Distributed Systems

Aleardo Manacero Jr.

UNIVERSITY OF OREGON

Replication - part 1

Introduction

- Using multiple servers to attend client requests allow for a better performance in the system
- Unfortunately, as shown in the study of transactions, server crashes may induce abortions or extra delays
- This can be solved if other versions of objects are present in the system
- This is provided by the replication mechanism

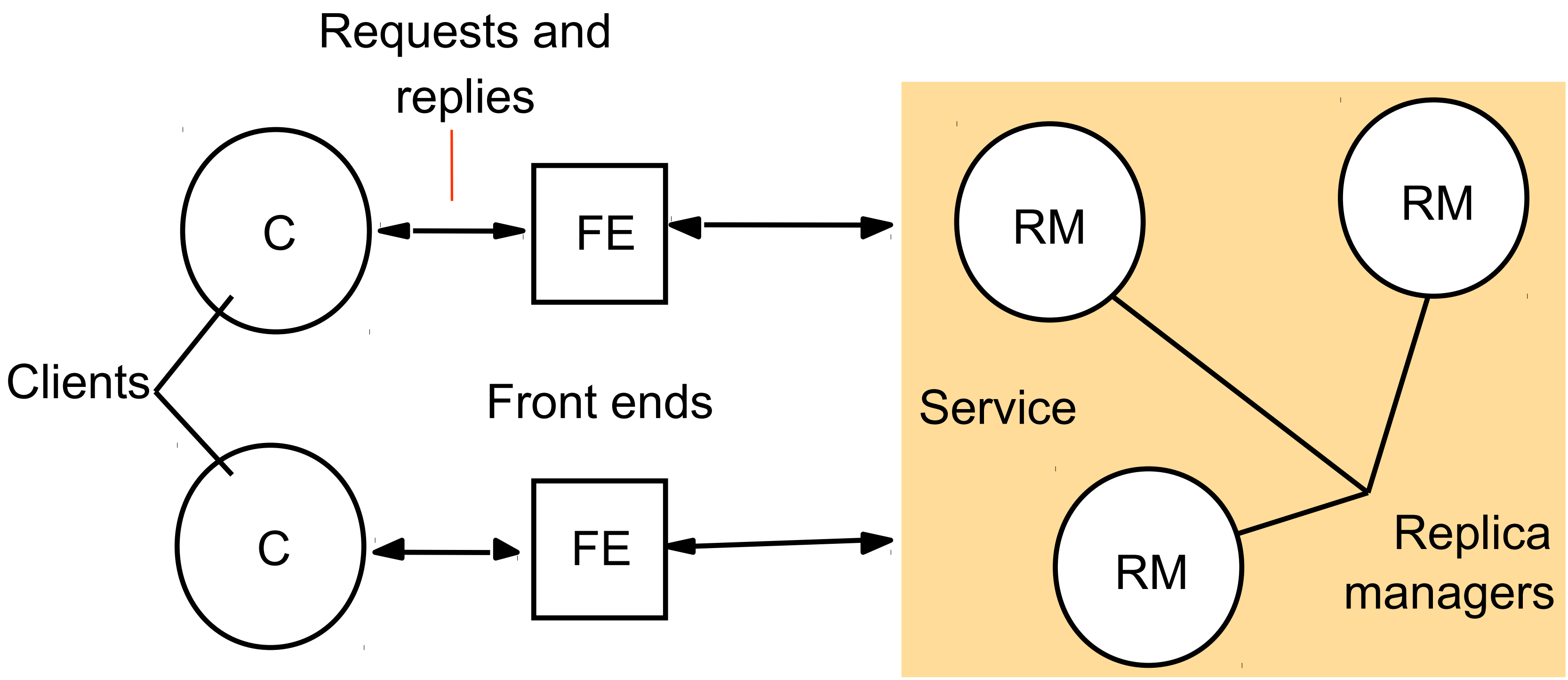
Introduction

- Replication allows for increases in the system's availability and consistency
- There are several motivations to implement replication, such as
  - Performance enhancement
  - Increased availability (either by server failures or network partitions)
  - Fault tolerance

System model for replication

- We will consider objects as a general entity comprising individual pieces up to complete files
- Each object is implemented through physical copies, the replicas
- The replicas of a given object may have different "values" in specific times, since the update of all replicas cannot be instantaneous
- This temporary inconsistency is tolerable for certain applications
- It is minimized by an efficient use of replica managers

