Distributed Systems Architectures

Prof. Claudio Bettini

EveryWare Lab, Dipartimento di Informatica
Università degli Studi di Milano

Copyright

System Architecture

- A DS architecture defines
  - the main entities of the system
    - processes/threads, sensor nodes, objects/components, services
  - the pattern of communications
  - how they communicate
  - the role of those entities (and how it possibly evolves)
    - clients, servers, ...
  - how they are mapped to physical infrastructure
    - replication, caching, ...

Overview

- Types of architectures based on roles
  - Centralized architectures
  - Decentralized architectures
  - Hybrid architectures

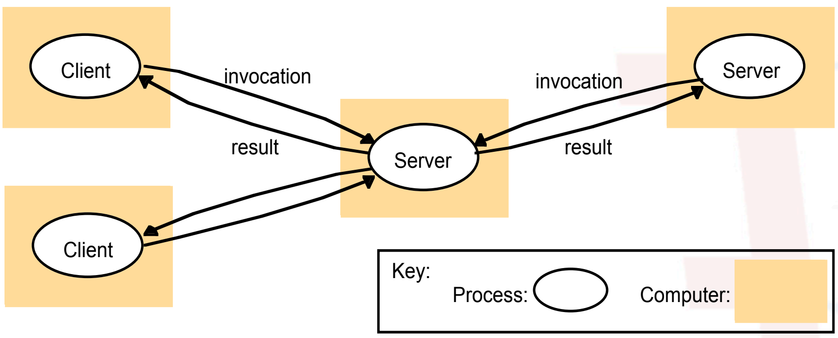
- Introduction to microservices, containers, and DS related Cloud technologies

Centralized Architectures

- Client – Server architectures (request - response pattern)
- Event Bus architectures (publish - subscribe pattern)

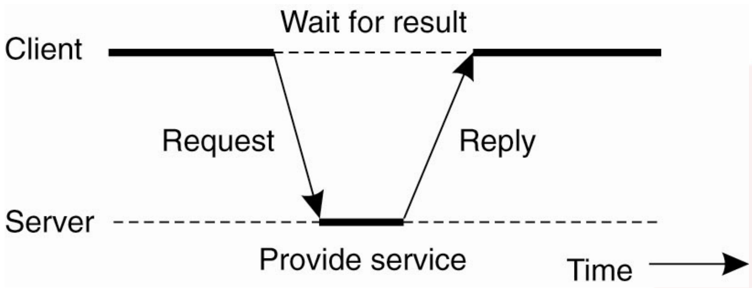
7

The client-server model



8

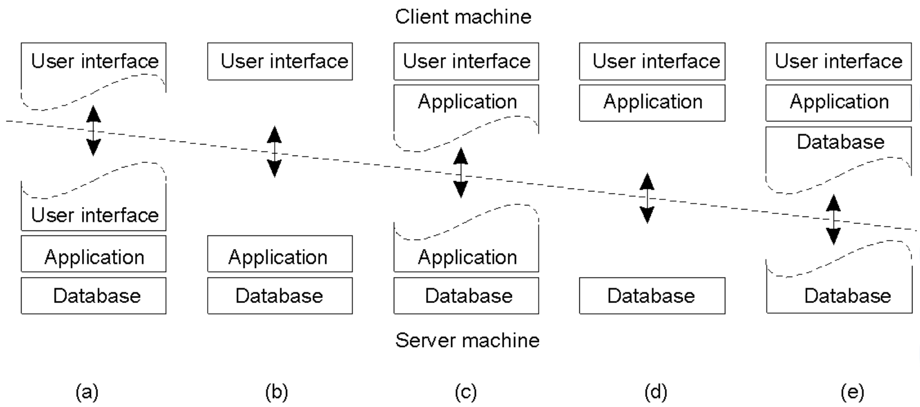
Client - Server



General interaction between a client and a server.

