2.2.1 Centralised architectures

Centralised architectures can be seen as equivalent to client-server systems that work on a request-response basis.

Implementations are mostly using messages, i.e. connectionless protocols.

Transmission failures must be treated with resend (transmission must be **idempotent** - repeatable without harm.)

Mostly reliable transmission using TCP/IP.

Server can be client to another server, application layering (tiers!)

Implies a lot of waiting.

Does not scale well.

2.2.2 Decentralised architectures

Can use **vertical distribution** (multi-tiered systems), different logic on different machines. Please note that we use vertical here for a different meaning than before, where it indicated something like protocol layers. Here we talk about dedicated functionality on different machines.

We can also use **horizontal distribution** (replicated client or server operating on different data). This is often geographical distribution, but need not be.

Cmmonly in use is an **overlay network** that hides the physical structure by adding a logical structure.

We distinguish **structured P2P architectures** and **unstructured P2P architectures**

Structured P2P architectures

Most popular technique uses distributed hash tables (DHT)

Randomly creates 128 bit or 160-bit key k using algorithms such as SHA1 for data and nodes. Two or more identical numbers are very unlikely. Most prominent system is **Chord**.

3 P2P architectures (week3)

We look at different types of P2P architectures, the structured and unstructured.

3.1 Structured P2P architectures

Normally, the nodes select unique identifiers and set up a virtual topology. We discuss two such topologies, the ring from Chord and Content-Addressable Networks (CAN) (cf. Distributed Systems; Tanenbaum, van Steen).

3.1.1 Chord

The Chord system arranges items in a ring.

Data item k is assigned to the node with the smallest identifier id, such that $id \geq k$.

For each item k the function $\mathrm{succ}(k) = id$ returns the node an item is assigned to.

To find item k the function LOOKUP(k) returns the address of succ(k) in $O(\log(N))$ time.

An entity with key k falls under the jurisdiction of the node with the smallest identifier $id \geq k$. This is the successor of k and denoted $succ(k)$.

For lookup, each Chord node maintains a finger table. In order to efficiently resolve an address. If FT_p denotes the finger table of node p then

$$FT_p[i] = succ(p + 2^{i-1})$$

In other words the i-th entry points to the first node succeeding p by at least 2^{i-1}.

To look up a key k, node p will then immediately forward the request to node q with index j in $p's$ finger table where

$$1 = FT_p[j] \leq k < FT_p[j+1].$$