Topics in Database Systems: Data Management in Peer-to-Peer Systems

Διαδικαστικά

- Αύριο, **Τετάρτη 18 Μαΐου 12:00 – 13:00 Ομιλία σε p2p**
- *Ερώτηση*: Τα δυο ποιο ενδιαφέροντα (κατά τη γνώμη σας) ερωτήματα που έθεσε ο ομιλητής

- "Feedback" για το άρθρο σας την επόμενη Τετάρτη 25/5, 3:00 – 6:00 αλφαβητικά

Την Παρασκευή (26/5) θα ανακοινώσω και την μερική σας βαθμολογία – *ελέγξτε αν μου έχετε στείλει όλες τις παρουσιάσεις*

Topics in Database Systems: Data Management in Peer-to-Peer Systems

Διαδικαστικά

- 2 ακόμα Μαθήματα

24/5 και

31/5 *ερωτήματα σε p2p*

Θα ανακοινώσω αύριο τις παρουσιάσεις για τις 24/5 (κάποιοι παράγραφοι)

Topics in Database Systems: Data Management in Peer-to-Peer Systems

Replication II

Replication Policy

- How many copies
- Where (owner, path, random path)

Update Policy

- Synchronous vs Asynchronous
- Master Copy

Replication II:
Epidemic Algorithms

Methods for spreading updates:

Push: originate from the site where the update appeared

To reach the sites that hold copies

Pull: the sites holding copies contact the master site

Expiration times

A. Demers et al, Epidemic Algorithms for Replicated Database Maintenance, SOSP 87

Update at a single site

Randomized algorithms for distributing updates and driving replicas towards consistency

Ensure that the effect of every update is *eventually* reflected to all replicas:

Sites become fully consistent only when all updating activity has stopped and the system has become quiescent

Analogous to epidemics

Methods for spreading updates:

Direct mail: each new update is immediately mailed from its originating site to all other sites

- Timely reasonably efficient
- Not all sites know all other sites
- Mails may be lost

Anti-entropy: every site regularly chooses another site *at random* and by exchanging content resolves any differences between them

- Extremely reliable but requires exchanging content and resolving updates
- Propagates updates much more slowly than direct mail

Methods for spreading updates:

Rumor mongering:

- Sites are initially "ignorant"; when a site receives a new update it becomes a "hot rumor"
- While a site holds a hot rumor, it *periodically* chooses another site at random and ensures that the other site has seen the update
- When a site has tried to share a hot rumor with *too many sites* that have already seen it, the site stops treating the rumor as hot and *retains the update* without propagating it further

Rumor cycles can be more frequent that anti-entropy cycles, because they require fewer resources at each site, but there is a chance that an update will not reach all sites

Anti-entropy and rumor spreading are examples of epidemic algorithms

Three types of sites:

- Infective: A site that holds an update that is willing to share is hold
- Susceptible: A site that has not yet received an update
- Removed: A site that has received and update but is no longer willing to share

Anti-entropy: simple epidemic where all sites are always either infective or susceptible

How to choose partners

Consider spatial distributions in which the choice tends to favor nearby servers

A set S of n sites, each storing a copy of a database

The *database copy* at site $s \in S$ is a time varying partial function

s.ValueOf: $K \rightarrow \{u:V \times t:T\}$

where K is a set of keys, V a set of values, T a set of timestamps

T is totally ordered by <

V contains the element NIL

s.ValueOf[k] = {NIL, t} means that item with k has been deleted from the database

Assume, just one item

s.ValueOf $\in \{u:V \times t:T\}$

thus, an ordered pair consisting of a value and a timestamp

The first component may be NIL indicating that the item was deleted by the time indicated by the second component

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

$\forall s, s' \in S$: s.ValueOf = s'.ValueOf

Operation invoked to update the database

Update[u:V] s.ValueOf {r, Now{()}

Direct Mail

At the site s where an update occurs:

For each $s' \in S$

PostMail[to:s', msg("Update", s.ValueOf)

Each site s' receiving the update message: ("Update", (u, t))

If s'.ValueOf.t < t

s'.ValueOf $\leftarrow$ (u, t)

- The complete set S must be known to s
- PostMail messages are queued so that the server is not delayed, but may fail when queues overflow or their destination are inaccessible for a long time
- n (number of sites) messages per update
- traffic proportional to n and the average distance between sites