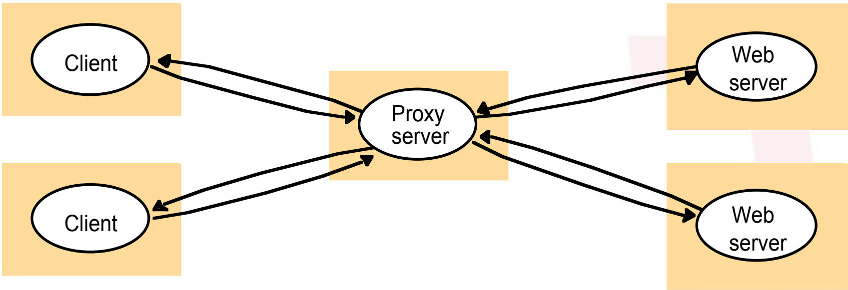
Every Ware Lab

Many variants of client-server models

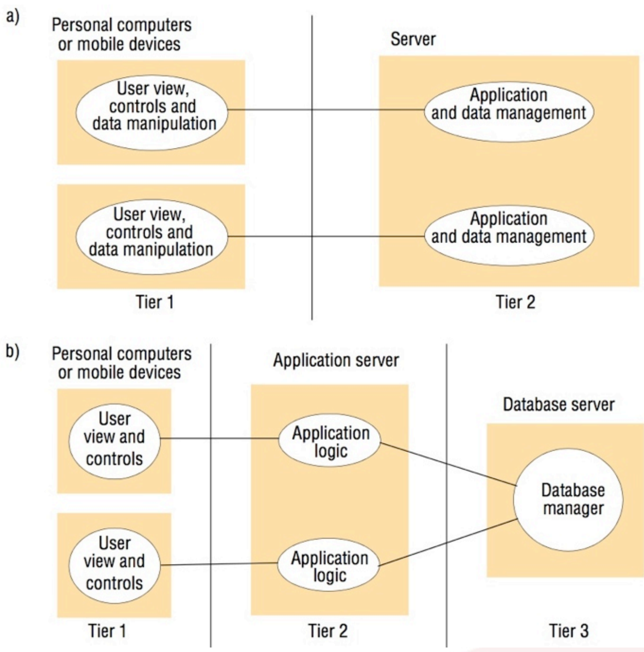


Every Ware Lab

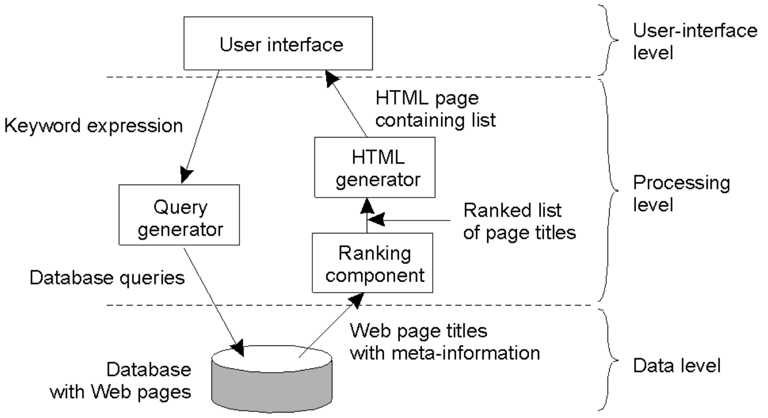
Caching in client-server



Multi-tier client-server



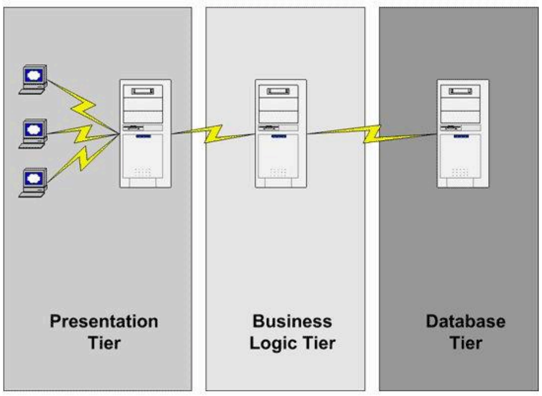
Example



The simplified organization of an Internet search engine into three different layers

