Introduction to Distributed and Parallel Computing
CS-401

Dr. Sanjay Saxena
Visiting Faculty, CSE, IIIT Vadodara
Assistant Professor, CSE, IIIT Bhubaneswar
Post doc – University of Pennsylvania, USA
PhD – Indian Institute of Technology(BHU), Varanasi

Remote Procedure Call (RPC)

Remote Procedure Call (RPC) is a powerful technique for constructing **distributed, client-server based applications**. It is based on extending the conventional local procedure calling so that the **called procedure need not exist in the same address space as the calling procedure**.

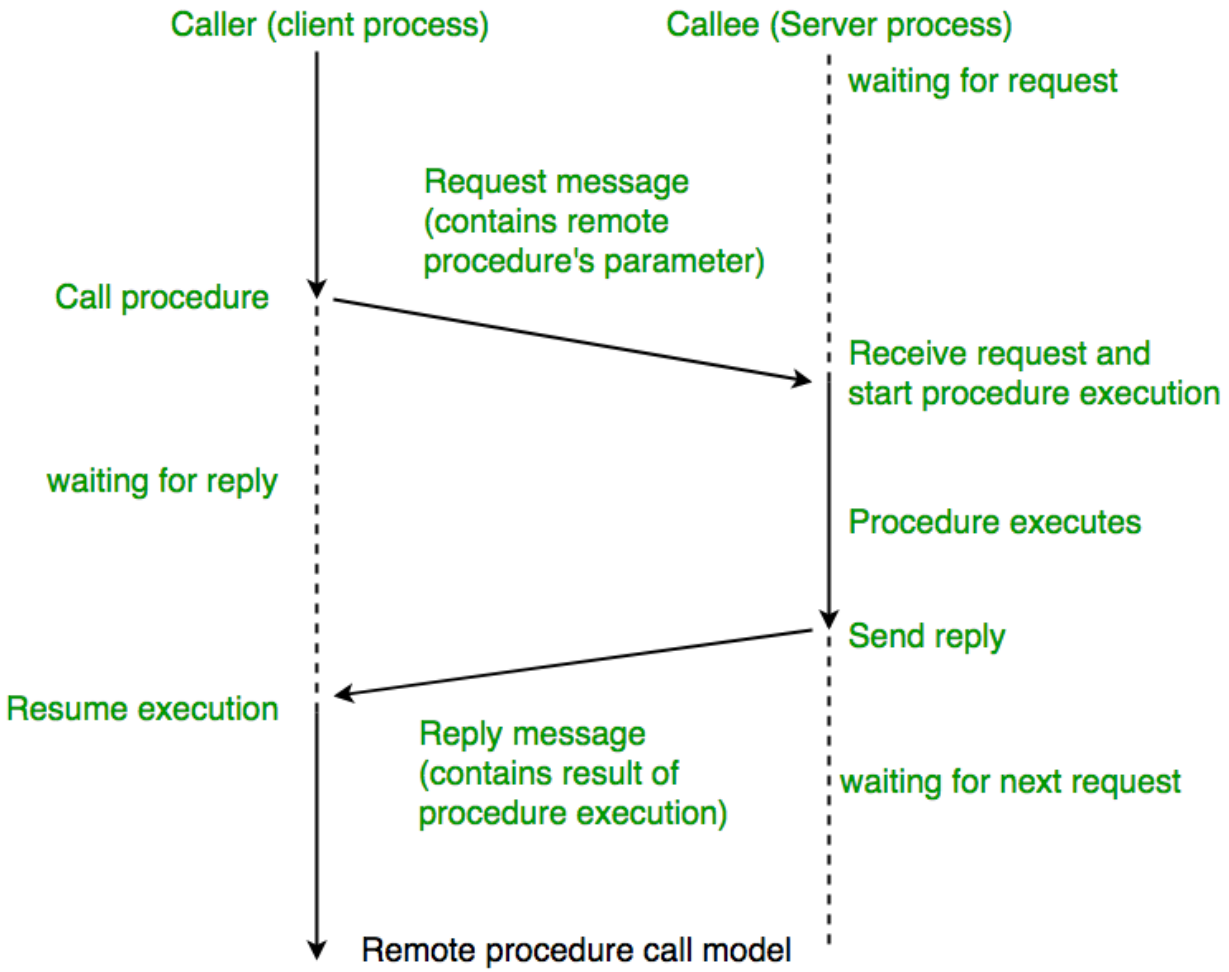
The two processes may be on the same system, or they may be on different systems with a network connecting them.

