Distributed Systems: Theory

Epidemic Protocols - Broadcast

Alberto Montresor
Università di Trento

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Introduction – Randomized protocols

- **Motivations**
  - Existing reliable protocols have scalability problems
  - Randomized protocols may have a smaller overhead
  - Trade-off between reliability and scalability
- **Problems**
  - *No deterministic* broadcast guarantee
  - Only *probabilistic* claims about reliability
- **Can be applied**
  - To large-scale distributed systems (millions of nodes)
  - When full reliability is not required

© Alberto Montresor

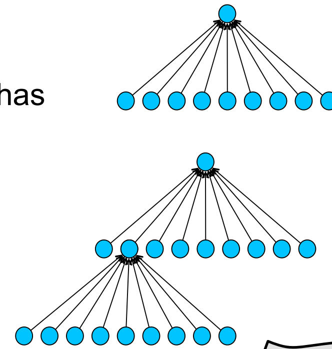
Setting the stage

- **XEROX Clearinghouse Servers**
  - Database replicated at *thousands of nodes*
  - Heterogeneous, *unreliable* network
  - *Independent* updates to single elements of the DB are injected at multiple nodes
  - Updates must propagate to all other nodes or be supplanted by a later updates of that same element
  - Replicas become consistent after no more new updates
  - Assuming a reasonable update rate, most information at any given replica is “current”

© Alberto Montresor

Setting the stage

- **Desirable goals**
  - All replicas converge quickly given no new updates
  - Scales gracefully with number of replicas
  - Efficient use of network
- **Can we use our URB algorithms?**
  - Ack implosion problem
    - Acks needed to be sure that a message has been delivered by all nodes
    - Acks needed for all received messages, not only for messages broadcast



© Alberto Montresor

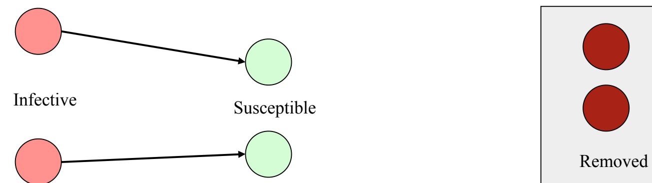
Probabilistic Broadcast Specification

- **PB1 – Probalistic validity:**
  - There is a given probability such that for any two correct processes *p* and *q*, every message PB-broadcast by *p* is eventually PB-delivered by *q*
- **PB2 - Integrity:**
  - *m* is PB-delivered by a process at most once, and only if it was previously PB-broadcast
- **Similar to Best-Effort Broadcast**
  - No agreement property
  - Probability in BEB is “hidden” in process life-cycle
  - Probability in PB is explicit

© Alberto Montresor

Model of epidemics

- *Epidemics* study the spread of a disease or infection in terms of populations of infected/uninfected individuals and their rates of change
- Following the epidemiology literature, we will name a node *n* as:
  - ***S**usceptible* *if n has not yet received an update*
  - ***I**nfective* if *n* holds an update it is willing to share
  - ***R**emoved* if *n* has the update but is no longer willing to share it



© Alberto Montresor

Model of epidemics

- **How does it work?**
  - Initially, a single individual is infective
  - Individuals get in touch with each other, spreading the update
- **Rumor spreading, or gossiping, is based on the same principles**
- **Can we apply the same ideas to distributed systems?**
  - Our goal is to spread the "infection" (update) as fast as possible!

© Alberto Montresor

System model

- **A database that is replicated at a set of *n* nodes $S = \{ s_1, \ldots, s_n \}$**
- **The copy of the database at node *s* can be represented by a time-varying partial function:**
  - $s.value: K \rightarrow (V \cdot T)$
    - *K* set of keys
    - *V* set of values
    - *T* set of timestamps
- **In the following slides**
  - We will omit the key and we will consider only a single key,value pair

© Alberto Montresor

System model

- **For simplicity we will assume a database that stores value and timestamp of a single entry at each node s**
  - `s.value = (v, t)`
- **To indicate a deletion at time t**
  - `s.value = (`**`null`**`, t)`
- **The update operation is formalized as**
  - `s.value ← (v, now())`
  - It is assumed by this work that `now()` is a function returning a globally unique timestamp (no details)
- **So, a pair with a larger timestamp is considered "newer"**

© Alberto Montresor

The goal

When a database is replicated at many sites, maintaining consistency in the presence of updates is a significant problem.

The goal of the update distribution process is to drive the system towards consistency:

$$\forall s, s' \in S : s.\texttt{value} = s'.\texttt{value}$$

And we want to achieve this with algorithms that are efficient, robust and scalable.

