distributed systems: "1. autonomous elaboration systems 2. that present themselves as a coherent system"											
	introduction										
		design goals									
			sharing of resources								
			distribution trasparency, apply to:								
				(data) access							
				location							
				relocation							
				migration							
				replication							
				concurrency							
				failure							
			being open								
			being scalable, how:								
				communication hiding							
				distribution of the algorithm							
				replica							
					srv replica						
					cache						
		types									
			distributed computing systems								
				cluster (high performance distributed computing)							
				grid							
			distributed information systems								
				TPS: distributed transaction processing; must be ACID:							
					atomicity: from outside transaction is seen indivisible						
					consistent: transaction doesn't violate system invariants						
					isolated: concurrent transactions don't interfere with each other						
					durable: once a transaction commits, the changes are permanent						
				EAI: enterprise application integration							
			pervasive systems								
	architectures										
		architectural styles (or software architectures): how relate components						component: module erogating and/or requesting services			
			layered architectures								
			Object-based (tight coupled)								
			Data-centered (data blackboard)								
			Event-based (bus architecture, loose coupled)								
			Shared-data space								
		system architecture (deployment)									
			centralized (client/server, vertical distribution, client type: thin/fat)								
				2/3 tired architecture							
			decentralized (peer to peer), horizontal distribution, servent)								
				overlay network							
					structured (DHT: distributed hash table, Direct acyclic graph, chord network)						
					unstructured (folooding, superpeer)						
			hybrid architecture								
	processes										
		process: program running (managed by OS)									
		thread: more instuction flows in a process (managed by user)									
		virtualization: presenting an API (or system calls) over another API (or system calls)									
			runtime system: uses OS of host								
			virtual machine monitor: uses hardware								
		clients: allow user interact with server									
			functional requirements								
				thin: User interface only							
				fat: ATM (automatic teller machine), TV set top box							
			non functional requirements requests:								
				location							
				migration							
				relocation							
				fault tolerance							
		servers: implement services used by users									
			types:								
				iterative: direct response to client							
				concurrent: pass the request to a worker Thread/process							
			port: on catalog or known								
			superserver								
			more channels to manage client commands to server while service is working								
			connection state								
				stateless							
				statefull							
			server clusters								
				HCP: High computing performance							
				load balancing							
					Dispatcher + cluster (of replicas)						
					distributed servers using MIPv6						
	communication										
		foundations									
			layered protocols								
				application	application						
				presentation							
				session	middleware						
				transport	OS						
				network							
				data link							
				physical							
			types of communication								
				persistent							
				transient							
				synchronous							
				asynchronous							
					synch.at request submission						

					synch.at request delivery						
		RPC: remote procedure call									
			parameter passing								
				copy by value							
				copy by reference							
				call by copy/restore							
		Berkeley Socket									
		Message oriented communicztion									
			MPI: message passing interface								
			message queueing interface: put, get, poll, notify								
			message brokers								
	naming: identifying end point of an entity using a stable mnemonic name										
		naming system: name <=> end point <=> entity <=> name									
		naming resolution system: returns the end point corresponding to a name									
		strategies to manage names / entity / endpoint									
			flat naming: by broadcasting / multicasting								
			forward pointers: when entity moves on, release a reference To the new address								
			home based: using MIPv6								
			distributed hyerarchical: DNS								
				recursive query: steps from component to component of DNS, final result to caller (not used)							
				iterative query: every step return to caller (usually used: it don't drawn the DNS name server)							
			DHT: distributed hash table								
	coordination, needed: 1.to access a resource 2. agree about events sequence										
		clock synchronization									
			physical clocks								
				quarz timer							
				solar day							
				TAI: International Atomic Time (SI => UTC: Universal Time Coordinate)							
				GPS							
			clock synchronization algoritms								
				network time protocol							
				Berkeley Algorithm							
		Lamport's logical clocks									
			totally ordered multicasting								
		Mutual exclusion									
			types:								
				token based							
				permission based							
			centralized algorithm								
			distributed algoritm (ricart and agrawala)								
			token-ring algorithm								
		election algorithm									
			the bully algorithm								
			a ring algorithm								
	consistency and replication										
		introduction:									
			reasons:								
				reliability (crash, corrupted data)							
				performance (in case of need to scale)							
			problem: keep all copies consistent requires global synchronization (costly on WAN) => relax consistency constraints								
		Data-centric consistency models									
			continuous consistency								
				deviation in numerical values btwn replicas							
				deviation in staleness btwn replicas							
				deviation with respect to the ordering of update op.							
			consistent **ordering of operations**								
				sequential consistency: "The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program"							
				causal consistency: "Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines"							
			eventual consistency: "data stores that have the property that in the absence of write-write conflicts, all replicas will converge toward identical copies of each other"								
		Client-centric consistency models (Bayou:)									
			monotonic reads: (same process on data item x) "reads will always return the same value or a more recent value"								
			monotonic writes: (same process on data item x) "write completes before any successive write"								
			read your writes: (same process on data item x) "the effect of a write will always be seen by a successive read"								
			writes follow reads: (same process on data item x) "a write following a previous read take place on the same or a more recent value than that was read"								
		replica management									
			finding the best location								
			content replication and placement								
				permanent replicas							
				Server-initiated replicas							
				Client-initiated replicas							
			content distribution								
				what propagate:							
					only notification of an update (invalidation protocols)						
					transfer data btwn copies						
					send the update operation						
				pull vs push protocols							
					Push-based approach (server-based protocols)						
					Pull-based approach (client-based protocols)						
				unicasting vs multicating							
			consistency protocols								
				(not seen) continuous consistency							
				Primary-based protocols							
				Local-write protocols							
				Paxos: a consensus protocol by L.Lamport and others where processes in error become to an halt. Decidability: 2*m+1 (m == num.of faulty systems)							
				Byzantine: a consensus protocol by L.Lamport and others where processes in error comunicate values. Decidability: 3*m+1 (m == num.of faulty systems)							