Anti-Entropy

At each site s periodically execute:

For some $s' \in S$

ResolveDifference[s, s']

Three ways to execute ResolveDifference:

Push

If s.Valueof.t > s'.Valueof.t

s'.ValueOf $\leftarrow$ s.ValueOf

Pull

If s.Valueof.t < s'.Valueof.t

s'.ValueOf $\leftarrow$ s.ValueOf

Push-Pull

s.Valueof.t > s'.Valueof.t $\Rightarrow$ s'.ValueOf $\leftarrow$ s.ValueOf

s.Valueof.t < s'.Valueof.t $\Rightarrow$ s.ValueOf $\leftarrow$ s'.ValueOf

Anti-Entropy

Assume that

- Site s' is chosen uniformly at random from the set S
- Each site executes the anti-entropy algorithm once per period

It can be proved that

- An update will eventually infect the entire population
- Starting from a single affected site, this can be achieved in time proportional to the log of the population size

Anti-Entropy

Let p_i be the probability of a site remaining susceptible after the i cycle of anti-entropy

For pull,

A site remains susceptible after the i+1 cycle, if (a) it was susceptible after the i cycle and (b) it contacted a susceptible site in the i+1 cycle

$$p_{i+1} = (p_i)^2$$

For push,

A site remains susceptible after the i+1 cycle, if (a) it was susceptible after the i cycle and (b) no infectious site choose to contact in the i+1 cycle

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

Pull is preferable than push

Complex Epidemics: Rumor Spreading

There are n individuals initially inactive (susceptible)

We plant a rumor with one person who becomes active (infective), phoning other people at random and sharing the rumor

Every person bearing the rumor also becomes active and likewise shares the rumor

When an active individual makes an unnecessary phone call (the recipient already knows the rumor), then with *probability 1/k* the active individual loses interest in sharing the rumor (becomes removed)

We would like to know:

- How fast the system converges to an inactive state (no one is infective)
- The percentage of people that know the rumor when the inactive state is reached

Complex Epidemics: Rumor Spreading

Let s, i, r be the fraction of individuals that are susceptible, infective and removed

$s + i + r = 1$

$ds/dt = - s\,i$

$di/dt = si - 1/k(1-s)\,I$

$$s = e^{-(k+1)(1-s)}$$

An exponential decrease with s

For k = 1, 20% miss the rumor

For k = 2, only 6% miss it

Criteria to characterize epidemics

Residue

The value of s when i is zero, that is, the remaining susceptible when the epidemic finishes

Traffic

m = Total update traffic / Number of sites

Delay

Average delay (t_{avg}) is the difference between the time of the initial injection of an update and the arrival of the update at a given site averaged over all sites

The delay until (t_{last}) the reception by the last site that will receive the update during an epidemic

Simple variations of rumor spreading

Blind vs. Feedback

Feedback variation: a sender loses interest only if the recipient knows the rumor

Blind variation: a sender loses interest with probability 1/k regardless of the recipient

Counter vs. Coin

Instead of losing interest with probability 1/k, we can use a counter so that we loose interest only after k unnecessary contacts

$$s = e^{-m}$$

There are nm updates sent

The probability that a single site misses all these updates is $(1 - 1/n)^{nm}$

Counters and feedback improve the delay, with counters playing a more significant role

Simple variations of rumor spreading

Push vs. Pull

Pull converges faster

If there are numerous independent updates, a pull request is likely to find a source with a non-empty rumor list

If the database is quiescent, the push phase ceases to introduce traffic overhead, while the pull continues to inject useless requests for updates

Counter, feedback and pull work better

Minimization

Use a push and pull together, if both sites know the update, only the site with the smaller counter is incremented

Connection Limit

A site can be the recipient of more than one push in a cycle, while for pull, a site can service an unlimited number of requests

Push gets better

Pull gets worst

Hunting

If a connection is rejected, then the choosing site can "hunt" for alternate sites

Complex Epidemic and Anti-entropy

Anti-entropy can be run infrequently to back-up a complex epidemic, so that every update eventually reaches (or is suspended at) every site

Deletion and Death Certificates

Replace deleted items with death certificates which carry timestamps and spread like ordinary data

When old copies of deleted items meet death certificates, the old items are removed.

