Chapter 1 Introduction

1. Why distributed?（相对于集中式系统，分布式系统的优点）

Economics（经济性）	微处理器能提供比大型机更好的性价比
Speed（速度）	分布式系统能提供比大型机更强的计算能力
Inherent distribution（固有的分布性）	有一些应用包含物理上分布的机器
Reliability（可靠性）	当某台机器崩溃时，整个系统仍能正常工作
Incremental growth（可扩展性）	计算能力逐步增加

相对于独立的 PC，分布式系统的优点

数据共享	允许用户共享一个数据库
外设共享	允许用户共享昂贵的外设，如彩色打印机
通信	使得个人与个人之间的通信更为方便，如 Email
灵活性	将工作负载更有效的分派到合适的机器上

分布式系统的缺点

软件	分布式系统的软件开发困难
通信网络	网络可能饱和或有损传输
安全	数据共享造成机密数据容易被窃取

2. 分布式系统的定义：

A distributed system is a collection of independent computers that appears to its user as a single, coherent system.

独立的计算机的集合，对这个系统的用户来说，系统就像一台计算机一样。

3. Challenges (D2 P18-19)

Heterogeneity	Security	Failure Handling	Transparency
Openness	Scalability	Concurrency	

4. Software concepts (Overview between DOS, NOS, and middleware)

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

DOS: Distributed Operating Systems

NOS: Network Operating Systems

Middleware：中间件定义成一个软件层，它的目的是屏蔽异构性，为应用程序员提供方便的编程模型。中间件表示成一组计算机上的进程或对象，他们互相交互，实现分布式应用的通信和资源共享支持。

Chapter 2 Communication

5. **RPC 的概念：**当机器 A 上的一个进程调用机器 B 上的一个过程，则调用进程挂起，被调用进程在 B 上执行。信息可通过参数从调用者传递到被调用者，和通过结果从被调用者返回。消息传递对程序员透明。

   Subtle problems：
   - 调用程序和被调程序运行在不同地址空间
   - 参数和结果需要在不同的积期间传递
   - 两台机器都可能崩溃，没种可能性都会引起不同的问题。

   Benefits:
   - Procedure call interface
   - Writing applications simplified
     - RPC hides all network code into stub functions
     - Programmers don't have to worry about details (Sockets, Port numbers, byte order)
   - RPC: presentation layer in OSI model

   Issues:
   - When parameters passing by value, multiple machine types are present in a distributed system.
   - When parameters passing by reference, it is replaced by copy/restore.
   - How does the client locate the server? – dynamic binding
   - In presence of failures:

     RPC 系统的 5 种错误

     1. 客户无法定位服务器

     解决：使用特定的返回值(error)/异常处理

     2. 客户发给服务器的请求消息丢失

     解决：设置一个 timer，超时重发

     3. 服务器发给客户的应答消息丢失

     解决：设置一个 timer,对于不幂等的请求，为客户请求分配序号，服务器区别不同的请求

     4. 服务器在收到消息后崩溃

     1 接受后，执行前崩溃

     2 执行后，发送前崩溃

     解决：等待服务器启动,然后重发请求/立即放弃并报告失败/不做任何保证

     5. 客户机在发送消息后崩溃

     解决：根绝 extermination: 在日志文件中纪录 RPC 请求，重启后清除孤儿

     再生 reincarnation: 将时间划分成不同时期，重启后广播新的时期开始。一旦收到广播消息，kill 所有 remote computations.

     温和再生 gentle reincarnation: 将时间划分成不同时期，重启后广播新的时期开始。收到广播消息时，定位 remote computations 的 owner，kill 没有 owner 的。

     过期 expiration: 赋予每个 RPC 一个一个执行时间 T，未完成任务需显式申请附加配额。
   - Performance：a lot slower
   - Security: (1) Message visible over network. (2) Need authenticate client and server.

