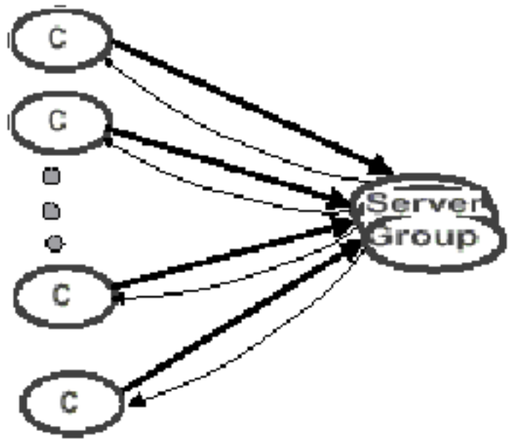
Group Models and Peer to Peer Computing

Jonathan Walpole

Department of Computer Science & Engineering
OGI/OHSU

Structured Group Models

Server Groups Model



(a)

Client-server architecture with replicated servers

- server group hides individual servers from clients
- open group architecture with well-known group name
- servers may be organised to improve availability or performance

Groups to Enhance Availability

Highly Available Service Architecture

Goal

- fail-over support for high availability during upgrades or faults

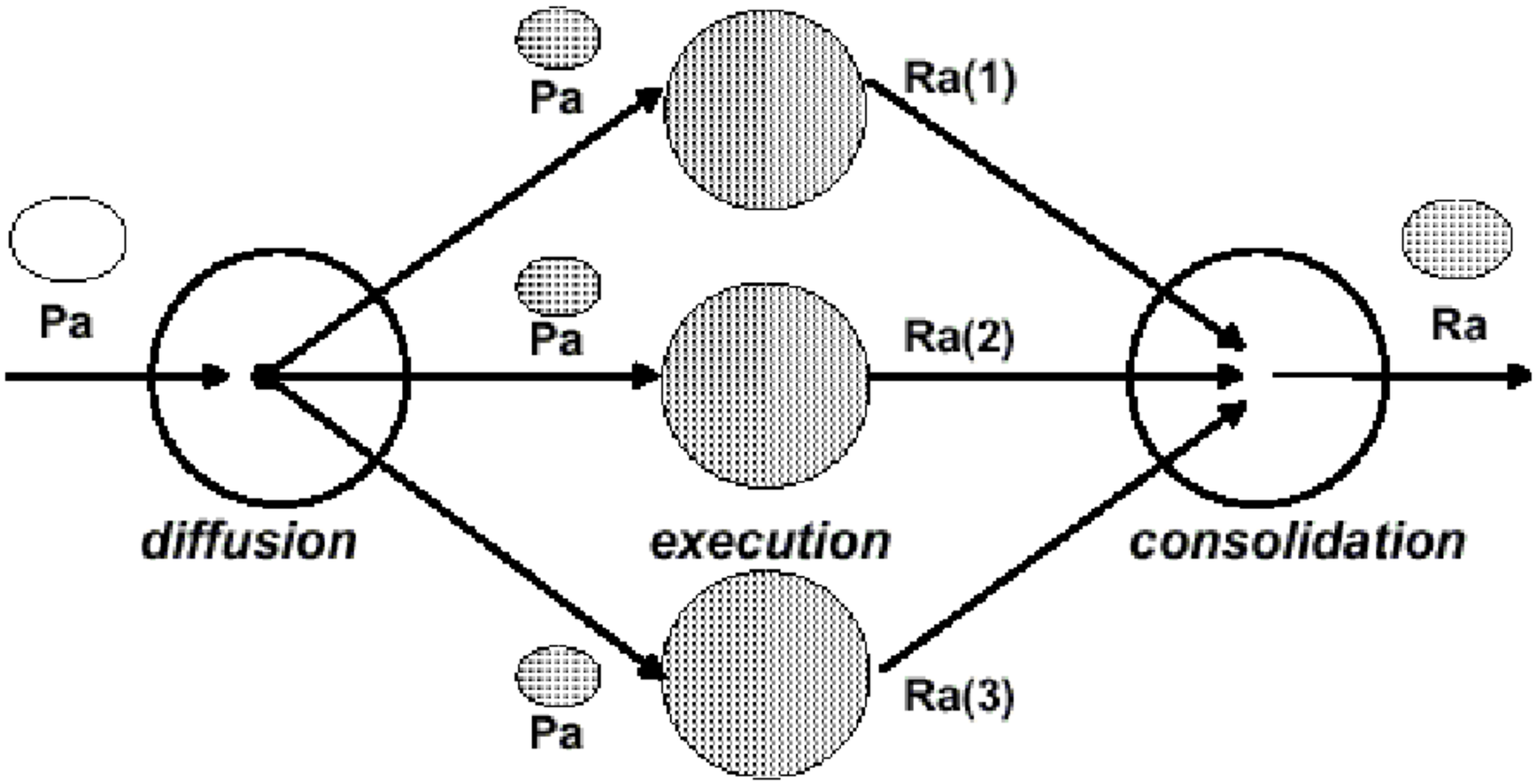
Consistency issues

- replicas must be synchronised to prevent repeated or missed actions
- needs virtual synchrony and totally ordered, reliable communication

Server group architecture

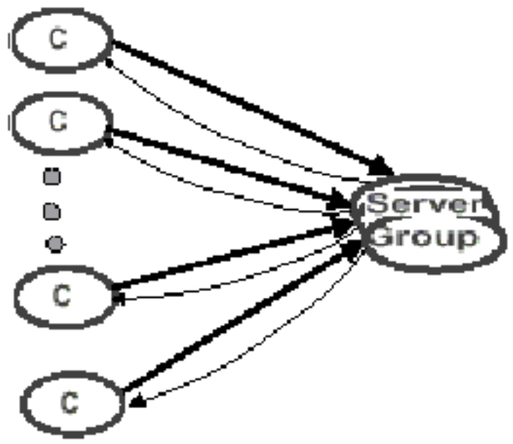
- centralized approach (master & slaves)
- decentralized approach (majority or weighted voting schemes)

Diffusion and Consolidation with Replicated Servers



Groups to Enhance Performance

Server Groups Model for Performance



(a)

