The UNIVERSITY *of* EDINBURGH
SCHOOL *of* COMPUTER SCIENCE

CS4/MSc

Distributed Systems

Jane Hillston

jeh@dcs.ed.ac.uk

Room 1604

(Lecture 15: Replication, 25th February 2002)

Replication

Replication is a technique for **enhancing** services: multiple copies of data are maintained at multiple computers. This can lead to:

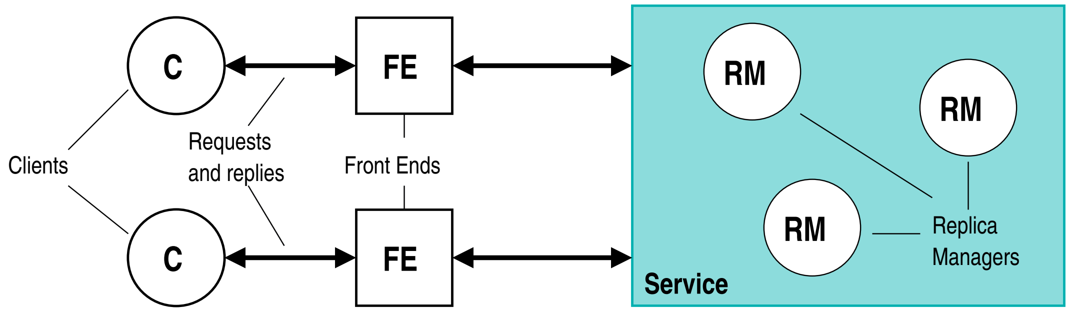
Performance Enhancement: For example, caching data in the web provides improved performance in terms of response time to users and lower utilisation of servers. In some cases the service itself may be replicated (e.g. DNS names may correspond to several IP addresses).

Fault Tolerance: A fault-tolerant service always guarantees strictly **correct** behaviour even in the presence of a certain number and type of faults. Correctness here may concern the integrity of the data with respect to client operations, and/or timeliness of the response.

Increased Availability: The availability of a service is the probability that a response is obtained within a reasonable time bound. Delays can be due to service features such as data locking as well as server and communication infrastructure failures.

System Model

When data is replicated it should be done in a **transparent** manner: the client should not normally be aware that multiple *physical* copies of the data exist, either in terms of submitted requests or returned values. This is achieved by interposing a **front end** between the client and the service.



There are also issues of **consistency** between the replicas although the degree of inconsistency tolerated will depend on the service and the motivation for the replication.

Executing a request on replicated objects

In general five phases are involved in the execution of a single request on a replicated object. The ordering of phases may vary.

Initiation: The front end issues the request to one or more replica managers (RMs).

Coordination: The RMs coordinate in preparation for executing the request consistently. They may agree on whether the request should be executed and the ordering of this request relative to others.

Execution: The RMs execute the request, possibly **tentatively**, i.e. in such a way that they can undo the effects later if necessary.

Agreement: The RMs reach concensus on the effects of the request (if any) that will be committed.

Response: One or more RMs respond to the front end. In some systems it is the responsibility of the front end to collect responses from a collection of RMs and select or synthesise a response for the client.

Replication in CORBA

- An **object group** is a set of objects (usually instances of the same class) that process the same set of responses concurrently. Each returns a response but clients do not need to be aware of the replication.
- The client objects invoke operations on a single, local object which acts as a proxy, not for a single remote object, but for the group of objects.
- The proxy sends the invocation to each of the members of the group using group communication, and collects responses.
- Several systems support replication in CORBA in the form of object groups.
- For example, **Electra** is a CORBA-compliant object group system. Here proxies can be run in **non-transparent** or **transparent** or mode depending on whether the client is replication-aware or not. The ORB is suitably extended to manage object groups.

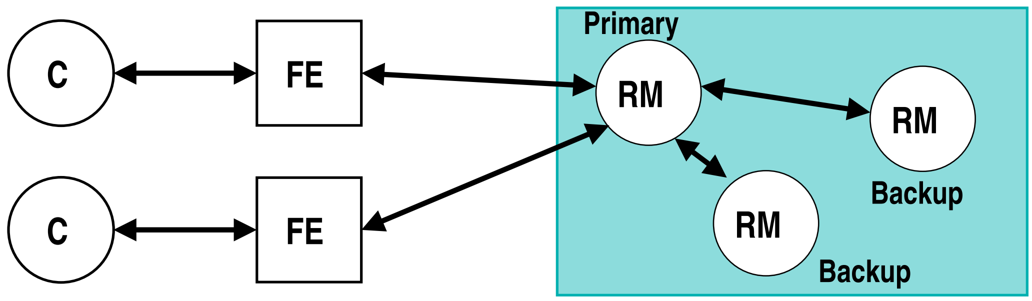
Replication for Fault Tolerance

- A **fault tolerant** service based on replication should be able to keep responding despite failures and clients should not be able to tell the difference between the replicated service and one provided by a single correct RM.
- Care is needed to avoid anomalies with respect to the consistency of the data.
- Various notions of correctness have been defined such as:

Linearizability: for any execution there is an interleaving of the client operations that meets the application expectations for a single correct copy of the objects, and maintains a relative ordering of the operations which is consistent with the real times at which the operations occurred.

