CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

MUTUAL EXCLUSION

Lecture D

MAEKAWA'S ALGORITHM AND WRAP-UP

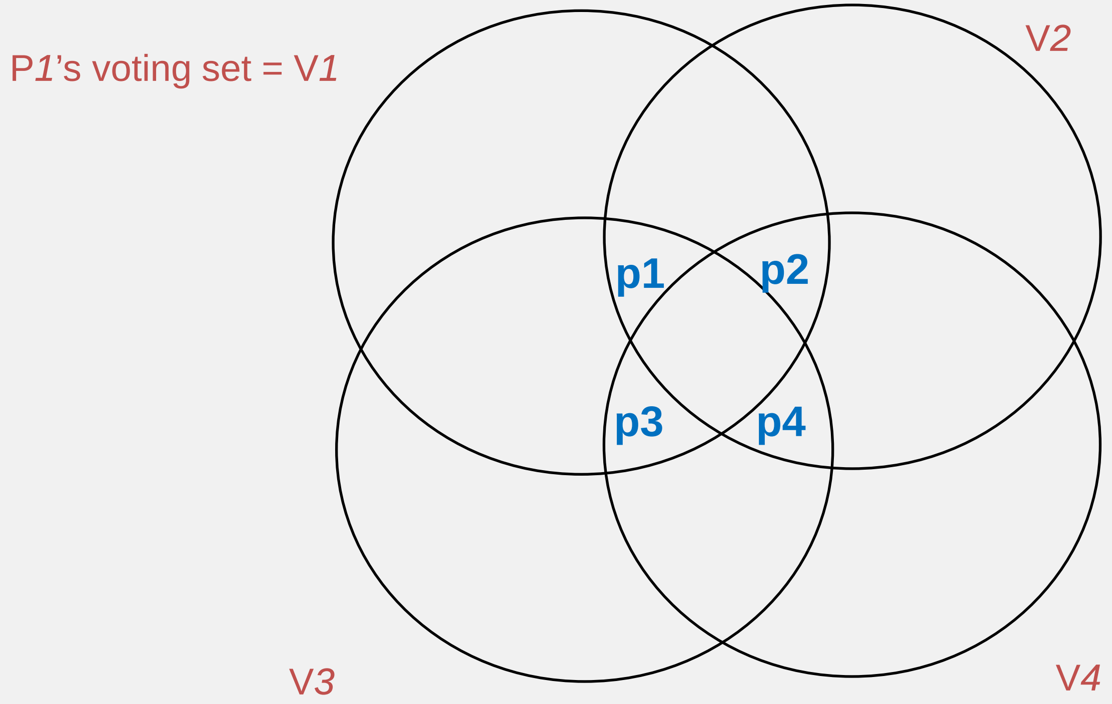
Key Idea

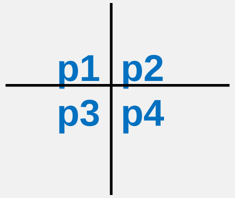
- Ricart-Agrawala requires replies from *all* processes in group
- Instead, get replies from only *some* processes in group
- But ensure that only one process is given access to CS (Critical Section) at a time

Maekawa's Voting Sets

- Each process Pi is associated with a <u>*voting set*</u> Vi (of processes)
- Each process belongs to its own voting set
- *The intersection of any two voting sets must be non-empty*
  - *Same concept as Quorums!*
- Each voting set is of size K
- Each process belongs to M other voting sets
- Maekawa showed that $K=M=\sqrt{N}$ works best
- One way of doing this is to put N processes in a $\sqrt{N}$ by $\sqrt{N}$ matrix and for each Pi, its voting set Vi = row containing Pi + column containing Pi. Size of voting set = $2*\sqrt{N}-1$

Example: Voting Sets with N=4





Maekawa: Key Differences From Ricart-Agrawala

- Each process requests permission from only its voting set members
    - Not from all
- Each process (in a voting set) gives permission to at most one process at a time
    - Not to all

