Distributed systems – theory

5. Group communications

Group communications

- RPC: two parties, client and server
- Counter-example: a group of file servers cooperating to offer a single, fault-tolerant file system
  - the client send a message to all the servers, to make sure that the request could be carried out even if one of them crashed
  - RPC cannot handle communication from one sender to many receivers (other than by performing separate RPCs with each one)

Groups

- = collection of processes that act together in some system or user-specified way.
- Aim: allow process to deal with collections of processes as a single abstraction -> a process can send a message to a group of servers without having to know how many there are or where they are, which may change from one call to the next
- Key property: when a message is sent to the group itself, all members of the group receive it
  - form of one-to-many communication and is contrasted with point-to-point communication.
- Dynamicity (analogy with social organization !)
  - New groups can be created and old groups can be destroyed.
  - A process can join a group or leave one.
  - A process can be a member of several groups at the same time.

  -> Mechanisms are needed for managing groups and group membership.

Implementations of group communications

1. Multicasting technique
   - create a special network address (for example, indicated by setting one of the high-order bits to 1), to which multiple machines can listen.
   - when a packet is sent to one of these addresses, it is automatically delivered to all machines listening to the address.
   - Implementing groups using multicast is straightforward: just assign each group a different multicast address.
2. Broadcasting technique
   - packets containing a certain address are delivered to all machines.
   - broadcasting can also be used to implement groups, but it is less efficient:
     - each machine receives each broadcast, so its software must check to see if the packet is intended for it.
     - If not, the packet is discarded, but some time is wasted processing the interrupt.
   - it still takes only one packet to reach all the members of a group

Implementations of group communications

1. Multicasting
2. Broadcasting, if multicasting is not allowed
3. Unicasting (point-to-point transmission), if mc or bc are not allowed
   - sending of a message from a single sender to a single receiver
   - the sender transmit separate packets to each of the members of the group.
   - for a group with *n* members, *n* packets are required, instead of one packet when either multicasting or broadcasting is used
   - although less efficient, this implementation is still workable, especially if most groups are small.

Design of group communications

- As regular message passing:
  - Buffered vs. unbuffered
  - Blocking vs. unblocking
  - Etc
- Additional choices
  - Closed Groups versus Open Groups.
  - Peer Groups versus Hierarchical Groups
- Other problems
  - Group Membership
  - Group Addressing
  - Send and Receive Primitives
  - Atomicity
  - Message Ordering
  - Overlapping Groups
  - Scalability

Closed Groups vs Open Groups

- closed groups:
  - in which only the members of the group can send to the group.
  - outsiders cannot send messages to the group as a whole, although they may be able to send messages to individual members
  - Example: A collection of processes working together to play a game of chess might form a closed group; they have their own goal and do not interact with the outside world.
- open groups:
  - any process in the system can send to any group
  - Example: support replicated servers, it is important that processes that are not members (clients) can send to the group.

Peer Groups vs. Hierarchical Groups

- Peer Groups
  - all the processes are equal.
  - no process is boss and all decisions are made collectively
  - peer group is symmetric and has no single point of failure.
  - if one of the processes crashes, the group simply becomes smaller, but continue.
  - Disadvantage: decision making is more complicated --to decide anything, a vote has to be taken, incurring some delay and overhead
- Hierarchical Groups
  - one process is the coordinator and all the others are workers.
  - when a request for work is generated, either by an external client or by one of the workers, it is sent to the coordinator who decides which worker is best suited to carry it out & forwards it there
  - loss of the coordinator brings the entire group to a grinding halt, but as long as it is running, it can make decisions without bothering everyone else.
  - E.g. a hierarchical group might be appropriate for a chess program:
    - The coordinator takes the current board, generates all the legal moves from it, and farms them out to the workers for evaluation.
    - During this evaluation, new boards are generated and sent back to the coordinator to have them evaluated.
    - When a worker is idle, it asks the coordinator for a new board to work on.
    - The coordinator controls the search strategy and prunes the game tree but leaves the actual evaluation to the workers.

Group Membership

- some method is needed
  - for creating and deleting groups, as well as
  - for allowing processes to join and leave groups.
- Approaches:
  1. have a group server to which all these requests can be sent.
  2. manage group membership in a distributed way.
     - In an open group, an outsider can send a message to all group members announcing its presence
     - In a closed group, something similar is needed
     - To leave a group, a member just sends a goodbye message to everyone.

Group Addressing – approaches

1. Give each group a unique address, much like a process address.
   - multicast allowed: the group address can be associated with a multicast address,
   - broadcast allowed: the message can be broadcast.
   - only unicasting allowed: need a list of machines that have processes belonging to the group.
2. Require the sender to provide an explicit list of all destinations.
3. Each message is sent to all members of the group using one of the methods described above, but with a new twist:
   - Each message contains a predicate (Boolean expression) to be evaluated.
   - The predicate can involve the receiver's machine number, its local variables, or other factors.
   - If the predicate evaluates to TRUE, the message is accepted.
   - If it evaluates to FALSE, the message is discarded.
   - Example: send a message to only those machines that have at least 4M of free memory & are willing to take on a new process.

Send and Receive Primitives

1. Merge the two form of comm. : group & point2point?
   - Send:
     - Parameter -- destination
       - A process address, a single message is sent to that one process.
       - A group address (or a pointer to a list of destinations), a message is sent to all members of the group
   - Receive:
     - completes when either a point-to-point message or a group message arrives.
2. New library procedures:
   - group-send
   - group-receive

Atomicity

- desirable because it makes programming distributed systems much easier.
- a process sends a message to the group, it does not have to worry about what to do if some of them do not get it
- Example – in a replicated distributed data base system:
  - a process sends a message to all the data base machines to create a new record in the database
    - The record is replicated in all copies

Message ordering

- Global time ordering:
  - The best guarantee is to have all messages delivered instantaneously and in the order in which they were sent.
  - All recipients get all messages in exactly the same order.
  - It conveniently ignoring the fact that there is no such thing as absolute global time!
- Absolute time ordering is not always easy to implement! -> some systems offer various watered-down variations. For example:
- Consistent time ordering:
  - If two messages, say A and B, are sent close together in time, the system picks one of them as being "first" and delivers it to all group members, followed by the other.
  - It may happen that the one chosen as first was not really first, but since no one knows this, the argument goes, system behavior should not depend on it.
  - Messages are guaranteed to arrive at all group members in the same order, but that order may not be the real order in which they were sent.

Overlapping Groups

- Although there is a time ordering within each group, there is not necessarily any coordination among multiple groups
- Some systems support well-defined time ordering among overlapping groups and others do not
- If the groups are disjoint, the issue does not arise.
- Implementing time ordering among different groups is frequently difficult to do.