6. 动态绑定：

Client 绑定到 Server：Server 先向本地 OS 请求一个端点（endpointt），然后在 DCE daemon 中注册该端点，记录在端点表中。Server 再向一个目录 Server 注册自己的网络地址和一个名字。Client 以 Server 提供的服务名字为 key 在目录 Server 中查找对应的网络地址，然后向 Server 上的 DCE daemon 请求服务端点。这些信息都具备后，执行 RPC 操作。

Advantages：ppt 02-68

Disadvantages:

7. Doors

A door is a generic name for a procedure in the address space of a server process that can be called by processes collocated with the server.

Benefit: They allow the use of a single mechanism, namely procedure call, for communication in a distributed system.

8. ROI 和 RPC 的比较

	Client	Server
ROI	Proxy（Interface）	Skeleton（Object）
RPC	Client Stub	Server Stub

传统的 RPC 和支持分布式对象的 RPC 的一个区别在于**后者提供系统范围（systemwide）的对象引用。**

9. RMI 调用语义

- 或许调用语义 Maybe：不采取任何容错措施。调用者不能判断一个远程方法是否已经执行过。面临故障：遗漏故障，系统崩溃。
- 至少一次调用语义 At-least-one：通过重发请求消息和重新执行过程。调用者或者接收到返回结果，说明至少执行过一次；或者接收到异常，通知他没有接收到执行结果。面临故障：系统崩溃，随机错误。
- 至多一次调用语义 At-most-onece：通过重发请求消息、过滤重复请求和重传应答实现。调用者或者接收到返回结果，说明恰好执行过一次；或者接收到异常，通知他没有接收到执行结果。

10. RMI 的实现

- 通信模块：实现请求-应答协议，它使用消息类型、请求 ID 和被调用对象的远程引用。服务器端通信模块为被调用对象类型选择调度程序，传输其本地引用，该本地引用曲子远程引用模块，用来替换请求消息中的远程对象标识符。
- 远程引用模块：翻译本地和远程对象引用以及创建远程对象引用。每个进程都有一个远程对象表，包括（1）该进程拥有的所有远程对象的表项。（2）每个本地代理的表项。
- 中间件对象，包括（1）代理：是远程方法调用对客户透明。（2）调度程序：接收来自通信模块的请求消息，并使用方法 ID 选择骨架中恰当的方法。（3）骨架：实现远程接口中的方法。骨架方法解码请求消息中的参数，并调用远程对象中的相应方法，等待调用完成，然后将结果和任何异常消息编码进应答消息，发送给代理。
- 厂方法：远程对象接口不能包括构造函数，厂方法代替构造函数创建远程对象。
- 绑定程序：维护从文本名字到远程对象引用的映射表。
- 服务器线程：服务器为每个远程调用的执行分配一个独立的线程，以避免延误其他本地或远程调用的执行。

11. DCE Distributed-Object Model

Distributed dynamic model: associated with only a single client

Distributed named object: created by a server to be shared by several clients.

12. Persistence and Synchronicity in 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 and receiving application are executing.

Asynchronous communication: a sender continues immediately after it ahs 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.

Chapter 4 Naming

13. Name space Distribution

Item	**Global**	**Administrational**	**Managerial**
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

14. Advantages and drawback of recursive name resolution:

Advantages: (1) Caching results is more effective compared to iterative name resolution.

(2) Communication costs may be reduced.

Drawback: It puts higher performance demand on each name server.

15. Why not centralized DNS?

(1) single point of failure
(2) traffic volume
(3) need distant centralized database
(4) maintenance
(5) doesn't scale

Chapter 5 Synchronization

16. Happened-Before Relationship

1: If a and b are two events in the same process, and a comes before b, then a -> b.

2: If a is the sending of a message, and b is the receipt of that message, then a -> b.

3: If a -> b and b -> c, then a -> c.

17. Lamport's Algorithm

Each process Pi maintains a local counter Ci and adjusts this counter according to the following rules:

1: For any two successive events that take place within Pi, Ci is incremented by 1.

2: Each time a message m is sent by process Pi, the message receives a timestamp Tm = Ci.