Management of replicated data



System model for replication

- Replica managers must operate in a recoverable approach, avoiding inconsistent results in case of a crash
- Non-deterministic effects over objects cannot be guaranteed
- The operation of a replica manager involves five phases: request, coordination, execution, agreement, response

Request

- Front-end issues the client's request to one or more replica managers by:
  - Communicating with a single manager, who can communicate with others
  - Multicasting the request to a set of managers

Coordination

- Managers must agree upon the execution of the request (if it will happen or not) and about the relative ordering of the request to other requests
- Ordering may be (for managers that handle r'):
  - **FIFO: if front-end requests r before r', correct replica managers handle r before r'**
  - Causal: if request r happened-before r', correct managers handle r before r'
  - Total: if a correct manager handles r before r', any correct manager handles r before r'

Execution

- The replica manager executes the request
- This may be done “tentatively”, where the manager can undo the effects of the request in case of failures

Agreement

- The request will be committed if the replica managers executing it reach consensus over the request

Response

- One or more managers responds to the front-end
- The choice between one or more responses depends on what is the system's goal
  - The fastest response is better for high availability
  - Consolidated responses are better for consistency (avoiding byzantine errors)

Group communication in replicated objects

- The existence of object copies controlled by separated replica managers lead to the concept of groups of managers (those that have copies of a given object)
- Therefore, all managers can communicate using group communication
- The impact of this in event ordering and replica consistency is discussed now

Problems with group communication

- We can devise two categories of problems:
  - Process suspicion
  - Network partition
- Process suspicions are managed by removal of managers from the group, but this reduces the effectiveness of the system
- Network partitions are managed by removing the set of managers in the unreachable network (maybe continuing to exist in the other side)

View delivery

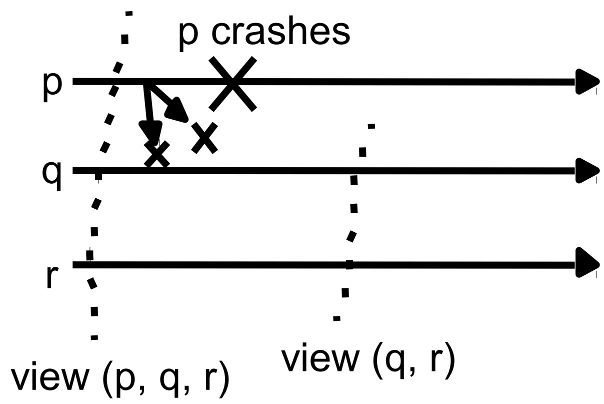
- View deliveries are used to allow the group members to know about group membership changes (processes entering/leaving it)
- These views are transmitted by multicast messages and must be received in a form satisfying to:
    - Order
    - Integrity
    - Non-triviality

View-synchronous communication

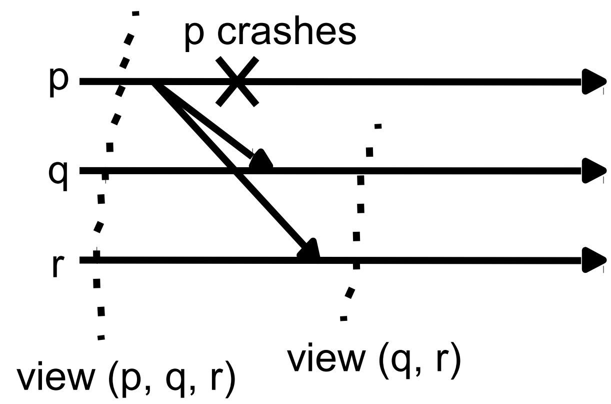
- View-synchronous group communication adds other requirements to member views:
  - Agreement, where if a correct process delivers m in view v(g), then all other correct processes deliver m in v(g)
  - Integrity, where if a process delivers m, then it will not do it again
  - Validity (closed groups), where if the system fails to deliver to a process q, then the next view will exclude q

View-synchronous group communication

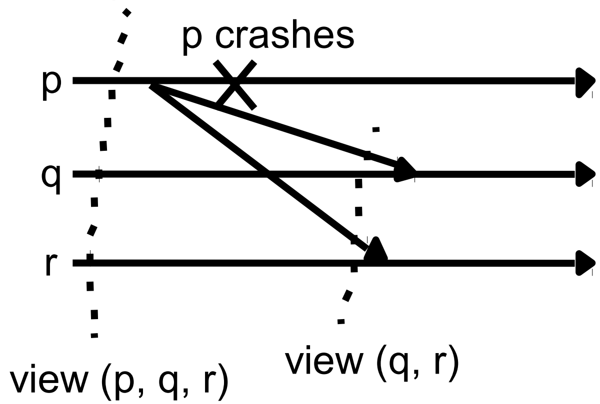
a (allowed).



