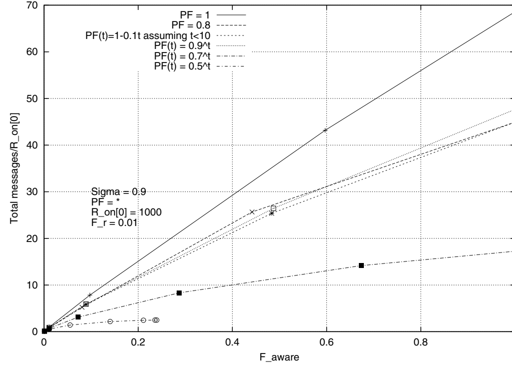


Figure 4. Varying $P_F(t)$

With the progress of the push rounds, a large population will become aware of the update (exponential growth initially), and a very small population will be left unaware. Consecutively, if all newly aware peers decide to continue gossiping, a large number of messages are generated, for a small target audience. Thus even if a small fraction of the newly aware peers gossip, it is sufficient to reach out all uninformed peers, and using a substantially lower number of messages. Fig. 4 indicates that the best strategy is to reduce the probability of forwarding updates with the increase in number of push rounds, which eliminates many unnecessary messages. On the downside, it is essential to properly tune $P_F(t)$, lest the update is not propagated to the whole population. We will briefly describe tuning of $P_F(t)$ in Section 6 for optimisations and self-tuning of parameters in a decentralised manner using only local information.

5.5. Result 5: Scalability

As stated previously, scalability has not been our principle concern with replication factor between 100-1000, but our push scheme also scales well, as observed for a total population varied between 10^4 to 10^8 with $R_{on}/R = 0.1, \sigma = 1, P_F(t) = 0.8*0.7^t+0.2$ and f_r chosen such that to ten online peers a message is sent, i.e., $R_{on} * f_r = 10$. The results are shown in Fig. 5. As may be observed the total number of messages per initially online peer has a decently low value. With the increase in total population, the number of messages per online peer is decreasing, since all parameters have been kept fixed. For a small population, we do not need a fanout of ten online peers, and choosing a smaller fanout increases the number of push rounds but decreases the message overhead as shown in Section 5.2. Thus we conclude that for a very large range of total population, the message overhead can be, with proper choice of fanout, limited to around 20 messages per initial online peer. Given the fact that this is so when there is no knowledge as to which replicas are actually online, and thus the best that can be done is to use ten messages, we think that our simple (look and implementation wise) push algorithm is quite robust, as well as scalable.

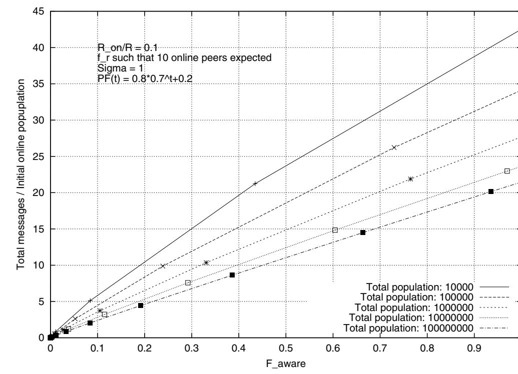


Figure 5. Scalability

5.6. Simple flooding (like in Gnutella) and variants

Since our Push phase algorithm uses Gnutella-like limited message broadcast (flooding with some specific fan out), which is known to have scalability problems [25, 28], it is imperative to point out the improvement achieved by very minor changes, since similar modifications may be made even in the Gnutella message flooding schemes to make it more efficient.