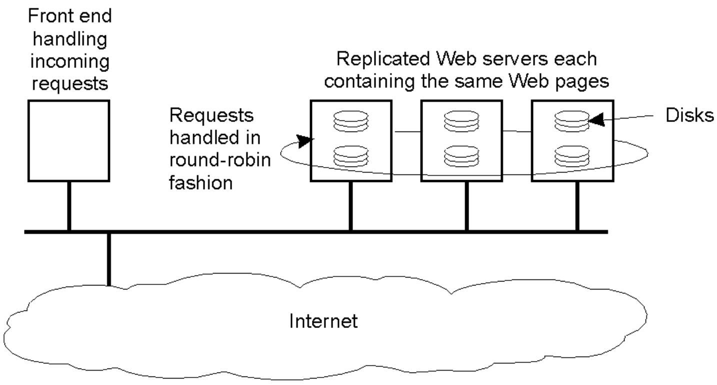
Vertical Distribution

- A different server/node for each functionality



Horizontal Distribution

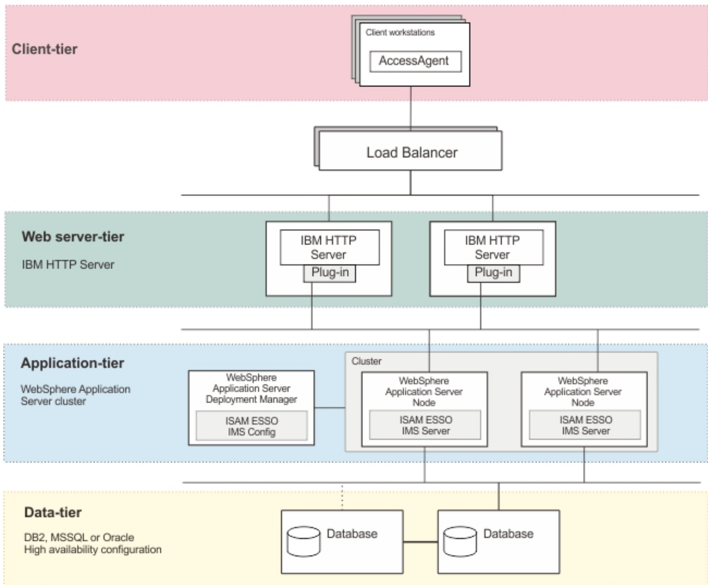
- The same functionality is distributed on multiple servers/nodes with load balancing.



Every Ware Lab

Vertical and horizontal distribution combined

- Each functionality duplicated on a separate group of servers with load balancing



Example from IBM Knowledge Center

Every Ware Lab

Microservice architecture

- Pushing forward the vertical distribution:
  - running each component/functionality of the application logic in a separate process, possibly on a separate machine
- (micro)Services must be independently replaceable and upgradable – packaged with all they need to be deployed
- (micro)Services communicate with lightweight mechanisms (REST API or RPC)
- Focused Scalability: act on the specific (micro)services that need to be scaled
- (micro)Services may be written in different programming languages

Microservices architecture

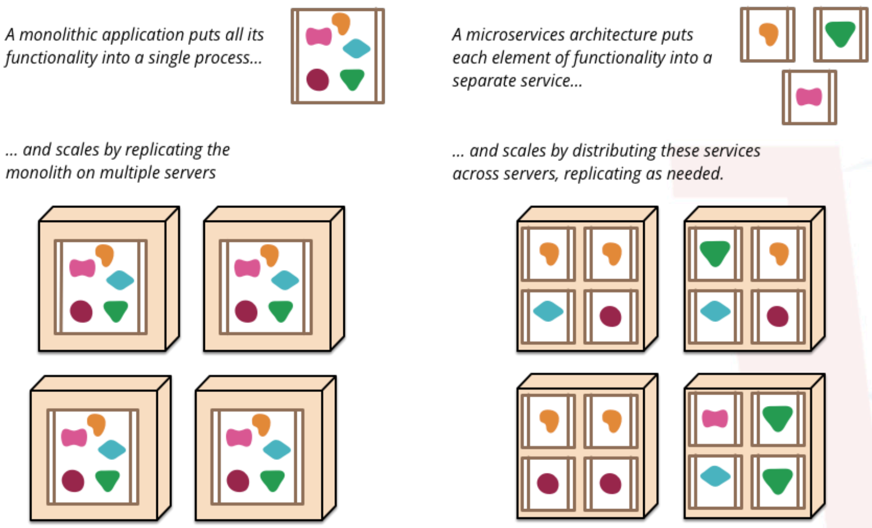


Figure from Martin Fowler

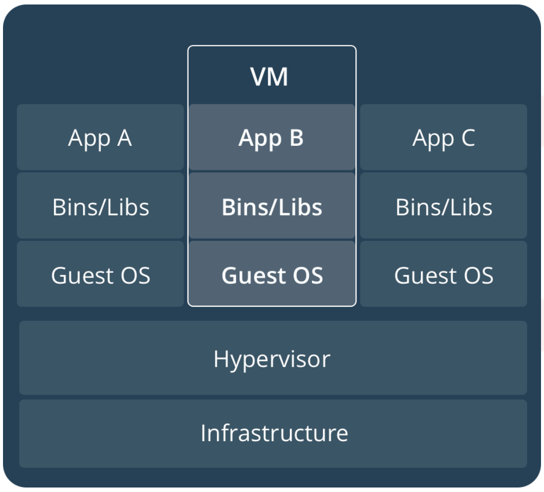
see also his video https://www.youtube.com/watch?v=lrlw-LGIJO4