© Alberto Montresor

Several algorithms for distributing updates

- **Best effort**
- **Anti-entropy (simple epidemics)**
  - Push
  - Pull
  - Push-pull
- **Rumor mongering (complex epidemics)**
  - Push
  - Pull
  - Push-pull
- **Eager epidemic dissemination**

© Alberto Montresor

Best Effort (Direct Mail)

- **How it works?**
  - Notify ALL other nodes of an update soon after it occurs.
  - When receiving an update, check if it is “news”
- **Node s executes:**

```
upon s.value ← (v, now()) do
   for each s’∈S do
        send(s’, <UPDATE, s.value>)
upon receive <UPDATE, (v, t)> do
     if (s.value.time < t) then
        s.value ← (v, t)
```

- **Not randomized nor epidemic algorithm: just the simplest**

© Alberto Montresor

Anti-entropy

- Every node regularly chooses another node at random and exchanges database contents, resolving differences:
- Node s executes:

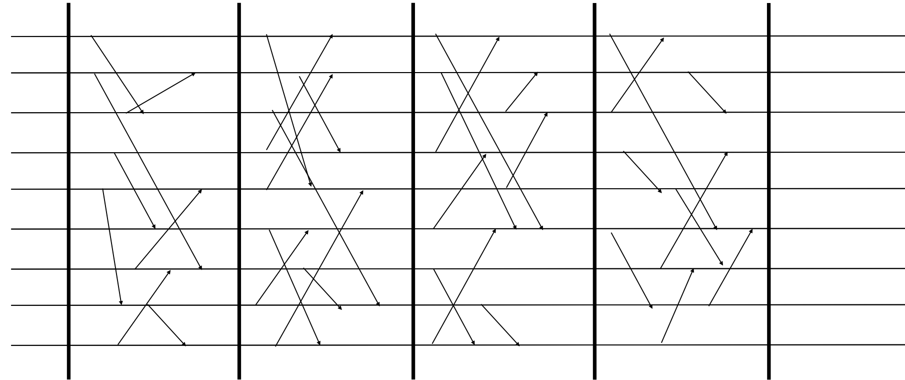
```
repeat periodically every Δ time units
    r = selectPeer(S-{s})     % random choice
    resolveDifference(s,r)
```

- All nodes are either susceptible or infective

© Alberto Montresor

An epidemic cycle

- During a cycle of length Δ, every node has the possibility of contacting one random node



© Alberto Montresor

Implementation of resolveDifference()

- **Push**

```
if (s.value.time > r.value.time )
     r.value ← s.value
```

- **Pull**

```
if (s.value.time < r.value.time )
   s.value ← r.value
```

- **Push-pull**

```
if (s.value.time > r.value.time )
     r.value ← s.value
else
   s.value ← r.value
```

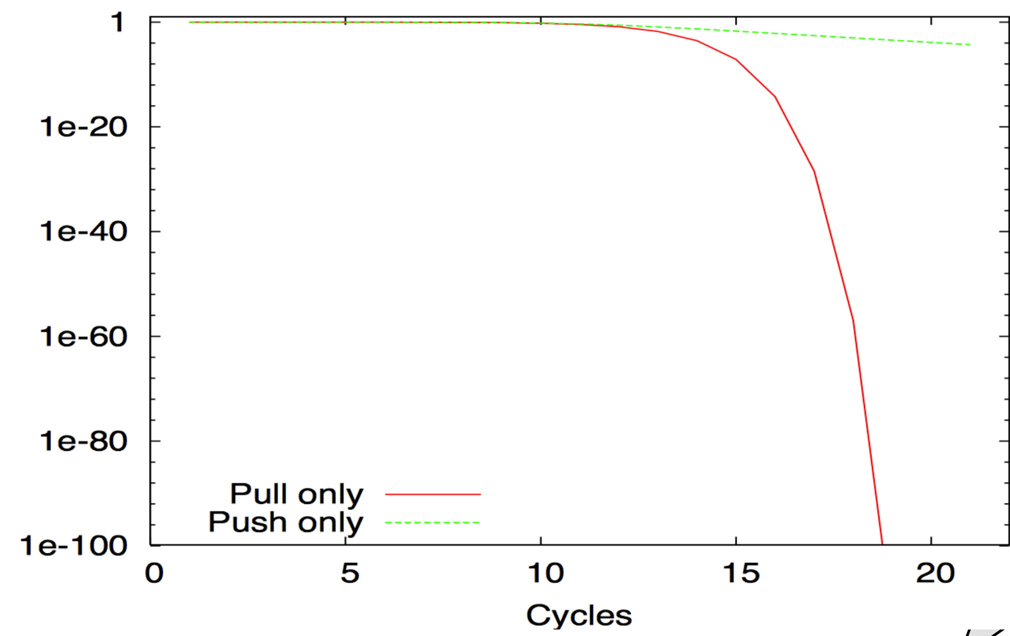
© Alberto Montresor

Anty-entropy: Convergence

- **To analyze convergence, we must consider what happens when only a few nodes remain susceptible**
  - Let *p(i)* be the probability of a node being (remaining) susceptible after the ith anti-entropy cycle.
    - Pull: $p(i+1) = p(i)^2$
    - Push: $p(i+1) = p(i)(1 - 1/n)^{n(1-p(i))}$
      For small p(i), $p(i+1) \sim p(i)/e$
  - Both starts to converge to 0, but pull is much more rapid, so in practice pull is used.
- **Push-pull: both mechanisms are used, convergence is even more rapid**