Though flooding (Gnutella) has been amply analysed by many researchers and music file sharing enthusiasts, the ping and pong messages required to establish such a connectivity/neighbourhood are mostly ignored, which makes Gnutella worse. Assume a random distribution for the replicas to stay online, with a probability p_{on}. We have $avg(R_{on}) = p_{on}R$. Then the expected number of peers that are reached in A attempts when actually K replicas are online is $(K * A)/R$. Thus the expected number of attempts to reach S online replicas $E_S(A) = \sum_S^R \frac{R*S*P(K)}{K}$. Assuming peers stay online according to a Poisson process, i.e. $P(K) = \frac{e^{-Rp_{on}}(Rp_{on})^K}{K!}$ we have

$E_S(A) \approx S/p_{on}[1-exp(-R*p_{on}) \sum_0^S (R*p_{on})^K/K!]$

We then use $f_r = E_S(A)/R$. Then the expected messages required in pure flooding (without duplicate avoidance) may be obtained from the geometric sum $1 + (R * f_r) + (R * f_r)^2 + ... + (R * f_r)^{RequiredPushRounds-1}$. In the case of Gnutella like duplicate avoidance, the total number of messages created per update will be exactly the average fanout multiplied by number of peers online, that is to say, there will be on an average f_r messages per online peer and the propagation of update will incur the same latency as in the case if flooding without duplicate avoidance, since duplicate avoidance only reduces the number of redundant messages without any effect on the spread of the update itself.

Recently a variant of pure flooding has been proposed by Haas et.al. [13] called $G(p, t)$ for the "Ad-hoc On Demand Distance Vector (AODV)" routing algorithm. There, for the first t rounds it follows a pure flooding, while in the next rounds $t' > t$, each node decides to continue flooding with a probability p. Simulation results have shown that such an approach reduces message overhead by a quarter to a third, as compared to pure flooding. Since this scheme is strictly a special case of our algorithm, it is obvious that our

IEEE COMPUTER SOCIETY

expected results are supposed to be better. In Table 2 we summarize the comparison in terms of total messages per initially online peer and latency (number of rounds). Our analytical result agrees with the simulation result of [13], as it may be seen in Table 2 that using $G(0.8, 2)$ eliminates substantial unnecessary messages as compared to duplicate avoidance like in Gnutella or even with partial list. However improvements with our scheme are dramatic, either when the whole population is online or when only 10% of them are online, and is significantly better even than $G(p, t)$[13], with a marginal drawback of an additional push round (latency) in each case.

Scheme	$\frac{Msgs}{R_{on}(0)}$	Push rounds
Gnutella	4	7
Using Partial List	3.92	7
Haas et.al.'s G(0.8,2) [13]	3.136	7
Our Scheme, $P_F(t) = 0.9^t$	2.215	8

$R_{on}/R = 10^3/10^3; \sigma = 1; F_r = 0.004(fanout = 4)$

Scheme	$\frac{Msgs}{R_{on}(0)}$	Push rounds
Gnutella	40	5
Using Partial List	35.22	5
Haas et.al.'s G(0.8,2) [13]	28.49	5
Our Scheme, $P_F(t) = 0.8^t$	16.35	6

$R_{on}/R = 10^2/10^3; \sigma = 1; F_r = 0.04$
$(ExpectedEffectiveFanout = 4)$

Table 2. Comparison

In conclusion, what may be argued is that once neighbours are located in Gnutella, there is no need to repeat this exercise. However since this scheme is meant for propagating updates, which are relatively infrequent, and using efficient indexing schemes such that message flooding is not required for searching, it is incorrect to assume that established online replicas continue to stay online. It is primarily this kind of unreliable environment, which had prompted us for our push scheme.

6. Optimisations and self-tuning

Apart from using certain optimisation techniques like directional gossiping [20], we may use certain ad-hoc techniques to reduce the total bandwidth usage.

Foremost we can use an acknowledgement (ack) that replica p sends back to replica f if p receives an update from f. Here p may adopt a policy to reply back only to the first or first k random replica f_1, from which it receives the update. Consequently, f_1 will have better chances to find online replicas in future updates. Since most of the messages are wasted in locating online replicas, this strategy may help. Furthermore, if there are other replicas f_i which had forwarded an update to p, they will assume (from the lack of an ack) that p is offline, and hence may decide not to send future updates, thereby reducing the number of duplicate messages sent to p. This strategy will only be effective for short time intervals, since over a period of time, p is expected to be online according to a random distribution for all the replicas. Moreover it is desirable that f_i again forwards updates to p in remote future since it is possible (quite likely) that f_1 is no more online.

