

Quorum

CSE 486: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo



Quorum

We saw quorum for [atomic commitment](#) in Raft.

There are other uses for quorum, including:

- Transaction commitment
- Read-write ordering
- Mutual exclusion

They're all related in mechanism and protocol.

Different uses have [different rules](#) for quorum.

Transaction Commitment

Transactions are a **sequence of operations** that:

- Are typically related in some way
- Must either **all succeed** or **all fail**
- Failed operations have **no effect**

Quorum can be used to **commit transactions**.

...Raft could be used for this, but there are **more efficient** protocols.

(We'll talk more about transactions later.)

Read/Write Ordering

Quorum can be used to order reads and writes such that:

- Before some time t , all reads are **before** a given write
- After time t , all reads are **after** a given write

This is a form of **serializability**.

(We'll talk more about serializability later.)

Mutual Exclusion

Quorum can be used to ensure **exclusive access**.

This is **also** essentially a commit:

- The committed value is the **current lock holder**.

This can provide mutual exclusion in the face of failure.

Maekawa's Algorithm

Maekawa proposed an **efficient algorithm** for mutual exclusion. [2]

It requires only $O(\sqrt{n})$ messages to be exchanged for n hosts!

It does this by **carefully selecting** the hosts to contact.

As n grows, this is **considerably easier** than Ricart and Agrawala. [3]

Permission Subsets

All processes in the algorithm are divided into **subsets**.

Every process p_i , $1 \leq i \leq n$, belongs primarily to some subset S_i .

For every process p_i and p_j , $1 \leq i, j \leq n$, $S_i \cap S_j \neq \emptyset$.

This means that p_i is **also in** other subsets!

A process must receive permission from its **entire subset** to enter the critical section.

Set Sizes

There are K members of each subset S_j .

Every process p_i is a member of D subsets.

If $K = n$ and $D = 1$, the algorithm is Ricart and Agrawala.

Choosing Subsets

Subsets are **chosen carefully**.

The paper describes the selection scheme:

The problem of finding a set of S_i 's that satisfies these conditions is equivalent to finding a finite projective plane of N points.

For now, let's **assume the sets can be created**.

Acquiring the Lock

Lock acquisition *similar to* Ricart and Agrawala:

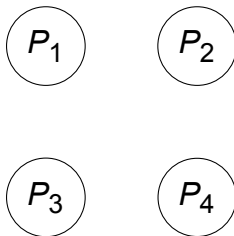
- Every node keeps a timestamp
- The node sends a timestamped request to all other nodes **in its subset**
- Eventually it receives K replies and enters its critical section

Lock release is likewise similar:

- The node sends a release to all other nodes **in its subset**

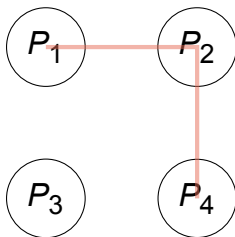
Example

Consider the subsets in four nodes:



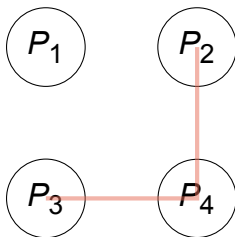
Example

Consider the subsets in four nodes:



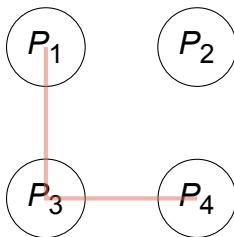
Example

Consider the subsets in four nodes:



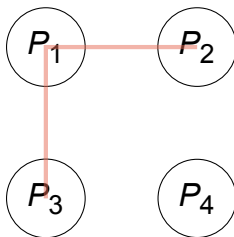
Example

Consider the subsets in four nodes:



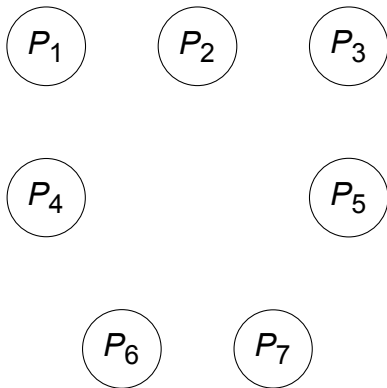
Example

Consider the subsets in four nodes:



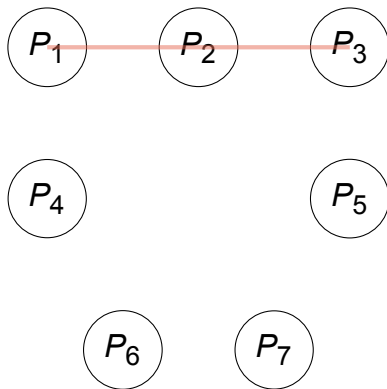
Example 2

Seven nodes gets more complicated:



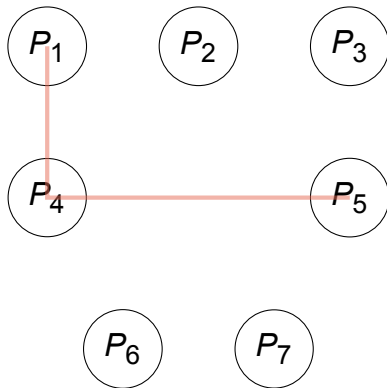
Example 2

Seven nodes gets more complicated:



