COMP2211 NS: Distributed Systems Replication

Frederick Li
School of Engineering and Computing Sciences
frederick.li@durham.ac.uk

This Lesson

- Concepts about replication
- Replication-based Distributed systems
- Types of replication system models that provide fault tolerance

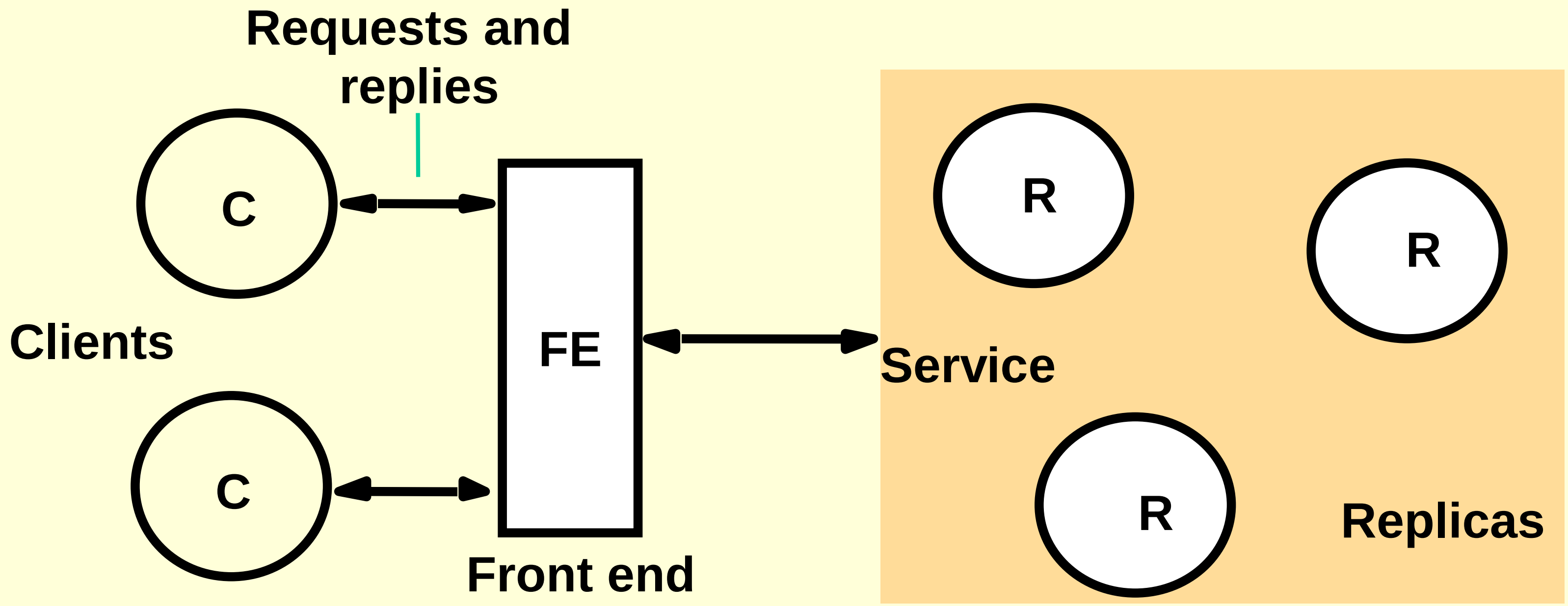
Replication

- Replication is a technique to offer data or services by maintaining multiple copies of them (Note that replicas are not required to use at the same time.)
- Replication is a key to the effectiveness of distributed systems in that it can provide enhanced performance, high availability and fault tolerance.
- Examples:
  - Enhance performance: Web caching, proxy server
  - Increase availability: automatically maintain data or service availability despite server failures
  - Fault tolerance: guarantees strictly correct behaviour despite a certain number and type of faults

Types and Requirements

- Types:
  - Computation (function / service) replication: multiple instances of the same computing task are executed
  - Data replication: same data is stored on multiple devices
- Requirements:
  - Replication transparency: A user sees one logical service, but not its physical copies
  - Data consistency: The same request will receive the same result even it is processed by different replicate copies of a service

System Model



- Replicas
  - Maintain same copies of information; provide same functions
  - Replicas are not necessarily consistent all the time (some may have received updates, not yet conveyed to the others)
  - Example of inconsistency: Think about Google Calendar

System Components

- Replicas (Rs)
  - Maintain replicas (data / functions) on computers
  - Process requests or store (may even propagate) results
  - Dynamic / static: set of Rs is fixed or variable
- Clients (C) request
  - those without updates are called *read-only* requests, the others are called *update* requests (they may include reads)
  - Read-only: handle by one replica
  - Update: may involve data propagation / synchronization, and concurrency control
- Front end (FE)
  - Make replication transparent
  - Maintain replica availability
  - Perform request distribution, and collate responses