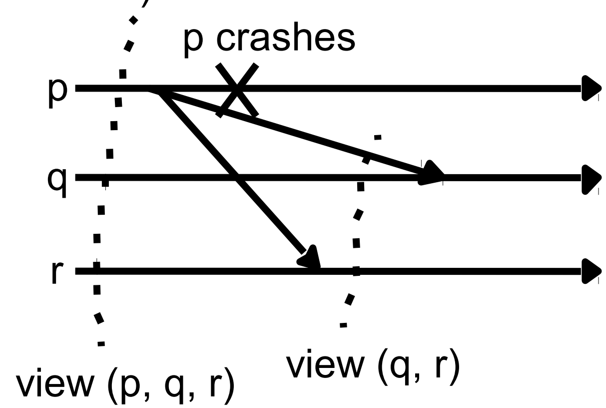
b (allowed).



c (disallowed).



d (disallowed).



Fault-tolerant services

- The goal of fault-tolerant services is to provide the service even in presence of some processes failures
- This is achieved by replicating data and functionality at replica managers
- Even with this scheme, inconsistencies may occur if naive update techniques are used

Linearizability

- Is a strict approach to guarantee correctness
- A replicated object is linearizable if, for any execution, the interleaving of the series of operations satisfies to:
  - Interleaved sequence of operations meets the specification of a single copy
  - The order of operations is consistent with the real times that they occurred

