Distributed Computing and Systems
Chalmers university of technology

Prof Philippas Tsigas
Distributed Computing and Systems Research Group

DISTRIBUTED SYSTEMS II

REPLICATION CNT. II

The Quorum consensus method for Replication

- To prevent transactions in different partitions from producing inconsistent results
  - make a rule that operations can be performed in only one of the partitions.
- RMs in different partitions cannot communicate:
  - each subgroup decides independently whether they can perform operations.
- A *quorum* is a subgroup of RMs whose size gives it the right to perform operations.
  - e.g. if having the majority of the RMs could be the criterion
- in quorum consensus schemes
  - update operations may be performed by a subset of the RMs
    - and the other RMs have out-of-date copies
    - version numbers or timestamps are used to determine which copies are up-to-date
    - operations are applied only to copies with the current version number

Gifford's quorum consensus file replication scheme

- a number of 'votes' is assigned to each physical copy of a logical file at an RM
  - a vote is a weighting giving the desirability of using a particular copy.
  - each *read* operation must obtain a read quorum of *R* votes before it can read from any up-to-date copy
  - each *write* operation must obtain a write quorum of *W* votes before it can do an update operation.
  - *R* and *W* are set for a group of replica managers such that
    - *W* > half the total votes
    - *R* + *W* > total number of votes for the group
  - ensuring that any pair contain common copies (i.e. a read quorum and a write quorum or two write quora)

Gifford's quorum consensus - performing *read* and *write* operations

- before a *read* operation, a read quorum is collected
  - by making version number enquiries at RMs to find a set of copies, the sum of whose votes is not less than *R* (not all of these copies need be up to date).
  - as each read quorum overlaps with every write quorum, every read quorum is certain to include at least one current copy.
  - the *read* operation may be applied to any up-to-date copy.
- before a *write* operation, a write quorum is collected
  - by making version number enquiries at RMs to find a set with up-to-date copies, the sum of whose votes is not less than *W*.
  - if there are insufficient up-to-date copies, then an out-of-date file is replaced with a current one, to enable the quorum to be established.
  - the *write* operation is then applied by each RM in the write quorum, the version number is incremented and completion is reported to the client.
  - the files at the remaining available RMs are then updated in the background.
- Two-phase read/write locking is used for concurrency control
  - the version number enquiry sets read locks (read and write quora overlap)

Gifford's quorum consensus: configurability of groups of replica managers

- groups of RMs can be configured to give different performance or reliability characteristics
  - once the *R* and *W* have been chosen for a set of RMs:
  - the reliability and performance of *write* operations may be increased by decreasing *W*
  - and similarly for reads by decreasing *R*
- the performance of read operations is degraded by the need to collect a read consensus
- examples from Gifford
  - three examples show the range of properties that can be achieved by allocating weights to the various RMs in a group and assigning *R* and *W* appropriately
  - weak representatives (on local disk) have zero votes, get a read quorum from RMs with votes and then read from the local copy

Gifford's quorum consensus examples (1979)

		Example 1	*Example 2*	*Example 3*
Latency (milliseconds)	Replica 1	75	75	75
	Replica 2	65	100	750
	Replica 3	65	750	750
Voting configuration	Replica 1	1	2	1
	Replica 2	0	1	1
	Replica 3	0	1	1
Quorum sizes	*R*	1	2	1
	W	1	3	3

Derived performance
latency
blocking probability - probability that a quorum cannot be obtained, assuming probability of 0.01 that any single RM is unavailable

Example 1 is configured for a file with high read to write ratio

Example 2 is configured for a file with a moderate read to write ratio

Example 3 is configured for a file with a very high read to write ratio. *Reads* can be done at any RM and the probability of the file being unavailable is small. But *writes* must access all RMs.

RM and one remote RM. If the local RM fails only *reads* are allowed

Distributed Replicated FIFO Queue

1. State Machine Approach (One copy of the Queue on each replica)
2. Quorun Consensus
    1. Can we use the approach above?

Distributed Replicated FIFO Queue

1. State Machine Approach (One copy of the Queue on each replica)
2. Quorun Consensus:
    1. Propably representing a state machine does not help here. Instead represent the queue as a log of versioned entries:
        1.enq(x)
        2.enq(y)
        3.deq(x)

FIFO Queue

Can we use the log representation of the FIFO queue to build a distributed highly available queue based on quorum consensus?

Enter enq or deq:

- Read queue version
- Compute new version
- Write new version

Make sure that all quorums intersect!

FIFO Queue

- Here is a new replication protocol:

Definition: To merge a log:
- Short entries in version order
- Discard Duplicates

  - Merge logs from the initial read operation
  - Reconstruct object's state from log
  - Apply operation and compute new entry
  - Append new entry to log and write log to the the final quorum
  - Each replica merges logs

Log Compaction

- Here is a more compact queue representation:
  - No deq records
  - The event horizon: enq version of most recently dequed item
  - The sequence of undequed enq entries

  To merge:
  - take latest event horizon
  - Discrad earlier enq entries
  - Sort remaining enq entries, discard duplicates

  Replicas can send event horizons in ”gossip” messages. Page (21)

Log Compaction

- Event horizons are type-specific, but many similar ideas can work

- Garbage collection:
  - Local: discard entries that can't effect the future
  - Non-local use background "gossip" to discard entries.

Quorum Assignments

- How are quorums chosen?
  - deq needs to know about earlier enq operations
  - deq needs to know about earlier deq operations
  - enq does not need to know about other operations

Depends-On Relation

- Let
  - D be a relation on opertions
  - h any operation sequence
  - and p any operation

  A view of h to p is
  - a sequence of g of h
  - contains every q such that pDq
  - If g contains q, then it contains any earlier r such that qDr

  Definition: D is a depends-on relation if whenever g.p is legal, so is h.p

Depends-On relation

- Quorum consensus replication is correct if and only if the quorum intersection is a depends-on relation

