Distributed Systems - Principles & Paradigms

Table of Contents

1. Introduction

Definition

A distributed system is a collection of independent computers that appear to its users as a single coherent system.

Goals

Connecting users & resources

+ cheaper
+ share costly resources
+ easy to exchange & collaborate information
=> groupware
-/+ security

Transparency

Ability of a computer system to present itself to users & applications as if it were only a single computer system.

Aspects:

- Access: Data representation.
- Location: => Logical names.
- Migration: Resource may move to another location.
- Relocation: Resource may be moved to another location while in use.
- Replication: Several copies of a resource.
- Concurrency: Resource may be shared by several competitive users => locking.
- Failure: Hide failure & recovery.
- Persistence: Hide whether a resource is in memory or on disk.
- => performance!

Openness

- Standard rules govern format, contents, meaning of messages sent & received.
- Interfaces (IDL).
- Complete specifications: Everything that is necessary to make an implementation has been specified.
- Neutral specifications.
- Completeness & neutrality important for interoperability & portability.
- Flexible (easy to configure the system) & extensible => system ist organized as a collection of relatively small & easy replaceable or adaptable components.

Scalability

Dimensions:

- Size: Add more users & resources.
- Geographical: Users & resources may lie far apart. Mostly synchronous communication. WAN: Communication ist unreliable, point-to-point.
- Administrative: Easy to manage. Conflicting policies.
- => loss of performance

Problems: Centralized services, centralized data, centralized algorithms.

Techniques:

- Communication latencies: Asynchronous communication.
- Distribution: Splitting a component into smaller parts. E.g. DNS.
- Replication: E.g. caching => consistency problems => synchronisation mechanisms.

Software concepts - Middleware

Will not manage an individual node => local OS. Between distributed applications & network OS services.

Models: Distributed file systems, RPC, distributed objects, distributed documents.

Services: Communication facilities, naming, persistence, distributed transactions, security.

Middleware & openness:

- - incompleteness
- - incompatible communication protocols
- - entities referred in different ways

Client-server model

Request-reply: Client requests a service from a server by sending a request & waiting for the result, server sends back a reply.

- Connectionless: + efficient
- Connection oriented: - performance

Application layering: User interface level, processing level (application), data level.

Architectures:

- 2-tier: Client & server; different organisations with: UI, application, DB.
- 3-tier: Server act as a client (UI -> application server -> DB server).
- Vertical distribution: Placing logically different components on different machines.
- Horizontal distribution: Client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set.
- Peer-to-peer distribution: No server, clients interact with each other (through same application).

2. Communication

Layered protocols

OSI (open system interconnection reference) model: Allow open systems to communicate. Rules, i.e. Protocols. Connection-oriented or connection-less.

7 Layers: Physical -> data link -> network -> transport -> session -> presentation -> application.

Each layer provides an interface to the one above it. Headers, trailer from data link layer.

Lower-level protocols:

- Physical layer: Transmitting 0 & 1.
- Data link layer: Group bits into units (frames, with sequence numbers). See that each frame is received correctly. Append a checksum (= trailer).
- Network layer: Routing (how to choose the best path from sender to receiver), e.g. IP (internet protocol).

Transport protocols:

- Transport layer: Provide a reliable connection (= connection-oriented). Break messages into pieces (sequence number). E.g. TCP (transmission control protocol), UDP (universal datagram protocol).
  UDP: Connectionless.
  TCP: Connection-oriented (= reliable). Overhead because of managing connections. 3-way-handshake.

Higher-level protocols:

- => In practice, only the application layer is used.
- Session protocols: Dialog control. Synchronisation facilities.
- Presentation protocols: Meaning of the bits. Records with fields; notify a receiver that a message contains a particular record in a certain format.
- Application protocols: E.g. FTP (file transfer protocol), HTTP.
- Middleware protocols: Logically lives in the application layer.

Remote Procedure Call

A program is allowed to call procedures on other machines.

RPC: A client calls a remote procedure, the server (where the called procedure is located) calls the local procedure & returns the result in a reply. The client blocks until this reply is returned.

Parameter passing: Packing parameters into a message is called parameter marshalling.

Asynchronous RPC: I.e. if there is no result to return. The client continues immediately after issueing the RPC request. The server immediately sends a reply back to the client the moment the RPC request is received, after that it calls the requested procedure. The reply acts as an acknowledgement to the client. The client will continue without further blocking as soon as it has received the servers ACK.

Problems:

- Calling & called procedures run on different machines, they execute in different address spaces.
- Parameters & results to be passed.
- Machines can crash.

Distributed objects

For remote object invocation. An object encapsulates data (state) & operations (methods). Methods are available through an interface.

Separation between interfaces & objects allows to place an interface at one machine, while the object itself resides on another machine. The state although is not distributed, it resides at a single machine.

Organization: Client invokes a method on a proxy on the client machine, marshalled invocation is passed from the proxy across the network to the server skeleton. The skeleton unmarshals the data & invokes the same method on the object (through interface).

Compile-time objects:

- Developer can mostly stay unaware of the distribution of objects.
- Compiling the class definition results in code that allows to instantiate an object.
- Dependency on a particular programming language (i.e. Java).

Runtime objects:

- How objects are written is left open.
- Use of an object adapter, a wrapper around the implementation, is necessary.
- Do not depend on a particular programming language.

Persistent objects: Continues to exist even if it is currently not contained in the address space of a server process.

Transient object: Exists only as long as the server that manages the object.

Message-oriented communication

Persistent communication: A message, that has been submitted for transmission is stored by the communication system as long as it takes to deliver it to the receiver.

Transient communication: A message is stored by the communication system only as long as the sending & receiving application are executing.

Asynchronous communication: A sender continues immediately after it has submitted its message for transmission.

Synchronous communication: The sender is blocked until its message is stored in a local buffer at the receiving host, or actually delivered to the receiver. The strongest form is when the sender is blocked until the receiver has processed the message.

Combinations:

- Persistent asynchronous communication.
- Persistent synchronous communication.
- Transient asynchronous communication: E.g. UDP, one-way-RPC.
- Transient synchronous communication: Receipt-based, delivery-based, response-based e.g. RPC, RMI.

Message-oriented transient communication

Berkeley sockets: Transport-level sockets. Interface.

Socket: A communication endpoint to which an application can write data that are to be sent out over the underlying network & from which incoming data can be read.
Insufficient: Wrong level of abstraction by supporting only simple send & receive primitives, sockets were not considered suitable for the proprietary protocols developed for high-speed interconnection networks.

Message-passing interface (MPI): For parallel applications. For transient communication. Assumes communication takes place within a known group of processes.

Message-oriented persistent communication

Message queuing systems, message-oriented middleware (MOM).

Extensive support for persistent asynchronous communication, message transfers are allowed to take minutes (instead of seconds or milliseconds like Berkeley sockets & MPI).

Message queuing: Applications communicate by inserting messages in specific queues. A sender is generally given only the guarantees that its message will eventually be inserted in the recipient's queue. Permits loosely-coupled communication.

Queuing layer: Look-up queue address in database.

Queue managers: Some operate as routers/relays.

Message brokers: Acts as an application-level gateway. Convert incoming messages to a format that can be understood by the destination application, uses a database of conversion rules.

3. Processes

Threads

OS has a process table, where information about a process is stored. A process is a program in execution.

Thread is similar to a process, it can also be seen as the execution of a (part of a) program. No attempt is made to achieve a high degree of concurrency transparency. Protecting data against inappropriate access by threads within a single process is left entirely to application developers. A thread context consists of nothing more than the CPU context & some information for thread management.

Threads can provide a convenient way of allowing a blocking system calls without blocking the entire process in which the thread is running. Threads improves performance through parallelism, blocking system calls (makes programming easier), retain idea of sequential processes.