© Alberto Montresor

Push vs pull



© Alberto Montresor

Anti-entropy: Comments

- **Benefits:**
  - Simple epidemics eventually “infect” all the population
  - For a push implementation, the expected time to infect everyone is $log_2(n) + ln(n)$
- **Drawbacks:**
  - Propagates updates much slower than direct mail (best effort)
  - Requires examining contents of database even when most data agrees, so it cannot practically be used too often
- **Normally used as support for best effort, i.e. left running in the background**

© Alberto Montresor

Anti-entropy: Optimizations

- **To avoid expensive databases checks:**
  - Maintain checksum, compare databases if checksums unequal
  - Maintain recent update lists for time T, exchange lists first
  - Maintain inverted index of database by timestamp; exchange information in reverse timestamp order, incrementally re-compute checksums
- **Note:**
  - These optimizations apply to the update problem for large DB
  - We will see how the same principle (anti-entropy) may be used for several other kind of applications

© Alberto Montresor

Rumor mongering

- Nodes initially "*ignorant*" or "susceptive"
- When a node receives a new update it becomes a "*hot rumor*" and the node "*infective*"
- A node that has a rumor periodically chooses randomly another node to spread the rumor
- Eventually, a node will "*lose interest*" in spreading the rumor and becomes "removed"
  - Spread too many times
  - Everybody knows it
- A sender can hold (and transmit) a list of infective updates rather than just one.

© Alberto Montresor

Rumor mongering: loss of interest

- **Counter vs. coin (random)**
  - Coin (random): lose interest with probability *1/k*
  - Counter: lose interest after *k* contacts
- **Feedback vs blind**
  - Feedback: lose interest only if the recipient knows the rumor.
  - Blind: lose interest regardless of the recipient.

© Alberto Montresor

Rumor mongering

- **How fast does the system converge to a state where all nodes are not infective? (inactive state)**
  - Eventually, everybody will lose interest
- **Once in this state, what is the fraction of nodes that know the rumor?**
  - The rumor may stop before reaching all nodes

© Alberto Montresor

Rumor mongering: analysis

- **Analysis from epidemics theory (not only computer science)**
- **Feedback, coin**
  - Let s, i and r denote the fraction of susceptible, infective, and removed nodes respectively. Then:

    $s + i + r = 1$
    $ds/dt = -si$
    $di/dt = +si - (1/k)(1 - s)i$

  - Solving the equations: $s = e^{-(k+1)(1-s)}$
    - Thus, increasing k we can make sure that most nodes get the rumor, exponentially better

© Alberto Montresor

Quality measures

- **Residue.**
  - The nodes that remain susceptible when the epidemic ends: value of s when i = 0
  - Residue must be as small as possible
- **Traffic.**
  - The average number of database updates sent between nodes
  - *m = total update traffic / # of nodes*
- **Delay. We can define 2 delays:**
  - t_{avg} : average time it takes for the introduction of an update to reach a node.
  - t_{last} : time it takes for the last node to get the update.

© Alberto Montresor

Simulation results

Using feedback and counter

Counter k	Residue s	Traffic m	Convergence	
			t_{avg}	t_{last}
1	0.176	1.74	11.0	16.8
2	0.037	3.30	12.1	16.9
3	0.011	4.53	12.5	17.4
4	0.0036	5.64	12.7	17.5
5	0.0012	6.68	12.8	17.7

Using blind and random

Counter k	Residue s	Traffic m	Convergence	
			t_{avg}	t_{last}
1	0.960	0.04	19	38
2	0.205	1.59	17	33
3	0.060	2.82	15	32
4	0.021	3.91	14.1	32
5	0.008	4.95	13.8	32