But when to delete death certificates?

Dormant Death Certificates

If the death certificate is older than the expected time required to propagate it to all sites, then the existence of an obsolete copy of the corresponding data item is unlikely

Delete very old certificates at most sites, retaining "*dormant*" copies at only a few sites (like antibodies)

Two thresholds, t1 and t2

+ a list of r retention sites names with each death certificate (chosen at random when the death certificate is created)

Once t1 is reached, all servers but the servers in the retention list delete the death certificate

Dormant death certificates are deleted when t1 + t2 is reached

Anti-Entropy with Dormant Death Certificates

Whenever a dormant death certificate encounters an obsolete data item, it must be "activated"

Spatial Distribution

The cost of sending an update to a nearby site is much lower that the cost of sending the update to a distant site

Favor nearby neighbors

Trade off between: Average traffic per link and Convergence times

Example: linear network, only nearest neighbor: O(1) and O(n) vs uniform random connections: O(n) and O(log n)

Issue: determine the probability of connecting to a site at distance d

For spreading updates on a line, d^{-2} distribution: the probability of connecting to a site at distance d is proportional to d^{-2}

In general, each site s independently choose connections according to a distribution that is a function of $Q_s(d)$, where $Q_s(d)$ is the cumulative number of sites at distance d or less from s

Spatial Distribution and Anti-Entropy

Extensive simulation on the actual topology with a number of different spatial distributions

A different class of distributions less sensitive to sudden increases of $Q_s(d)$

Let each site s build a list of the other sites sorted by their distances from s

Select anti-entropy exchange partners from the sorted list according to a function f(i), where i is its position on the list

(averaging thee probabilities of selecting equidistant sites)

Non-uniform distribution induce less overload on critical links

Spatial Distribution and Rumors

Anti-entropy converges with probability 1 for a spatial distribution such that for every pair (s', s) of sites there is a nonzero probability that s will choose to exchange data with s'

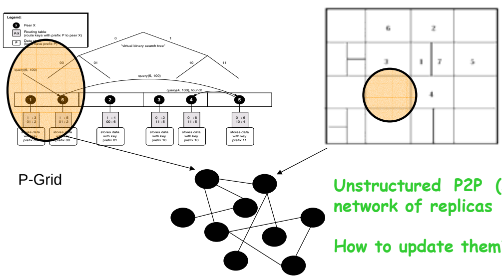
However, rumor mongering is less robust against changes in spatial distributions and network topology

As the spatial distribution is made less uniform, we can increase the value of k to compensate

Replication II:
A Push&Pull Algorithm

Updates in Highly Unreliable, Replicated Peer-to-Peer Systems [Datta, Hauwirth, Aberer, ICDCS04]

Replication in P2P systems

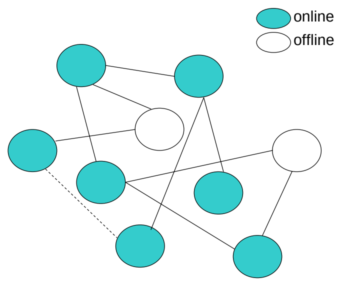


Unstructured P2P (sub-) network of replicas

How to update them?

Updates in replicated P2P systems

- P2P system's search algorithm will find a random online replica responsible for the key being searched.
- The replicas need to be consistent (ideally)
- Probabilistic guarantee: Best effort!



Updates in Highly Unreliable, Replicated Peer-to-Peer Systems [Datta, HauwirthAberer, ICDCS04]

Problems in real-world P2P systems

- All replicas need to be informed of updates.
- Peers have low online probabilities and quorum can not be assumed.
- Eventual consistency is sufficient.
- Updates are relatively infrequent compared to queries.
- Communication overhead, latency and percentage of replicas getting updates determine the critical metrics for performance.

Updates in Highly Unreliable, Replicated Peer-to-Peer Systems [Datta, HauwirthAberer, ICDCS04]

Problems in real-world P2P systems (continued)

- Replication factor is substantially higher than what is assumed for distributed databases.
- Connectivity among replicas is high.
- Connectivity graph is random.