Multithreaded clients: I.e. web browsers & HTML documents (including images).

Multithreaded servers: Dispatcher thread & worker threads.

Servers

A server is a process implementing a specific service on behalf of a collection of clients. It waits for an incoming request from a client & ensures that the request is taken care of, after this it waits for the next incoming request.

Contacting a server: Clients send requests to an endpoint (port) at the machine where the server is running. Servers listen to a specific endpoint.

Design issues:

- Iterative server: Server itself handles the request.
- Concurrent server: Does not handle the request itself (passes it to a separate thread or another process), e.g. multithreaded server.
- Superserver: Listen to a number of endpoints, when a request comes in, fork a process to take further care of the request.
- Out-of-band data: Data that is to be processed by the server before any other data from the client.
- Stateless server: Does not keep information on the state of its clients. Can change its own state without having to inform any client. No recovery. E.g. a web server.
- Stateful server: Maintains information on its clients. Better performance than stateless servers. Recover strategies necessary. E.g. a file server that allows a client to keep a local copy of a file, even for update.
- Cookie: Small piece of data containing client-specific information that is of interest to the server.

Object servers:

- Server tailored to support distributed objects. Specific services are implemented by the

objects that reside in the server.

- Invoking objects (activation policies): Server needs to know which code to execute. Support just one policy or different.
- Object adapter (object wrapper): To group objects per policy, an object adapter has one or more objects under its control. There can be different object adapters in the same server at the same time. Invocation request is first dispatched to the appropriate object adapter.

Code migration

Situations, where passing programs simplifies the design of a distributed system.

Reasons: Traditionally code migration took place in the form of process migration in which an entire process was moved from one machine to another. That is costly. Basic idea: Overall system performance can be improved if processes are moved from heavily-loaded to lightly-loaded machines. In many modern distributed systems optimizing computing capacity is less an issue than, for example, trying to minimize communication. Performance improvement through code migration is often based on qualitative reasons.

Models:

- Weak mobility: Minimum for code migration, transfer only the code segment, along with perhaps some initialization data. A transferred program is always started from its initial state. Its simple, requires only that the target machine can execute that code. E.g. Java applets.
- Strong mobility: Execution & code segment can be transferred. A running process can be stopped, subsequently moved to another machine, and then resume execution where it left off. More powerful than weak mobility, but much harder to implement.
- Sender-initiated: Migration is initiated at the machine where the code currently resides or is being executed. Typically it is done when uploading programs to a compute server. Would require a server to know all its clients.
- Receiver-initiated: Migration is initiated by the target machine. Simpler to implement than sender-initiated. E.g. Java applets.

Process-to-resource bindings:

- Binding by identifier: Process refers to a resource by its identifier, the referenced resource is important, e.g. URL.
- Binding by value: When a program relies on standard libraries, these libraries should be locally available, the content is important.
- Binding by type: References to local devices, e.g. monitors, printers.

Resource-to-machine bindings:

- Unattached resources: Can be easily moved between different machines, e.g. (data) files associated only with the program that is to be migrated.
- Fastened resources: High costs, e.g. local databases.
- Fixed resources: Bound to a specific machine or environment, cannot be moved, e.g. local devices.

Migration stack: Copy of the program stack, but in a machine-independent way, updated when a subroutine is called. When migrating code, the migration stack & global program-specific data are transferred to the destination.

4. Naming

The implementation of a naming system is itself often distributed across multiple machines.

Naming entities

Naming entities with a fixed location.

Name: String of bits or characters, refer to an entity. Addresses & identifiers are two important types of names, each used for very different purpose. Always organized in a name space.

Access point: Special kind of entity. An entity may easily change it. It may be reassigned to a different entity.

Address: Name of an access point. Treated as a special type of name.

Location independent: Name for an entity that is independent from its addresses. Easier & more flexible to use.

Identifier: Name with three properties:

- Identifier refers to at most one entity.
- Each entity is referred to by at most one identifier.
- Identifier always refers to the same entity (never reused).

Human-friendly names: Tailored to be used by humans. Represented as a character string.

Name space: Names are organized into it. Can be represented as a labeled, directed graph with two types of nodes (leaf & directory node). Can be organized strictly hierarchical.

Leaf node: Named entity. Has the property that it has no outgoing edges. Stores information on the entity it is representing (e.g. its address, state).

Directory node: Has a number of outgoing edges, each labeled with a name. Each node is considered as yet another entity & has an associated identifier.

Directory table: Table in directory node. Stores outgoing edges as a pair (edge label, node identifier).

Root (node): Node with only outgoing & no incoming edges. For simplicity naming systems have only one root.

Path name: Sequence of labels corresponding to the edges in that path.

Absolute path name: When the first node is the root of the naming graph.

Relative path name: First node is not the root of the naming graph.

Global name: Denotes the same entity, no matter where that name is used in a system.

Local name: Interpretation depends on where that name is being used. A relative name whose directory in which it is contained is (implicitly) known.

Name resolution: Name lookup. Given a path name, it is possible to look up any information stored in the node referred to by that name. Can take place only if we know

how & where to start.

Closure mechanisms: Knowing how & where to start name resolution. Partly implicit. May be very different.

Aliases: Another name for the same entity, e.g. environment variable.

Hard links: A node can be referred to by two different path names (both paths points to the same node).

Symbolic links: Representing an entity by a leaf node, but instead of storing the address or state of that entity, the node stores an absolute path name.

Mounted file system: Letting a directory node store the identifier of a directory node from a different (foreign) name space.

Mount point: Directory node storing the node identifier.

Mounting point: Directory node in the foreign name space.

Principle of mounting: Can be generalized to other name spaces. A directory node that acts as a mount point need to store all the necessary information for identifying & accessing the mounting point in the foreign name space.

Information needed to mount a foreign name space: Name of an access protocol, name of the server, name of the mounting point in the foreign name space.

Implementation of a name space: Three logical layers:

- Global layer: Highest-level nodes. Root node & other directory nodes logically close to the root. Characterized by their stability. Should be high available. Can be cached & replicated (for availability & performance), e.g. organizations.
- Administrational layer: Directory nodes that together are managed within a single organization. Represent groups of entities that belong to the same organization. Relatively stable. Should be high available. Can be cached & replicated (for availability & performance).
- Managerial layer: Nodes that may typically change regularly, e.g. hosts in local network, shared files.

Zones: Nonoverlapping parts the name space is divided into in DNS. Part of the name space that is implemented by a separate name server.

Name resolver: Responsible for ensuring that the name resolution process is carried out. Each client has access to a name resolver.

Iterative name resolution: A name resolver hands over the complete name to the root name server, the root name server resolves the path name as far as it can. Then the name resolver passes the remaining path name to the next name server and so on. Name servers in the global layer of a name space support only iterative name resolution.

Recursive name resolution: Instead of returning each intermediate result back to the clients name resolver, a name server passes the result to the next name server it finds. Puts a higher performance demand on each name server. Caching results is more effective (whole URL can be cached), communication costs may be reduced.

Domain name service (DNS)

Used for looking up host addresses & mail servers.

DNS name space is hierarchically organized as a rooted tree, a label is a case-insensitive string of alphanumeric characters (max. 63), max. path-length is 255. A subtree is a domain, a path name to the root is a domain name (absolute or relative). Content of a node is a resource record (different record types: for zone, host, domain, ...). A hosts primary name is called a canonical name (CNAME), an alias for a host is possible. Maintains an inverse mapping of IP addresses to host names (PTR).