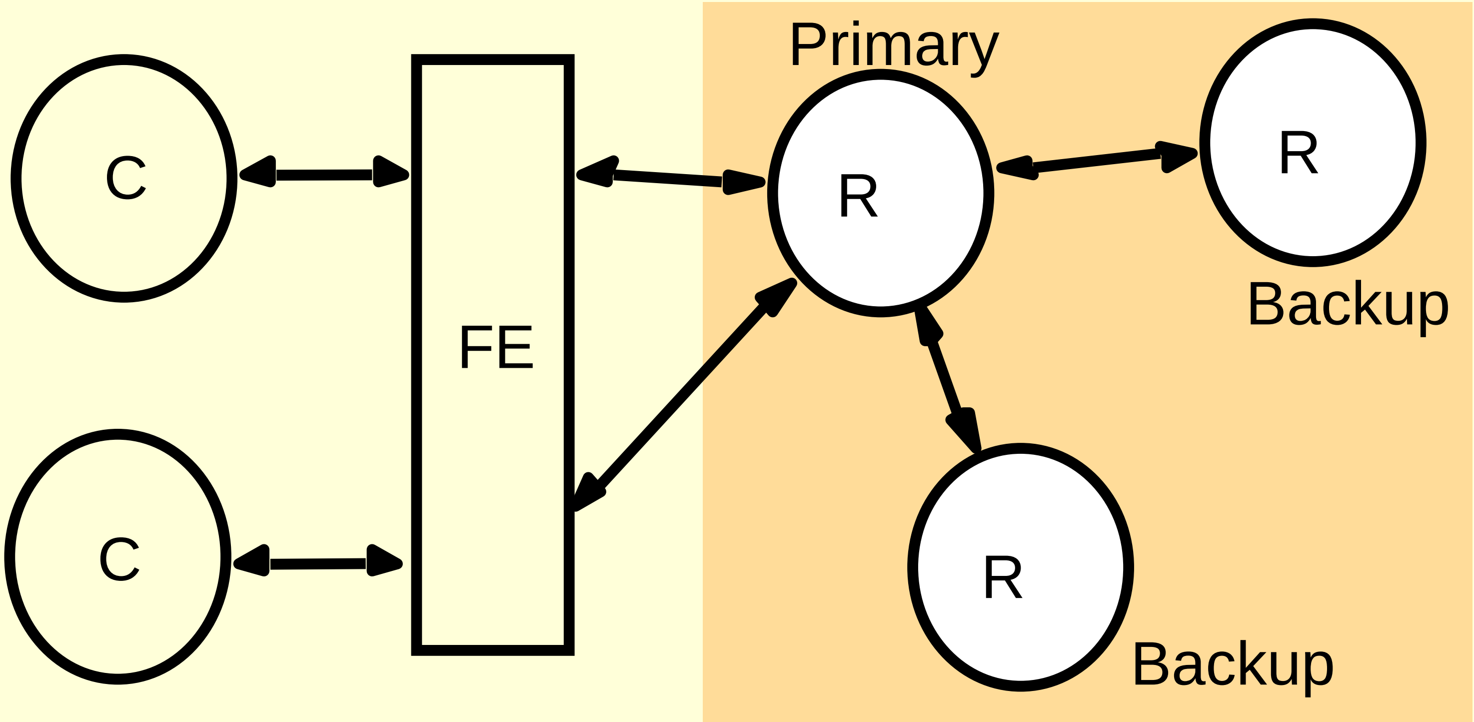
Workflow of the system model

1. Incoming request
   – Receive by the FE, and the FE will forward the request to R(s)
2. Coordination
   – R(s) accepts a request
   – Decide the ordering of a request relative to other requests
3. Execution
   – R(s) processes the request
4. Agreement
   – R(s) reach consensus on the effect of the requests
5. Response
   – One or more Rs reply to the FE
   – FE may process the response before returning it to the client

Fault-Tolerance Services

- Provide a correct service despite up to *f* process failures
- Each replica is assumed to behave according to the specification of the distributed system, when they have not crashed
  - e.g., a specification of bank accounts: ensure funds transferred between bank accounts can never disappear, and that only deposits and withdrawals affect the balance of an account.
- A service based on replication is correct if:
  - it keeps responding despite failures, and
  - if clients cannot tell the difference between the service they obtain from an implementation with replicated data and one provided by a single correct replica manager

Passive (primary-backup) model for fault tolerance



- There is at any time a single primary R and one or more secondary (backup, slave) Rs
- 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

Workflow of Passive replication

- 1. Request:
  - a FE issues the request, containing a unique identifier, to the primary R
- 2. Coordination:
  - the primary performs each request atomically, in the order in which it receives it relative to other requests *(Message ordering)*
  - it checks the unique id; if it has already done the request, it re-sends the response. *(Message Loss)*
- 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

