

# Epidemic Algorithms in Distributed Computing

J. Ramanand (05329402),  
KReSIT, IIT Bombay

IT630 (Principles and Practices of Distributed Computing)  
Term Paper

April 5, 2006

## Abstract

This term paper briefly surveys the application of **Epidemic and Gossip Algorithms** to the field of Distributed Computing. It begins by explaining the concepts behind these classes of algorithms, their key properties and their perceived utility. It then provides instances of their application in solving various problems in the domain of distributed systems.

## 1 Introduction to Epidemic and Gossip Algorithms

### 1.1 Epidemic Algorithms

Epidemic algorithms model scenarios in which two entities interact, of which one is “infected” and the other is “healthy”. By way of the interaction, the “infection” is passed on to the “healthy” entity which later proceeds to infect other connected entities. Real-life epidemics have been thoroughly studied in the medical field of epidemiology and mathematical models have been constructed in their analysis. The “infection” is usually an information nugget that is available with the infected node but not the healthy node.

### 1.2 Gossip Algorithms

Epidemic algorithms are usually run using the idea of “Gossiping”, in which a node routinely picks another node (usually probabilistically) and exchanges information with it. Gossiping almost always involves only two nodes and hence is akin to a telephone call between two people. The extent of information exchange is dependent on the algorithm and the resources (time, bandwidth) available. The selection of which node to gossip with is largely independent of other knowledge.

### 1.3 Epidemic Communication

In such epidemic scenarios, communication between entities is not deterministic, but is randomised. Communication is initiated by a “rumour-monger” who picks randomly a node and then establishes contact with it. The common variants of communication are:

1. *Anti-entropy*: Each node chooses another node randomly and exchanges all relevant information with the other such that both sides do not have any differences.
2. *Rumour mongering*: A node is interested in communication when it is in possession of a “hot rumour” i.e. a new piece of information. Then, it periodically contacts other sites at random and passes on the latest rumour. Anti-entropy methods are usually reliable but come with heavy communication overheads. Rumour mongering, by exchanging only recent updates, is less expensive.

Both these methods are “push” based solutions. Pull based solutions have also been proposed and studied. In these, a node periodically and randomly contacts other nodes to find out if they have any new or different information from its own.

Important aspects of such methods are:

1. Message passing overheads
2. Termination when the rumour becomes “cold” (i.e. whether all nodes are in possession of it or the rumour is very old)
3. Time taken for the information to reach all nodes
4. Fraction of the uninformed nodes at any given point

## 1.4 Advantages

Epidemic algorithms provide the following advantages [1]:

1. *Probabilistic model*: This helps reason about the spread of information through a system over time
2. *Asynchronous communication pattern*: An effective epidemic communication system allows operation in a ‘fire-and-forget’ mode, where, even if the initial sender fails, all surviving nodes will receive the update (or none will).
3. *Autonomous and decentralized actions*: Epidemics techniques could enable actions to be taken autonomously based on the data received without the need for additional communication
4. *Robust with respect to message loss and node failures*: In well-designed systems, once a message has been received by at least one of the peers of a gossiping process, it is almost impossible to prevent the spread of the information through the system. In some systems, this can be true even in high-loss scenarios.
5. *Rigorous mathematical underpinnings*: Mathematical techniques can be used to reason about the operation of the protocols under all sorts of conditions.
6. *Simplicity in implementation*

## 1.5 Disadvantages

1. Could lead to high communication costs
2. Potential unreliability as to whether information has reached all nodes
3. Analysis can be harder as scenarios are not deterministic

## 2 System Model

From an abstraction point of view for analysis, the problem is usually cast as follows [5]: There are  $n$  players which exchange information in parallel communication rounds. In each round  $t$ , the players are connected by a communication graph  $G_t$  generated by “random phone calls” i.e. there is an edge between two nodes if they participate in a phone call. The creation edge depends on the algorithm used for gossip communication. In the simplest form, each player  $u$  selects a partner  $v$  for communication at random and  $u$  calls  $v$ . This “rumour-mongering” can be initiated by any player in any round and data can be transmitted in both direction along all the edges of the graph

in that round. For modelling simplicity, it is usually assumed that a node participates only in one call in a round.

By gossip theory, assuming there are no redundant calls and an ideal sequence of calls is followed, all rumours can be exchanged to all gossips in  $2n - 4$  calls if  $n > 4$  [2]. If there is a single rumour to be propagated among the gossips, the minimum number of rounds required will be  $O(\log n)$  (assuming a simple binary tree communication pattern among players). However, models in distributed scenarios use a random and probabilistic model of communication with the possibility of process failures thrown in, so despite that, algorithms must try and ensure termination of rumour propagation in  $O(\log n)$  such that all players are informed with high probability.

Attempts to “infect” other players can be dampened over a period of time to reduce network traffic. A player can stop transmission of the infection if he contacts an already infected player. There are several ways to do this: the process can immediately remove itself, or remove itself with a certain probability, or cease transmission after a count of such unnecessary calls exceeds a threshold.