Record types:

Type of record	***Associated entity***	***Description/example***
SOA	Zone	Information on the represented zone. cs.vu.nl SOA star (1999121502,7200,3600,2419200,86400)
A	Host	IP address of the host this node represents. star.cs.vu.nl A 192.31.231.66
MX	Domain	Mail server to handle mail addresses to this node. star.cs.vu.nl MX 1 star.cs.vu.nl
SRV	Domain	A server handling a specific service.
NS	Zone	Name server that implements the represented zone. cs.vu.nl NS star.cs.vu.nl
CNAME	Node	Symbolic link with the primary name of the represented node. www.cs.vu.nl CNAME soling.cs.vu.nl
PTR	Host	Canonical name of a host. vucs-das.cs.vu.nl PTR 0.26.37.130.in-addr.arpa
HINFO	Host	Information on the host. star.cs.vu.nl HINFO Sun Unix
TXT	Any kind	Any entity-specific information considered useful. cs.vu.nl TXT "Vrije Universiteit - Math. & Comp. Sc."

DNS name space can be devided into a global & an administrational layer. Managerial layer is not part of DNS (formed by local file systems). Each zone is implemented by a name server.

Zone transfer: Secondary name server request the primary server to transfer its content. Database is a collection of files (resource records).

X.500

Consists of a number of records (directory entries), a directory entry is comparable to a resource record in DNS. Each record is made up of a collection of attribute-value pairs (single-valued & multiple-valued attributes).

Directory entries:

Attribute	*Abbr.*	*Example*
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Math. & Comp. Sc.
CommonName	CN	Main server
Mail_Server	--	130.37.24.6, 192.31.231.42
FTP_Server	--	130.37.24.11
WWW_Server	--	130.37.24.11
Host_Name	--	star
Host_Address	--	192.31.231.42

Collection of all directory entries is called a directory information base (DIB) where each record is uniquely named. Unique name is a sequence of naming attributes, each attribute is called a relative distinguished name (RDN).

/C=NL/O=Vrije Universiteit/OU=Math. & Comp. Sc. is analogous to nl.vu.sc

Hierarchy of the collection of directory entries, called directory information tree (DIT). It forms the naming graph on an X.500 directory service in which each node represents a directory entry or an X.500 record. DIT is partitioned & distributed across several servers, known as directory service agents (DSA). Each part of a partitioned DIT corresponds to a zone in DNS. Clients are represented by directory user agents (DUA), similar to a name resolver. Facilities for searching through a DIB (search for a directory entry given a set of criteria for attributes), searching is an expensive operation.

Lightweight directory access protocol (LDAP): A simplified protocol. Is an application-level protocol that is implemented on top of TCP. Parameters can be passed as strings (instead of using encodings required by OSI).

Locating mobile entities

Mobile entities change name-to-address mappings regularly. Naming services are not well suited for this.

Moving an entity: Record the address or the name (symbolic link) of the new machine in the DNS database. The "old" name cannot be used for any other entity, it is expected not to change for the lifetime. This is a direct mapping between human-friendly names & addresses of entities, which leads to problems for mobile entities. Better is to separate naming from locating entities by introducing identifiers (e.g. entity IDs), so each entity has exactly one identifier, the identifier is never assigned to a different entity, it is optimized for machine processing.

Locating an entity is handled by means of a separate location service, getting the name is done by a naming service => two-level mapping.

Simple solutions

Address resolution protocol (ARP): Only in local-area networks. To find the data-link address of a machine when given only the IP address. Broadcast a message containing the IP address to each machine & each machine is requested to check whether it listens to the requested IP address. If so, they send a reply packet containing, for example, the Ethernet address.

Broadcast becomes inefficient when the network grows (too many hosts may be interrupted by requests they cannot answer). Its possible to switch to multicasting, by which only a restricted group of hosts receives the request. Or use multicast to locate entities in point-to-point networks. Internet supports network-level multicasting, multicast groups are identified by multicast addresses.

Forwarding pointers: Only in local-area networks, simple. When an entity moves from A to B, it leaves behind a reference to its new location at B. Look up the entity by following the chain of forwarding pointers. SSP chains is an approach for distributed objects, where a forwarding pointer is implemented as a proxy-skeleton pair. Transparent to a client.

Home-based approaches

Broadcasting or multicasting difficult to implement efficiently in large-scale networks (long chain of forwarding pointers). Approach: Home location which keeps track of the current location of an entity. Mobile IP: All communication is directed to the mobile hosts home agent. Whenever the mobile host moves to another network it requests a temporary address it can use for communication. This care-of address is registered at the home agent. Home agent receives a packet for the mobile host, it looks up the hosts location. If on the local network, the packet is forwarded. Otherwise its tunneled to the hosts current location (sent to the care-of address) & the sender of the packet is informed of the hosts current location. Fixed home location can be a problem when the location is not available (contacting the entity is then impossible).

Hierarchical approaches

Network is divided into a collection of domains. Single top-level domain, each domain can be subdivided into smaller domains. A lowest-level domain is called a leaf domain & corresponds to a local-area network in a computer network or a cell in a mobile telephone network. Each entity currently located in a domain is represented by a location record. An entity may have multiple addresses. If a lookup request does not find the entity in a leaf domain (i.e. leaf domain has no location record for the entity), the node forwards the request to its parent & so on. When the request reaches a directory node that stores a location record for the entity the request gets down to the childs of the node.

Pointer caching: In a subdomain store a pointer to another subdomain where the entity currently resides.

Root node is required to store a location record for each entity & process requests for each entity. Handling requests may become a bottleneck. Solution: Partition the root node & other high-level directory nodes into subnodes.

Removing unreferenced entities

Distributed garbage collectors, remote objects (proxy-skeleton pairs), skeleton is doing the

administration necessary for garbage collection (together with proxy). Objects with no remote reference should be removed from the system. A cycle of objects referring to each other should also be removed. Graph: Root set (objects which need not be referenced, i.e. User), reachable entities from the root set, unreachable entities from the root set, entities forming an unreachable cycle.

Reference counting: Skeleton maintains reference counter. Its necessary to detect duplicate messages (incrementing & decrementing of counter). Problem with copying a remote reference to another process.

Weighted reference counting: Only decrementing operations can take place, only a limited number of references can be created. Each object has a fixed, total weight. When the object is created, the total weight is stored in its associated skeleton, along with a partial weight, which is initialized to the total weight. When a new remote reference is created, half of the partial weight stored in the skeleton is assigned to the new proxy. When a remote reference is duplicated, half of the partial weight of the proxy is assigned to the copied proxy. When a reference is destroyed, a decrement message is sent to the objects skeleton, which subtracts the partial weight of the removed reference from the total weight. When the total weight is zero, the object can be removed. If the partial weight of the skeleton or the remote references is zero, no more references can be created or duplicated.

Reference listing: The skeleton keep track of the proxies that have a reference to it. The skeleton maintains a list of all proxies that point to it. Adding & deleting references is idempotent, that is they can be repeated without affecting the end result. Does not require communication to be reliable, duplicate messages can be sent (an ACK is needed). I.e. Java RMI.

Tracing-based garbage collection: Method by which all entities in a distributed system are traced. Checking which entities can be reached from the root set & subsequently removing all others.

Mark-and-sweep collectors: Two phases. Mark phase: Tracing entities by following chains of references originating from entities in the root set. Each entity that can be reached is marked. Sweep phase: Examining memory to locate entities that have not been marked & remove them. Requires the reachability graph to remain the same during both phases (execution of program needs to be temporarily stopped).

