1

Oliver Kugel

Remote Procedure Calls (RPCs)

- RPCs occur when a computer causes a procedure / function to execute in another address space, commonly on another computer on a shared network}.
- The programmer doesn't explicitely code the details for the remote interaction.
- RPCs provide access transparency (local service looks like remote service)
- Goal of RPCs: Hide the network from the program

Sequence of events during an RPC

1. The client calls the Client stub. This is a local procedure call with parameters pushed on the stack in the normal way.
2. Marshalling. Client stub packs parameters into a message, makes a system call and the OS sends the message over the network to the server.
   Packing parameters = marshalling.
3. The server's OS passes the incoming packets to the server stub.
4. Unmarshalling. The server stub unpacks the parameters from the message.
5. The server stub calls the server procedure.
6. The reply (i.e. return result) traces the same steps in reverse direction.

Synchronous vs. asynchronous RPC

Synchronous RPC	Asynchronous RPC
Synchronous RPC behaves like local one. Program waits for return results and only carries on after it's received them.	After an asynchronous RPC program only waits for acceptance but then carries on, on arrival of the return result the remote machine interrupts the local program, which then sends an ACK.

Java RMI

- Where with RPCs you can solely call remote functions exported into a server, with Remote Method Invocation (RMI) you can have references to remote objects and invoke their methods.
  instance methods

RMI Registry

This is the place for the server to register the services it offers and the place for clients to query those services.

Words on Parallel Computing

- Parallel computing means using multiple processors (cores) within the same computer.
- Two common problems: Some computations are hard or impossible to parallelise. More cores doesn't always mean more speed up.

Amdahl's Law

Shows that unless a program is 100% efficient at using multiple cores, you will receive less and less of a benefit by adding more cores.

$$\text{running time} = b + t \cdot \left(1 - x + \frac{x}{p}\right)$$

b: time it takes to setup, synchronise and communicate between cores.

x: proportion of your application that you can speed up by parallelisation.

p: number of cores

t: a program runs on one machine in time t.

speed up = sequential time / parallel time

Algorithms for choosing a Coordinator Node

Ring-based algorithm

(1) We arrange all processes/nodes in a logical ring and assume no failures. All nodes are set to be "non-participants".

(2) Initiating process makes himself "participant" and sends its identifier in an election message to its neighbour.

(3) Receiver compares the received with its own identifier.
if ($ID_{message} > ID_{own}$) : forward message unchanged
else : replace $ID_{message}$ with ID_{own} and forward message
Receiver is now a participant.

(4) If ($ID_{message} = ID_{own}$)
The respective node has won and becomes the coordinator

(5) Coordinator sends an elected message around the ring to tell everyone ("I'm the coordinator, this is my ID.").

Bully algorithm

(1) P sends an election message to all processes/nodes with higher numbers. (P is the initiator).

(2) If noone responds P wins the election, becomes coordinator and informs all other nodes via a coordinator message

(3) If a higher-numbered node Q answers P via an answer message, P doesn't win and Q begins the election process again. This repeats until one process wins.

Note 1: Failures are tolerated since wait-times for nodes can be controlled via time-outs.

Note 2: All nodes must know about the IDs (e.g. IP addresses) of the participating nodes and can communicate with them.

Algorithms for Clock Synchronisation

Cristian's algorithm

① Client C with time t_c sends a time request to server S.

② S sends C his time t_s.

③ C receives t_s at time t_r

④ C sets his time to $t_s + \frac{RTT}{2}$, $RTT = t_r - t_c$ round trip time

Berkeley algorithm

① There is one master node that polls its slave nodes.

② Each slave replies to the master with its time.

③ The master averages the slaves' and its own time and eliminates any times with excessive RTTs. to compute one final time.

④ The master sends each slave a delta which tells them how much to add or take off to/from their clocks.

Network time protocol (NTP)

- Works for networks of a larger scale (unlike Cristian, Berkeley)
- There are three methods of synchronization

Multicast mode	• Time server sends his time to all servers on LAN at once. • Each server receives and resets its clock (assuming little delay)
Procedure call mode	• Effectively Cristian's algorithm • Requesting node sets its time to $t_s + \frac{RTT}{2}$
Symmetric mode	• Most accurate, messages are exchanged and data is built up to improve accuracy of synchronization over time. Messages contain timing info about the previous message received (time sent, time received, etc.)