When making a Remote Procedure Call:

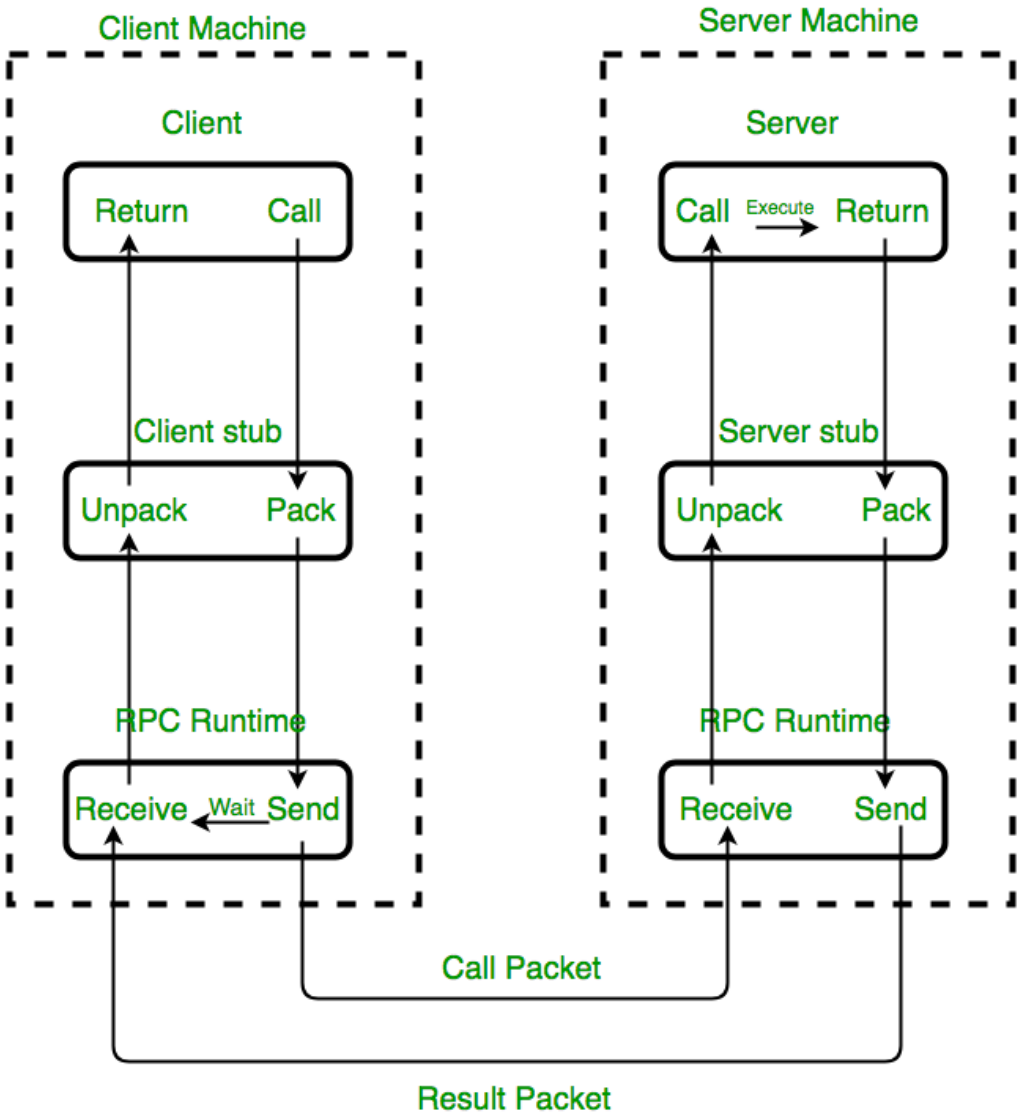
1. The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.

