LECTURE 3

Remote Procedure Calls
What are RPCs?

- Distributed systems are especially suited for providing services (e.g., MapReduce).
- Explicit message passing can be used for communications between processes – but do not conceal communications.
- How do we access services?
  - Simply allow programs to call procedures that are located and execute on other machines
    - Process 1 on Machine A calls Procedure2 on Machine B.
    - Blocks and waits for the procedure to return
    - Continues execution
  - Message passing hidden from programmer
- This method is called Remote Procedure Call or RPC for short
- RPC is widely used and forms the basis for communications in many distributed systems.
Challenges

- While the basic idea is simple, hard to implement.
- Procedure executes on a different machine with a different address space.
- Passing parameters and results is necessary.
- Fault tolerance and synchronization
Arguments

- Conventionally, arguments are passed by value or by reference (pointers).
  - e.g., `append(data, dBList);`
- Copy-by-value or Copy-by-reference
- The data may be values of local variables and `dBList` maybe a pointer to a list.
- These are then pushed to the Stack
- Programmer doesn’t need to Know what append does. Only pass arguments.
Principles of RPC

- The procedure is executed on a remote machine.
  - What do we do?
- There is a different version of append() running on the client.
  - Called a client stub
- It does not perform an append operation – instead it packs the parameters into a message which is sent to a server.
- Using send()
Principle of RPC (2)

- The client then blocks itself until the reply from server gets back.
- Server’s OS passes message to a server stub.
  - Equivalent to client stub.
  - Waits for incoming requests by calling receive().
- Unpacks parameters from message and then calls the server procedure in the usual way.
  - As if being called directly by the client.
- Performs its work and returns result.
Principles of RPC (3)

- When the result is received, (using receive()) it unblocks the client stub.
- The client stub inspects message, unpacks result, and copies it to the caller.
  - It returns in the usual way.
  - The caller gets control – and all it knows is that the data has been appended to dBList.
Steps of RPC

1. Client call to procedure
2. Stub builds message
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to “doit”
Parameter marshaling

- Packing parameters into messages is called parameter marshaling.
- Needed because the parameters (e.g., data and dBList) sent over the network are correctly interpreted.
- Little Endian vs Big Endian byte ordering
  - Standardized to Big Endian
- Transform the data to a machine and network independent format (essential)
  - Can be done with programming support.
How are pointers or references passed?

- Cannot just forbid pointer and reference parameter passing.

Solution: Copy the entire data structure

- Effectively replaces copy-by-reference to copy-by-value/restore (return restores the reference)

- Not semantically exact but good enough

- When client knows that the referred data is read only – no need for restore.
More complex parameter passing

The Machine A is executing a RPC on Machine C; There is a local object and a remote object.
Interface Definition Language

- Often specified by means of an Interface Definition Language (IDL).
  - Example the DCE (Distributed Computing Environment) RPC – from the Open Software Foundation
- Defining the message format
- Representation of different (simple) data structures such as integers, characters etc.
- Protocol of use (e.g., TCP)
void someFunction(char x; float y; int z[5])

(a)

- Rules for how to encode.
- Example: Character in rightmost byte of a word, with the following 3 bytes empty.
- A float is a whole word
- An array is a group of words equal to the array length; preceded by size.
Asynchronous RPC

- Sometimes the RPC does not have to return a result to the client
- Blocking an issue
- Asynchronous RPC eliminates this issue.

- Server ACKS request at which point client continues.

![Diagram of Asynchronous RPC](image)
Asynchronous RPCs with returns

- Client may not want to wait for the RPC to return
  - For example, MapReduce where client interacts with many servers simultaneously.
- Combine RPC with callback (a user defined function which is invoked upon a certain event like message receipt)
Multicast RPC

- Allows sending procedure requests to multiple servers (e.g., MapReduce)