The number of duplicate messages received by a replica p also provides an essential, locally available metric that p may utilise to tune parameters $P_F(t)$ and f_r. (In the case that replicas adopt a policy of sending multiple acks then the number of acks may be used similarly.) Though $P_F(t)$ has been shown to be a deterministic function of t, it can be assigned an ad-hoc value as well without affecting the general inferences drawn from such simplistic functions. Most importantly, $P_F(t)$ should be reduced significantly with increment of t, specially since there are fewer unaware replicas with every t, and hence the need to propagate message is lesser. Another information available to the replicas is the message length $L(t)$ which provides an estimate of the extent of propagation of update message, and hence to tune f_r and P_F.

Similarly, we may significantly decrease the number of Pull messages. It is not necessary for a replica p coming online to instantaneously pull updates. It can wait till it receives update from some replica f and pull updates from f. This saves the unnecessary messages which are otherwise wasted to find an up to date online replica. However this lazy and optimistic approach has a performance tradeoff during queries. This is because if there is a query Q for p, then it will not be able to answer the query (since it is not aware whether it has an up to date information), but instead will itself have to initiate a pull.

7. Related work

Updates in the presence of replication is a widely researched field. This section positions our approach with respect to the research done in the areas of database systems, group communication, and P2P systems. Most of the related work has been done in the context of database systems. Recently, group communication techniques (lazy epidemic algorithms) have been investigated for this purpose as well. Only little work on replication and updates is available from the P2P domain.

7.1. Replication and updates in databases

Several recent approaches exist that attempt to address some of the problems given in Section 2, but cannot meet all requirements. For example, iAnywhere Solutions [10, 17] propose a central server-based scheme for mobile data management with wireless and offline data access. Clearly, such centralised schemes do not suit a totally infrastructure-less environment as we assume. [19] describes hierarchy-less distribution of data, but the approach is confined to highly available sites. [12, 23] propose optimistic replica management schemes in a peer-to-peer way (or using hybrid schemes), and primarily address mobility through reconciliation techniques which may be considered as variants of anti-entropy. They use a pull-based reconciliation scheme which thus exhibits limited consistency guarantees. In Xerox PARC's Bayou project [29], a weakly connected replicated storage system, update conflict management has been

IEEE COMPUTER SOCIETY

addressed through tentative and committed writes to provide best effort consistency, along with anti-entropy based conflict resolution. It is similar to the approach presented in [11]. However, it assumes significantly less replicas, less updates (and hence conflicts), and while the system supports frequent temporary network partitions, it assumes that disconnections are rather short.

Data replication in Mariposa [27] uses economic measures to determine when to replicate data and uses unidirectional periodic reconciliation techniques and rule-based conflict resolution. Other economic paradigms to maintain distributed data replicas include [15, 16, 21] where a primary copy model is used to provide one-copy serializability. These approaches optimise resource usage but inherently assume the availability of the resources and replicas in general. The Ficus [22] replicated Internet-file system tries to scale to large numbers of users and files. It uses optimistic P2P-based file replication based on the assumption that in file sharing systems, conflicts are rare, and can often be resolved.

7.2. Group communication and lazy epidemic schemes

Many conventional database replication schemes and file sharing schemes often use either group communication methods or rumor concepts to propagate updates [6], assuming that such primitives are in themselves robust enough. Group communication primitives typically can tolerate a specific number of faults but are not applicable in such highly unreliable environments that we assume. Even gossip-based approaches, for example, probabilistic broadcast [4], are insufficient, and a hybrid push/pull scheme is required. The novel approach of our work is the use of push/pull in the context of replicas being offline long and frequently, and in the significant reduction of message overhead in the push phase. Our approach may be considered as a generic version of [13].