Server groups process different requests in parallel

High Performance Service Architecture

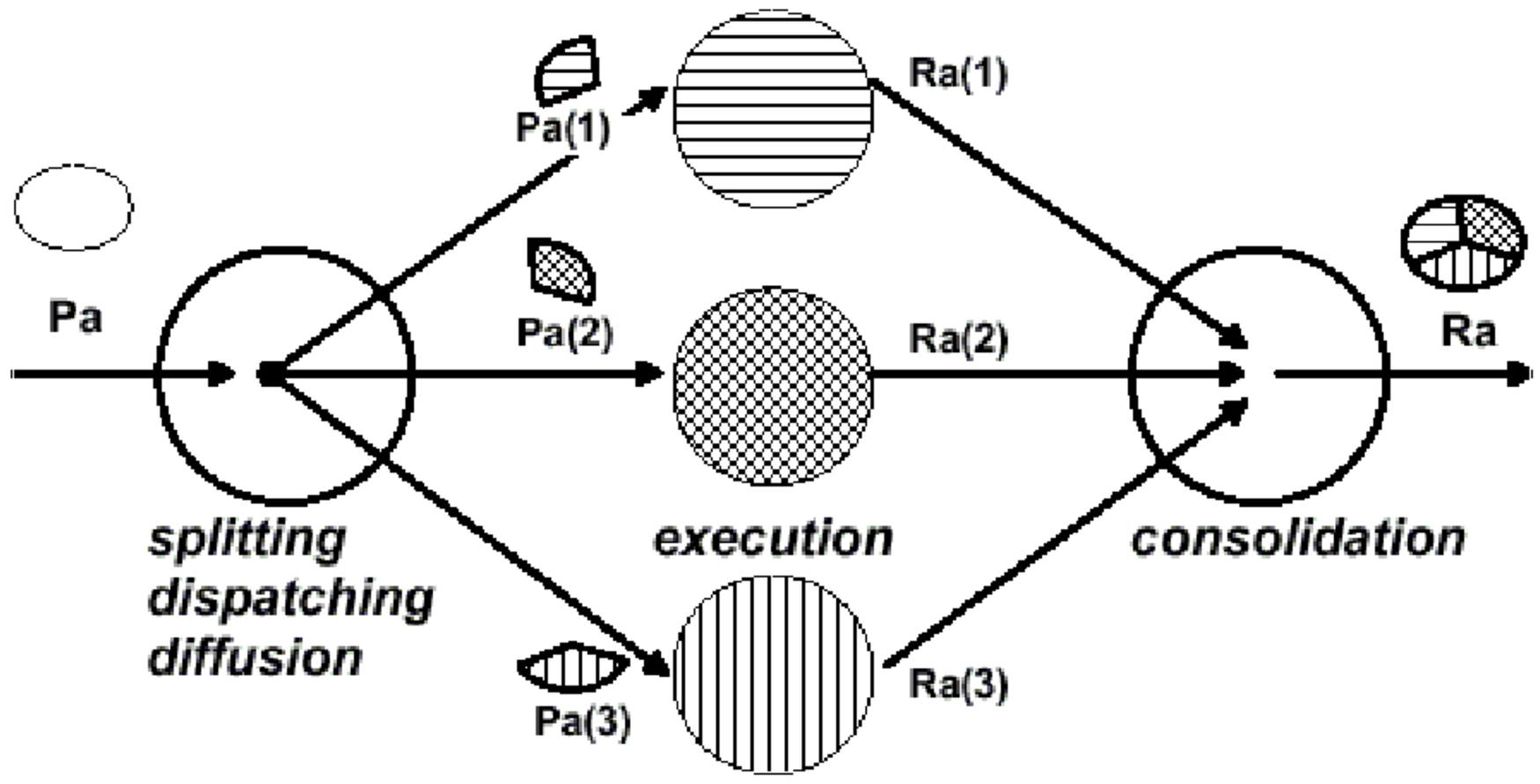
Each server handles a different request

- Diffusion mechanism hands out requests
- Consolidation is trivial - servers send result directly to client

Or, each server computes a different part of the result

- Diffusion mechanism distributes request parts
- Consolidation mechanism combines result parts