3: Whenever a message m is received by a process Pj, Pj adjusts its local counter Cj:

Cj = max{ Cj + 1, Tm + 1}

18. Total Ordering with Logical Clocks

(Ti, i) < (Tj, j) if and only if

1: Ti < Tj or

2: Ti = Tj and i < j

19. Totally-Ordered Multicast

1: Process Pi sends timestamped message msg_i to all others. The message itself is put in a local queue $queue_i$.

2: Any incoming message at Pj is queued in $queue_j$, according to its timestamp. And multicasts an acknowledgement to the other processes.

3: Pj passes a message msg_i to its application if:

(1) msg_i is at the head of $queue_j$ and

(2) for each process P_k, there is a message msg_k or $acknowledgement_k$ in $queue_j$ with a larger timestamp.

20. Vector Timestamps

Pi maintains a vector Vi with the following two properties:

1: Vi[i] is the number of events that have occurred so fat at Pi

2: If Vi[j] = k, then Pi knows that k events have occurred at Pj

- When a process Pj receives a message m from Pi with vector timestamp vt (m), it (1) updates each Vj[k] to max {Vj[k], vt[k]} and (2) increments Vj[j] by 1.
- To support causal delivery of messages, assume you increment your own component only when sending a message. Then, Pj postpones delivery of m until:
  - vt(m)[i] = Vj[i] + 1 (states that m is the next message that Pj was expecting form process Pi.)
  - vt(m)[k] <= Vj[k] for k != i (states that Pj has not seen any messages that were not seen by Pi when it sent message m.

21. **Chandy and Lamport's 'snapshot' algorithm**

Marker receiving rule for process pi

On pi's receipt of a marker message over channel c:

if (pi has not yet recorded its state) it
records its process state now;
records the state of c as the empty set;
turns on recording of messages arriving over other incoming channels;
else
pi records the state of c as the set of messages it has received over c
since it saved its state.
end if

Marker sending rule for process pi

After pi has recorded its state, for each outgoing channel c:

pi sends one marker message over c
(before it sends any other message over c).

22. 选举算法（选举出协调者，发起者（比如记录快照的发起者）等）

*选出拥有最大进程号的进程

Bully Algorithm：

1. Any process can just start an election by sending an election message to all other processes (assuming you don't know the weights of the others).
2. If a process Pheavy receives an election message from a lighter process Plight, it sends a take-over message to Plight. Plight is out of the race.
3. If a process doesn't get a take-over message back, it wins, and sends a victory message to all other processes.

Ring Algorithm:

Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator.

1. Any process can start an election by sending an election message to its successor. If a successor is down, the message is passed on to the next successor.
2. If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known.
3. The initiator sends a coordinator message around the ring containing a list of all living processes. The one with the highest priority is elected as coordinator.

23. Mutual Exclusion:

Ricart & Agrawala

Replies (i.e. grants) are sent only when:

- The receiving process has no interest in the shared resource; or
- The receiving process is waiting for the resource, but has lower priority (known through comparison of timestamps).

In all other cases, reply is deferred, implying some more local administration.

Problems:

- The single point of failure has been replaced by n points of failure. If any process crashes, it will fail to respond to requests. This silence will be interpreted as denial of permission, thus blocking all subsequent attempts by all processes to enter all critical regions.
- The probability of one of n processor failing is at least n times as large as a single coordinator failing.
- If reliable multicasting is not available, each process must maintain the group membership list itself, including processes entering the group, leaving the group, and crashing.

In a word, Ricart & Agrawala algorithm is slower, more complicated, more expensive and less robust than centralized algorithm.

Ring Algorithm

Pros:

- Guarantee mutual exclusion.
- No starvation.

Cons:

- Regeneration of the token if it is ever lost: Detecting that the token is lost is difficult, since the amount of time between successive appearances of the token on the network is unbounded.
- Recovery if a process crashes: It is easier than in other cases though.