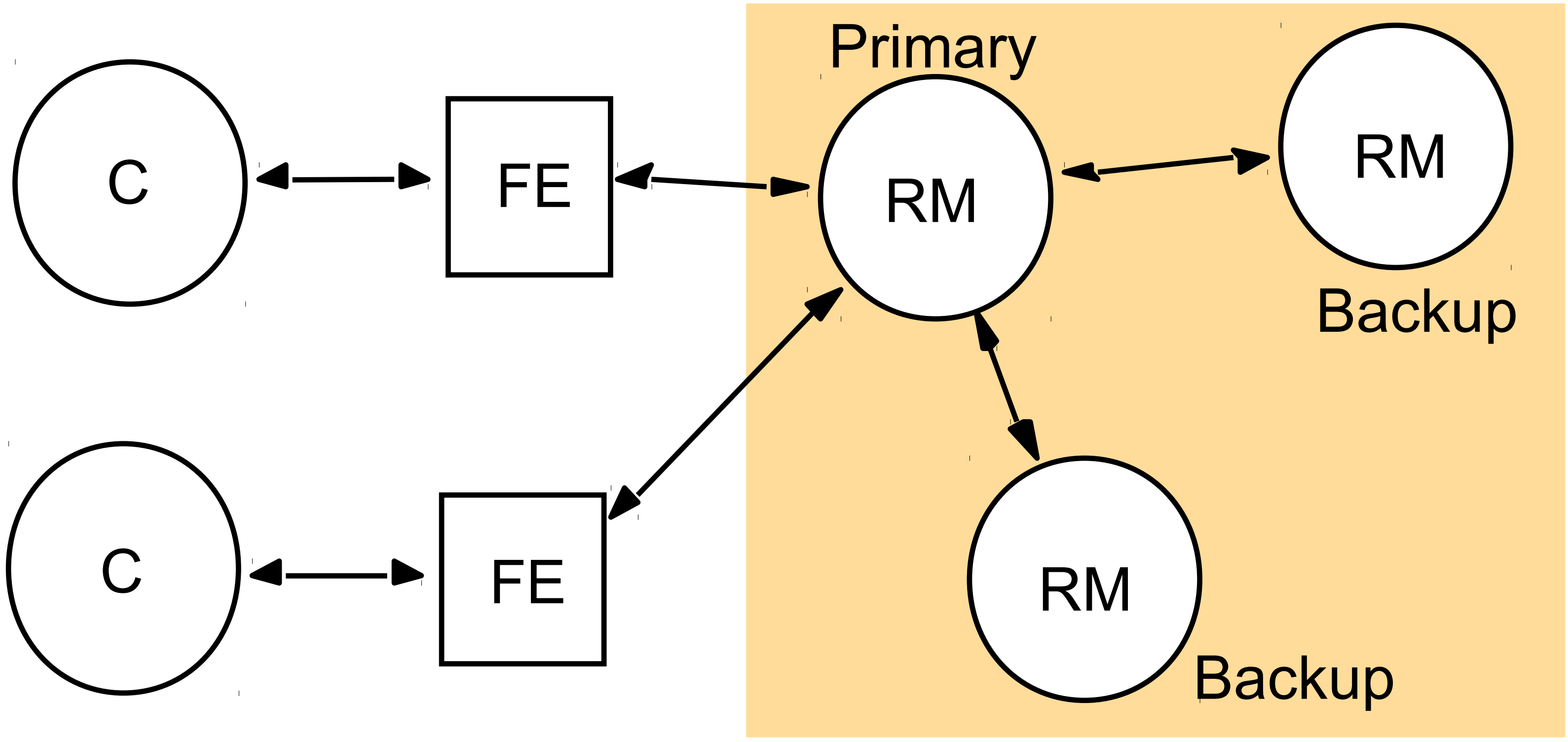
Models for fault tolerance

- Two different models are used with replication for fault-tolerance
- Passive replication, also known as primary-backup, uses a single primary manager and secondary managers (slaves or backups)
- Active replication considers all managers as peers, organized as a group

Passive replication

- In passive replication the front-end sends the request to the primary replica manager
- If the request is an update in the object, the primary sends it to all the backups, who acknowledge to that

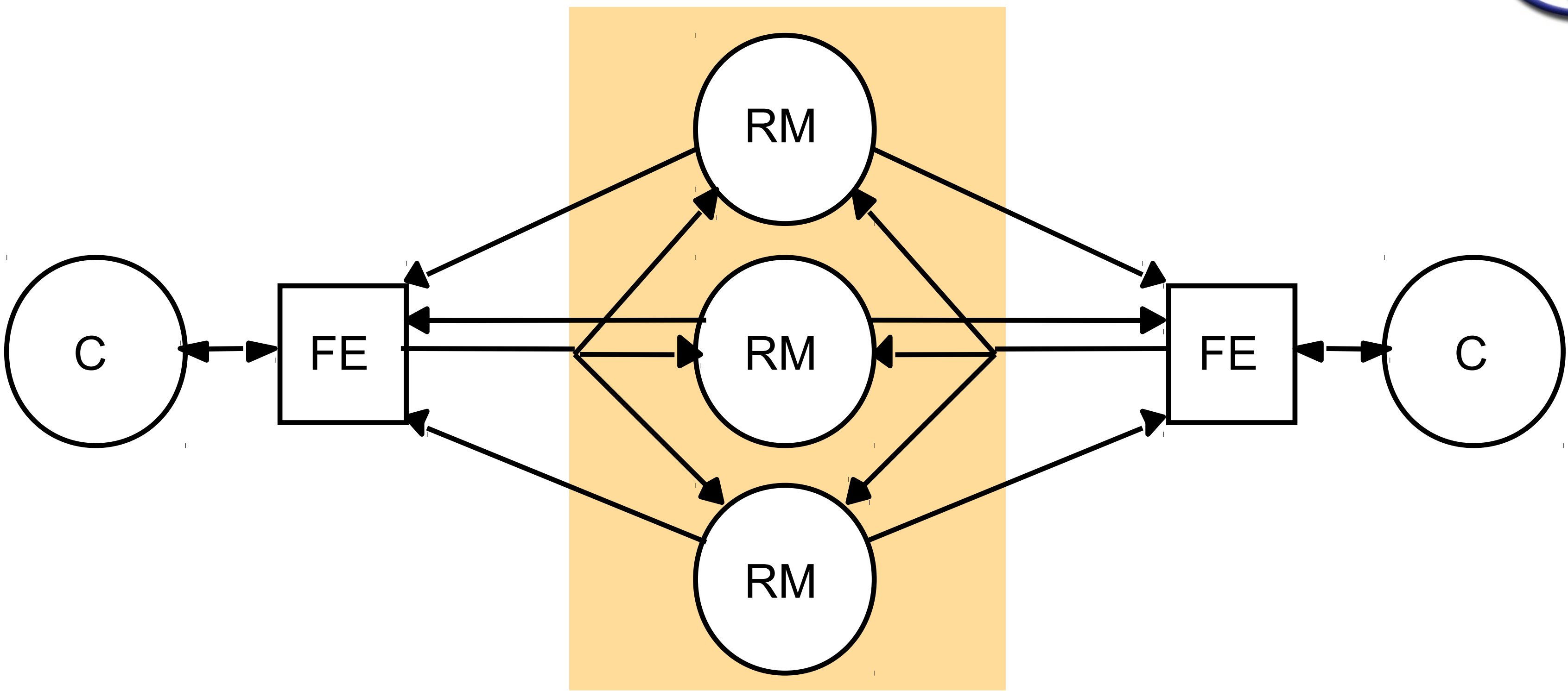
Passive model for fault tolerance



Active replication

- In passive replication the front-end multicasts the request to the group of replica managers
- The use of multicast avoids the need for coordination and agreement
- The front-end must deal with crashes among the managers, possibly through the Byzantine algorithm

Figure 18.4
Active replication



Almost done !!