Coordination

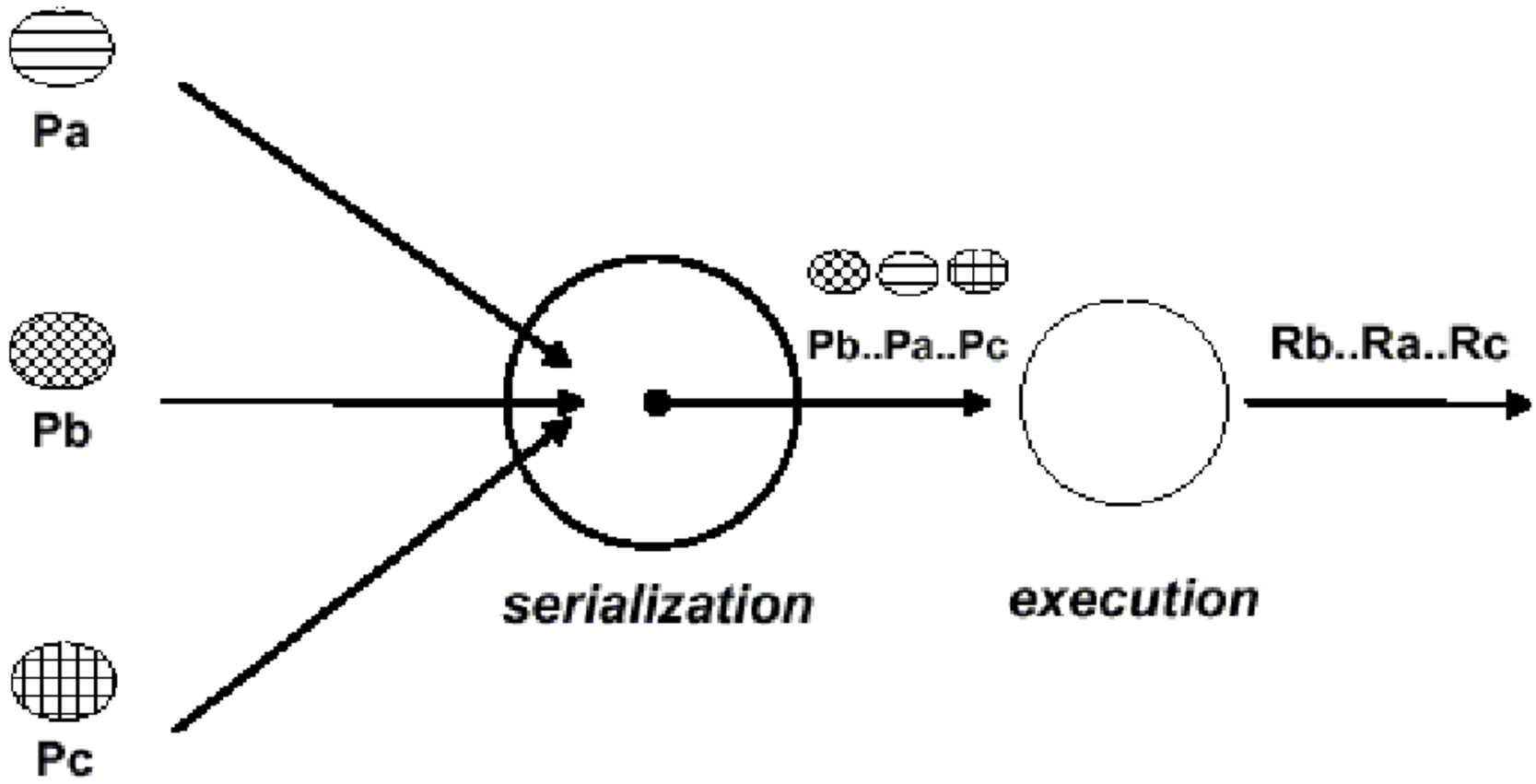


Transactions

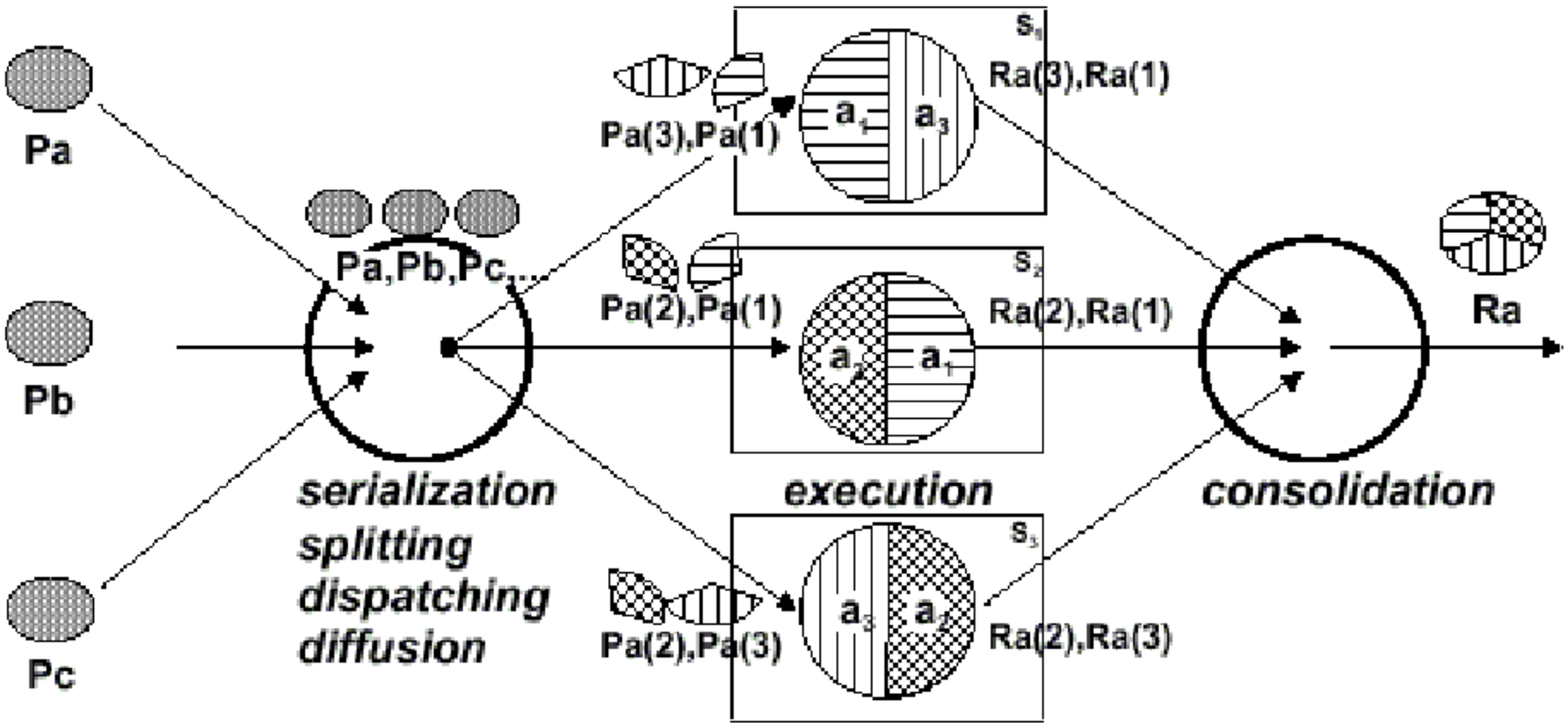
Data consistency semantics may impose constraints on the order in which requests or request parts can be processed