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

Chapter 6 Consistency and Replication

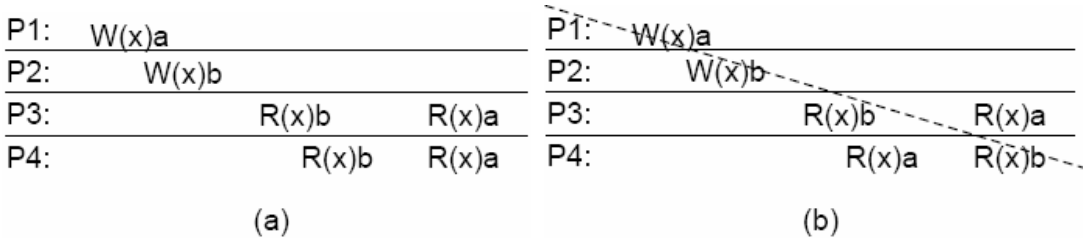
24. **复制的优点：**reliability；performance。缺点：复制透明性；一致性问题。
25. **多个 clients 并发访问远程对象的两种控制方式：**一是对象自己处理（使用一个互斥机制）；二是对象驻留的 server 用一个 adapter 来负责。

    解决复制对象一致性问题的两种方法：一是对象知道自己能被复制，对象之间可以通信，优点是支持特定对象的赋值策略；二是由分布式系统自己管理（middleware 有一个复制协议），优点是应用程序开发者任务简单。复制和 cache 是两种扩展技术。
26. **Consistency model:** a contract between a (distributed) data store and processes, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency.
27. **Data-Centric Consistency Models**

Strict Consistency: Any read to a shared data item X returns the value stored by the most recent write operation on X.

Problem: Relies on absolute global time.

Sequential Consistency: The result of any execution is the same as if the operations of all processes were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program.

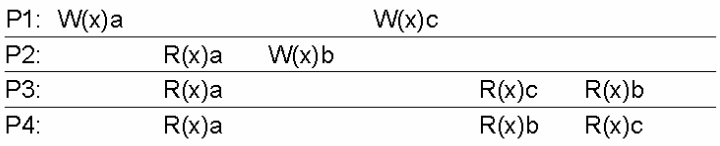


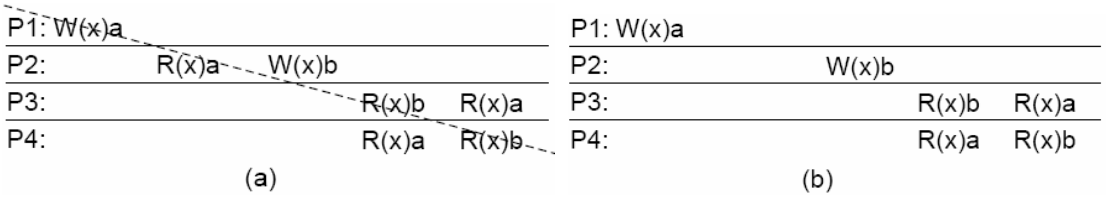
Linearizable: Sequential plus operations are ordered according to a global time.

Comparing sequential consistency and serializability in DB transactions: Main difference is granularity: sequential consistency is defined in terms of read and write operations; serializability is defined in terms of transactions.

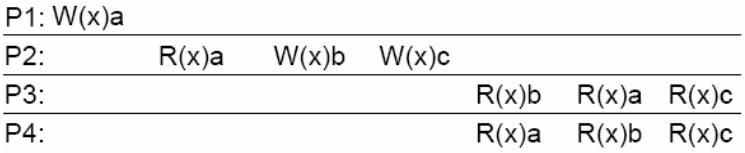
Casual Consistency: Writes that are potentially casually related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.

Implementing: one way by vector timestamps





FIFO Consistency: Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.