5. Synchronisation

Clock synchronisation

- Cristian's algorithm: Centralized. One machine has a WWV receiver (time server), try to stay the other machines synchronized with it. Periodically each machine sends a message to the time server asking for the current time. Time server responds as fast as possible. Time must never run backward, so if a clients time is "more" than the servers, its a problem. A solution is to slow down (or advancing) the interrupt until the times are synchronized. The response from the time server takes time. So the message propagation time is estimated with $(T_1 - T_0)/2$. If an approximate response time is known, messages in which $T_1 - T_0$ exceeds some threshold are discarded.
- Berkeley algorithm: Centralized. No machine has a WWV receiver. Time server (time daemon) is active, polling every machine from time to time to ask what time it is there. Based on the answers, it computes an average time & tells the other machines to advance or slow down their clocks.
- Averaging algorithms: Decentralized. Dividing time into fixed-length resynchronization intervals. At the beginning of each interval, every machine broadcasts its current time. After a machine broadcasts its time, it collects all other broadcasts that arrive during some interval. Then it computes a new time from them. By averaging all values or by discarding m highest & m lowest & averaging the rest.
- Multiple external time sources: More than one machine has a WWV receiver.

Logical clocks

Logical clocks: The internal consistency of the clocks matters, not if they are particularly close to the real time.

- Lamport timestamps: Synchronize logical clocks. Transitive relation "happens-before" (a happens before b, a & b are events). Processes need to agree on a time value for an event. Each message carries the sending time according to the sender's clock. When a message arrives & the receiver's clock shows a lower value, the receiver fast forwards its clock to be one more than the sending time. Between two events the clock must tick at least once. Addition: No two events occur at exactly the same time. The number of the process is attached to the time, separated by a decimal point (e.g. event in process 1 - 40.1, event in process 2 - 40.2). Thus there is a total ordering of all events in the system.
- Vector timestamps: All events in a distributed system are totally ordered with the property that if event a happened before event b, then a will also be positioned in that ordering before b. Do not capture causality. If b is an reaction to a, it should follow a. If the two are independent, their order of delivery should not matter. If VT(a) < VT(b) for some event a & b, then event a is known to causally precede event b.

Election algorithms

Electing a coordinator among processes, i.e. one process to act as coordinator.

- Bully algorithm: When any process notices that the coordinator is no longer responding to requests, it initiates an election by sending an election message to all processes with

higher numbers. The process with the highest process number is the new coordinator. The new coordinator then sends all processes a message to announce its victory. If a process that was previously down comes back up, it holds an election.

- Ring algorithm: Processes are physically or logically ordered (each knows its successor). When any process notices that the coordinator is no longer responding to requests, it sends an election message (with own process number) to its successor. If the successor is down, the sender skips it & goes to the next member along the ring. Each sender adds its own process number to the message. When the message gets back to the process that started it, that process sends a message to tell all processes who the new coordinator is (highest process number) & who the members of the new ring are.

6. Consistency & replication

Introduction

Reasons for replication:

- Reliability: If one replica crashes we can switch to one of the other replicas. Better protection against corrupted data (failing write operation, at least n copies are the same & that is the right data).
- Performance: When the distributed sytem needs to scale in numbers (many clients need the data, replicate server & divide the work) & geographical area (time to access the data decreases).

Problem: Consistency. Modifications have to be carried out on all copies.

Object replication:

- Protect object from simultaneous access: Object itself can handle concurrent invocations (i.e. Java methods can be "synchronized"). Or object is unprotected but the server in which the object resides makes the concurrency control (object adapter with a single thread per object).
- Replication-aware objects: Object is aware of its replicas, it is responsible for ensuring that its replicas stay consistent. Possible to adopt object-specific replication strategies.
- Distributed system responsible: Distributed system ensures that concurrent invocations are passed to the various replicas in the correct order. Simpler for application developers.

Scaling techniques: Replication & caching for performance. Keep objects close to processes using them => reduction of access time. Keeping copies up to date may require more network bandwidth. Low access-to-update ratio => many updated versions will never be accessed & the network communication for those was useless. Update operation can be a single atomic operation but when a large number of replicas in a large-scale network is involved the operation may be slow. Global synchronization takes a lot of communication time (wide-area network). Solution: Loosen the consistency constraints. Copies may not always be the same everywhere but performance is better. Look for access & update patterns.

Distribution protocols

Types of copies:

- Permanent replicas: Data store, initial set of replicas. Small number of such replicas. E.g. Web site: replicated accross a limited number of servers or a few number of mirrored web sites (geographically spread).
- Server-initiated replicas: Copies of data store that exist to enhance performance, created at the initiative of the data store. Keep read-only copies close to clients. Push caches: Sudden bursts come from an unexpected location far from the server, so a temporary replica in this region is installed. Problem: Where & when replicas should be created or deleted. One solution (web hosting services, static content): Server keeps track of access counts per file & where access requests come from. When the number of requests drops below a deletion threshold, that file can be removed. Replication

threshold for replicating the file.

- Client-initiated replicas: (Client) caches. Improve access time to data. Local storage facility that is used to temporarily store a copy of the data a client has just requested. Managing the cache is left entirely to the client. Its not keeping the cached data consistent. Kept in cache for a limited time. Cache hit: When requested data can be fetched from the local cache. To improve number of cache hits, caches can be shared between clients. I.e. a shared cache for files are useless, for websites useful.

Update propagation

Update operations are generally initiated at a client & forwarded to one of the copies. From there the update should be propagated to the other copies (while ensuring consistency).

- Propagate only a notification of an update: Invalidation protocols. Other copies are informed that an update has taken place & the data they contain are no longer valid. When an operation on an invalidated copy is requested, that copy need to be updated first. Use little network bandwidth, good when read-to-write ratio is small.
- Transfer data from one copy to another: Useful when read-to-write ratio is high. Propagating modified data or just logs (what changed). Possible to pack multiple modifications into a single message (saving communication overhead).
- Propagate the update operation to other copies: Active replication. Updates can often be propagated at minimal bandwidth costs. More processing power may be required by each replica.
- Push-based approach: Server-based protocols. Updates are propagated to other replicas without those replicas even asking for the updates. Often used between permanent & server-initiated replicas. Applied when replicas need to maintain a high degree of consistency (replicas need to be kept identical), when read-to-update ratio is high. Server needs to keep track of all client caches (overhead at the server).
- Pull-based approach: Client-based protocols. Server or client requests another server to send any updates it has at that moment. Used by client caches. Applied when read-to-update ratio is low. Response time increases in the case of a cache miss.
- Lease: Promise by the server that it will push updates to the client for a specified time. When a lease expires, the client is forced to poll the server for updates & pull the modified data. Or request a new lease. Mechanism for dynamically switching between push-based & pull-based strategy. Age-based lease, renewal-frequency based lease, state-space overhead.
- Unicast communication: Server sends one message to each of the n other servers. Efficiently combined with pull-based approach.
- Multicast communication: Underlying network takes care of sending a message efficiently to multiple receivers. Cheaper. Efficiently combined with a push-based approach.

Epidemic protocols: Need only be guaranteed that all replicas are eventually identical. Concers only about sending as few messages as possible to all replicas. Updates are often aggregated into a single message & exchanged between two servers. Based on the theory of epidemics, which studies the spreading of infectious diseases (updates). Try to

"infect" all replicas with updates as fast as possible.

7. Fault tolerance

Partial failure: One component of a distributed system fails. This may affect the proper operation of other components, while yet other components are totally unaffected. Construct a distributed system that it can automatically recover from partial failures without seriously affecting the overall performance.

Introduction

Fault tolerance is related to dependable systems.