Updates in Highly Unreliable, Replicated Peer-to-Peer Systems [Datta, HauwirthAberer, ICDCS04]

Update Propagation combines

- A push phase is initiated by the originator of the update that pushes the new update to a subset of responsible peers it knows, which in turn propagate it to responsible peers they know, etc (similar to flooding with TTL)

- A pull phase is initiated by a peer that needs to update its copy. For example, because (a) it was offline (disconnected) or (b) has received a pull request but is not sure that it has the most up-to-date copy

Push and pull are consecutive, but may overlap in time

Algorithms

Push:

- If replica p gets Push(U, V, Rf, t) for a new (U, V) pair
  - Define Rp= random subset (of size R*fr) of replicas known to p
  - With probability PF(t): Push(U, V, Rf $\cup$ Rp, t+1) to Rp \ Rf

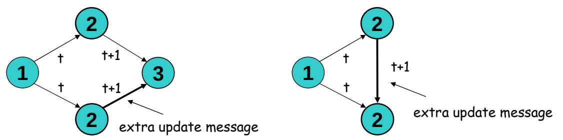
U item

V version

t (counter, similar to TTL)

Rf partial list of peers that have received the update

Selective Push



avoid parallel redundant update: messages are propagated only with probability PF < 1 and to a fraction of the neighbors

avoid sequential redundant update: partial lists of informed neighbors are transmitted with the message

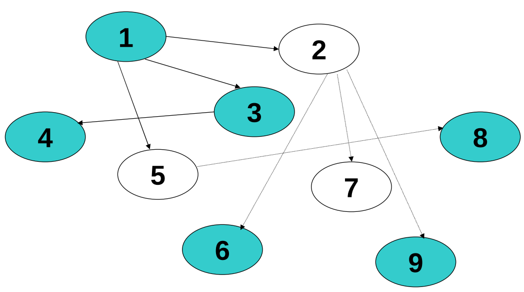
Algorithms

Pull:

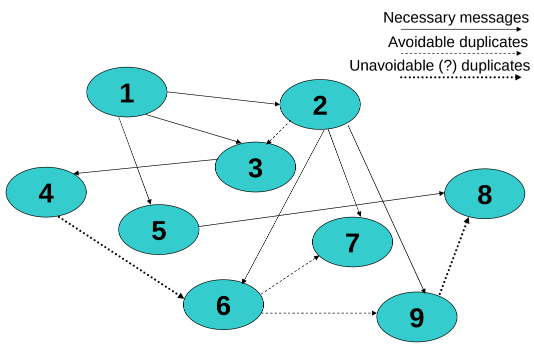
- If p coming online, or got no Push for time T
  - Contact online replicas
  - Pull updates based on version vectors

Strategy: Push update to online peers asap, such that later, all online peers always have update (possibly pulled) w.h.p.

Scenario1: Dynamic topology

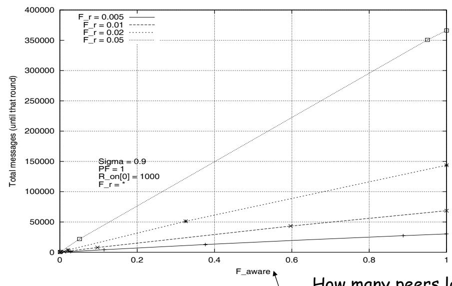


Scenario2: Duplicate messages



Results: Impact of varying fanout

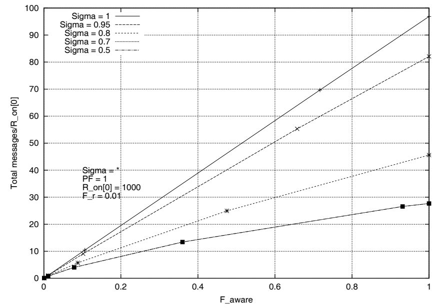
A limited fanout (fr) is sufficient to spread the update, since flooding is exponential. A large fanout will cause unnecessary duplicate messages



How many peers learn about the update

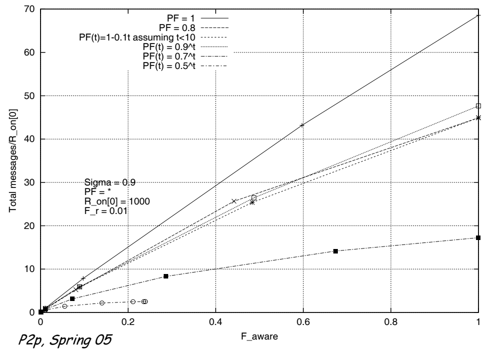
Results: Impact of probability of peer staying online in consecutive push rounds

Sigma (σ) probability of online peers staying online in consecutive push rounds:



Results: Impact of varying probability of pushing

Reduce the probability of forwarding updates with the increase in the number of push rounds



Related work

- Replication and updates in databases
  - Centralized
  - Fewer replicas, Infrequent 'failures'
  - Xerox Parc' Bayou (frequent but brief network partitions)
- Group communication and lazy epidemic algorithms
  - All participants known
  - Faults are exception
  - Bimodal multicast, Randomized Rumor Spreading
- Peer-to-peer systems
  - Napster/Gnutella/Kazaa: No notion of updates
  - Freenet: Propagate update downstream (no guarantees, particularly for offline peers or network changes)
  - OceanStore: Updates create new versions. 2-tier architecture using inner-ring and secondary replicas (caches) for consistency. No analysis. Since inner-ring uses Byzantine agreement protocol, implicitly assumes limited number of peer failures.

CUP: Controlled Update Propagation in Peer-to-Peer Networks [RoussopoulosBaker02]

PCX: Path Caching with Expiration

Cache index entries at intermediary nodes that lie on the path taken by a search query

Cached entries typically have *expiration times*

Which items need to be updated as well as whether the interest in updating particular entries has died out

CUP: Controlled Update Propagation

Asynchronously builds caches of index entries while answering search queries

It then propagates updates of index entries to maintain these caches

CUP: Controlled Update Propagation in Peer-to-Peer Networks [RoussopoulosBaker02]

Every node maintains two logical channels per neighbor:

- a query channel: used to forward search queries
- an update channel: used to forward query responses *asynchronously* to a neighbor and to update index entries that are cached at the neighbor (to proactively push updates)

Queries travel to the node holding the item

Updates travel along the reverse path taken by a query

Query coalescing: if a node receives two or more queries for an item pushes only one instance

All responses go through the update channel: use interest bits so it knows to which neighbors to push the response

CUP: Controlled Update Propagation in Peer-to-Peer Networks [RoussopoulosBaker02]

For each key K, node n stores

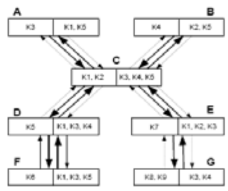
a flag that indicates whether the node is waiting to receive an update for K in response to a query

an interest vector: each bit corresponds to a neighbor and is set or clear depending on whether the neighbor is or is not interested in receiving updates for K

a popularity measure or request frequency of each non-local key K for which it receives queries

The measure is used to re-evaluate whether it is beneficial to continue caching and receiving updates for K

CUP: Controlled Update Propagation in Peer-to-Peer Networks [RoussopoulosBaker02]



For each key, the authority node that own the key is the root of the CUP tree

Updates originate at the root of the tree and travel downstream to interested nodes

Types of updates: deletes, refresh, append

CUP: Controlled Update Propagation in Peer-to-Peer Networks [RoussopoulosBaker02]

Handling Queries for K:

1. Fresh entries for key K are cached

2. Key K is not in cache

added and marked it as pending (to coalesce potential bursts)

3. All cached entries for K have expired

Handling Updates for K:

An update of K is forwarded only to neighbors have registered interest in K

Also, an adaptive control mechanism to regulate the rate of pushed updates

CUP: Controlled Update Propagation in Peer-to-Peer Networks [RoussopoulosBaker02]

Adaptive control mechanism to regulate the rate of pushed updates

Each node N has a capacity U for pushing updates that varies with its workload, network bandwidth and/or network connectivity

N divides U among its outgoing update channels such that each channel gets a share that is proportional to the length of its queue

Entries in the queue may be re-ordered

V. Gopalakrihnan et al, Adaptive Replication in Peer-to-Peer Systems, ICDCS 2004

App-cache

Copies of the requested file are placed in the caches of all servers traversed as the query is routed from the source to the server that finally replies with the file

The LAR protocol

Two types:

replicas of files (contain the data itself are advertised on the query path)

cache hints (caches of routing/index information to decide which of the replicas to use during routing)

Cache entries: data item id, home address, a set of known replica locations, LRU policy