p2p Timeline

- ICQ: 1996
- AIM: 1997
- Napster: 1999
- Gnutella & Limewire: 2000
- Bittorrent: 2002
- Kazaa: 2004
- Chord: 2009
- Rapidshare: Napster shut down
- Pirate Bay: 2002
- Bitcoin: 2009
Utilities

- Gas
- Electricity
- Water
- Phone
- Internet
- Data Storage?

OceanStore

Heterogeneous network

Nomadic data

IOT connections

Untrusted clients, untrusted servers
Overview

- Objects have GUIDs (self-certifying a la Mazieres)
- Partition tolerance (replication)
- Routing (randomized distribution, Bloom filters)
- Access control (reads restricted at clients, writes restricted at servers)
- Updates (pairwise anti-entropy sessions a la Bayou)
- Temperature (active vs. archive a la Elephant, deep archive foreshadows f4)
- Introspection (learn from how system is used, adapt accordingly)
**Consistency**

**Primary tier**
- Few, high-bandwidth, high-connectivity replicas
- Collective master
- Uses Byzantine agreement
- Multicasts decisions to secondary tier

**Secondary tier**
- Many, potentially low-bandwidth, low-connectivity replicas
- Uses anti-entropy sessions (epidemic algorithm)
Updates

Figure 5: The path of an update. (a) After generating an update, a client sends it directly to the object’s primary tier, as well as to several other random replicas for that object. (b) While the primary tier performs a Byzantine agreement protocol to commit the update, the secondary replicas propagate the update among themselves epidemically. (c) Once the primary tier has finished its agreement protocol, the result of the update is multicast down the dissemination tree to all of the secondary replicas.
Byzantine Agreement
The general decides to attack & needs to communicate plans to lieutenants.

But some lieutenants might be incompetent & bungle things; or they might be traitors & deliberately subvert orders.

Sometimes generals are corrupt, too.

Loyal lieutenants receive orders & pass them along accordingly.

Byzantine Agreement

Solution must guarantee:

1. All loyal lieutenants (correct processes) eventually reach a decision regarding the value of the order they have been given.

2. All loyal lieutenants (correct processes) have to decide on the same value of the order they have been given.

3. If the general (source process) is loyal (correct), all lieutenants (processes) have to decide on the value originally given by the general (source process).

Byzantine Agreement

Recursive algorithm

Step 1: general sends a message to lieutenants.

Step 2: messages from step 1 are collected together in the form of the vectors that encode lieutenant info

Step 3: lieutenants pass their vectors to each other

Step 4: lieutenants examine $i$th elements of vectors received from step 3. If any value has a majority, it is selected. If none has majority, value is marked unknown.

Byzantine Agreement

(a) Diagram showing the network connections and faulty processor.

(b) Table showing messages Received (Got) at processors 1, 2, 3, and 4.

1 Got (1, 2, x, 4)
2 Got (1, 2, y, 4)
3 Got (1, 2, 3, 4)
4 Got (1, 2, z, 4)

(c) Table showing messages Received (Got) at processors 1, 2, 3.

1 Got
2 Got
3 Got

1, 2, x, 4
(1, 2, x, 4)
(1, 2, y, 4)

(a, b, c, d)
(e, f, g, h)
(i, j, k, l)
Byzantine Agreement

Works given $L$ participants, with $T$ traitors, where $L > 3T$.

E.g. If fewer than one third of the participants are traitors.

Anti-entropy
Pairwise Anti-Entropy Sessions (Bayou)

```
# learn writes unknown to R & send

anti-entropy(S,R) {
    Get R.WL from R

    w = first write in S.write-log

    WHILE(w) DO
        IF w is new for R
            SendWrite(R,w)
        END
        w = next write in S.write-log
    END
}
```

Finding

- shared objects in a network
- not all unique!
- given a target object, how can we find it (or a convenient copy)?
Bloom Filters
Bloom Filters

Apply hashes, map objects to bit vector.

Later we can check to see if an object is definitely not or maybe in the set.

h1("oracle") = 1
h2("oracle") = 4
h3("oracle") = 5

h1("database") = 2
h2("database") = 5
h3("database") = 10

h1("filter") = 4
h2("filter") = 7
h3("filter") = 10
Attenuated Bloom Filters

- n1 is looking for object X, whose GUID hashes to bits 0, 1, and 3.
- n1’s local Bloom filter shows it does not have X.
- n1’s neighbor filter indicates that n2 might be an intermediate node en route to X.
- n2’s local Bloom filter shows it does not have X.
- n2’s neighbor filter indicates n4 doesn’t have X either, but n3 might.
- n3 verifies it has X.
Routing
Plaxton Tree Routing

- reads are fast
- insert or delete are expensive
- storage overhead grows.

Idea 2: store in an arbitrary node
- accesses could be slow

Plaxton Tree Routing

Algorithm:

- Give objects and nodes names independent of location (SHA-1 fixed length). Even distribution of objects in namespace.
- For every object, form a rooted tree. Root doesn’t store objects, but instead pointers to the server that stores the object. From root you can navigate to the server storing that object.
- Embed an n-node virtual height-balanced tree $T$ into the network. Each node $u$ maintains information about copies of the object in the subtree of $T$ rooted at $u$.
- An object’s root is a node whose name matches the object’s name in the largest number of trailing bit positions (aka suffix routing). If there are multiple candidates, pick the one with the largest ID.

Plaxton Tree Routing

Plaxton tree (some call it Plaxton mesh) is a data structure that allows peers to efficiently locate objects, and route to them across an arbitrarily-sized network, using a small routing map at each hop. Each node serves as a client, server and a router.

To access a copy, u searches the subtree under it. If the copy or a pointer to the object is available, then u uses that information, otherwise, it passes the request to its parent.

Introspection

- Genetic algorithm.
- Local events can inform local optimizations.
- Patterns that occur across multiple localities can be used to optimize the entire system.

Figure 8: Fast event handlers summarize and respond to local events. For efficiency, the “database” may be only soft state (see text). Further processing analyzes trends and aggregate information across nodes.
API

- Help applications developers understand interactions with system.
- Sessions
- Session guarantees
- Updates
- Callbacks
- Legacy compatibilities (e.g. RDBMS, Unix)
Potential issues

● Requires coordination across service providers (government?)
● Encryption (too much? not enough?)
● Bottlenecks (e.g. speed of writes?)
● Incomplete prototype
● Experimental results?
Thank you