Analysis is usually simplified by focusing on the propagation of a single rumour in the system and is concerned with the lifetime of the rumour and number of transmissions. Though the above formulation seems to indicate a synchronous model, it can be cast into an equivalent asynchronous problem. Reasoning over such models cannot be done deterministically, but involve probabilistic expressions. However, the advantage is that the implementation is simplified.

## 3 Application 1: Replication in Databases

### 3.1 Introduction

The problem is that of replicating a database at many sites. This calls for algorithms to maintain consistency among all sites given updates to the database. Each update is made to a single site and must be propagated to all the other sites eventually. The sites can be fully consistent when all updating activity has stopped and the system has become quiescent. In practice, sites will settle for having most of their information as current. [3] describe an epidemic algorithm to serve these replication needs such that it is efficient, robust and scalable. This algorithm mainly uses a rumour mongering scheme backed by occasional (expensive) anti-entropy exchanges for reliability.

### 3.2 Anti-Entropy scheme

The anti-entropy scheme consists of a site  $s$  contacting site  $s'$  (pull) or being contacted by site  $s'$  (push). A comparison of the database values is made and if it can be inferred that the  $s'.values$  is more recent than  $s.values$ , the  $s'.values$  are copied to  $s$ . The choice of which site to pull from or push to is made randomly. This is an example of a simple epidemic and by epidemiology theory, it is known that expected time for propagating an update to all sites is proportional to  $O(\ln n)$ . The following short analysis shows that a pull mechanism is likely to be better than push.

Let  $p_i$  be the probability of a site being susceptible (i.e. the site has not received the infection yet) after the  $i^{th}$  cycle of anti-entropy. For pull, a site remains susceptible after the  $i + 1^{st}$  cycle if it was susceptible after the  $i^{th}$  cycle and it contacted a fellow susceptible site in the  $i + 1^{st}$  cycle. This can be expressed by:

$$p_{i+1} = p_i * p_i$$

If  $p_i$  is small, this converges very soon.

Now, in case of push, a site is susceptible after the  $i + 1^{st}$  cycle if it was susceptible after the  $i^{th}$  cycle and no infected site contacted it in the current cycle. Hence, the recurrence relation becomes:

$$p_{i+1} = p_i * (1 - 1/n)^{(n(1-p_i))}$$

which converges to 0 but much less rapidly.

### 3.3 Rumour mongering

From a network traffic point of view, an anti-entropy method of comparing database contents is extremely expensive. A rumour spreading model was proposed instead. In this, an update to the database is shared with other sites by the infected sites who make a random phone call to do so. Additionally, when an infected site makes a redundant phone call to an already infected recipient, it loses interest in sharing the rumour. This loss of interest can be dampened gradually or immediately, removing the site from rumour mongering activity. The analysis then shifts to understanding convergence of the rumour to all sites and the fraction of removed processes at quiescence.

[3] show the number of susceptible processes  $s$  is equal to  $e^{-((k+1)(1-s))}$  where a process removes itself from epidemic activity with a probability  $1/k$  if it contacts another infected process. By tuning  $k$ , an optimal solution for minimizing the number of messages as well as reaching convergence can be attempted. This formulation may however mean that a single, hot rumour known at nearly all the sites dies out quickly, so there may be some sites that may not

receive the rumour. However, network traffic is not affected.

The paper also looks at the method of propagating deletes in the database. The above models cannot propagate “absence” by themselves - instead, the deleted entry is likely to be patched in back from another site. To handle this, a “death certificate” is generated and propagated as any other rumour.

[6] present other replication algorithms with pessimistic and optimistic flavours.

### 3.4 Review

This was one of the earliest papers applying epidemic theories to a distributed systems problem, and was influential in showing the possibilities of these algorithms. The paper does not distinguish between different operations such as updates or inserts (except for deletes). In certain replication schemes, it may be important to have different rates at which to gossip depending on the importance of the rumour - the protocol does not seem to address this. There may be expensive bandwidth costs in practices if updates to databases are frequent or large in size.

## 4 Application 2: Failure Detection

### 4.1 Introduction and Protocol

Epidemic algorithms were applied to failure detection to try and improve the problems of poor scalability in existing methods. Accurate failure detection has usually been difficult because sometimes processes may be slow, or the network connection to it may be slow, or a subset of processes may be partitioned from the rest, but they may be flagged as failed. Though false detection of a small number of processes is acceptable if minimal progress has to be made, scaling upto large numbers of processes increases this by a large amount.

