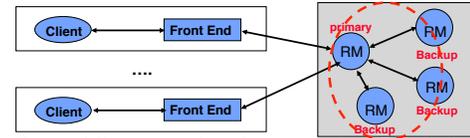


CSE 486/586 Distributed Systems Gossiping

Steve Ko
Computer Sciences and Engineering
University at Buffalo

CSE 486/586, Spring 2013

Recall: Passive Replication

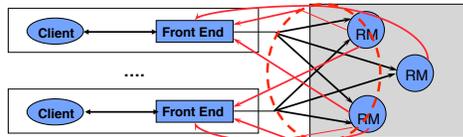


- **Request Communication:** the request is issued to the primary RM and carries a unique request id.
- **Coordination:** Primary takes requests atomically, in order, checks id (resends response if not new id.)
- **Execution:** Primary executes & stores the response
- **Agreement:** If update, primary sends updated state/result, req-id and response to all backup RMs (1-phase commit enough).
- **Response:** primary sends result to the front end

CSE 486/586, Spring 2013

2

Recall: Active Replication



- **Request Communication:** The request contains a unique identifier and is multicast to all by a reliable totally-ordered multicast.
- **Coordination:** Group communication ensures that requests are delivered to each RM in the same order (but may be at different physical times!).
- **Execution:** Each replica executes the request. (Correct replicas return same result since they are running the same program, i.e., they are replicated protocols or replicated state machines)
- **Agreement:** No agreement phase is needed, because of multicast delivery semantics of requests
- **Response:** Each replica sends response directly to FE

CSE 486/586, Spring 2013

3

Eager vs. Lazy

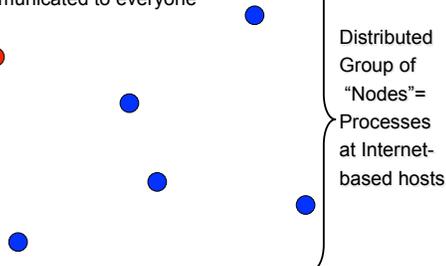
- Eager replication, e.g., B-multicast, R-multicast, etc. (previously in the course)
 - Multicast request to all RMs immediately in active replication
 - Multicast results to all RMs immediately in passive replication
- Alternative: **Lazy replication**
 - Allow replicas to converge eventually and lazily
 - Propagate updates and queries lazily, e.g., when network bandwidth available
 - FEs need to wait for reply from only one RM
 - Allow other RMs to be disconnected/unavailable
 - May provide weaker consistency than sequential consistency, but **improves performance**
- Lazy replication can be provided by using the **gossiping**

CSE 486/586, Spring 2013

4

Revisiting Multicast

Node with a piece of information to be communicated to everyone

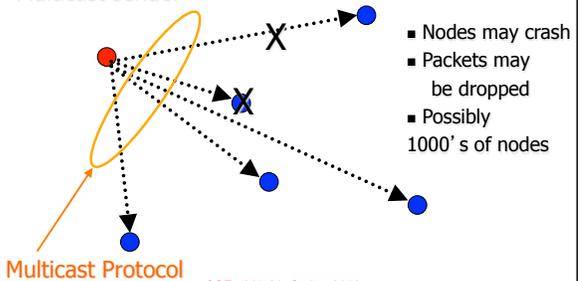


CSE 486/586, Spring 2013

5

Fault-Tolerance and Scalability

Multicast sender



Multicast Protocol

CSE 486/586, Spring 2013

6

B-Multicast

- Simplest implementation
- Problems?

CSE 486/586, Spring 2013 7

R-Multicast

- Stronger guarantees
- Overhead is quadratic in N

CSE 486/586, Spring 2013 8

Any Other?

- E.g., tree-based multicast

- e.g., IPmulticast, SRM, RMTP, TRAM, TMTP
- Tree setup and maintenance
- Problems?

CSE 486/586, Spring 2013 9

Another Approach

Multicast sender

CSE 486/586, Spring 2013 10

Another Approach

Periodically, transmit to b random targets

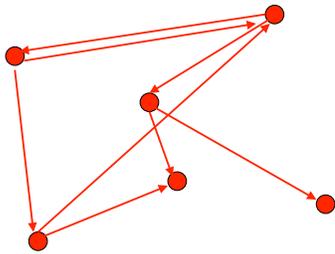
CSE 486/586, Spring 2013 11

Another Approach

Other nodes do same after receiving multicast

CSE 486/586, Spring 2013 12

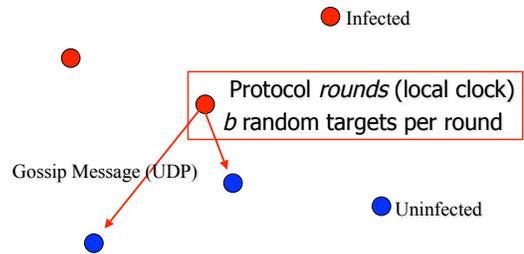
Another Approach



CSE 486/586, Spring 2013

13

“Gossip” (or “Epidemic”) Multicast



CSE 486/586, Spring 2013

14

Properties

- Lightweight
- Quick spread
- Highly fault-tolerant
- Analysis from old mathematical branch of *Epidemiology* [Bailey 75]
- Parameters c, b :
 - c for determining rounds: $(c \cdot \log(n))$, b : # of nodes to contact
 - Can be small numbers independent of n , e.g., $c=2; b=2$;
- Within $c \cdot \log(n)$ rounds, [low latency]
 - all but $\frac{1}{n^{cb-2}}$ of nodes receive the multicast [reliability]
 - each node has transmitted no more than $c \cdot b \cdot \log(n)$ gossip messages [lightweight]