Randomised rumor spreading algorithms may be categorized [9] by the gossip termination decision criteria used by peers. The first category is defined by whether nodes use feedback from other nodes (for example, whether they already know the rumor or not) and thus decide on their future course, or not (generally called "blind" then). The second category of algorithms uses either probabilistic (coin flipping) or deterministic (counter) measures to determine when to stop. Many rumor spreading algorithms are hybrids of these two categories and results indicate that feedback and counters improve the latency of rumor spreading.

By using the partial random list of replicas to which a rumor has been sent, we are also sending information about replicas hitherto unknown to certain nodes, thus gradually propagating global information, and the idea is similar to work done in the context of resource discovery, called the name dropper scheme [14].

The directional gossiping approach [20] exploits knowledge of the logical connectivity/topology of the system to minimise the number of messages required for update distribution. Unfortunately, this approach cannot be applied in the scenarios we address because replicas go online/offline frequently which changes the topology considerably so that topological knowledge cannot be exploited.

In another analysis of randomised rumor spreading [18], it has been shown that a hybrid push/pull algorithm has performance benefits since push grows fast (quadratically in the beginning, and then exponentially) when there are very few nodes with the rumor, and a very large target audience, while pull is efficient when most nodes already have a rumor, and very few still need it. Their algorithm, besides being very complex assumes continuous availability of all peers, and can tolerate a limited number of permanent failures, but cannot deal with online/offline behaviour at all. However, their hybrid push/pull scheme motivated us to employ the same strategy. In our work we exploit the advantage of push/pull, though there is a subtle difference of objectives. We exploit the exponential nature of push to achieve a rapid spread of updates among online nodes, so that any node coming online later may easily pull the same.

7.3. Peer-to-peer systems

Generally state-of-the-art P2P systems consider the data they offer to be very static or even read-only. Unsurprisingly, most of them thus do not address updates. Typically, centralised (or hierarchical) P2P systems, such as was Napster or now is FastTrack, maintain a centralised index of data items available at online peers. If an update of a data item occurs this means that the peer that holds the item changes it. Subsequent requests would get the new version. However, updates are not propagated to other peers which replicate the item. As a result multiple versions under the same identifier (filename) may co-exist and it depends on the peer that a user contacts whether the latest version is accessed. The same holds true for most decentralised systems such as Gnutella [8].

The Freenet [7] P2P system uses a heuristic strategy to route updates to replicas which is uncertain to guarantee eventual consistency. Searches replicate data along query paths ("upstream"). In the case of an update (which can only be done by the data's owner) the update is routed "downstream" based on a key-closeness relation. Since the routing is heuristic, the network may change, and no precautions are taken to notify peers that come online after an update has occurred, consistency guarantees are limited.

In OceanStore [24] every update creates a new version of the data object (versioning). Consistency is achieved by a two-tiered architecture: A client sends an update to the object's "inner ring" (some replicas who are the primary storage of the object and perform a Byzantine agreement protocol to achieve fault-tolerance and consistency) and some secondary replicas that are mere data caches in parallel. The inner ring commits the update and in parallel an epidemic algorithm distributes the tentative update among the secondary replicas. Once the update is committed, the inner ring multicasts the result of the update down the dissemination tree. To our knowledge analysis of the latency and consistency guarantees for this update scheme has not been published yet.

IEEE COMPUTER SOCIETY

8. Future work

Tuning the push phase may not only be done through feedback mechanisms (to determine when to stop pushing), but also by a speculative (feed-forward) mechanism. In this paper, we have used heuristics to find proper parameters, but we plan to explore the possibility of both feed-back and feed-forward to evolve a proper mechanism of parameter tuning using local knowledge. To verify the correctness of the analysis if some of the simplifying assumptions are relaxed, we plan to use simulations, which will also help us investigate whether there is bimodal[2] behavior [4, 13] even in the assumed environment of very low peer presence. Also the effect of non-uniform online probability of peers needs to be explored. In such a scenario a relatively reliable network backbone would exist and thus would make possible further performance improvements. We plan to use our P-Grid peer-to-peer system as a testbed for the implementation and practical tests of the algorithm.