Containers

- Abstraction at the application layer that packages code and dependencies together
- Virtualization at the process level in user space
- Goals:
  - offering efficient support for the micro-services architecture
  - increasing the portability, enabling easy migration
  - reducing problems at deployment time

Every Ware Lab

Containers vs VMs



Every Ware Lab

Figure from www.docker.com/what-docker#/overview

Containers vs VMs

Docker is an example of a container platform

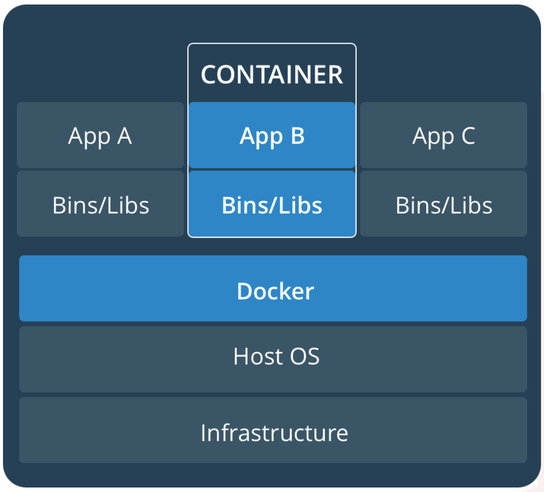
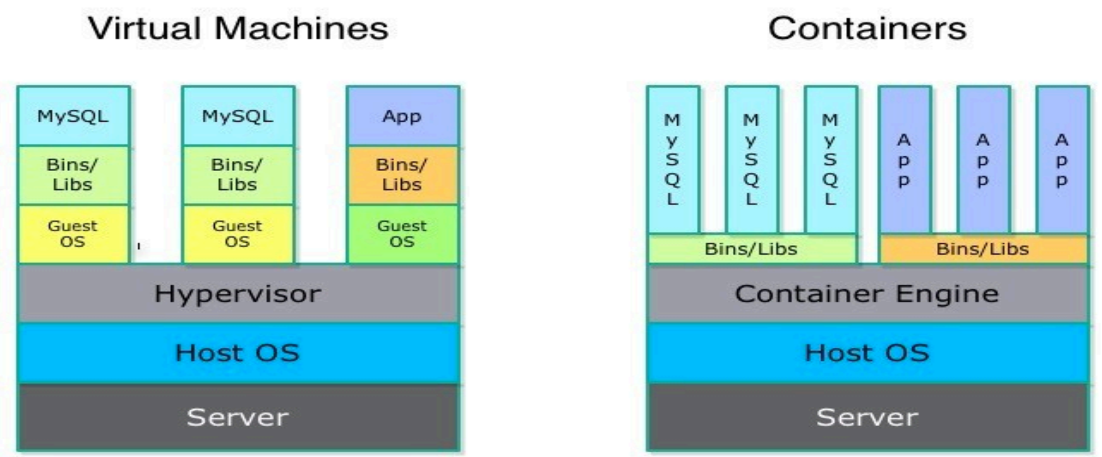


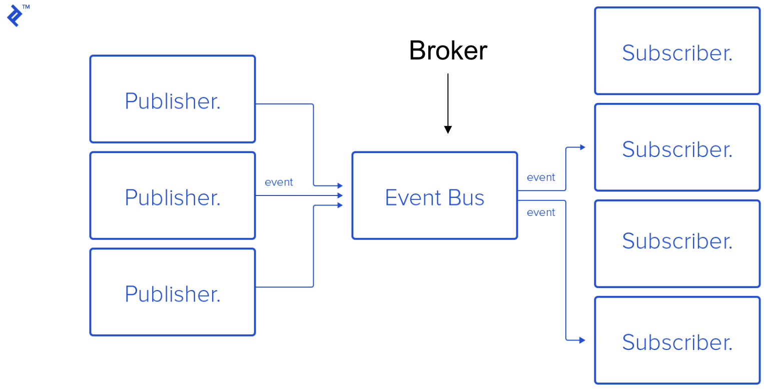
Figure from www.docker.com/what-docker#/overview