The FE has to find the primary, e.g. after it crashes and another takes over

The passive (primary-backup) model for fault tolerance

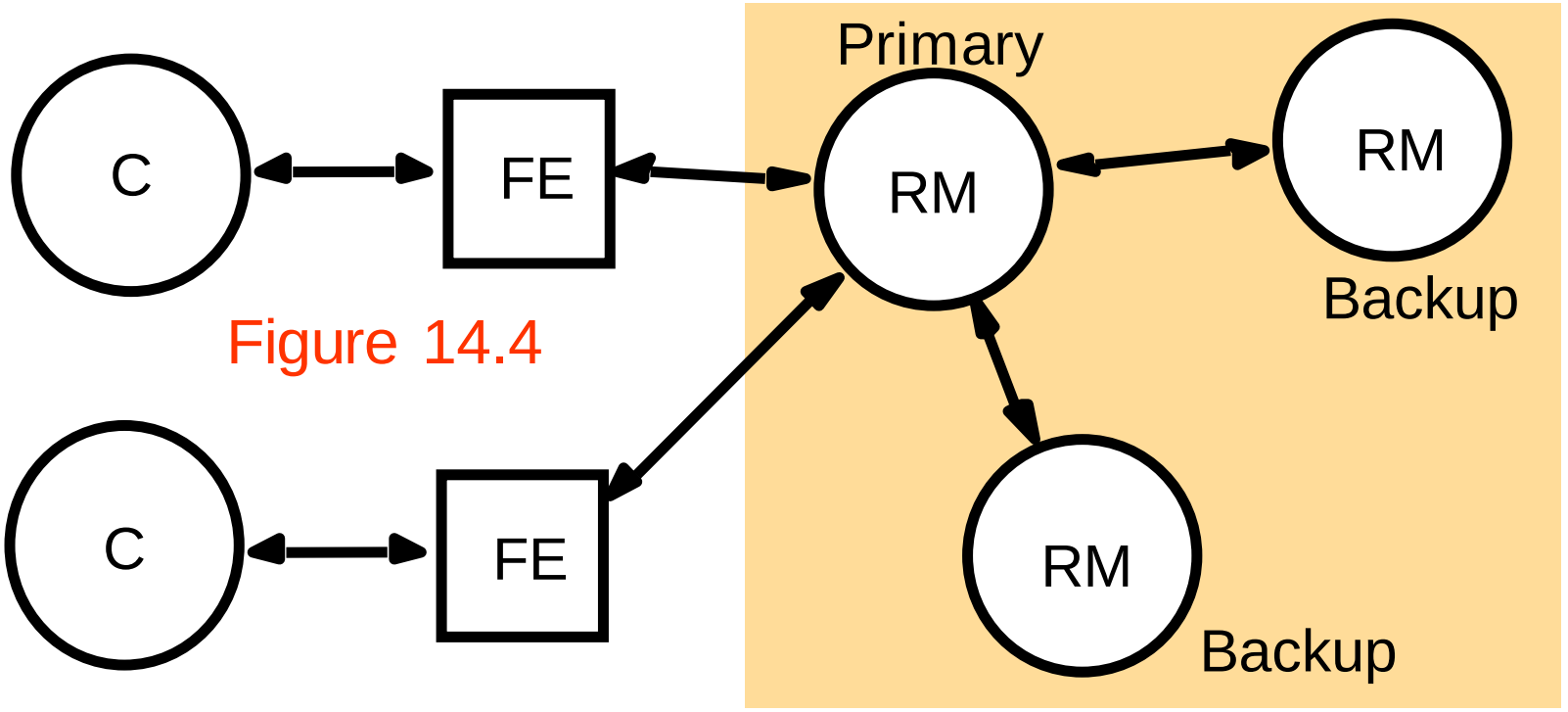

Figure 14.4

- There is at any time a single primary RM and one or more secondary (backup, slave) RMs
- FEs communicate with the primary which executes the operation and sends copies of the updated data to the result to backups
- if the primary fails, one of the backups is promoted to act as the primary.

Passive (primary-backup) replication. Five phases.

- The five phases in performing a client request are as follows:
- 1. Request:
  - a FE issues the request, containing a unique identifier, to the primary RM
- 2. Coordination:
  - the primary performs each request atomically, in the order in which it receives it relative to other requests
  - it checks the unique id; if it has already done the request it re-sends the response.
- 3. Execution:
  - The primary executes the request and stores the response.
- 4. Agreement:
  - If the request is an update the primary sends the updated state, the response and the unique identifier to all the backups. The backups send an acknowledgement.
- 5. Response:
  - The primary responds to the FE, which hands the response back to the client.

Passive (primary-backup) replication (discussion)

- This system implements linearizability, since the primary sequences all the operations on the shared objects
- If the primary fails, the system is linearizable, if a single backup takes over exactly where the primary left off, i.e.:
  - the primary is replaced by a unique backup
  - surviving RMs agree which operations had been performed at take over
- view-synchronous group communication can achieve this
  - when surviving backups receive a view without the primary, they use an agreed function to calculate which is the new primary.
  - The new primary registers with name service
  - view synchrony also allows the processes to agree which operations were performed before the primary failed.
  - E.g. when a FE does not get a response, it retransmits it to the new primary
  - The new primary continues from phase 2 (coordination -uses the unique identifier to discover whether the request has already been performed.

View-synchronous Group Communication

Systems with dynamic groups extend this model by providing explicit join and leave operations to adapt the group membership over time. Moreover, such systems can exclude faulty servers automatically from the membership. Still, reaching agreement on the group membership in the presence of failures is not trivial.

Two approaches have been considered:

1. Run a consensus protocol among the all previous group members to agree on the future group membership. This is the canonical approach, tolerates further failures during the membership change, but involves the potentially expensive consensus primitive.
2. Integrate consensus with the membership protocol and run it only among the (hopefully) correct members. Since this consensus algorithm needs not tolerate failures, it can be simpler; but because further failures may still occur, it provides different guarantees.

The second approach is taken by view-synchronous group communication systems and related group membership algorithms.

Discussion of passive replication

- To survive *f* process crashes, *f*+1 RMs are required
  - it cannot deal with byzantine failures because the client can't get replies from the backup RMs
- To design passive replication that is linearizable
  - View synchronous communication has relatively large overheads
  - Several rounds of messages per multicast
  - After failure of primary, there is latency due to delivery of group view
- variant in which clients can read from backups
  - which reduces the work for the primary
  - get sequential consistency but not linearizability
- Sun NIS uses passive replication with weaker guarantees
  - Weaker than sequential consistency, but adequate to the type of data stored
  - achieves high availability and good performance
  - Master receives updates and propagates them to slaves using 1-1 communication. Clients can uses either master or slave
  - updates are not done via RMs - they are made on the files at the master