Requirements:

- Availability: System is ready to be used immediately, is operating correctly at any given moment. System will most likely be working at a given instant in time.
- Reliability: System can run continously without failure. Will most likely continue to work without interruption during a relatively long period of time.
- Safety: When a system temporarily fails to operate correctly, nothing catastrophic happens.
- Maintainability: How easy a failed system can be repaired.
- Security

Faults & failures:

- Failure: System fails when it cannot meet its promises.
- Error: Part of a systems state that may lead to a failure.
- Fault: Cause of an error.
- Fault tolerance: System can provide its services even in the presence of faults.
- Transient faults: Occur once & then disappear. If operation is repeated, the fault goes away.
- Intermittent faults: Occur, then vanishes of its own accord, then reappears & so on. Difficult to diagnose.
- Permanent faults: Continue to exist until the faulty component is repaired.

Failure models:

- Crash failure: Server halts, but was working correctly until it stopped.
- Ommission failure: Server fails to respond to a request. Receive ommission: Server fails to receive incoming messages. Will generally not affect the current state of the server. Send ommission: Server fails to send messages. Affects the current state.
- Timing failure: Servers response lies outside a specified time interval.
- Response failure: Servers response is incorrect. Value failure: Value of the response is wrong. State transition failure: Server reacts unexpectedly to an incoming request.
- Arbitrary (byzantine) failure: Server may produce arbitrary responses at arbitrary times.

- Fail-stop server: Server will stop producing output in such a way that its halting can be detected by other processes.
- Fail-silent system: Server don't announce that it is going to stop. Processes conclude that a server has halted, but perhaps it is just unexpectedly slow.
- Fail-safe system: Server is producing random output, but this output can be recognized by other processes as plain junk.

Failure masking by redundancy:

- Information redundancy: Extra bits are added to allow recovery from garbled bits.
- Time redundancy: An action is performed & then, if needed it is performed again. E.g. using transactions.
- Physical redundancy: Extra equipment or processes are added to make it possible for the system as a whole to tolerate the loss or malfunctioning of some components. E.g. add extra processes to a system. Triple modular redundancy (TMR).

Process resilience

Key approach to tolerating a faulty process it to organize several identical processes into a group. When a message is sent to the group, all members of the group receive it. Process groups may be dynamic, a process can be a member of several groups at the same time (analogous to social organizations).

- Flat groups: All the processes are equal, all decisions are made collectively. Symmetrical, no single point of failure. Decision making is complicated (voting).
- Hierarchical groups: One process is the coordinator, all others are workers. Coordinator chooses the worker. Single point of failure (coordinator), decision making easy.
- Group server: Maintains a complete data base of all groups. Creating & deleting groups, allowing processes to join & leave groups. Easy to implement. Single point of failure.
- Manage group membership in a distributed way: Use multicast messages.

Agreement in faulty systems:

- Needed when: Electing a coordinator, deciding whether or not to commit a transaction, dividing up tasks among workers, synchronization.
- Two-army problem: 2 armies, 1 messenger. How does one army know that the messenger got through?
- Byzantine generals problem: 2 armies, n generals are head of one army, m generals are traitors (faulty). Communication is done pairwise & is reliable. How do the loyal generals reach agreement?

Reliable client-server communication

Point-to-point communication: Making use of a reliable transport protocol, such as TCP (uses ACK & retransmissions). Crash failure are often not masked => automatically set up a new connection.

Client-server communication: RPC. Hide communication by making remote procedure calls look just like local ones. When errors occur, the difference between local & remote calls are not easy to mask. Five different classes of failures:

- Client cannot locate server: Raise an exception or signal. But not every language has exceptions/signals. Transparency is destroyed.
- Request message from client to server is lost: Operating system or client stub starts a timer when sending the request. If the timer expires before a reply or ACK comes back, the message is sent again. If the request was not lost, the server should be able to detect retransmissions.
- Server crashes after receiving a request:
  - At least once semantics: Send messages as long as a reply has been received. Guarantees that the RPC has been carried out at least one time, but possibly more.
  - At most once semantics: Give up immediately & report back failure. Guarantees RPC has been carried out at most one time, possibly none.
  - Guarantee nothing.
  - Exactly once semantics: Server has 2 possibilities: Send a completion message before requesting an operation or after the operation is completed. Client has 4 possibilities for reissuing a request: Never, always, only if it did not receive an ACK, only if it has received an ACK. => 8 combinations. There is no combination of client & server strategy that will work correctly under all possible event sequences.
- Reply message from server to client is lost: Rely on a timer. If no reply is coming in the timer interval the request is sent again. Idempotent request: Has no side effects, can be repeated without problems. Try to structure requests so they are idempotent, but thats not everywhere possible. Safeguard so that the server can distinguish the initial transmission from retransmissions.
- Client crashes after sending request: Orphan: Unwanted computation, wastes CPU, lock files or something else. Its also possible that a reply comes back after the reboot. None of the following methods are desirable.
  - Extermination: Client make logs (on disk). After reboot, the log is checked & the orphan is killed off. Expense of writing a disk record for every RPC. Orphans may do another RPCs, called grandorphans that are difficult or impossible to locate.
  - Reincarnation: Divide time into sequentially numbered epochs. After reboot client declares the start of a new epoch. All remote computations on behalf of that client are then killed. Or the reply has an obsolete epoch number.
  - Gentle reincarnation: On an epoch broadcast each machine checks to see if it has a remote computation & tries to locate its owner. If the owner cannot be found the computation is killed.

- Expiration: Each RPC is given a standard amount of time to do the job. If it cannot finish, it must ask for another quantum. If the client waits this time before rebooting, all orphans are surely gone.

Reliable group communication

Every message should be delivered to each current group member. Sometimes its necessary that all group members receive messages in the same order.

Simple solution: Sender assigns a sequence number to each message it multicasts. Each message is stored locally in a history buffer at the sender. The message is kept in the buffer until each receiver has ACK it. Cannot support large numbers of receivers.

Scalability in reliable multicasting:

- Nonhierarchical feedback control: Reduce the number of feedback messages => feedback suppression, scalable reliable multicasting (SRM) protocol. Receivers never ACK the successful delivery but reports when a message is missing. How message loss is detected is left to the application. When a receiver notices that it missed a message, it multicasts its feedback (after some random delay). This allows other group members to suppress its own feedback. Retransmissions are always multicasted to the entire group. Group members are forced to receive messages that are useless to them (because they already got the first transmission).
- Hierarchical feedback control: For very large groups. The group of receivers is partitioned into subgroups, which are organized into a tree. Within each subgroup any reliable multicast scheme can be used. Each subgroup appoints a local coordinator, which is responsible for handling retransmissions. The local coordinator have its own history buffer. If the coordinator itself missed a message, it asks the coordinator of the parent subgroup to retransmit it.

Atomic multicast

Atomic multicast problem: Often needed is the guarantee that a message is delivered to either all processes or none. Additionally it is generally required that all messages are delivered in the same order to all processes.

A multicast message is uniquely associated with a list of processes to which it should be delivered. This delivery list corresponds to a group view. Each process on that list has the same view, they agree on the group members. View change: A process wants to join or leave the group. It acts as a barrier across which no multicast can pass. All multicasts that are in transit while a view change takes place are completed before the view change comes into effect.

Virtually synchronous: Guarantees that a message multicast to a group view is delivered to each nonfaulty process. Ordering of messages:

- Unordered: No guarantees are given concerning the order in which received messages are delivered to different processes.
- FIFO-ordered: Communication layer is forced to deliver incoming messages from the same process in the same order as they have been sent. No constraint regarding the delivery of messages sent by different processes.

- Causally-ordered: Potential causality between different messages is preserved.
- Totally-ordered: Additional requirement that when messages are delivered, they are delivered in the same order to all group members.
- => 6 forms of reliable multicasting

Atomic multicast: Virtually synchronous reliable multicast offering totally-ordered delivery of messages.

8. Security

Introduction

Security is related to dependability. A dependable computer system is one that we justifiably trust to deliver its services. Dependability includes availability, reliability, safety & maintainability. Confidentiality & integrity should also be taken into account.