Replication ~~and Redundancy~~

~~Definition~~ Reason 1 : Redundancy

Redundancy is a key technique to increase availability. If one server crashes, then there is a replica that can still be used. Thus, failures are tolerated by the use of redundant components.

> **Physical**
> E.g. use 3 engines eventhough you only need 1
>
> **Time**
> Perform an action again, and again, ...
>
> **Information**
> E.g. send extra bits over the network, allow recovery

Probability of availability

$$
\begin{aligned}
P(\text{service is available}) &= 1 - P(\text{all replicas have failed}) \\
&= 1 - \big(P(\text{replica 1 has failed}) \\
&\qquad \cdot P(\text{replica 2 has failed}) \\
&\qquad \cdot P(\text{replica 3 has failed}) \\
&\qquad \cdot \text{ and so on}\big)
\end{aligned}
$$

$$
P(\text{replica } x \text{ has failed}) = \frac{\text{mean time to repair failure}}{\text{mean time between failures} + \text{mean time to repair failure}}
$$

Reason 2: Performance

- By placing a copy of the data closeby the process using them, the time to access the data decreases. This is also useful to achieve scalability.
- One approach to this is caching: web browsers store locally a copy of a previously fetched web page to avoid the latency of fetching the resource again.

Consistency problem

- If a copy (i.e. a replication) is modified, this copy becomes different from the rest. Consequently, modifications have to be carried out on all copies, to ensure consistency.
- When and how these modifications need to be carried out, determines the price of replication.

Consistency Models

Definition

A consistency model is a contract between processes and the data store. It says that if processes agree to obey certain rules, the store promises to work correctly.

Strict consistency

- Any read to a shared data item x returns the value stored by the most recent write operation on x.
- This means there is an absolute time ordering of all shared accesses.
- This doesn't make sense in a distributed system since it requires absolute global time. This is impossible (limit: speed of light).

Sequential consistency

- The result of any execution is the same as if operations of all processes were executed in some sequential order and the operations of each process appear in the order specified by the program.
- ~~Bsp~~ Analogie: Zwei Kartenstapel, jeweils einer in jeder Hand, Karten werden ineinander gemischt. Die Reihenfolge des Zusammenmischens ist beliebig, aber die Reihenfolge der beiden Stapel wird beibehalten.

- Example:

Causal consistency

- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order by different processes.
- Example:

Eventual consistency

- Given a sufficiently long period of time over which no changes are sent, all updates are expected to propagate, and eventually all replicas will be consistent.

Where to place replica servers

1. Find the total cost of accessing each site from all the other sites
2. Choose the site with the minimum total cost, make it a replica server (i.e. host the other guys' contents)
3. Go to 1, taking into account sites hosting replicas, choose by min(value, value-to-nearest-replica).
4. Do this until you have the desired number of replicas.

Denial of Service Attack in Lab2

- Ignore the 1s-delay-rule.
- Send requests within an infinite loop, while(true) ...
- Make more people do the same.

Two Generals' Problem

Description

- Two armies, surrounding a city, are prepared to attack the city. They must attack at the same time in order to succeed, otherwise they will fail.
- They send messages to each other through an unreliable medium, e.g. a message-carrier-boy walking through the city.
- There is no way to guarantee that both generals agree a message was delivered, especially the last message!

With traitors – Byzantine Generals' Problem

- Several divisions of the byzantine army are camped outside an enemy city, each division commanded by its own general
- The generals communicate via an unreliable channel
- They must decide upon a common plan of action (attack or retreat)
- Some of the generals may be traitors, forwarding wrong information.
- A reliable agreement is possible for $g > 3 \cdot t$, where g = number of generals, t = number of traitors among them.
- An agreement is impossible for $g \leq 3 \cdot t$
- Possible solutions: redundancy (e.g. send every message 100 times, expect more than one ACK, ...) and majority voting (which message is most probable to be true?)