Computing with cloud containers



From Lakshmi Narsimhan's blog article, MSys technologies

Event Bus architecture

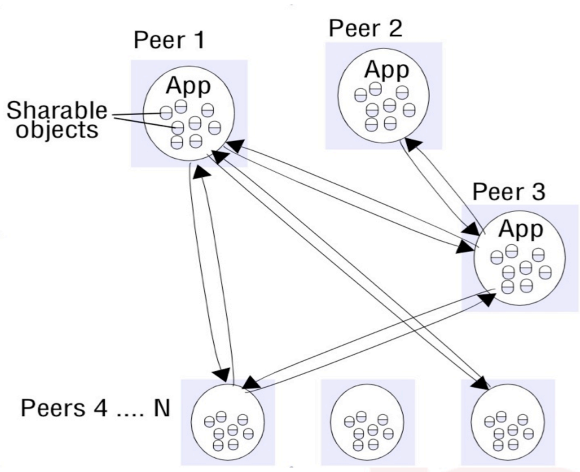


Publish - Subscribe communication pattern

Decentralized Architectures

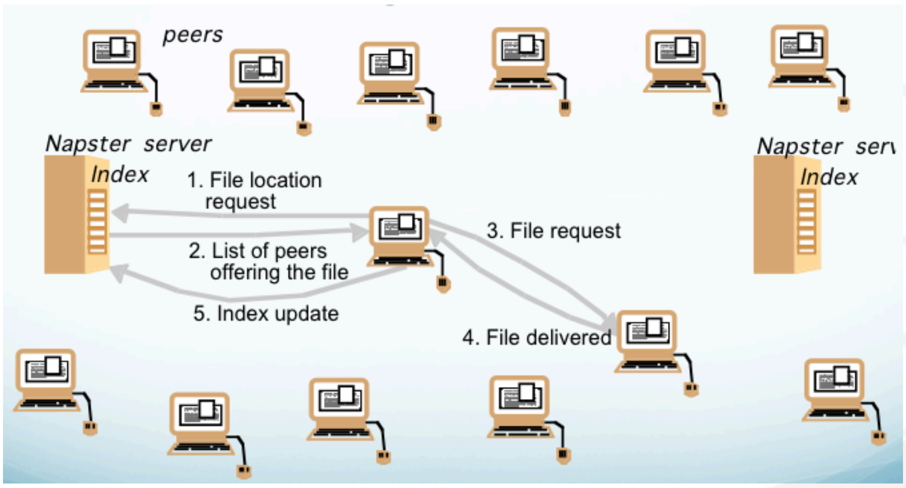
The peer-to-peer model

- All nodes have the same functional capabilities
- Designed so that each user contributes resources
- Operation does not depend on any centrally administered system



Every Ware Lab

Napster: a well-known P2P system



Every Ware Lab

P2P: a key problem

- How to place data objects across many hosts in order to
  - Achieve load balancing while accessing data
  - Ensuring availability avoiding overheads

Three generations of P2P systems

- First file sharing systems
  - Napster
- Protocols are refined to improve scalability, fault tolerance, anonymity
  - Gnutella, FreeNet, bitTorrent
- P2P Middleware
  - Pastry, Tapestry, Chord, CAN, Kademlia

Goal of P2P Middleware

- Enable clients to transparently
  - locate and communicate with any resource
  - add and remove resources
  - add and remove peers

Overlay networks

- Optimization criteria: global scalability, load balancing, locality of interactions, support of encryption and authentication, anonymity

- P2P systems are often organized in an *overlay network*: A network in which nodes are formed by processes and links represent possible communication channels

- An overlay network is a logical network over an existing lower level network (e.g., the Internet)

Routing Overlay

- A distributed algorithm to locate nodes and objects in an overlay network
- Routing requests from clients to hosts holding objects of interest at the application level instead of the network level (IP)
- Routing is based on global identifiers and should direct to one of the replicas of the object
- Routing overlay protocols also deal with insertion/removal of objects and nodes

Two types of overlay networks

- Structured
  - Overlay Network deterministically built (often via hash table) in order to obtain efficient routing towards the node containing the required data
- Unstructured
  - Overlay Network built with randomized algorithms
  - Each peer knows only its neighbors
  - Sometimes they have a hierarchical structure through superpeer nodes

Chord: a P2P system with structured overlay

- Main Goal: quickly mapping a resource to a node
  - e.g., where to find a file (address of the node that stores the file)
- Main Idea: distributed hash table (DHT)
  - limit the number of node addresses that each node knows
  - make the the mapping in time logarithmic in the number of nodes