- serialization degrades performance
- serializability allows parallelism but guarantees that outcome is equivalent to some serial order
- transactions use synchronization primitives, such as locks, on individual data items, and follow locking rules to ensure ordering of groups of requests

Serialization

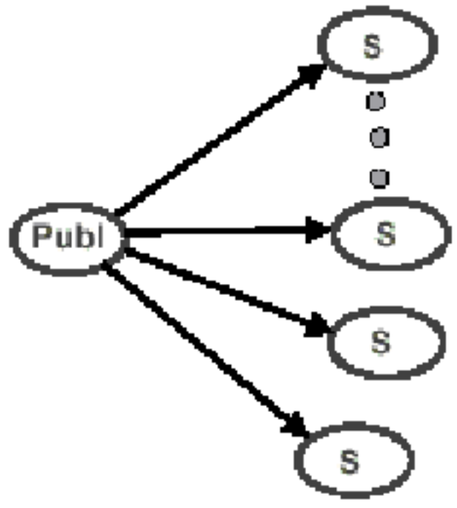


Reliable, High Performance Server Groups



Groups for Dissemination

Dissemination Model



Publish-subscribe architecture

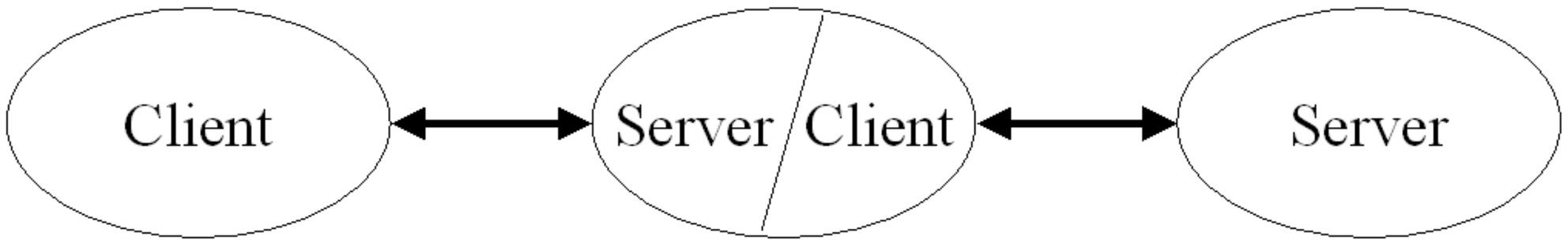
- clients subscribe to a service and join a group
- servers “push” results to the group
- example content distribution networks (CDNs)

Unstructured Group Models

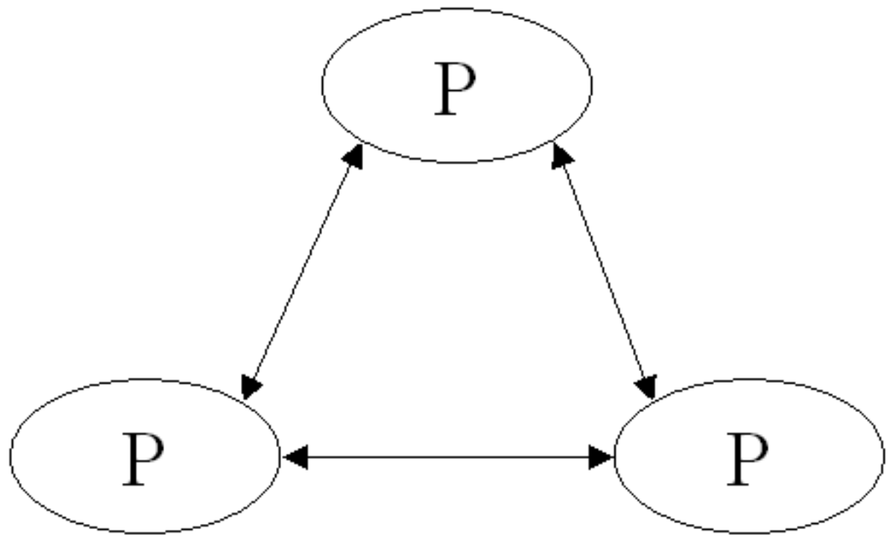
Proxies

Proxy

- a node that plays the role of both a client and a server
- in true peer to peer systems all nodes can be thought of as proxies