- Confidentiality: Property of a computer system whereby its information is disclosed only to authorized parties. Ensures that messages cannot be intercepted & read by eavesdroppers.
- Integrity: Alterations to a systems assets can be made only in an authorized way. Ensures that messages are protected against modification.

Security threats

- Interception: Situation that an unauthorized party has gained access to a service or data. E.g. Communication has been overheard by someone else, data are illegally copied.
- Interruption: Situation in which services or data become unavailable, unusable, destroyed. E.g. DoS attacks.
- Modification: Unauthorized changing of data or tampering with a service so that it no longer adheres to its original specifications. E.g. Intercepting & changing transmitted data, changing a program so that it secretly logs the activities of its users.
- Fabrication: Situation in which additional data or activity are generated that would normally not exist. E.g. breaking into a system by replaying previously sent messages.

Security policy: Describes precisely which actions the entities (users, services, data, machines, ...) in a system are allowed to take & which ones are prohibited.

Security mechanisms

- Encryption: Transforms data into something an attacker cannot understand. Allows to check whether data have been modified. Provides means to implement confidentiality, support for integrity checks.
- Authentication: Verify the claimed identity of a user, client, server, ... Users are authentificated by means of passwords.
- Authorization: Check whether a client is authorized to perform the requested action.
- Auditing: Trace which clients accessed what & which way. Audit logs can be useful for the analysis of a security breach.

Design issues

- Focus of control: Protection of an application. Different approaches: Concentrate on the protection of the data (i.e. database systems), ensure data integrity. Concentrate on the protection against invalid operations. By specifying exactly which operations may be invoked & by whom. Concentrate on the protection against unauthorized users. Taking measures by which only specified people have access to the application. Control is

focused on defining roles that users have.

- Layering of security mechanisms: Computer networks are often organized into layers following some reference model.
- Simplicity: If a system designer can use a few, simple mechanisms that are easily understood & trusted to work, the better it is.

Cryptography

- Encryption - decryption: Sender encrypts the message, receiver must decrypt the message. Cryptographic methods parameterized by keys.
- Plaintext: Original form of the message.
- Ciphertext: Encrypted form of the message.
- Intruder: May intercept the message (eavesdropping). If the transmitted message has been encrypted in such a way that it cannot be easily decrypted without having the proper key, interception is useless. Modifying ciphertext that has been properly encrypted is difficult because the intruder will first have to decrypt the message before it can meaningfully modify it. & he will also have to properly encrypt it again. He cannot easily insert encrypted messages if he doesn't know the key.
- Symmetric cryptosystems: Secret-key systems, shared-key systems. Same key is used to encrypt & decrypt a message. Sender & receiver are required to share the same key, this shared key must be kept secret. $P = D_K(E_K(P))$.
- Asymmetrc cryptosystems: Public-key systems. Keys for encryption & decryption are different, but together form a unique pair. One of the keys is kept private, the other is made public. $P = D_{KD}(E_{KE}(P))$.
- Hash functions: A hash function takes a message of arbitrary length as input & produces a bit string having a fixed length as output. One-way functions: It is computationally infeasible to find the input that corresponds to a known output. Weak collision resistance: Given an input & its associated output it is computationally infeasible to find another, different input such that the output is the same. Strong collision resistance: Given only the hash function it is computationally infeasible to find any two different input values such that the output is the same.
- Encryption function: It should be computationally infeasible: To find the key when given the plaintext & associated ciphertext, to find another key such that the ciphertexts are the same when given a plaintext & a key.

Data encryption standard (DES)

For symmetric cryptosystems. Operate on 64-bit blocks of data. A block is transformed into an encrypted block of output in 16 rounds, where each round uses a different 48-bit key for encryption. Each of this 16 keys is derived from a 56-bit master key. Before the 16 rounds the input block is first subject to an initial permutation of which the inverse is later applied to the encrypted output & so leading to the final output.

Input -> initial permutation -> 16 rounds -> final permutation -> output.

Algorithm is difficult to break using analytical methods. Using a brute-force attack by searching for a key has become easy. Using DES three times in a special encrypt-decrypt-

encrypt mode with different keys appears to be safe for the moment. There are variants with larger keys & larger data blocks. The algorithm has been designed to be fast (even on smart cards).

Public-key cryptosystems: RSA

Fact that no methods are known to efficiently find the prime factors of large numbers. Private & public keys are constructed from very large prime numbers. Breaking RSA is finding those two numbers, but this has shown to be computationally infeasible.

Generating private & public key:

- Choose two very large prime numbers (p, q).
- Compute n = p*q, z = (p-1)*(q-1).
- Choose a number d that is relatively prime to z.
- Compute the number e such that e*d=1 mod z.

One of the numbers d or e can be used for decryption, the other for encryption. Only one of these two is made public. Message m is divided into fixed-length blocks m_i where each block, interpreted as binary number, should lie in the interval $0 \leq m_i < n$. Encryption: Sender calculates for each block m_i the value $c_i=m_i^e$ (mod n), which is sent to the receiver. Decryption takes place by computing $m_i=c_i^d$ (mod n).

Computationally more complex than DES.

Hash functions: MD5

Computing a 128-bit, fixed length message digest from an arbitrary length binary input string. Input string is first padded to a total length of 448 bits, after which the length of the original bit string is added as a 64-bit integer. The input is converted to a series of 512-bit blocks.

Secure channels

Authentication

Authentication & message integrity cannot do without each other. Bob may know for sure that Alice is the sender but if there is no guarantee that the message has not been modified during transmission its useless. Alice sets up a channel, once it has been set up, Alice knows for sure that she is talking to Bob. To ensure integrity of the data messages it is common practice to use secret-key cryptography by means of session keys. Session key: Shared key that is used to encrypt messages for integrity & confidentiality. It is used only for as long as the channel exists.

Authentication based on a shared secret key: Challenge-response protocol. Setting up a communication channel between A & B:

- A sends his identity to B.
- B sends a challenge R_B to a A (e.g. a random number).
- A encrypt the challenge with his secret key & sends this to B.
- B decrypts the message with his shared key to see if it contains the challenge. If so, he

knows that he is talking to A.

- A sends another challenge R_A to B.
- B encrypt the challenge with his secret key & sends this to A.
- A decrypts it with his shared key. If it contains the challenge, A knows that he is talking to B.
- => 5 messages, A & B know they are surely talking to each other.

"Optimization" using only 3 messages, but can be defeated by a reflection attack. Because the two parties use the same challenge in two different runs of the protocol.

Authentication using a key distribution center: Using a shared secret key may lead to scalability problems. Each host is required to share a secret key with each other host. Use a centralized approach by means of a key distribution center (KDC). The center shares a secret key with each of the hosts, but no pair of hosts is required to have a secret key. Setting up a communication between A & B: A tells the KDC that he want to talk to B, the KDC hands out a key to both A & B that they use for communication (=> 3 messages). The message from the KDC to A is encrypted with the secret key A shares with the KDC. Same for B.

Authentication using public-key cryptography: Setting up a communication between A & B:

- A sends a challenge R_A to B encrypted with B's public key.
- B decrypts the message & returns the challenge to A, his own challenge R_B, a generated session key for their communication, all encrypted with A's public key.
- A decrypts the message using his private key & responds by encrypting the challenge R_B with the session key.
- => 3 messages

Message integrity & confidentiality

Confidentiality is easily established by encrypting a message before sending it.

Digital signatures: A pays an amount of money for an article sold by B. The deal was done through e-mail. A needs to be assured that B won't change the amount into something higher. B needs to be assured that A cannot deny ever having sent the message. => A digitally signs the message in such a way that his signature is uniquely tied to its content. The unique association between a message & its signature prevents modifications. If the signature can be verified to be genuine, A cannot later deny that he signed the message.