Every Ware Lab

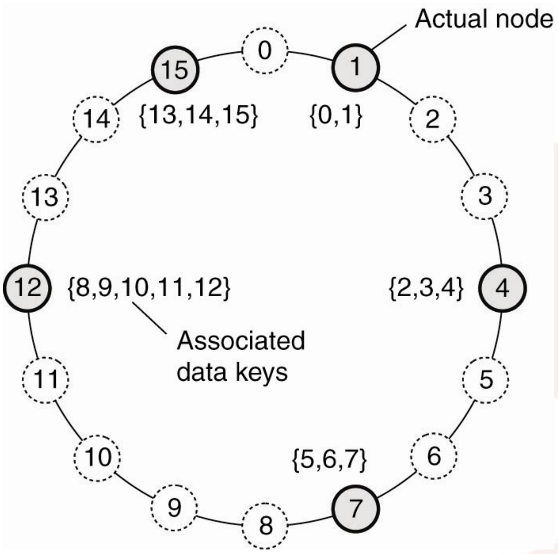
Chord: how

- A fixed address space
  - addresses up to 160 bits
- Each node gets as *id* an address in the space
  - e.g., by hashing its IP&port
- Each data item gets a *key* (address) in the same space
  - e.g., a file gets a key by hashing its file name
- A data item with key *k* is managed by the first node with *id* >= k, called *succ(k)*
- Chord provides LOOKUP(k) to efficiently find the address

Every Ware Lab

Chord: example

- Assignment of data to nodes in Chord



Chord: Key resolution

- Finger (hash) table of node with *id = p*:
  - If *m* is the number of bits in the address space each table has *m* entries
  - FT(i) is the i-th entry of the table and stores the id and address (IP,port) of succ(*p* + 2^{i-1})
    - operations are MOD 2^m
    - In the next slide: m=5. if p=18, the 4-th element is succ(18+2^3)=succ(26)=28

NOTE: the topology of the overlay is no more a ring since each node knows the address of m other nodes.

Chord: Key resolution

Computing LOOKUP(k):

Suppose the search starts at node p and FT() is its finger table.

- If k=p, the data is on p, otherwise the search is forwarded to a node q
- If FT(j) <= *k* < FT(j+1), then q=FT(j), otherwise
- if p < k and (k < FT(1) or there is no node between k and FT(1)) then q=FT(1)
- otherwise, if k >= FT(m), then q=FT(m)

Note: all arithmetics is MOD 2^m

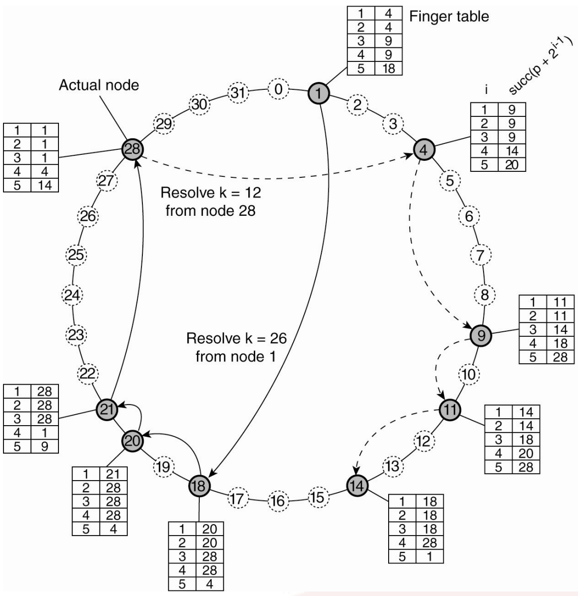
Note: it is always safe (but slower) to forward to FT(1), i.e., the next node in the ring.

Every Ware Lab

Distributed Hash Tables

General Mechanism
enabling O(logN) search

Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.



Every Ware Lab

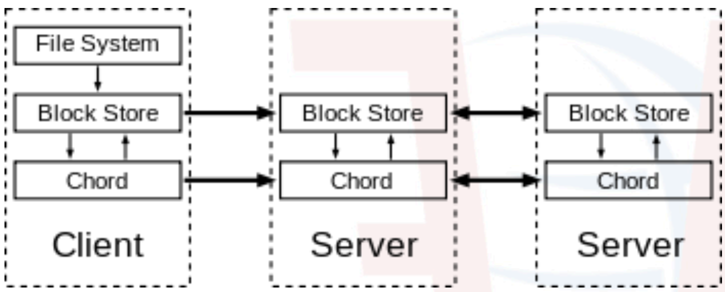
Chord: node insertion and deletion

- Chord must update the routing information when a node joins or leaves the network
- Addresses of successor/predecessor nodes must be updated, resources must be moved, and finger tables must be updated.
- A join or leave requires several messages in the network O(log^2 N)