2. When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.

NOTE: RPC is especially well suited for client-server **(e.g. query-response)** interaction in which the flow of control **alternates between the caller and callee**. Conceptually, the client and server do not both execute at the same time. Instead, the thread of execution jumps from the caller to the callee and then back again.



Working of RPC



Implementation of RPC mechanism

The following steps take place during a RPC :

- A client invokes a **client stub procedure**, passing parameters in the usual way. The client stub resides within the client's own address space.
- The client stub **marshalls(pack)** the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.
- The client stub passes the message to the transport layer, which sends it to the remote server machine.
- On the server, the transport layer passes the message to a server stub, which **demarshalls(unpack)** the parameters and calls the desired server routine using the regular procedure call mechanism.
- When the server procedure completes, it returns to the server stub **(e.g., via a normal procedure call return)**, which marshalls the return values into a message. The server stub then hands the message to the transport layer.
- The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.
- The client stub demarshalls the return parameters and execution returns to the caller.

Key Considerations for Designing and Implementing RPC Systems are:

- **Security:** Since RPC involves communication over the network, security is a major concern. Measures such as authentication, encryption, and authorization must be implemented to prevent unauthorized access and protect sensitive data.
- **Scalability:** As the number of clients and servers increases, the performance of the RPC system must not degrade. Load balancing techniques and efficient resource utilization are important for scalability.
- **Fault tolerance:** The RPC system should be resilient to network failures, server crashes, and other unexpected events. Measures such as redundancy, failover, and graceful degradation can help ensure fault tolerance.
- **Standardization:** There are several RPC frameworks and protocols available, and it is important to choose a standardized and widely accepted one to ensure interoperability and compatibility across different platforms and programming languages.
- **Performance tuning:** Fine-tuning the RPC system for optimal performance is important. This may involve optimizing the network protocol, minimizing the data transferred over the network, and reducing the latency and overhead associated with RPC calls.

ADVANTAGES :

- RPC provides **ABSTRACTION** i.e message-passing nature of network communication is hidden from the user.
- RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often.
- RPC enables the usage of the applications in the distributed environment, not only in the local environment.
- With RPC code re-writing / re-developing effort is minimized.
- Process-oriented and thread oriented models supported by RPC.

Synchronization in Distributed Systems

- Distributed System is a collection of computers connected via a high-speed communication network.
- In the distributed system, the hardware and software components communicate and coordinate their actions by message passing.
- Each node in distributed systems can share its resources with other nodes. So, there is a need for proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used.
- Synchronization in distributed systems is achieved via clocks. The physical clocks are used to adjust the time of nodes. Each node in the system can share its local time with other nodes in the system.
- The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system. Clock synchronization can be achieved by 2 ways: External and Internal Clock Synchronization.

1. **External clock synchronization** is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
2. **Internal clock synchronization** is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

- There are 2 types of clock synchronization algorithms: Centralized and Distributed.

➢ **Centralized** is the one in which a time server is used as a reference. The single time-server propagates it's time to the nodes, and all the nodes adjust the time accordingly. It is dependent on a single time-server, so if that node fails, the whole system will lose synchronization. Examples of centralized are-Berkeley the Algorithm, Passive Time Server, Active Time Server etc.

➢ **Distributed** is the one in which there is no centralized time-server present. Instead, the nodes adjust their time by using their local time and then, taking the average of the differences in time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol), etc.