9. Conclusions

This paper described an efficient, generic push/pull gossiping algorithm for highly unreliable, replicated environments. It provides an analytical model to demonstrate the significant reduction of message overhead using certain optimising techniques (partial lists) and proper tuning of the gossiping (push) phase which in consequence improves the scalability of the algorithm. The analytical model for the gossiping algorithm is a significant contribution in contrast to most of the literature in this area which relies on simulation results. Since our algorithm is generic the analytical model is valid for many of the other variants of flooding algorithms and so are the results of our analysis. We have demonstrated that our algorithm is robust and applicable in unreliable environments such as current peer-to-peer systems. Another major advantage of the algorithm is that it is totally decentralised and uses no global knowledge but exploits local knowledge instead. This makes it suitable for state-of-the-art systems in the P2P, mobility, and ad-hoc networking domains. Finally, it introduces the notion of speculation (feed-forward) into the field of epidemic algorithms.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *CoopIS*, 2001.

[2] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *CIKM*, 2001.

[3] K. Aberer, M. Hauswirth, M. Punceva, and R. Schmidt. Improving data access in P2P systems. *IEEE Internet Computing*, 6(1), 2002.

[4] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *TOCS*, 17(2), 1999.

[5] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *SIGMETRICS*, 2000.

[6] S.-W. Chen and C. Pu. A structural classification of integrated replica control mechanisms. Technical Report CUCS-006-92, Dept. of CS, Columbia Univ., 1992.

[2]Bimodal behaviour denotes a reliability model which corresponds to a family of bimodal probability distributions, i.e., the traditional "all or nothing" guarantee becomes "almost all or almost none."

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, 2001.

[8] Clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.2), 2001. http://www.clip2.com/GnutellaProtocol04.pdf.

[9] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC*, 1987.

[10] E. Giguère. Mobile data management: Challenges of wireless and offline data access. In *ICDE*, 2001.

[11] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In *SIGMOD*, 1996.

[12] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile data access through optimistic peer-to-peer replication. In *Workshop on Mobile Data Access*, 1998.

[13] Z. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. In *INFOCOM*, 2002.

[14] M. Harchol-Balter, T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *PODC*, 1999.

[15] Y. Huang and O. Wolfson. A competitive dynamic data replication algorithm. In *ICDE*, 1993.

[16] Y. Huang and O. Wolfson. Object allocation in distributed databases and mobile computers. In *ICDE*, pages 20–29, 1994.

[17] Y. Huang, O. Wolfson, and A. P. Sistla. Data replication for mobile computers. In *SIGMOD*, 1994.

[18] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *FOCS*, 2000.

[19] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *VLDB*, 2000.

[20] M. J. Lin and K. Marzullo. Directional gossip: Gossip in a wide-area network. In *European Dependable Computing Conference, LNCS*, 1999.

[21] E. Pacitti, P. Minet, and E. Simon. Fast algorithms for maintaining replica consistency in lazy master replicated databases. In *VLDB*, 1999.

[22] T. W. Page, R. G. Guy, J. S. Heidemann, D. Ratner, P. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek. Perspectives on optimistically replicated peer-to-peer filing. *Software—Practice and Experience*, 28(2), 1998.

[23] D. Ratner, G. J. Popek, and P. Reiher. Roam: A scalable replication system for mobile computing. In *Mobility in Databases and Distributed Systems*, 1999.

[24] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5), 2001.

[25] J. Ritter. Why gnutella can't scale. no, really. http://www.darkridge.com/~jpr5/doc/gnutella.html, 2001.

[26] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *ICDCSW*, 2001.

[27] J. Sidell, P. M. Aoki, A. Sah, C. Staelin, M. Stonebraker, and A. Yu. Data replication in mariposa. In *ICDE*, 1996.

[28] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html, 2001.

[29] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP*, 1995.

[30] K. Truelove. Opennap use crashes, Nov. 2001. http://www.openp2p.com/pub/a/p2p/2001/05/11/opennap.html.

1063-6927/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY