Types of Replication

Two types of replication

- Index (metadata): replicate index entries
- Data/Document replication: replicate the actual data (e.g., music files)

Types of Replication

Caching vs Replication

Cache: Store data retrieved from a previous request (client-initiated)
Replication: More proactive, a copy of a data item may be stored at a node even if the node has not requested it

Reasons for Replication

Reasons for replication

- Performance
  - load balancing
  - locality: place copies close to the requestor
ground: more choices of next step in search
  - reduce number of search
- Availability
  - In case of failures
  - Peer departures
Besides storage, cost associated with replication: Consistency Maintenance
  - Make reads faster in the expense of slower writes

Issues

Which items (data/metadata) to replicate

- Popularity
  - In traditional distributed systems, also rate of read/write
- Where to replicate

"Database-Flavored" Replication Control Protocols

Let's assume the existence of a data item $x$ with copies $x_1, x_2, ..., x_n$

$x$: logical data item

$x_i$: physical data item

A replication control protocol is responsible for mapping each read/write on a logical data item (R($x$) W($x$)) to a set of read/writes on a (possibly) proper subset of the physical data item copies of $x$
One Copy Serializability

Correctness

A DBMS for a replicated database should behave like a DBMS managing a one-copy (i.e., nonreplicated) database as far as users can tell.

One-copy serializable (1SR)

the schedule of transactions on a replicated database be equivalent to a serial execution of those transactions on a one-copy database.

RowA

Read One/Write All (ROWA)

A replication control protocol that maps each read to only one copy of the item and each write to a set of writes on all physical data item copies.

Even if one of the copies is unavailable an update transaction cannot terminate.

Write-All-Available

Write-all-available

A replication control protocol that maps each read to only one copy of the item and each write to a set of writes on all available physical data item copies.

Quorum-Based Voting

Read quorum \( V_r \) and a write quorum \( V_w \) to read or write a data item.

If a given data item has a total of \( V \) votes, the quorums have to obey the following rules:

1. \( V_r + V_w > V \)
2. \( V_w > V/2 \)

Rule 1 ensures that a data item is not read or written by two transactions concurrently (R/W).

Rule 2 ensures that two write operations from two transactions cannot occur concurrently on the same data item (W/W).

Distributing Writes

Immediate writes

Deffered writes

Access only one copy of the data item, it delays the distribution of writes to other sites until the transaction has terminated and is ready to commit. It maintains an intention list of deferred updates.

After the transaction terminates, it sends the appropriate portion of the intention list to each site that contains replicated copies.

Optimizations - aborts cost less - may delay commitment - delays the detection of copies.

Primary copy

Use the same copy of a data item.
Eager vs Lazy Replication

**Eager replication**: keeps all replicas synchronized by updating all replicas in a single transaction.

**Lazy replication**: asynchronously propagate replica updates to other nodes after replicating transaction commits.

*In p2p, lazy replication is the norm.*

Update Propagation

- **Who initiates the update:**
  - Push by the server item (copy) that changes
  - Pull by the client holding the copy
- **When**
  - Periodic
  - Immediate
  - When an inconsistency is detected
  - Threshold-based: Freshness (e.g., number of updates or actual time)
  - **Value**
  - Time-to-live: Items expire after that time
- **Stateless or State-full**

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

Replication in Structured P2P

From CHORD and CAN first papers

CHORD

- **Invariant to guarantee correctness of lookups:**
  - Keep successors nodes up-to-date
- **Method:** Maintain a successor list of its “r” nearest successors on the Chord ring
- **Why? Availability**
  - How to keep it consistent: Lazy thought a periodic stabilization

CHORD

- **Data replication**
  - Method: Replicate data associated with a key at the K nodes succeeding the key
  - **Why? Availability**

CAN

- **Metadata replication**
  - **Multiple realities**
    - With r realities each node is assigned r coordinated zones, one on every reality and holds r independent neighbor sets
    - Replicate the hash table at each reality
  - **Availability:** Fails only if nodes at both r nodes fail
  - **Performance:** Better search, choose to forward the query to the neighbor with coordinates closest to the destination
CAN

Metadata replication

Overloading coordinate zones

Multiple nodes may share a zone

The hash table may be replicated among zones

Higher availability

Performance: choices in the number of neighbors, can select nodes closer in latency

Cost for Consistency

P2p, Spring 05

20

CAN

Metadata replication

Multiple Hash Functions

Use $k$ different hash functions to map a single key onto $k$ points in the coordinate space

Availability: fail only if all $k$ replicas are unavailable

Performance: choose to send it to the node closest in the coordinate space or send query to all $k$ nodes in parallel ($k$ parallel searches)

Cost for Consistency

Query traffic (if parallel searches)

P2p, Spring 05

21

CAN

Metadata replication

Hot-spot Replication

A node that finds it is being overloaded by requests for a particular data key can replicate this key at each of its neighboring nodes

Then with a certain probability can choose to either satisfy the request or forward it

Performance: load balancing

P2p, Spring 05

22

CAN

Metadata replication

Caching

Each node maintains a cache of the data keys it recently accessed

Before forwarding a request, it first checks whether the requested key is in its cache, and if so, it can satisfy the request without forwarding it any further

Number of cache entries per key grows in direct proportion to its popularity

P2p, Spring 05

23

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

Q. Lv et al, "Search and Replication in Unstructured Peer-to-Peer Networks", ICS’02

P2p, Spring 05

24

Search and Replication in Unstructured Peer-to-Peer Networks

Type of replication depends on the search strategy used

(i) A number of blind-search variations of flooding

(ii) A number of (metadata) replication strategies

Evaluation Method: Study how they work for a number of different topologies and query distributions

P2p, Spring 05
Methodology

Aspects of P2P

Performance of search depends on:

- Network topology: graph formed by the p2p overlay network
- Query distribution: the distribution of query frequencies for individual files
- Replication: number of nodes that have a particular file

Assumption: fixed network topology and fixed query distribution

Results still hold, if one assumes that the time to complete a search is short compared to the time of change in network topology and in query distribution.

Network Topology

(1) Power-Law Random Graph

A 9239-node random graph

Node degrees follow a power law distribution

\[ \omega / i^a \]

when ranked from the most connected to the least, the \( i \)-th ranked has \( \omega / i^a \), where \( \omega \) is a constant

Once the node degrees are chosen, the nodes are connected randomly.

Network Topology

(2) Normal Random Graph

A 9836-node random graph

Network Topology

(3) Gnutella Graph (Gnutella)

A 4736-node graph obtained in Oct 2000

Node degrees roughly follow a two-segment power law distribution

Network Topology

(4) Two-Dimensional Grid (Grid)

A two dimensional 100x100 grid

Table 1: Key statistics of the network topologies
Query Distribution

Let $q_i$ be the relative popularity of the $i$-th object (in terms of queries issued for it)

Values are normalized: $\sum_{i=1}^{m} q_i = 1$

1. Uniform: All objects are equally popular
   $q_i = 1/m$

2. Zip-like
   $q_i \propto 1/i^\alpha$

Replication

Each object $i$ is replicated on $r_i$ nodes and the total number of objects stored is $R$, that is

$\sum_{i=1}^{m} r_i = R$

1. Uniform: All objects are replicated at the same number of nodes
   $r_i = R/m$

2. Proportional: The replication of an object is proportional to the query probability of the object
   $r_i \propto q_i$

3. Square-root: The replication of an object $i$ is proportional to the square root of its query probability $q_i$
   $r_i \propto \sqrt{q_i}$

Query Distribution & Replication

When the replication is uniform, the query distribution is irrelevant (since all objects are replicated by the same amount, search times are equivalent)

When the query distribution is uniform all three replication distributions are equivalent

Thus, three relevant combinations:

1. Uniform/Uniform
2. Zip-like/Proportional
3. Zip-like/Square-root

Metrics

- $Pr(\text{success})$: probability of finding the queried object before the search terminates
- $\#\text{hops}$: delay in finding an object as measured in number of hops
- $\#\text{msgs per node}$: Overhead of an algorithm as measured in average number of search messages each node in the p2p has to process
- $\#\text{nodes visited}$
- Percentage of message duplication
- Peak $\#\text{msgs}$: the number of messages that the busiest node has to process (to identify hot spots)

Simulation Methodology

For each experiment,
First select the topology and the query/replication distributions
For each object $i$ with replication $r_i$, generate $\text{numPlace}$ different sets of random replica placements (each set contains $r_i$ random nodes, on which to place the replicas of object $i$)
For each replica placement, randomly choose $\text{numQuery}$ different nodes from which to initiate the query for object $i$
Thus, we get $\text{numPlace} \times \text{numQuery}$ queries
In the paper, $\text{numPlace} = 10$ and $\text{numQuery} = 100 \rightarrow 1000$ different queries per object
Limitation of Flooding

Choice of TTL
- Too low, the node may not find the object, even if it exists.
- Too high, burdens the network unnecessarily.

Overhead
Also depends on the topology.

Search for an object
that is replicated at
0.125% of the nodes (~11 nodes if total 9000)
Note that TTL depends on the topology.
Also depends on replication (which is however unknown).

Limitation of Flooding

There are many duplicate messages (due to cycles) particularly in high connectivity graphs.

Multiple copies of a query are sent to a node by multiple neighbors.

Duplicated messages can be detected and not forwarded.
But, the number of duplicate messages can still be excessive and worsens as TTL increases.

Limitation of Flooding: Comparison of the topologies

Power-law and Gnutella-style graphs particularly bad with flooding.
Highly connected nodes means higher duplication messages, because many nodes' neighbors overlap.

Random graph best,
Because in truly random graph the duplication ratio (the likelihood that the next node already received the query) is the same as the fraction of nodes visited so far, as long as that fraction is small.

Random graph better load distribution among nodes.

Two New Blind Search Strategies

1. Expanding Ring – not a fixed TTL (iterative deepening).
2. Random Walks (more details) – reduce number of duplicate messages.
Expanding Ring or Iterative Deepening

Note that since flooding queries node in parallel, search may not stop even if the object is located.

Use successive floods with increasing TTL

- A node starts a flood with a small TTL
- If the search is not successful, the node increases the TTL and starts another flood
- The process repeats until the object is found

Works well when hot objects are replicated more widely than cold objects.

Expanding Ring or Iterative Deepening (details)

Need to define

- A policy at which depths the iterations are to occur (i.e., the successive TTLs)
- A time period $W$ between successive iterations
- After waiting for a time period $W$, if it has not received a positive response (i.e., the requested object), the query initiator resends the query with a larger TTL.

Nodes maintain ID of queries for $W + \varepsilon$.

A node that receives the same message as in the previous round does not process it, it just forwards it.

Expanding Ring

Start with TTL = 1 and increase each time by a step of 2.

For replication over 10%, search stops at TTL 1 or 2.

Comparison of message overhead between flooding and expanding ring.

Even for objects that are replicated at 0.125% of the nodes, even if flooding uses the best TTL for each topology, expanding ring still halves the per-node message overhead.

Expanding Ring

More pronounced improvement for Random and Gnutella graphs than for the PLGR partly because the very high degree nodes in PLGR reduce the opportunity for incremental retries in the expanding ring.

Introduce slight increase in the delays of finding an object:

From 2 to 4 in flooding to 3 to 6 in expanding ring.

Random Walks

Forward the query to a randomly chosen neighbor at each step.

Each message a walker $k$-walkers

The requesting node sends $k$ query messages and each query message takes its own random walk.

$k$ walkers after $T$ steps should reach roughly the same number of nodes as $1$ walker after $kT$ steps.

So cut delay by a factor of $k$.

16 to 64 walkers give good results.
Random Walks

When to terminate the walks
- TTL-based
- Checking: the walker periodically checks with the original requestor before walking to the next node (again use a large TTL, just to prevent loops)

Experiments show that checking once at every 4th step strikes a good balance between the overhead of the checking message and the benefits of checking.

When compared to flooding:
- The 32-walker random walk reduces message overhead by roughly two orders of magnitude for all queries across all network topologies at the expense of a slight increase in the number of hops (increasing from 2-6 to 4-15)

When compared to expanding ring,
- The 32-walkers random walk outperforms expanding ring as well, particularly in PLRG and Gnutella graphs.

Random Walks

Keeping State
- Each query has a unique ID and its k-walkers are tagged with this ID
- For each ID, a node remembers the neighbor it has forwarded the query
- When a new query with the same ID arrives, the node forwards it to a different neighbor (randomly chosen)

Improves Random and Grid by reducing up to 30% the message overhead and up to 30% the number of hops
- Small improvements for Gnutella and PLRG

Principles of Search

- Adaptive termination is very important
- Expanding ring or the checking method
- Message duplication should be minimized
- Preferably, each query should visit a node just once
- Granularity of the coverage should be small
- Increase of each additional step should not significantly increase the number of nodes visited

Replication

How many copies?
Theoretically addressed in another paper, three types of replication:
- Uniform
- Proportional
- Square-Root

Replication: Problem Definition

How many copies of each object so that the search overhead for the object is minimized, assuming that the total amount of storage for objects in the network is fixed.
Replication Theory

Assume $m$ objects and $n$ nodes.
Each object $i$ is replicated on $r_i$ distinct nodes and the total number of objects stored is $R$, that is
\[ \sum_{i=1}^{m} r_i = R \]
Assume that object $i$ is requested with relative rates $q_i$, we normalize it by setting
\[ \sum_{i=1}^{m} q_i = 1 \]
For convenience, assume $1 \leq r_i \leq n$.

Searches consist of randomly probing sites until the desired object is found.
The probability $Pr(k)$ that the object is found at the $k$'th probe is given
\[ Pr(k) = (1 - r_i/n)^{k-1} \cdot \frac{r_i}{n} \]

Average search size $A$ of all the objects (average number of nodes probed per object query)
Average search size is the inverse of the fraction of sites that have replicas of the object
\[ A_i = \frac{n}{r_i} \]
Average search size for all the objects
\[ A = \Sigma_i q_i A_i = n \Sigma_i q_i/r_i \]

Uniform Replication

Create the same number of replicas for each object
\[ r_i = \frac{R}{m} \]
Average search size for uniform replication
\[ A_i = \frac{n}{r_i} = \frac{m}{r_i} \]
\[ A_{\text{uniform}} = \Sigma_i q_i \frac{m}{r_i} = \frac{m}{r} \]
Which is independent of the query distribution.
It makes sense to allocate more copies to objects that are frequently queried, this should reduce the search size for the more popular objects.

Proportional Replication

Create a number of replicas for each object proportional to the query rate
\[ r_i = R q_i \]
Average search size for uniform replication
\[ A_i = \frac{n}{r_i} = \frac{n}{R q_i} \]
\[ A_{\text{proportional}} = \Sigma_i q_i \frac{nR}{q_i} = m/n \]
Which is again independent of the query distribution.
Why? Objects whose query rate are greater than average (>1/m) do better with proportional, and the other do better with uniform.
The weighted average balances out to be the same.
So what is the optimal way to allocate replicas so that $A$ is minimized?
Square-Root Replication

Find \( r_i \) that minimizes \( A \),

\[
A = \sum_i q_i A_i = n \sum_i q_i/r_i
\]

This is done for \( r_i = \lambda \sqrt{q_i} \) where \( \lambda = R/\sum_i \sqrt{q_i} \).

Then the average search size is

\[
A_{\text{square}} = 1/\rho (\sum_i \sqrt{q_i})^2
\]

Other Metrics: Discussion

Utilization rate, the rate of requests that a replica of an object \( i \) receives

\[
U_i = \frac{R q_i/r_i}{\sum_i q_i/r_i}
\]

- For uniform replication, all objects have the same average search size, but replicas have utilization rates proportional to their query rates.
- Proportional replication achieves perfect load balancing with all replicas having the same utilization rate, but average search sizes vary with more popular objects having smaller average search sizes than less popular ones.

Pareto Distribution

Replication (summary)

Each object \( i \) is replicated on \( r_i \) nodes and the total number of objects stored is \( R \), that is

\[
\sum_{i=1}^n r_i = R
\]

1. Uniform: All objects are replicated at the same number of nodes

\[
r_i = \frac{R}{m}
\]

2. Proportional: The replication of an object is proportional to the query probability of the object

\[
r_i \propto q_i
\]

3. Square-root: The replication of an object \( i \) is proportional to the square root of its query probability \( q_i \)

\[
r_i \propto \sqrt{q_i}
\]

Pareto Distribution

Replication: Summary

<table>
<thead>
<tr>
<th></th>
<th>Uniform</th>
<th>Proportional</th>
<th>Square-Root</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\rho^{-1}m)</td>
<td>(q_i \rho^{-1}m)</td>
<td>(\rho^{-1} (\sum_i \sqrt{q_i})^{-2})</td>
</tr>
<tr>
<td></td>
<td>(m/r_i)</td>
<td>(q_i m/r_i)</td>
<td>(\rho^{-1} (\sum_i \sqrt{q_i})^{-2})</td>
</tr>
<tr>
<td></td>
<td>(U_i = R q_i/r_i)</td>
<td>(q_i m)</td>
<td>(1/\rho (\sum_i \sqrt{q_i})^2)</td>
</tr>
</tbody>
</table>

Achieving Square-Root Replication

How can we achieve square-root replication in practices?

- Assume that each query keeps track of the search size
- Each time a query is finished the object is copied to a number of sites proportional to the number of probes

On average object \( i \) will be replicated on \( c n/r_i \) times each time a query is issued (for some constant \( c \))

It can be argued that this gives square root.
Achieving Square-Root Replication

What about replica deletion?

The lifetime of replicas must be independent of object identity or query rate.
FIFO or random deletions is ok
LRU or LFU no

Replication - Conclusion

Square-root replication is needed to minimize the overall search traffic:
an object should be replicated at a number of nodes that is proportional to the number of probes that the search required

Replication - Implementation

Two strategies are easily implementable

Owner Replication
When a search is successful, the object is stored at the requestor node only (used in Gnutella)

Path Replication
When a search succeeds, the object is stored at all nodes along the path from the requestor node to the provider node (used in Freenet)

If a p2p system uses k-walkers, the number of nodes between the requestor and the provider node is 1/k of the total nodes visited.
Then, path replication should result in square-root replication.
Problem: Tends to replicate nodes that are topologically along the same path.

Random Replication
When a search succeeds, we count the number of nodes on the path between the requestor and the provider.
Say p
Then, randomly pick p of the nodes that the k walkers visited to replicate the object.
Harder to implement.

Replication: Evaluation

Study the three replication strategies in the Random graph network topology.
Simulation Details
- Place the m distinct objects randomly into the network
- Query generator generates queries according to a Poisson process at 5 queries/sec
- Zipf distribution of queries among the m objects (with a=1.2)
- For each query, the initiator is chosen randomly
- Then a 32-walker random walk with state keeping and checking every 4 steps
- Each sites stores at most objAllow (40) objects
- Random Deletion
- Warm-up period of 10,000 secs
- Snapshots every 2,000 query chunks
Replication: Evaluation

For each replication strategy

- What kind of replication ratio distribution does the strategy generate?
- What is the average number of messages per node in a system using the strategy?
- What is the distribution of number of hops in a system using the strategy?

Both path and random replication generates replication ratios quite close to square-root of query rates.

Path replication and random replication reduces the overall message traffic by a factor of 3 to 4.

Much of the traffic reduction comes from reducing the number of hops.

Path and random, better than owner.
For example, queries that finish with 4 hops, 71% owner, 86% path, 91% random.