Actions

- state = Released, voted = false
- enter() at process P*i*:
  - state = Wanted
  - Multicast Request message to all processes in V*i*
  - Wait for Reply (vote) messages from all processes in V*i* (including vote from self)
  - state = Held
- exit() at process P*i*:
  - state = Released
  - Multicast Reply to all processes in V*i*

Actions (2)

- When P*i* receives a request from P*j*:

if (state == Held OR voted = true)
    queue request
else
    send Reply to P*j* and set voted = true

- When P*i* receives a Reply from P*j*:

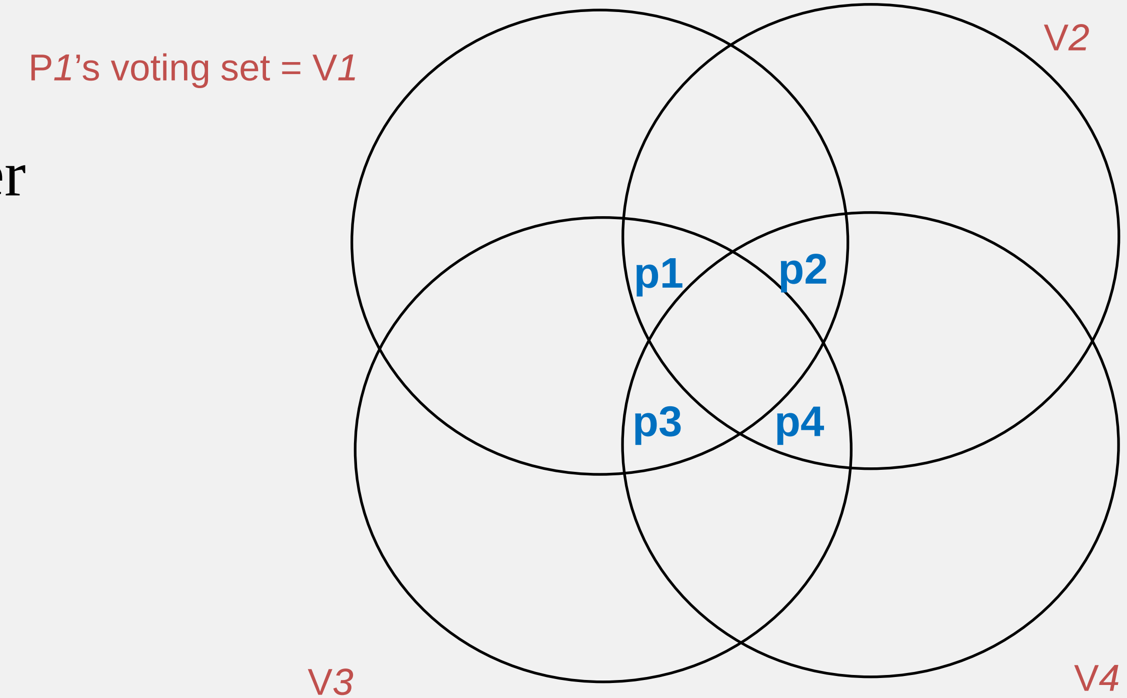
if (queue empty)
    voted = false
else
    dequeue head of queue, say P*k*
    Send Reply *only* to P*k*
    voted = true

Safety

- When a process P*i* receives replies from all its voting set V*i* members, no other process P*j* could have received replies from all its voting set members V*j*
    - V*i* and V*j* intersect in at least one process say P*k*
    - But P*k* sends only one Reply (vote) at a time, so it could not have voted for both P*i* and P*j*

Liveness

- A process needs to wait for at most (*N-1*) other processes to finish CS
- But does not guarantee liveness
- Since can have a *deadlock*
- Example: all 4 processes need access
  - P1 is waiting for P3
  - P3 is waiting for P4
  - P4 is waiting for P2
  - P2 is waiting for P1
  - No progress in the system!
- There are deadlock-free versions



Performance

- Bandwidth
  - $2\sqrt{N}$ messages per enter()
  - $\sqrt{N}$ messages per exit()
  - Better than Ricart and Agrawala's ($2*(N-1)$ and $N-1$ messages)
  - $\sqrt{N}$ quite small. $N \sim 1$ million => $\sqrt{N} = 1K$
- Client delay: One round trip time
- Synchronization delay: 2 message transmission times

Why $\sqrt{N}$?