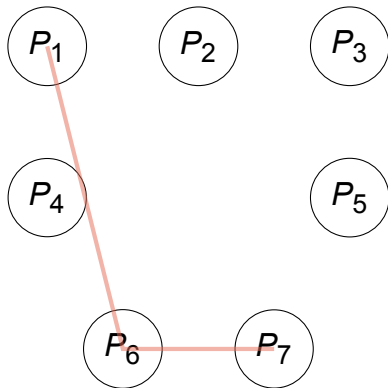
Example 2

Seven nodes gets more complicated:



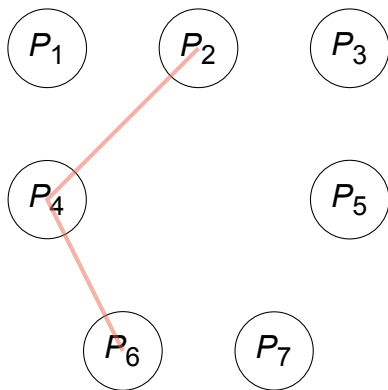
Example 2

Seven nodes gets more complicated:



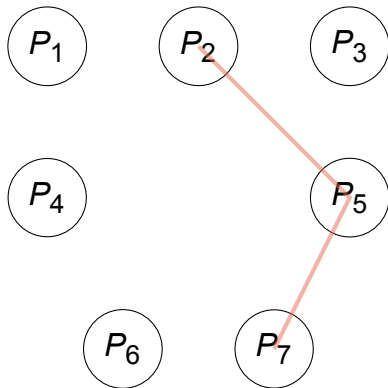
Example 2

Seven nodes gets more complicated:



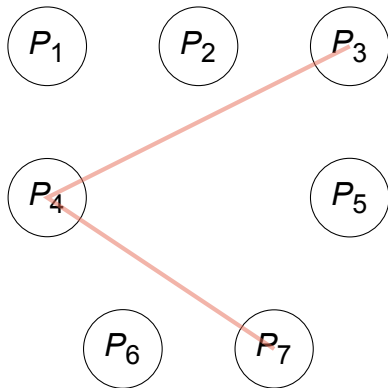
Example 2

Seven nodes gets more complicated:



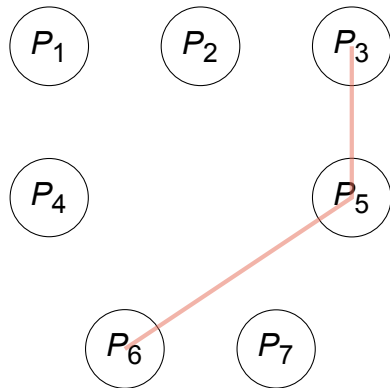
Example 2

Seven nodes gets more complicated:



Example 2

Seven nodes gets more complicated:



Deadlock

Consider three processes: p_1, p_2, p_3 .

Three subsets: $(p_1, p_2), (p_2, p_3), (p_3, p_1)$.

Now consider that **each process locks at the same time**.

Every subset has one locking process, and is waiting on another.

This is deadlock!

Relinquishment

Maekawa must **modify Ricart and Agrawala** to prevent deadlock.

Instead of **withholding a response**, processes send a **FAILED message**.

Processes waiting to lock **keep track of FAILED** messages.

If a process becomes aware of **two processes** requesting its permission to lock, it sends an **INQUIRE message** to the process with the lower-priority lock.

If that process received **any FAILED message**, it relinquishes.

(There are more complications.)

Observations

This is safe because **every quorum overlaps**.

If **any quorum** holds the lock, no other quorum can complete.

This depends on **perfect subset selection**.

There may be **several possible solutions** for any set of processes!

Process membership and subsets are **static configurations**.

Properties

Safety: Similar to Ricart & Agrawala

Liveness:

- Deadlock for any quorum with a failed process
- Special tools to prevent deadlock otherwise

Fairness: Complicated (because of relinquishment), but pretty good

Synchronization Delay: One-way message delay

Throughput: Complicated, but faster than Ricart and Agrawala

Message Complexity: Complicated ($O(\sqrt{n})$)

Summary

- Quorum can solve many problems
- Different quorums have different uses
- Maekawa's mutual exclusion uses quorum for mutexes
- Mutexes can be solved with relatively few members in a quorum

References I

Required Readings

- [1] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Chapter 9: 9.7, 9.8. Cambridge University Press, 2008. ISBN: 978-0-521-18984-2.

Optional Readings

- [2] Mamoru Maekawa. “A \sqrt{N} Algorithm for Mutual Exclusion in Decentralized Systems”. In: 3.2 (May 1985), pp. 145–159. DOI: 10.1145/214438.214445. URL: https://search.lib.buffalo.edu/permalink/01SUNY_BUF/12pkqkt/cdi_crossref_primary_10_1145_214438_214445.

References II

- [3] Glenn Ricart and Ashok Agrawala. “An optimal algorithm for mutual exclusion in computer networks”. In: 24.1 (Jan. 1981). Ed. by R. Stockton Gaines, pp. 9–17. DOI: 10.1145/358527.358537. URL: https://search.lib.buffalo.edu/permalink/01SUNY_BUF/12pkqkt/cdi_crossref_primary_10_1145_358527_358537.

Copyright 2021, 2023, 2024 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see <https://www.cse.buffalo.edu/~eblanton/>.