Every Ware Lab

Example of Chord application

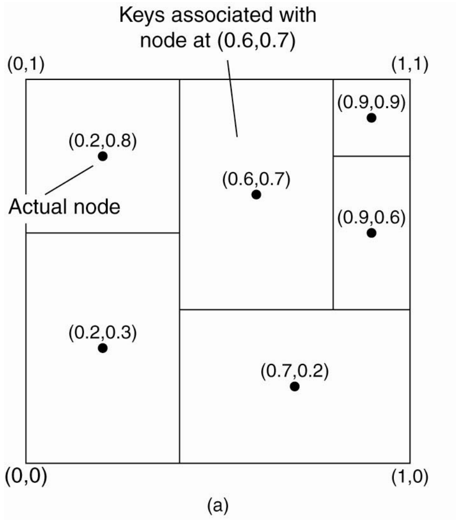
- Highest level on client provides FS interface
- FS maps operations on low level block operations
- Chord used to balance blocks distribution and find the server where block is stored



Every Ware Lab

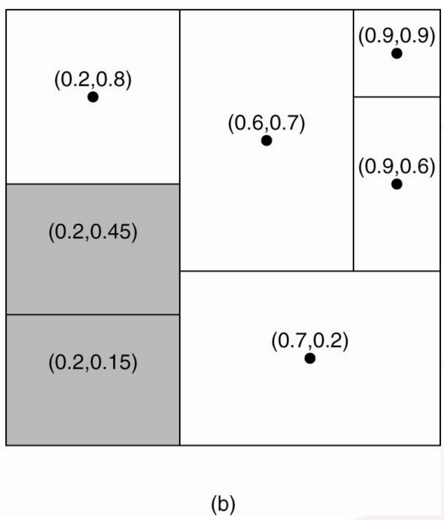
Structured overlays: CAN

- Assignment of data to nodes in CAN.
- CAN uses a bi-dimensional address space
- Each data key is a *point*
- Each node manages all the point in its *region*



CAN: adding a node

- Region split upon insertion of a node in CAN



Limitations of structured overlays

- Maintenance of complex overlay structures can be difficult and costly to achieve (specially in highly dynamic environments)
- Need for self organizing nodes, naturally resilient to failures

Unstructured overlays

- Routing is based on randomized algorithms
  - Each node stores a list of neighbour nodes randomly built (its “view” of the overlay)
  - Resources are also often randomly assigned to nodes
  - Search for a resource starts from a node and propagates according to local views
  - Search is often limited to a number of hops or timeout. Resource replication helps improving search success rate

Unstructured overlays: peer sampling

- Local knowledge of the overlay is updated based on the peer sampling service:
    - nodes periodically exchange their random views and update their local views thereby creating a new random sample.
    - These random views define an approximately random overlay network.

Structured vs Unstructured Overlays in P2P

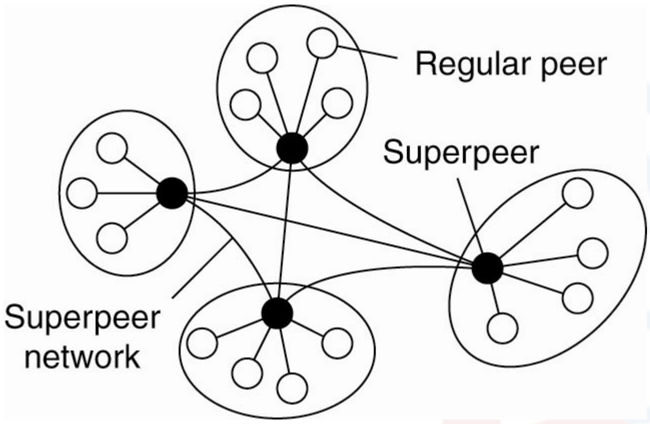
	Structured peer-to-peer	*Unstructured peer-to-peer*
Advantages	Guaranteed to locate objects (assuming they exist) and can offer time and complexity bounds on this operation; relatively low message overhead.	Self-organizing and naturally resilient to node failure.
Disadvantages	Need to maintain often complex overlay structures, which can be difficult and costly to achieve, especially in highly dynamic environments.	Probabilistic and hence cannot offer absolute guarantees on locating objects; prone to excessive messaging overhead which can affect scalability.