CSE 486/586, Spring 2013

15

Fault-Tolerance

- Packet loss
 - 50% packet loss: analyze with b replaced with $b/2$
 - To achieve same reliability as 0% packet loss, takes twice as many rounds
- Node failure
 - 50% of nodes fail: analyze with n replaced with $n/2$ and b replaced with $b/2$
 - Same as above

CSE 486/586, Spring 2013

16

Fault-Tolerance

- With failures, is it possible that the epidemic might die out quickly?
- Possible, but improbable:
 - Once a few nodes are infected, with high probability, the epidemic will not die out
 - So the analysis we saw in the previous slides is actually behavior *with high probability* [Galey and Dani 98]
- The same applicable to:
 - Rumors
 - Infectious diseases
 - A worm such as Blaster
- Some implementations
 - Amazon Web Services EC2/S3 (rumored)
 - Usenet NNTP (Network News Transport Protocol)

CSE 486/586, Spring 2013

17

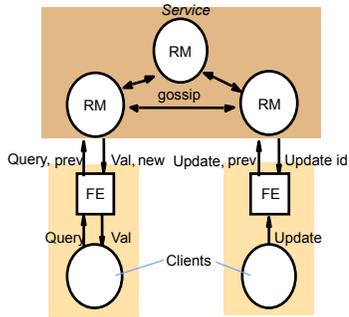
Gossiping Architecture

- The RMs exchange “gossip” messages
 - Periodically and amongst each other.
 - Gossip messages convey updates they have each received from clients, and serve to achieve convergence of all RMs.
- Objective: provisioning of highly available service. Guarantee:
 - **Each client obtains a consistent service over time:** in response to a query, an RM may have to wait until it receives “required” updates from other RMs. The RM then provides client with data that at least reflects the updates that the client has observed so far.
 - **Relaxed consistency among replicas:** RMs may be inconsistent at any given point of time. Yet all RMs **eventually** receive all updates and they apply updates with ordering guarantees. Can be used to provide sequential consistency.

CSE 486/586, Spring 2013

18

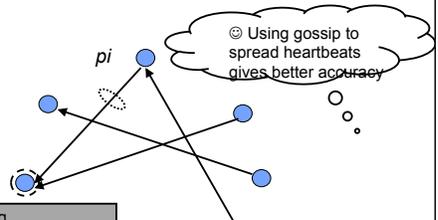
Gossip Architecture



CSE 486/586, Spring 2013

19

Using Gossip for Failure Detection: Gossip-style Heartbeating

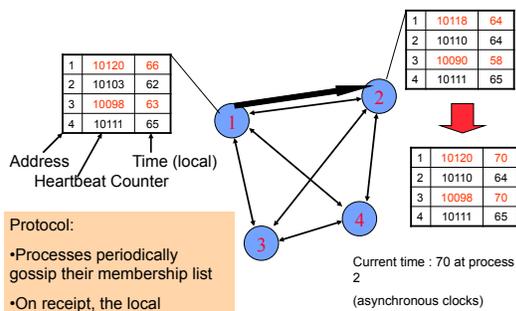


- All-to-all heartbeating
- Each process sends out heartbeats to every other process
 - Con: Slow process/link causes false positives

586, Spring 2013

20

Gossip-Style Failure Detection



CSE 486/586, Spring 2013

21

Gossip-Style Failure Detection

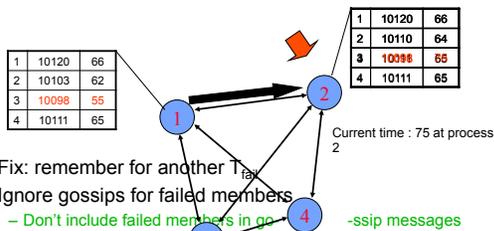
- If the heartbeat has not increased for more than T_{fail} seconds (according to local time), the member is considered failed
- But don't delete it right away
- Wait another $T_{cleanup}$ seconds, then delete the member from the list

CSE 486/586, Spring 2013

22

Gossip-Style Failure Detection

- What if an entry pointing to a failed process is deleted right after T_{fail} seconds?



CSE 486/586, Spring 2013

23

Summary

- Eager replication vs. lazy replication
 - Lazy replication propagates updates in the background
- Gossiping
 - One strategy for lazy replication
 - High-level of fault-tolerance & quick spread
- Another use case for gossiping
 - Failure detection

CSE 486/586, Spring 2013

24

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).