Transactions

Definition

A transaction is an individual, indivisible operation that provides the ACID properties.

ACID

ACID stands for Atomicity, Consistency, Isolation and Durability

all or nothing

never leave system in inconsistent state

don't affect the "outside world", be visible only when you're done

don't change until the next transaction

ACID properties are partially implemented by the two-phase-commit algorithm.

Two-phase-commit

1. A co-ordinator node requests a transaction and sends a request to all participants.
2. All participants respond if they're willing + able to execute the request and send VOTE_COMMIT or VOTE_ABORT
3. The participants log their current state and perform the transaction.
4. All participants log their vote.
5. The coordinator looks at the votes. If all participants have voted to commit the coordinator sends a GLOBAL_COMMIT to everyone. Otherwise he sends a GLOBAL_ABORT.
6. Participants receive the decision from the coordinator and record it locally. If it was an ABORT, participants roll back to their previous state.

which they logged in step 3

Little's Law

The long-term average number of customers n in a stable system is equal to the long-term average effective arrival rate λ multiplied by the average time a customer spends in the system. t.

$$\boxed{n = \lambda \cdot t}$$

Fallacies of Distributed Systems

1. Latency is greater than zero.
2. Bandwidth is less than infinite.
3. Transport cost is greater than zero.
4. There is more than one administrator.
5. Topology does change.
6. The network is not homogeneous.
7. The network is not secure.
8. The network is not reliable.

What is Middleware?

Middleware is software that acts as a bridge between an operating system (or database) and user applications, especially on a network. It also describes software that enables communication and management of data in Distributed Applications. It's the dash in Client-Server or the "-to-" in Peer-to-Peer.

RPC Semantics

At-least-once semantics

- If the client stub doesn't receive a response within some specific time, the stub resends the request as many times as necessary until it receives a response from the service.
- This may cause the remote procedure to execute a request more than once.
- This only works well for idempotent operations.
  Idempotent = ~~applying the same operation~~ repeating the same request several times has the same effect as doing it just once.
- This approch only fails if the executing server never responds.

At-most-once semantics

- If the client stub doesn't receive a response within some specific time it will not retry before it has received a negative acknowledgment of the invocation.
- This "guarantees" the remote procedure is executed either 0 or 1 time.
- This works well for non-idempotent operations where it is impotant that a remote procedure is only invoked once, e.g. incrementing a bank account balance.

Name server	Directory server
• Takes a name, and returns one or more attributes of the named object.	• Takes attribute values, and returns sets of attributes of objects with those attribute values.
• Analogy: White pages telephone book	• Analogy: yellow pages telephone book

Logical Clocks

Definition

Logical clocks are constantly updated timestamps that enable a basic partial ordering of events, in local as well as distributed systems.

Lamport clocks

1. Each processor/core $\boxed{i}$ has a logical clock $\boxed{LC_i}$
2. An event occurs on processor $\boxed{i}$ $\Rightarrow$ $\boxed{LC_i = LC_i + 1}$
3. When a processor $\boxed{x}$ sends a message to $\boxed{y}$, it also sends its logical clock $\boxed{LC_x}$
4. y receives the message $\Rightarrow$

```
if ( LC_y > (LC_x + 1))
    LC_y = LC_y  // no change
else
    LC_y = LC_x + 1
```

Vector clocks

- A vector clock is similar to the lamport clock above, but <u>each process keeps track of the clock of each other process</u>. It is in essence n lamport clocks for each process, where n = number of processes.
- When a process receives a message, it <u>merges</u> its clock-vector with the clock-vector in the message, finding the maximum of each item.
- Vector clocks overcome the shortcoming of Lamport Clocks which is $\boxed{L(e) < L(e')\text{ does not imply }e\text{ happened before }e'}$.
- ~~Vectore~~ Vector clocks <u>capture causality</u>, which Lamport Clocks do not.
- However, Vector Clocks are more <u>expensive</u> in terms of bandwidth.

Interface definition language (IDL)

IDL is a generic term for a language that lets a program (or object) written in one language communicate with another program (or object) written in an unknown language.
In distributed object technologies it's important that new objects be able to be sent to any platform environment and discover how to run in that environment.