[4] present an algorithm based on epidemic/gossiping strategies. In this, each process maintains a list of entries for each process containing the process id/address and a heartbeat counter. Every  $T_{gossip}$  seconds, each member increments its own heartbeat counter, and selects one other member at random to send its list to. Upon receipt of such a gossip message, a member merges the list in the message with its own list, and adopts the maximum heartbeat counter for each member. Each member may occasionally broadcast its list in order to be located initially and also to recover from network partitions. Each member also maintains, for each other member in the list, the last time that

its corresponding heartbeat counter has increased. If the heartbeat counter has not increased for more than  $T_{fail}$  seconds, then the member is considered failed.  $T_{fail}$  is selected so that the probability that anybody makes an erroneous failure detection is less than some small threshold  $P_{mistake}$ .

After a member is considered faulty, it cannot immediately be forgotten about. The problem is that not all members will detect failures at the same time, and thus a member A may receive a gossip about another member B that A has previously detected as faulty. If A had forgotten about B, it would reinstall B in its membership list, since A would think that it was seeing B's heartbeat for the first time. A would continue to gossip this information on to other members, and, in effect, the faulty member B never quite disappears from the membership.

Therefore, the failure detector does not remove a member from its membership list until after  $T_{cleanup}$  seconds ( $T_{cleanup} \leq T_{fail}$ ).  $T_{cleanup}$  is chosen so that the probability that a gossip is received about this member, after it has been detected as faulty, is less than some small threshold  $P_{cleanup}$ .  $P_{cleanup}$  can be made equal to  $P_{fail}$  by setting  $T_{cleanup}$  to  $2 * T_{fail}$ .

The paper presents analysis in an asynchronous setting along with results to show that detection of failures takes time  $O(n \log n)$  and that it is resilient to process failures until about half the members have failed.

## 4.2 Review

This paper's key contribution is in showing how gossip algorithms can enable processes to make autonomous decisions. However, the paper is only restricted to handling stop failures. To its credit, the protocol is very simple and reports seem to indicate good scalability.

## 5 Other Applications

1. Group Membership and Multicast problems
2. Aggregate computations
3. Information exchange in peer-to-peer networks
4. Garbage Collection in distributed systems

## 6 Proposed Novel Application

A possible application of Epidemic algorithms could be to monitor *trustworthiness* of processes. The motivation for maintaining such information about processes is the likelihood of Byzantine Failures in the

system or in case of breaches of security in large distributed settings where a process is affected by malicious viruses that turn it antagonistic to the system. Normal processes could then use the trust scores to determine if data received from a process is to be treated with confidence - if the process has a poor trust score, then the data can be discarded. This would also be relevant if a process has a choice of peers to "pull" data from; using these scores, a ranking of credibility can be determined and hence a preferred sequence of sources can be identified.

The model would be for processes to maintain a score of trust (say from 0 to 1) for each of the other processes in the system. The information can then be exchanged amongst processes using an epidemic/gossip protocol. Initially, the default score would be 1 i.e. the highest. During the course of operation, a process's score about its peers is modified. This is influenced either by exchanging rumours with other peers and learning from their reports of current trustworthiness, or by a process independently changing its opinion about a peer. One way in which this independent decision can be made is by comparing "knowledge" about some known "fact" received from a failed and malicious peer. If the opinions diverge by a value exceeding a threshold, a probabilistic decision to decrease the peer's trust rating could be made. If several processes notice the incorrect or inconsistent behaviour of the failed process, the protocol should ensure that eventually gossiping should propagate and strengthen this view. A potential issue here is to ensure that malicious gossiping from the failed process does not influence the degradation in its trust score in the system.

[To the best of this author's knowledge, such an epidemic based model is not currently available.]

## 7 Conclusions

Epidemic and gossip algorithms have been successfully applied to distributed systems. They promise answers to problems of scalability without adding too much complexity, and seem to be applicable to most classical distributed problems. However, more work needs to be done on defining acceptable bounds on likely errors as these are probabilistic algorithms. This term paper traced through the main concepts involved in these algorithms, provided a couple of seminal research applications in this area, and ended with a proposal for a new application for this class of algorithms.

## References

- [1] Werner Vogels, *History of Epidemics (in Distributed Systems)*, April 2004, <http://www.allthingsdistributed.com/historical/archives/000451.html>.
- [2] B. Baker, R. Shostak, *Gossips and telephones*, Discrete Mathematics 2(1972), pp. 191–193.
- [3] A. Demers, D. Greene, A. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, *Epidemic algorithms for replicated database maintenance*, In Proc. ACM Symp. on the Principles of Distr. Computing, pages 1–12, August 1987.
- [4] Robbert van Renesse, Yaron Minsky, and Mark Hayden, *A Gossip-Based Failure Detection Service*, Proc. of Middleware '98, the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 55–70, Sept 1998.
- [5] R. Karp, C. Schindelhauer, S. Shenker, B. Vocking, *Randomized rumor spreading*, Proc. IEEE Symp. Foundations of Computer Science, 2000.
- [6] D. Agrawal, A. El-Abbadi, and R. Steinke. *Epidemic algorithms in replicated databases*, In Proc. 16th ACM SIGACT-SIGMOD Symp. Princip. of Database Systems (PODS), Tucson, Arizona, May 1997.