- Each voting set is of size K
- Each process belongs to M other voting sets
- Total number of voting set members (processes may be repeated) = $K*N$
- But since each process is in M voting sets
  - $K*N/M = N \Rightarrow K = M$ (1)
- Consider a process Pi
  - Total number of voting sets = members present in Pi's voting set and all their voting sets = $(M-1)*K + 1$
  - This must equal the number of processes
  - To minimize the overhead at each process (K), need each of the above members to be unique, i.e.,
    - $N = (M-1)*K + 1$
    - $N = (K-1)*K + 1$ (due to (1))
    - $K \sim \sqrt{N}$

Failures?

- There are fault-tolerant versions of the algorithms we’ve discussed
  - E.g., Maekawa

- One other way to handle failures: Use Paxos-like approaches!

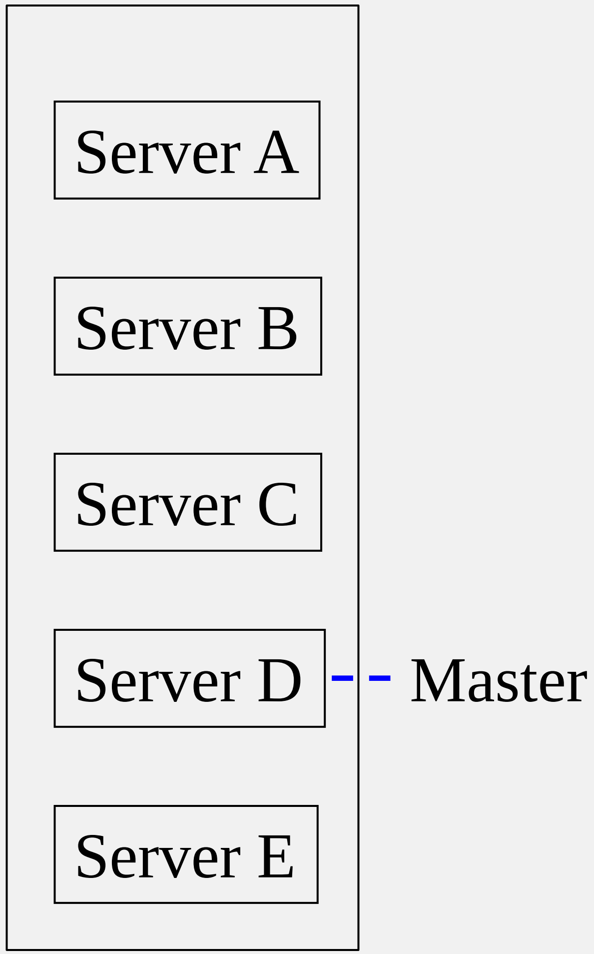
Chubby

- Google’s system for locking
- Used underneath Google’s systems like BigTable, Megastore, etc.
- Not open-sourced but published
- Chubby provides *Advisory* locks only
    - Doesn’t guarantee mutual exclusion unless every client checks lock before accessing resource

Reference: http://research.google.com/archive/chubby.html

Chubby (2)

- Can use not only for locking but also writing small configuration files
- Relies on Paxos
- Group of servers with one elected as Master
  - All servers replicate same information
- Clients send read requests to Master, which serves it locally
- Clients send write requests to Master, which sends it to all servers, gets majority (quorum) among servers, and then responds to client
- On master failure, run election protocol
- On replica failure, just replace it and have it catch up



Summary

- Mutual exclusion important problem in cloud computing systems
- Classical algorithms
  - Central
  - Ring-based
  - Ricart-Agrawala
  - Maekawa
- Industry systems
  - Chubby: a coordination service
  - Similarly, Apache Zookeeper for coordination