Superpeers

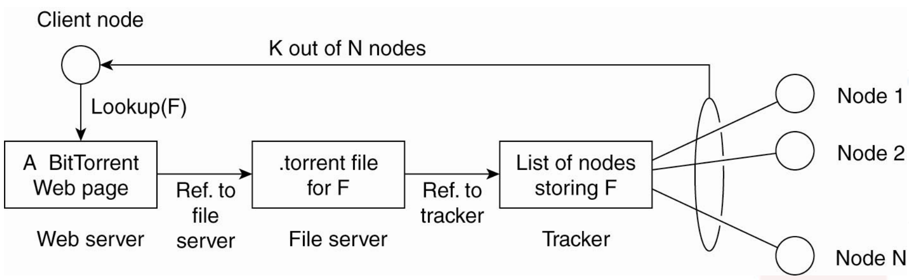
A hierarchical organization of nodes into a superpeer network.

Possible Goals:

- keeping an index of data for a subnetwork
- optimization and load balancing based on local knowledge



Hybrid architectures in Collaborative Distributed Systems



Bootstrapping using a client-server protocol: the example of BitTorrent

[adapted with permission from Pouwelse et al. (2004)].

More on www.bittorrent.org

DS related Cloud services

DS related Cloud platform services

- A set of services built on distributed systems
  - Computing & Communicating
  - Storage & DBMS
  - Identity & Security
  - Management
- They offer several types of transparency
- Major players
  - Amazon AWS
  - Google Cloud platform
  - Microsoft Azure

Every Ware Lab

Service examples

- Storage
  - Google Distributed File System (GFS, Colossus, …)
  - Distributed RDBMS/NoSQL DBMS (Spanner)
- Communication
  - Google PUB/SUB
    - Message-oriented middleware on cloud
    - Supports many-to-many, asynchronous messaging
- Computation and Data Analysis
  - Google DataFlow
    - Stream data processing

See details at https://cloud.google.com

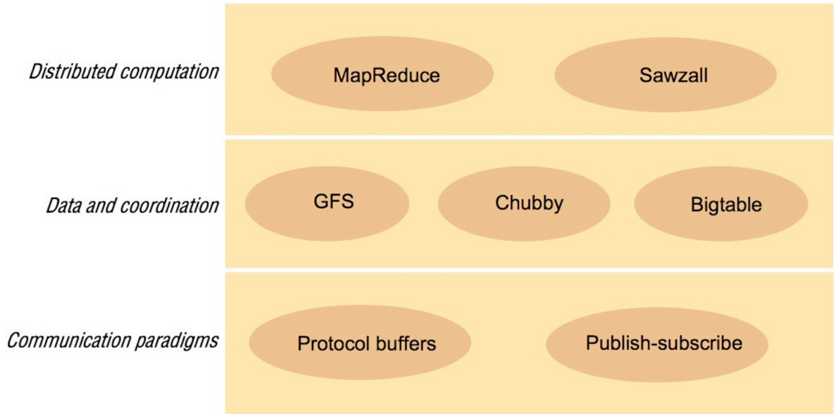
Every Ware Lab

Extensibility and portability in cloud platforms

- Currently a problem for complex solutions running on a specific cloud platform (AWS, Google, Azure, …)
- Cloud containers and microservices favor portability
- Tools for orchestration and distribution over multiple machines of containerized applications help

Every Ware Lab

Examples of Google DS technologies



Every Ware Lab

The Google File System (GFS)

Motivations:

- Storage of very large files
- Use of large clusters of “commodity" computers (high failure rate)
- Files are mostly read or appended to
- High data throughput desired

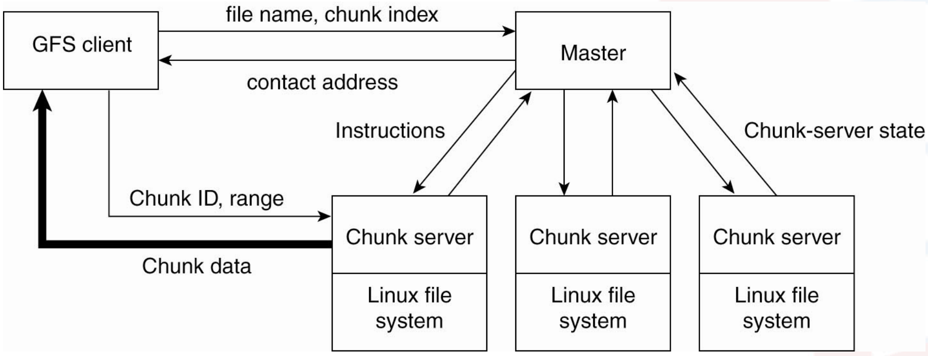
Main Idea:

- Files are divided into chunks with unique labels and distributed over chunkservers
- A master node stores the metadata needed to map chunk labels to chunkservers

Every Ware Lab

GFS Architecture

- The organization of a Google cluster of servers.



https://research.google.com/archive/gfs.html