RSA: A encrypts the message first with A's private key then with B's public key. B decrypts it using B's private key & A's public key. Along with the encrypted message the message itself is sent encrypted with B's public key. If the messages are the same, everything is ok. Signing the hole message is costly.

Message digest: Is cheaper. A compute a message digest & encrypt it with his private key. The encrypted digest & the message in plaintext is sent to B. B decrypts the digest with A's public key & separately calculate the message digest. If both digests match, everything is ok.

Session keys: Instead of a session key its possible to use the same keys for confidentiality

as those that are used for setting up the secure channel. But session keys are better.

- When a key is often used, it becomes easier to reveal it. Its safer to use the authentication keys as little as possible.
- Exchanging keys using some relatively time-expensive out-of-band mechanisms such as regular mail or telephone should be kept to a minimum.
- Its a protection against replay attacks (against replaying an entire session). As protection against replaying individual messages timestamps or sequence numbers as part of the message content are included.
- When the key is compromised, an intruder may be able to decrypt old messages (from old conversations). With session keys at worst a single session is affected.
- If A don't trust B so much he don't want to give so much information away (data encrypted with long-lasting keys). Such keys will be reserved for highly-confidential messages exchanged with parties A really trusts.
- Replacing authentication keys is relatively expensive.

Access control

- Access rights: A request can be carried out only if the client has sufficient access rights for that request.
- Access control: Verifying access rights.
- Authorization: Granting access rights.
- Subject: Issues a request to access an object. Processes acting on behalf of users, objects that need the service of other objects.
- Object: Encapsulating its own state, implementing operations on that state. operations are made available through interfaces.
- Reference monitor: Program enforcing protection. Records which subject may do what. Decides whether a subject is allowed to have an operation carried out. Monitor is called each time an object is invoked.

Access control matrix: Each subject is represented by a row, each object by a column. An entry M[s,o] lists which operations subject s can request to be carried out on object o. With thousands of users the access control matrix would be too big & many entries will be empty. A better way are ACLs or lists of capabilities because they ignore empty entries.

Access control list (ACL): Each object have its own ACL where the rights of the subjects are shown. Reference monitor checks if object & subject (request) is listed in ACL.

List of capabilities: Each subject has a list of capabilities for each object. Reference monitor checks if the requested operation is listed in the capabilities.

Protection domains: For reducing ACLs. A protection domain is a set of object-access rights pair. It specifies for a given object exactly which operations are allowed to be carried out. Requests for carrying out an operation are issued within a domain. Uses of protection domains: Group users (possibly hierarchical), certificates (subject carries a certificate listing the groups it belongs to), roles (a user can have different roles).

Grouping objects: Based on the operations they provide.

Firewalls: Special reference monitor. Controls the external access to any part of a distributed system. It disconnects any part of the distributed system from the outside world. It inspects all incoming packets & discard the ones with no access rights.

Packet-filtering gateway: Type of a firewall that operates as a router & makes decisions as to whether or not to pass a network packet based on the source & destination address (contained in the header).

Application-level gateway: Type of a firewall that inspects the content of an incoming or outgoing message. E.g. filtering spam e-mails.

Proxy gateway: Special type of application-level gateway. Works as a front end to a specific kind of application & ensures that only those messages are passed that meet certain criteria. E.g. prevent applets or scripts to be downloaded.

Security management

Key management

Key establishment: Diffie-Hellman key exchange: A & B agree on two large numbers (n, g). Each one of them picks his own secret large number (x, y).

- A sends g^x mod n to B, along with g & n. Can be sent as plaintext (its impossible to compute x from this informations).
- B calculates g^{xy} mod n & sends g^y mod n to A.
- A computes g^{xy} mod n, which is now the shared secret key.
- => 2 messages

If viewed as a public-key cryptosystem x is A's private key, g^x mod n his public.

Key distribution

Symmetric cryptosystem: To get initial keys A & B need to get in touch with each other using some other communication means that the network.

Public-key cryptosystem: Receivers need to be sure that the key is paired to a claimed private key. The public key may be sent plaintext. Public-key distribution takes place by means of public-key certificates. Certificate contains of a public key & a string identifying the associated entity. Public key & identifier have together been signed by a certification authority (with the CA's private key), this signature has been placed on the certificate as well. Using the certificate: A client uses the CA's public key to verify the certificate signature.

Lifetime of certificates

If the private key is compromised the client want to revoke the certificate.

- Certificate revocation lists (CRL): Are published regularly by the CA. When a client checks a certificate, it will have to check the CRL before. A compromised certificate can be falsely used until a new CRL is published.
- Restrict the livetime of a certificate. CRLs are still necessary.

Authorization management

Managing access rights. Resources are spread across several machines. Create an account for each user on each machine. Or create a single account on a central server. Or use capabilities.

Example: Kerberos

Security system that assists clients in setting up a secure channel with a server. Security is based on shared secret keys. Components: Authentication server (AS), ticket granting service (TGS). AS handles login requests from users. Authenticates a user & provides a key that can be used to set up secure channels with servers. TGS handles setting up secure channels. Hands out tickets that are used to convince a server that the client is really who it claims to be.

Example: Electronic payment systems

Payment involves at least two parties, a customer & a merchant. Mostly also a bank is involved.

Digital money: Using a smart card capable of storing or store it on the hard disk. Should not be used twice. Should not be possible to make counterfeit money. Authentication if the money is "real". Digital signature as part of the transaction so that the transaction can't be denied. Payments need to be carried out as (atomic) transactions. So the merchant can't deny he has been paid.

Secure electronic transactions (SET): Standard way for purchasing goods over a network using credit cards. It includes authentication in every step. Dual signature: Constructing a message digest of the order & one of the payment information. Then construct another digest over the two digests. This last digest is signed with A's private key. A sends this message to B. Steps:

- A sends order & payment information to B.
- B sends payment information & request authentication to bank.
- Bank sends back ok & capture message (used later on by B to actually get the payment from the bank; signed & encrypted by the bank) to B.
- B sends ok to A.
- B sends a "pay me" & capture message to bank.
- Bank sends ok to B.
- => 5 messages. For every message to/from the bank a new session key is used.

9. Distributed object-based systems

Common object request broker architecture (CORBA)

Specification of a distributed system, drawn up by a comittee. Global architecture: 4 groups of architectural elements are connected to the object request broker (ORB). Groups are: Application objects, vertical facilities (domain specific), horizontal facilities (general purpose), common object services.

- ORB: Core of any CORBA distributed system. Responsible for enabling communication between objects & their clients while hiding issues related to distribution & heterogeneity.
- Horizontal facilities: General purpose high-level services that are independent of application domains, i.e. user interface.
- Vertical facilities: High-level services that are targeted to a specific application domain, i.e. banking.

Object model: Implementation of an object resides in the address space of a server. Objects & services are specified in the interface definition language (IDL). Underlying any process is the ORB.

Services: Collection (grouping objects), query (querying collections of objects), concurrency control, transaction, event (asynchronous communication), notification, life cycle (creation, deletion, copying, moving of objects), licensing, naming, property (associating attribute-value pairs with objects), trading (publish & find services an object has to offer), persistence, relationship, security (secure channels, authorization, auditing), time (current time).

Object invocation models:

- Synchronous: At-most-once semantics. Caller blocks until a response is returned or an exception is raised.
- One-way: Best effort delivery semantics. Caller continues immediately without waiting for any response from the server.
- Deferred synchronous: At-most-once semantics. Caller continues immediately & can later block until response is delivered.