Programming with message-passing libraries

Message-passing libraries make the task of programming parallel or distributed applications running in distributed systems easier

One of such libraries is the MPI

The MPI

Message Passing Interface, or MPI, was developed to be a standard for parallel programming, and allows for different actual implementations of the library standard, such as openmpi and mpich2

It can be used in environments ranging from multicore machines, to massively parallel machines, including distributed systems, clusters, and so on

The MPI

- It was delevoped based on other existing libraries and languages, such as PVM
- The parallel programming model is restricted to the SPMD model, that is, to a single program, multiple data model
- MPI libraries can be used by programs written in C, C++, Fortran or Java

Libray escope

In a parallel program MPI functions can be used in the interval marked by the instructions

MPI_Init (&argc, &argv)

e

MPI_Finalize

Initializing MPI

MPI_Init receives a copy of the arguments passed by the program call (*argc* e *argv*)

During its execution it initializes a variable called *MPI_COMM_WORLD*, which stores data about the processes started

Programming with MPI

- Control functions

```
MPI_Comm_size(MPI_COMM_WORLD, &size)
/* returns the number of processes started */

MPI_Comm_rank(MPI_COMM_WORLD, &myid)
/* returns the identity (rank) of the process */

MPI_Cart_create(COMM,dims,dsize,wrap,reorder,cart)
/* creates a cartesian topology of dims dimensions */
```

Programming with MPI

- Simple communication functions

MPI_Send(msg, length, type, id, tag, comm)
/* sends msg to process **id** */

MPI_Recv(msg, length, type, id, tag, comm, status)
/* receives msg from process **id** */

MPI_Sendrecv(smsg, slength, stype, dest, stag,
rmsg, rlength, rtype, source, rtag, comm, status)
/* sends smsg to process **dest** and receives rmsg from process **source** */

Programming with MPI

Meaning of communication parameters

- msg is the data (in a buffer)
- length is the size of the data
- type identifies the data type
- id identifies the processes communicating
- tag identifies the message
- status stores the result of the communication

Predefined data types

Predefined MPI datatypes

MPI datatype	*C datatype*
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Programming with MPI

- More functions

MPI_Barrier(group)

/* makes every process in *group* wait for the remaining processes in the group */

MPI_Bcast(msg, count, type, root, comm)

/* Communication by broadcast, started by process with rank *root*, and must be executed in all processes from group *comm* */

Programming with MPI

- Continuing ....

  MPI_Scatter(msg, count, type, rmsg, rcount, rtype, root, comm)

  /* sends a segment (of size count) of msg to each process in comm, which receives the segment in rmsg */

  MPI_Gather(msg, count, type, rmsg, rcount, rtype, root, comm)

  /* receives a segment (msg) from each process and stores them in rmsg)

Programming with MPI

More functions

MPI_Reduce(operand,result,count,type,op,root,comm)

/* same as MPI_Gather, but executing an operation (op) over all received data */

MPI_Allreduce(operand, result, count, type, op, comm)

/* same as MPI_Reduce, but returning the final result to all processes */

op can be one of the reduction operations presented in the next table

Reduction operators

Predefined reduction operators in MPI

Operation Name	*Meaning*
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical and
MPI_BAND	Bitwise and
MPI_LOR	Logical or
MPI_BOR	Bitwise or
MPI_LXOR	Logical exclusive or
MPI_BXOR	Bitwise exclusive or
MPI_MAXLOC	Maximum and location of maximum
MPI_MINLOC	Minimum and location of minimum

Other functions

Besides the functions seen so far MPI provides several other functionalities, such as:

- Definition of the groups of communicating processes (*communicators*)
- Manipulation of communicators topology
- Synchronous and assynchronous communication
- Definition of structured data types

References

ftp://info.mcs.anl.gov/pub/mpi contains the mpich implementation

http://www.mcs.anl.gov/mpi MPI website at Argonne Natl Lab, with lots of extra data

http://www.usfca.edu/~peter/ppmpi contains the sources of code examples in the “Parallel Programming with MPI” of Peter Pacheco

THAT'S IT FOR TODAY !!