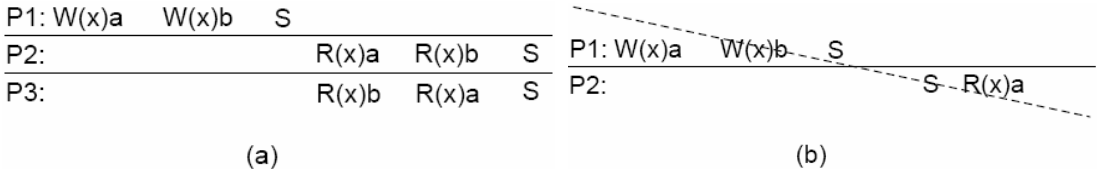


Weak Consistency:

- Accesses to synchronization variables associated with a data store are sequentially consistent
- No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere
- No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.

Distinction:

- Weak consistency enforces consistency on a group of operations.
- Limit only the time when consistency holds



Release Consistency:

Divide access to a synchronization variable into two parts: an acquire and a release phase. Acquire forces a requester to wait until the shared data can be accessed; release sends requester's local value to other servers in data store.

1．在访问共享变量前，所有先前的 acquire 都必须完成。

2．在进行 release 前，先前的所有读写操作都必须结束。

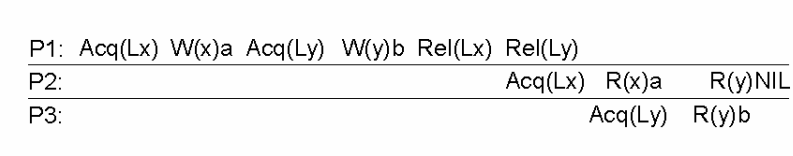
3．acquire 和 release 访问必须满足处理机（FIFO）一致性。

Acquire and release phase make it no different with sequentially consistent.

Entry Consistency:

- With release consistency, all local updates are propagated to other copies/servers during release of shared data.
- With entry consistency, each shared data item is associated with a synchronization variable.
- When acquiring the synchronization variable, the most recent values of its associated shared data item are fetched.

Note: Where release consistency affects all shared data, entry consistency affects only those shared data associated with a synchronization variable.



(a) Consistency models not using synchronization operations.

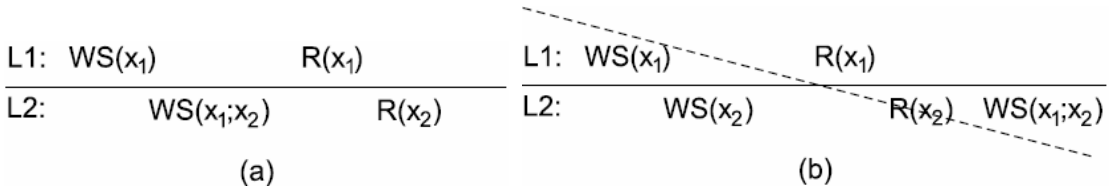
Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order.
FIFO	All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order

(b) Models with synchronization operations.

Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

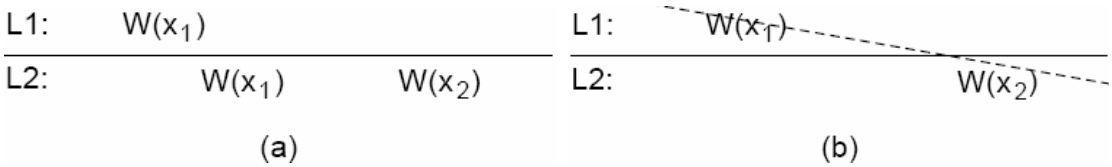
28. Client-Centric Consistency Models

Monotonic Reads: If a process reads the value of a data item x, any successive read operation on x by that process will always return that same or a more recent value.

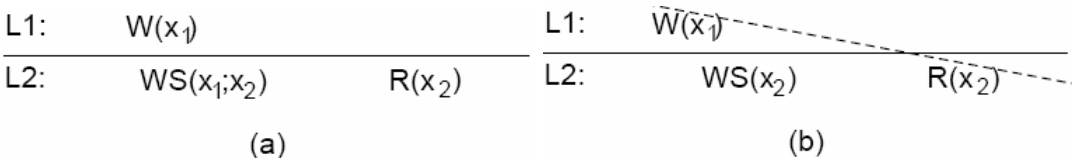


Monotonic Writes: A write operation by a process on a data item x is completed before any successive write operation on x by the same process.