Discussion of Passive Replication

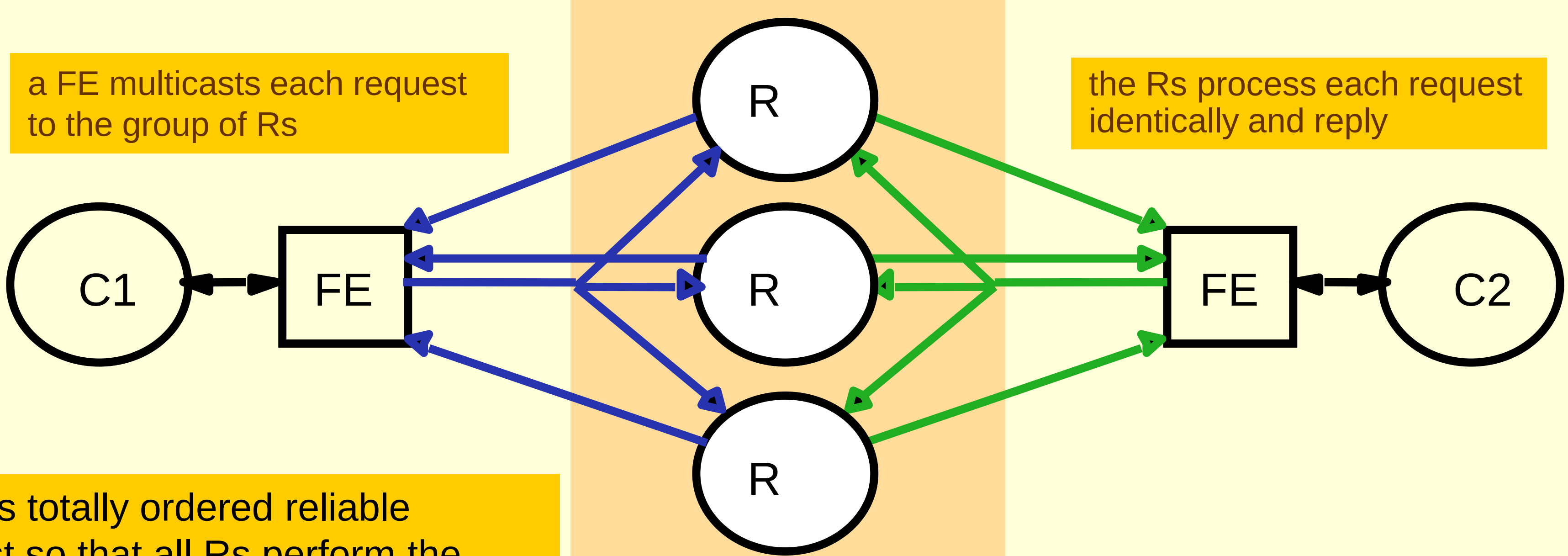
- Non-deterministic behavior at primary replica
  - e.g. due to multi-threading
  - No fatal problem: as other replicas (backups) only slavishly record states determined by the primary's actions
- Replica crashes
  - Survive up to f replica crash, when the system comprises $f + 1$ replicas
- Front-end functionality
  - Requires little functionality: only need to lookup a new primary replica when the current one is not available
- System overhead
  - Relatively large due to data propagation

Active replication

- The Rs are *state machines* all playing the same role and organised as a group.
  - all start in the same state and perform the same in the same order so that their state remains identical
- If an R crashes it has no effect on performance of the service because the others continue as normal

a FE multicasts each request to the group of Rs

the Rs process each request identically and reply



Requires totally ordered reliable multicast so that all Rs perform the same operations in the same order

Active replication - five phases in performing a client request

- 1. Request
  - FE attaches a unique *id* and uses *totally ordered reliable multicast* to send request to Rs. FE can at worst, crash. It does not issue requests in parallel
- 2. Coordination *(Message ordering)*
  - the multicast delivers requests to all the Rs in the same (total) order.
- 3. Execution
  - every R executes the request. They are state machines and receive requests in the same order, so the effects are identical. The *id* is put in the response
- 4. Agreement
  - no agreement is required because all Rs execute the same operations in the same order, due to the properties of the totally ordered multicast.
- 5. Response
  - FEs collect responses from Rs. FE may just use one or more responses. If it is only trying to tolerate crash failures, it gives the client the first response. (Byzantine failure: address here)

Discussion of Active Replication

- Assumption
  - A solution to totally ordered and reliable multicast is available
- Fail situation
  - Can mask up to f Byzantine failures, if the system incorporates at least $2f + 1$ replicas
  - Front end waits until it has collected $f + 1$ identical responses and passes that response back to the client, and discards other responses to the same request
- Read-only request
  - Front-end may send read-only requests only to individual replica
  - Lose fault tolerance, but remain sequentially consistent
  - Can easily mask replica failure in this case, by submitting the read-only request to another replica

Byzantine fault

- A **Byzantine fault** is an arbitrary fault that occurs during the execution of an algorithm by a distributed system. It encompasses both:
  - **Omission failures** (e.g., crash failures, failing to receive a request, or failing to send a response) and;
  - **Commission failures** (e.g., processing a request incorrectly, corrupting local state, and/or sending an incorrect or inconsistent response to a request).
- **Byzantine failure-tolerant algorithms**: characterized by their resilience *t*, the number of faulty processes with which an algorithm can cope

Reference

- **Distributed systems : concepts and design / George Coulouris ... [et al.],** Addison-Wesley (Pearson), c2012.

  Chapter 18