Tracking Objects in Distributed Systems

Arvind Krishnamurthy
Fall 2004

Distributed Systems

- Consider the Shared virtual memory system (SVM) by Kai Li:
  - It tracks pages in a cluster system
  - Simplest version uses a centralized manager to keep track of all objects in the system
  - Refined version uses a distributed manager
    - Distribution is for load-balance and not because the system is a "distributed system"

- Trademarks of a distributed system
  - Heterogeneous in nature
  - Loosely coupled systems; interconnection network might be just the Internet
  - Separate entities might control the different components of a system
  - Controlling entities might be receptive to incentives, might be “byzantine”
Tracking Objects in Distributed Systems

If one were to design the equivalent of the SVM in a distributed system:
- What issues should we worry about?
- How will the issues affect the design of the system?

Motivating Application

- Distributed file sharing
- Large number of users managing an even larger number of objects
- Users/individual computers might join and leave the system at any point in time
- Tracking of data objects could be divorced from storing the data objects (orthogonal issues)
- Minimize the amount of state managed by the system
- Minimize the amount of communication required to track the current system state
- Many research systems; popular approach “distributed hash tables” exemplified by Chord
**Chord Overview**

- Provides lookup service:
  - Lookup(key) → IP address
  - Chord does not store the data

- \( m \) bit identifier space for both keys and nodes
- **Key identifier = SHA-1(key)**
  
  \[
  \text{Key} = \text{SHA-1}(\text{mozilla.rpm}) \quad \text{ID} = 60
  \]

- **Node identifier = SHA-1(IP address)**
  
  \[
  \text{IP} = \text{SHA-1}(198.10.10.1) \quad \text{ID} = 123
  \]

**Hashing Scheme**

- A key is stored at its **successor**: node with next higher ID
**Simple Scheme**

- Every node knows of every other node
  - that is, "N10" knows "N90" is "198.20.20.1"
  - requires global information
- Routing tables are large $O(N)$
- Lookups are fast $O(1)$

**Other Extreme**

- Every node knows its successor in the ring
- requires $O(N)$ lookups
Intermediate solution: “finger tables”

- Every node knows \( m \) other nodes in the ring
  - That is, it knows the node that is maintaining \( K + 2^i \)
  - where \( K \) is mapped id of current node
- Increase distance exponentially

Faster Lookups

- Lookups take \( O(\log N) \) hops
- Halve the distance to target with each hop

Question: what topic of parallel computing does this resemble most?
Joining the ring

- Three step process:
  - Initialize all fingers of new node
  - Update fingers of existing nodes
  - Transfer keys from successor to new node

- Less aggressive mechanism (lazy finger update):
  - Initialize only the finger to successor node
  - Periodically verify immediate successor, predecessor
  - Periodically refresh finger table entries

Joining the ring: step 1

- Initialize the new node’s finger table
  - Locate any node \( p \) in the ring
  - Ask node \( p \) to lookup fingers of new node \( N_{36} \)
  - Return results to new node

1. Lookup(37, 38, 40, ..., 100, 164)
Joining the ring (contd.)

- Step 2: Updating fingers of existing nodes
  - new node calls update function on existing nodes
  - existing nodes can recursively update fingers of other nodes
- Step 3: transfer keys from successor node to new node
  - only keys in the range are transferred

Handling Failures

- Failure of nodes might cause incorrect lookup

- N80 doesn’t know correct successor, so lookup fails
  - Successor fingers are enough for correctness
Handling Failures

- Use successor list
  - Each node knows $r$ immediate successors
  - After failure, will know first live successor
  - Correct successors guarantee correct lookups

- Guarantee is with some probability
  - Can choose $r$ to make probability of lookup failure arbitrarily small

Lookup Cost

- Cost is $O(\log N)$ as predicted by theory
- Constant is $1/2$