Object adapter: Calls a method on an object. Responsible for providing a consistent image of what an object is. Wrapper. Portable object adapter (POA): Responsible for making server-side code appear as CORBA objects to clients. Object implementations are partly provided by servants. A servant is that part of an object that implements the methods that clients can invoke.

Naming: A process uses a language-specific implementation of an object reference, mostly pointers. This is a pointer to a local representation of the object. That reference cannot be passed from process A to process B, instead process A have to marshal the pointer into a process-independent representation (provided by the ORB). This reference can be passed to process B which can unmarshal it. The ORB have its own language-independent representation of an object reference. Interoperable object reference (IOR): Language-independent representation of an object reference. Organization of IOR: Repository identifier, profile ID & profile (n times). Profile includes IIOP version, host, port,

object key (POA identifier, object identifier), components. CORBA offers a naming service that allows clients to look up object references using a character-based name. A name is a sequence of name components (id-kind pair).

Distributed component object model (DCOM)

From Microsoft. Intricate system, in which many similar things can be done in many different ways. Primarily offers access transparency. Does not form a complete distributed system.

General organization: COM (core library, object activation, persistent storage), OLE (embedding, document linking, drag & drop) & ActiveX (documents, scripting). DCOM adds the possibility for a process to communicate with components that are placed on another machine.

Object model: Centered around the implementation of interfaces. Only binary interfaces (table with pointers to the implementations of the methods). Microsoft IDL (MIDL) for generating binary interfaces. Each interface has a globally unique interface identifier (IID).

Services: DCOM, COM, COM+, ActiveX. Collection, concurrency, transaction, event, notification, externalization, life cycle, licensing, namin, persistence, security.

Object invocation models:

- Synchronous: At-most-once semantics.
- Callback interfaces: Supported by connectable objects.
- Cancel a pending synchronous call: Cancel object.
- Asynchronous: Require client & object up & running.

Naming: Low-level naming. DCOM objects make use of the Windows Active Directory. Moniker: Persistent reference, can be stored on disk. Contains all information to reconstruct the referenced object. Different kinds of monikers, e.g. file, URL, class moniker.

Active directory: Worldwide directory service. Strongly coupled to the organization of a distributed system based on Windows 2000. Such a system is partitioned into domains. Each domain has one or more domain controllers (local directory services) which keep track of the users & resources in a domain. Each domain has an associated DNS name. Every domain controller is organized as an LDAP directory server & is registered as LDAP services in DNS. Active directory allows different domains to be grouped into trees (domain tree).

10.Distributed document-based systems

The WWW

Huge distributed system consisting of millions of clients & servers for accessing linked documents. Each document is stored as a file.

Uniform resource locator (URL): To refer to a document. Specifies where the document is located (DNS name of the server & file name) & the application-level protocol for transferring the document.

A client interacts with web servers through a browser (special application). A browser is responsible for properly displaying a document. Accepts input from a user (type in URL).

Document model: All information is represented by means of documents. A document can contain references to other documents (hyperlinks). Most documents are expressed in hypertext markup language (HTML). HTML provides keywords to structure a document into different sections. Parts of a document can be expressed in the form of a script (scripting language, i.e. JavaScript). When a document is parsed, it is internally stored as a rooted tree (parse tree). Each node represents an element of that document. The representation of the parse tree has been standardized => document object model (DOM), also referred as dynamic HTML. DOM provides a standard programming interface to parsed web documents. Scripts can be used to inspect & modify the document that they are part of. Alternative language that matches the DOM: Extensible markup language (XML). XML is used to structure a document, it contains no keywords to format a document. It can be used to define arbitrary structures. Formatting language for XML: Extensible style language (XSL). Can be used to describe the layout of an XML document.

Document types

Multipurpose internet mail extension (MIME) type: Expresses the type of a document. Distinguishes various types of message contents. There are top-level types & subtypes. The type of a document is represented as a combination of top-level type & subtype, i.e. application/PDF.

Type	***Subtype***	***description***
Text	Plain	Unformatted text.
	HTML	Text with HTML markup commands.
	XML	Text with XML markup commands.
Image	GIF	
	JPEG	
Audio	Basic	
	Tone	Specific audible tone.
Video	MPEG	
	Pointer	Representation of a pointer device for presentations.

Type	*Subtype*	*description*
Application	Octet-stream	Uninterpreted byte sequence.
	Postscript	Printable document in postscript.
	PDF	Printable document in PDF.
Multipart	Mixed	Independent parts in the specified order.
	Parallel	Parts must be viewed simultaneously.

Architecture: Enhancement to basic architecture: CGI, scripts, applets, servlets.

Common gateway interface (CGI): It defines a standard way by which a web server can execute a program taking user data as input. When the server sees the request, it starts the program named in the request & passes it the parameter values. The program does its work & returns the results in the form of a document that is sent back to the users browser to be displayed.

Servers can also process a fetched document before passing it to the client. A document may contain a server-side script, which is executed by the server when the document has been fetched locally. The result of executing a script is sent along with the rest of the document to the client. The script itself is not sent.

Its possible to pass precompiled programs to a client in the form of applets. An applet is a small stand-alone application that can be sent to the client & executed in the browsers address space.

Server-side counterpart of applets: Servlets. A servlet is a precompiled program that is executed in the address space of the server.

Architectural overview:

- Client: Browser: Request processing, applets, post processing.
- Server: Web server (servlets, request relay, post processing), CGI program. Servlet, CGI & request relay may communicate with a database.

Hypertext transfer protocol (HTTP): Client-server protocol. A client sends a request message to a server & waits for a response message. It is stateless. It is based on TCP. Whenever a client issues a request to a server, it sets up a TCP connection to the server & sends its request message along that connection. The same connection is used for receiving the response. Persistent connections: A connection can be used to issue several requests without the need for a separate connection.

HTTP methods:

- Head: Request to return the header of a document.
- Get: Request to return a document to the client.
- Put: Request to store a document.
- Post: Provide data that are to be added to a document.
- Delete: Request to delete a document.

HTTP messages: HTTP recognizes only request & response messages. Request

message consists of three parts: Request line (operation, reference, version), request message headers (message header name, value), message body. Response message consists of three parts, too: Status line (version, status code, phrase), response message headers (message header name, value), message body.

Plug-in: Small program that can be dynamically loaded into a browser for handling a specific document type. A plug-in should be locally available, possibly after being specifically transferred by a user from a remote server. Plug-ins offer a standard interface to the browser & expect a standard interface from the browser.

Web proxy: Client-side process. Used to allow a browser handle application-level protocols other than HTTP. By now, proxies are used for providing a cache shared by a number of browsers.

Uniform resource identifier (URI): Single naming scheme to refer to documents. Two forms: Uniform resource locater (URL) & uniform resource name (URN). The syntax of a URI is determined by its associated scheme (e.g. http, ftp, telnet). The name of the scheme is part of the URI.

URL: Identifies a document by including information on how & where to access the document. It is a location-dependent reference to a document. Where a document is located: Scheme (http, ftp, telnet, file, data, tel, modem), DNS name of the server or IP address, port, pathname. E.g. http://www.cs.vu.nl/home/steen/mbox, tel:+31201234567.

URN: Acts as true identifier. It is used as a globally unique, location-independent, persistent reference to a document. Consists of: "urn", name space, name of the resource. E.g. urn:isbn:0-13-349945-6.

Caching: Client-side caching occurs at two places: Browsers are equipped with a simple caching facility, clients site often runs a web proxy, which can implement a shared cache.

Secure socket layer (SSL): Approach for setting up a secure channel. An update to SSL is the transport layer security (TLS). SSL & TLS are usually based on TCP. Transport layer - TLS - HTTP/FTP/telnet/...