- Centralized clock synchronization algorithms suffer from two major drawbacks:

1. They are subject to a single-point failure. If the time-server node fails, the clock synchronization operation cannot be performed. This makes the system unreliable. Ideally, a distributed system should be more reliable than its individual nodes. If one goes down, the rest should continue to function correctly.
2. From a scalability point of view, it is generally not acceptable to get all the time requests serviced by a single-time server. In a large system, such a solution puts a heavy burden on that one process.

- Distributed algorithms overcome these drawbacks as there is no centralized time-server present. Instead, a simple method for clock synchronization may be to equip each node of the system with a real-time receiver so that each node's clock can be independently synchronized in real-time. Multiple real-time clocks (one for each node) are normally used for this purpose.

Logical Clock in Distributed System

- **Logical Clocks** refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

- **Example :**

  If we go outside then we have made a full plan that at which place we have to go first, second and so on. We don’t go to second place at first and then the first place. We always maintain the procedure or an organization that is planned before. In a similar way, we should do the operations on our PCs one by one in an organized way.

- Suppose, we have more than 10 PCs in a distributed system and every PC is doing it’s own work but then how we make them work together. There comes a solution to this i.e. LOGICAL CLOCK.

- **Method-1:**

  To order events across process, try to sync clocks in one approach.

  This means that if one PC has a time 2:00 pm then every PC should have the same time which is quite not possible. Not every clock can sync at one time. Then we can’t follow this method.

- **Method-2:**

  Another approach is to assign Timestamps to events.

- Taking the example into consideration, this means if we assign the first place as 1, second place as 2, third place as 3 and so on. Then we always know that the first place will always come first and then so on. Similarly, If we give each PC their individual number than it will be organized in a way that 1st PC will complete its process first and then second and so on.
- BUT, Timestamps will only work as long as they obey causality.

- **What is causality ?**

  Causality is fully based on HAPPEN BEFORE RELATIONSHIP.

- Taking single PC only if 2 events A and B are occurring one by one then TS(A) < TS(B). If A has timestamp of 1, then B should have timestamp more than 1, then only happen before relationship occurs.

- Taking 2 PCs and event A in P1 (PC.1) and event B in P2 (PC.2) then also the condition will be TS(A) < TS(B). Taking example- suppose you are sending message to someone at 2:00:00 pm, and the other person is receiving it at 2:00:02 pm.Then it’s obvious that TS(sender) < TS(receiver).

Election algorithm and distributed processing

- **Distributed Algorithm** is an algorithm that runs on a distributed system. Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on another machine. Many algorithms used in the distributed system require a coordinator that performs functions needed by other processes in the system.

Election algorithms are designed to choose a coordinator.

Election Algorithms: Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor.

- Election algorithm basically determines where a new copy of the coordinator should be restarted.
- Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number.Then this number is send to every active process in the distributed system. We have two election algorithms for two different configurations of a distributed system.

1. The Bully Algorithm – This algorithm applies to system where every process can send a message to every other process in the system. **Algorithm –** Suppose process P sends a message to the coordinator.

- If the coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
- Now process P sends an election messages to every process with high priority number.
- It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
- Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
- However, if an answer is received within time T from any other process Q,
    1. (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
    2. (II) If Q doesn't responds within time interval T' then it is assumed to have failed and algorithm is restarted.

2. The Ring Algorithm – This algorithm applies to systems organized as a ring(logically or physically). In this algorithm we assume that the link between the process are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is **active list**, a list that has a priority number of all active processes in the system.

Algorithm –

1.If process P1 detects a coordinator failure, it creates new active list which is empty initially. It sends election message to its neighbour on right and adds number 1 to its active list.

2.If process P2 receives message elect from processes on left, it responds in 3 ways:

1. (I) If message received does not contain 1 in active list then P1 adds 2 to its active list and forwards the message.
2. (II) If this is the first election message it has received or sent, P1 creates new active list with numbers 1 and 2. It then sends election message 1 followed by 2.
3. (III) If Process P1 receives its own election message 1 then active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects highest priority number from list and elects it as the new coordinator.

Thanks & Cheers!!

Small aim is a crime; have great aim.
Bharat-Ratan A. P. J. Abdul Kalam