A write operation on a copy of data item x is performed only if that copy has been brought up to date by means of any preceding write operation, which may have taken place on other copies of x.

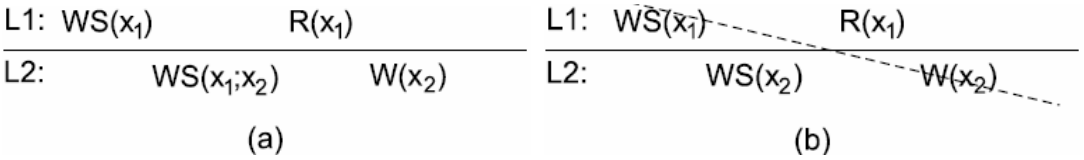


Read Your Writes: The effect of a write operation by a process on data item x, will always be seen by a successive read operation on x by the same process.



Writes Follow Reads: A write operation by a process on a data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read.

Any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by that process.



29. **分布式协议：**三种数据拷贝的类型：永久复制（多个拷贝分布再 LAN 的多个 server 上；镜像分布再 Internet 中）；server 发起的复制（要精确判断何时何地创建或撤销副本）；client 发起的复制（主要是 cache）。
30. **更新传播的三种传播方式：**仅传播一个更新通知；传播更新后数据；传播更新操作种类。
31. **Pull versus Push Protocols**

Issue	**Push-based**	**Pull-based**
State of server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

Adaptive leases：

- Age-based leases: An object that hasn't changed for a long time, will not change in the near future, so provide a long-lasting lease
- Renewal-frequency based leases: The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be
- State-based leases: The more loaded a server is, the shorter the expiration times become

32. **一致性协议**

- Primary-based 协议（每个数据项 x 有一个相关的 Primary 负责对 x 的读写操作）
  - Remote-write 协议（所有读写操作都在一个远程 server 上进行）
  - Primary-backup Remote-write 协议（可以本地读，但写必须由 primary 管理）
  - Local-write 协议（将数据项迁移到本地 server 后本地操作）
  - Remote-write Local-write 协议（本地写操作后更新 backup server）
- Replicated-write 协议（写操作可对多个副本同时进行）
  - Active replication（多个副本对象对其他对象的调用以及多个副本对象返回结果时要选择一个 coordinator）
  - Quorum-based 协议（客户再对副本进行读写操作前必须得到 server 的批准，两个不等式）

Chapter 7 Fault Tolerance

33. Dependability

Availability: Readiness for usage

Reliability: Continuity of service delivery

Safety: Very low probability of catastrophes

Maintainability: How easy can a failed system be repaired

34. Terminology

Failure: When a component is not living up to its specifications, a failure occurs

Error: That part of a component's state that can lead to a failure

Fault: The cause of an error

Fault tolerance: build a component in such a way that it can meet its specifications in the presence of faults

35. Failure Models

Crash failures: A component simply halts, but behaves correctly before halting

Omission failures: A component fails to respond Timing failures: The output of a component is correct, but lies outside a specified real-time interval (performance failures: too slow)

Response failures: The output of a component is incorrect (but can at least not be accounted to another component)

Value failure: The wrong value is produced

State transition failure: Execution of the component's service brings it into a wrong state

Arbitrary failures: A component may produce arbitrary output and be subject to arbitrary timing failures

Observation: Crash failures are the least severe; arbitrary failures are the worst

36. Process Resilience

Flat groups: Good for fault tolerance as information exchange immediately occurs with all group members; however, may impose more overhead as control is completely distributed (hard to implement).

Hierarchical groups: All communication through a single coordinator => not really fault tolerant and scalable, but relatively easy to implement.