Peer to Peer Model



Communication in peer to peer systems

- use underlying group communication support
- define overlay networks to implement group communication

Broadcast and Multicast Communication

IP Multicast

IP-level support for multicast

- special IP address for multicast group (class D)
- multicast forwarding support built in to routers
- nodes may join and leave the group dynamically
- membership changes affect forwarding behavior at routers
- groups dynamically created and deleted via session directory
- local groups limit propagation with TTL on packets

Properties of IP Multicast

- messages only travel once over each link
- reliability?
- congestion control?

Multicast Overlays

Overlay networks

- Overlay routers are end hosts instead of IP-level routers
- may use higher level protocols such as TCP between routers
- can create virtual networks mapped onto the underlying Internet
- packets go once over each link in the overlay, but may go many times of the same underlying link!

Overlays are at the core of true peer to peer systems

- ideally they are efficient, scaleable, self-repairing and attack proof
- new nodes should be able to discover their local “routers”
- nodes should be able to join / leave without disrupting the network

Example Peer to Peer Systems

Napster

Napster

- Centralized approach to coordination and query processing
- Point to point (peer to peer) data transfer
- Other systems such as TrueDisk take a similar approach, but with caching & translation also done at the server
- Single point of failure/attack
  - server may be replicated, but service is easy to shut down
- Widely used example of a *structured peer to peer* system

Gnutella

Gnutella

Unstructured, “decentralized”, true peer to peer system

Overlay network of peers instead of central server

Peer discovery based on flooding ping/pong approach

Discovery of data based on flooding queries

Robust, but not scalable, even with TTL on messages

- exponential traffic growth limits scalability
- slow participants get swamped

Enhancements to Gnutella

Reflector proxies to protect low bandwidth participants

Bandwidth capacity matching to group participants

- based on pong filtering

Pong caching for suppression of ping propagation

- trades stale view of overlay for improved bandwidth efficiency

Query-hit caching for suppression of query propagation

- trades stale view for improved bandwidth efficiency
- good for power-law networks
- how to discover new nodes?

Expanding ring using incrementally expanding TTL

- good for small-world networks

Enhancements to Gnutella

Random walk

- trades response time for bandwidth efficiency
- can be biased towards well connected nodes
- can use parallel walks
- good for power-law and small world networks

Flow control

- refines walk to prevent hot-spots

Query routing

- requires that peers share information about resources in order to construct routing tables
- routing enhances probability of finding the desired item

Freenet

Freenet

True P2P, no centralized directory

Discovery based on query routing instead of broadcast

- distributed discovery and routing infrastructure

Distributed file caching for replication/availability

- transparent to users
- caching strategy and routing approach are complementary

Freenet

Naming via location-independent keys

- keys produced by hashing file keywords or file content
- combining them with public/private keys to create protected/unique name spaces

Each node has files and a routing table <key, node addr>

Freenet

File placement / propagation:

- create key
- insert key locally (if unique) and set TTL
- forward key to "nearest" node in route table (closest hash value)
- if unique it is inserted, otherwise returned
- if still unique when TTL expires placement has succeeded
- the file is stored at each node along the propagation path
- results in file copies being placed on nodes with "similar" files (giving some kind of locality)
- LRU used for storage management (same as caching)
- no guarantee of permanence!

Freenet

Discovery: given a key, look locally, then look for nearest key (closest hash value) in route table and route request there.

- TTL expiry generates a “failure” reply message
- failure messages cause the receiving node to try another candidate with the next closest key
- no more candidates to try results in a failure reply message

Caching: cache file on every node in reverse path and adjust routing tables

- localizes copies around nodes that use them
- uses LRU replacement algorithm to manage space
- quality of routing improves / becomes specialized over time