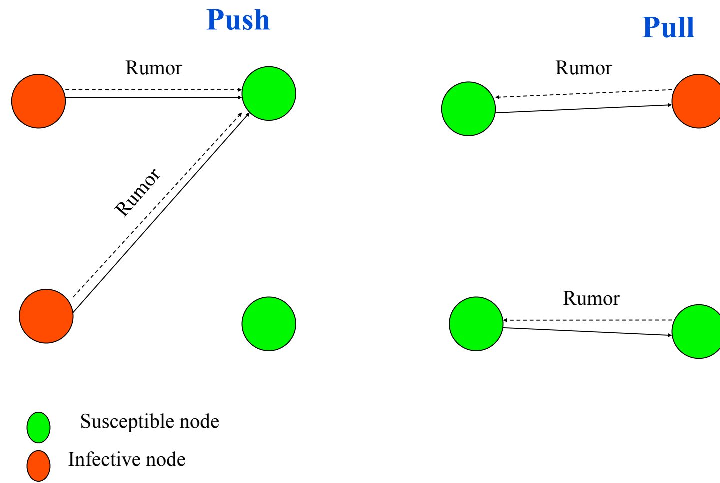
© Alberto Montresor

Push and pull

- **Push (what we have assumed so far)**
  - If database becomes quiescent, this scheme stops trying to introduce updates.
  - If there are many independent updates, more likely to introduce unnecessary messages.
- **Pull**
  - If many independent updates, pull is more likely to find a source with a non-empty rumor list
  - But if database quiescent, it spends time doing unnecessary update requests.

© Alberto Montresor

Push and pull



© Alberto Montresor

Push and pull

- **Empirically, in the database system of the authors (frequent updates)**
  - Pull has a better residue/traffic relationship than push
- **Performance of pull epidemic on 1000 nodes using feedback & counters**

Counter k	Residue s	Traffic m	Convergence	
			t_{avg}	t_{last}
1	0.031	2.70	9.97	17.63
2	0.00058	4.49	10.07	15.39
3	0.000004	6.09	10.08	14.00

© Alberto Montresor

Mixing with anti-entropy

- **Rumor Mongering**
  - spreads updates fast with low traffic.
  - however, there is still a nonzero probability of nodes remaining susceptible after the epidemic
- **Anti-entropy**
  - can be run (infrequently) in the background to ensure all nodes eventually get the update with probability 1.
  - Since a single rumor that is already known by most nodes dies out quickly

© Alberto Montresor

Deletion and death certificates

- **Deletion**
  - We cannot delete an entry just by removing it from a node - the absence of the entry is not propagated.
  - If the entry has been updated recently, there may still be an update traversing the network!
- **Death certificate**
  - Solution: replace the deleted item with a *death certificate* that has a timestamp and spreads like an ordinary update.

© Alberto Montresor

Deletion and death certificates

- **Problem:**
  - we must, at some point, delete DCs or they may consume significant space.
- **Strategy 1:**
  - retain each DC until all nodes have received it. Requires a protocol to determine which nodes have it and to handle node failures
- **Strategy 2:**
  - hold DCs for some time (e.g. 30 days) and discard them. Pragmatic approach, still have the "resurrection" problem; increasing the time requires more space.

© Alberto Montresor

Dormant certificates

- **Observation:**
  - if a DC is older than the time it takes to propagate it to all nodes, it is highly unlikely anyone still has an old copy.
  - So, we can delete very old DCs but retain only a few “dormant” copies in some nodes.
  - If an obsolete update reaches a dormant DC, it is “awakened” and re-propagated.
- **Analogy with epidemiology:**
  - the awakened DC is like an antibody triggered by an immune reaction

© Alberto Montresor

Dormant certificates

- **Formalization**
  - When creating a death certificate, randomly choose *r* retention nodes, and propagate the DC just like an update
  - All nodes keep the DCs until a time t_1 has elapsed since its creation
  - After t_1 is reached, nodes not in the retention list delete the DCs
  - After $t_1 + t_2$ time has elapsed, all DCs are discarded

© Alberto Montresor

Dormant Certificates in Anti-Entropy

- **A dormant DC**
  - Can be activated by setting its timestamp to the current clock
  - But if there is a reinsertion of the entry in between, it would be cancelled!
- **Store a second timestamp (*activation timestamp*) with the DC, leaving original timestamp unchanged.**
  - A DC still cancels an entry that is older than its original timestamp
  - A DC can be removed or put to sleep when its activation timestamp grows older than t_1 and deleted after $t_1 + t_2$
  - Anti-entropy only propagates non-dormant DCs

© Alberto Montresor

Dormant Certificates in Rumor Mongering

- **Activation timestamps work just as well**
  - A DC is created as an active rumor
  - When awakening a dormant DC, it is set as an active rumor again
  - Its activation timestamp is set to the current time
  - The regular rumor spreading mechanism then distributes the DC again

© Alberto Montresor

Spatial Distribution

- **In the previous exposition**
  - The network has been considered uniform (i.e. all nodes equally reachable)
- **In reality**
  - More expensive to send updates to distant nodes
  - Especially if a critical link needs to be traversed
  - Traffic can clog these links

© Alberto Montresor

Epidemics vs probabilistic broadcast

- **How to broadcast messages?**
  - Anti-entropy:
    - DB is a collection of message received
  - Rumor mongering
    - Updates = Message broadcast

© Alberto Montresor