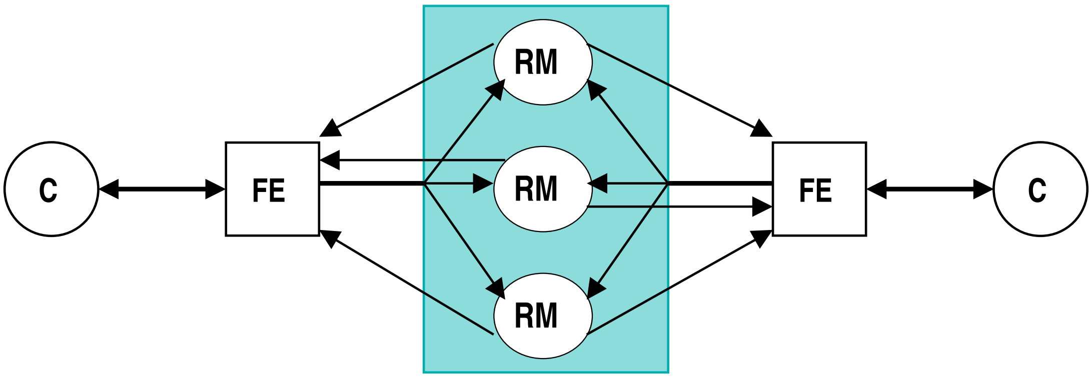
Sequential Consistency: is a weakened form of linearizability which imposes only that the relative ordering of operations is consistent with the program order in which each individual client executed them.

Passive Replication



In the **passive** model of replication for fault tolerance (also known as the **primary-backup** model), one RM is distinguished as the **primary** one. All front ends communicate with the primary RM which is responsible for executing all requests and then updating the other RMs, known as **backups** or **slaves**. If the primary RM fails it is replaced by one of the backups. At this point the surviving RMs must also agree on the set of operations that had been completed before the replacement primary takes over.

Active Replication



In the **active** model of replication for fault tolerance, the RM are state machines that play equivalent roles and are organised as a group. Front ends multicast their requests to the group and each RM processes the request independently but identically and replies. Active replication can tolerate byzantine failures because the front end can collect and compare responses.

Replication for High Availability

- When replication is included for high availability the intention is to give clients access to the service for as much of the time as possible.
- If data are replicated at two or more failure-independent servers then client software should be able to access data at an alternative server should the default server fail or become inaccessible.
- Furthermore replicated service can help to keep the response times low.
- Consistency issues are less of a priority; instead the priority is to return a response to the client. Thus collective agreement may not be reached before a response is made.
- Propagation of updates may be **lazy** rather than the **eager** style adopted in most fault tolerant replicated services.

The Gossip Architecture

- The gossip architecture is a framework for providing a highly available service which replicates data close to the points where groups of clients request it.
- RMs exchange "**gossip**" messages periodically in order to convey the updates they have each received from clients.
- A gossip service provides two basic types of operation:

  queries are read-only operations; and
  updates modify but do not read the state.

- A front end can send a query or update to any RM that it chooses (based on availability and response time).
- RM may be temporarily unable to communicate with each other due to failures. Nevertheless the system guarantees a form of consistency called **relaxed consistency**.

Gossip System Guarantees

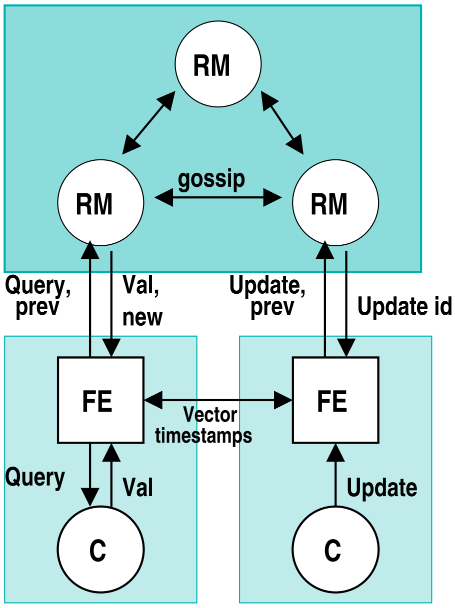
Each client obtains a consistent service over time: Responding to a query a RM provides a client with data that reflects at least the updates that client has observed so far.

Note that this is guaranteed even though a client may communicate with different RM at different times.

Relaxed consistency between replicas: All RM eventually receive all updates and they apply updates with ordering guarantees that make the replicas sufficiently similar to meet the needs of the application.

This is weaker than sequential consistency and may mean that clients observe stale data. It is supported by **causal update ordering**: if the issue of request r *happened-before* the issue of request r' then any correct RM handles r before r'. However stronger ordering conditions may be applied at higher operational costs.

The Gossip Architecture



The gossip service front end handles client operations using an application-specific API and turns them into gossip operations (queries or updates). RM updates are **lazy** in the sense that gossip messages may be exchanged only occasionally. Each front end keeps a **vector timestamp** *prev*, reflecting the latest data values accessed by the client/front end. When clients communicate directly they piggyback their vector timestamps, which are then merged.

Gossip Replica Manager: main components

Value This records the application state as maintained by this RM.

Value timestamp This represents the updates currently reflected in the value.

Update log All update operations are recorded in the log as soon as they arrive although they may be applied in a different order. An update is termed **stable** if it can be applied consistently with the desired ordering. An arriving update is stored until it is stable and can be applied. It then remains in the log until the RM receives confirmation that all other RMs have received the update.

Replica timestamp This timestamp represents those updates that have been accepted by the RM (not necessarily applied).

Executed operation table Maintained to prevent an update being applied twice and checked before an update is added to the log.

Timestamp table Contains a vector timestamp for each other RM, derived from gossip messages.

Gossip Replica Manager