

**LEARNMORE NDLOVU**

**INFORMATION TECHNOLOGY**

**H160170B**

**DISTRIBUTED SYSTEMS  
ASSIGNMENT 3**

Architecture of replication management systems

**What is Replication?**

It involves sharing of information so as to ensure consistency between redundant resources (software or hardware components) to improve reliability, fault-tolerance or accessibility.

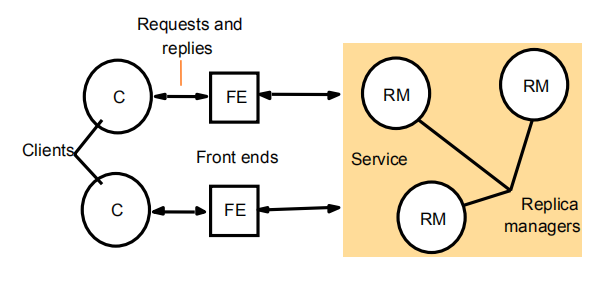
**Benefits of this technique:**

Performance Enhancement: More load can be tolerated because workload is shared among several processes.

Increased Availability: Helps a service overcome individual server failures.

Fault Tolerance: Ensures correctness in addition to availability.

**A basic architectural model for the management of replicated data.**



**Challenges of replication**

Complex mechanisms

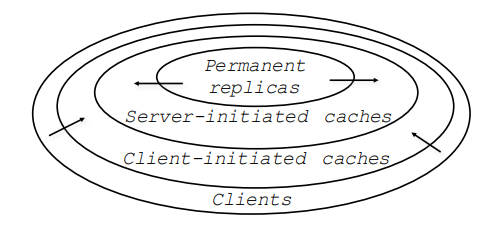
* Placement of replicas (and search for them)
* Propagation of data (e.g., updates) among the replicas
* Consistency maintenance
* Monitoring and failover mechanisms

These protocols also consume bandwidth

Some of this complexity is exposed to the clients

* impossible to achieve complete replication transparency

**Placement of replicas**



Permanent replicas

* Clusters of servers
* Geographically dispersed web mirrors (Akamai)

Server-initiated caches

* Placement of hosting servers
* Placement of caches

Client-initiated caches

* Enterprise proxies or web browser caches

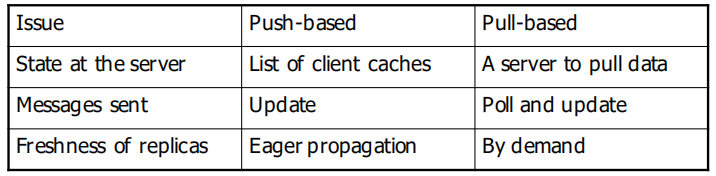
**Propagation of updates among the replicas**

Push-based propagation

* A replica pushes the update to the others
* May push the new data or parameters of the update operation

Pull-based propagation

* A replica requests another replica to send the newest data it has



**Pushing data vs pushing updates**

* Pushing updates reduces traffic
* Requires more processing power on each replica
* Requires deterministic operations

**Hybrid push-pull approaches**

* Lease-based propagation
* Pushing invalidations
* A replica that performs the update notifies other replicas
* A replica informed that a newer version is available will fetch the new version at a later point

**Replication Models**

Master – Slave Model

* In this model one of the copy is the master replica and all the other copies are slaves. In this model the functionality of the slaves are very limited, thus the configuration is very simple. The slaves essentially are read-only. Most of the master-slaves services ignore all the updates or modifications performed at the slave, and “undo” the update during synchronization, making the slave identical to the master.

Client – Server Model

* The client-server model like the master- slave designates one server, which serves multiple clients.
* In Client- server replication all the updates must be propagated first to the server, which then updates all the other clients.
* Since all updates must go through the server, the server acts as a physical synchronization point. In this model the conflicts which occur are always be detected only at the server and only the server needs to handle them.

Peer–to–Peer Model

* Here all the replicas or the copies are of equal importance or they are all peers.
* In this model any replica can synchronize with any other replica, and any file system modification or update can be applied at any replica.
* Peer-to-peer systems can propagate updates faster by making use of any available connectivity. They provide a very rich and robust communication framework.
* They are more complex in implementation and in the states they can achieve. One more problem with this model is scalability.

Consistency and its Models

Consistency Problem Replication improves reliability and performance … but when a replica is updated, it becomes different from the othersso we need to propagate updates in a way that temporal inconsistencies are not noticed however this may degrade performance severely

Strict Consistency

* Every read of x returns a value corresponding to the result of the most recent write to x.
* True replication transparency, every process receives a response that is consistent with the real time.
* All writes are instantaneously visible to all process.
* Assumes absolute global time ○ Due to message latency, strict consistency is difficult to implement.

In general, A:write t(x,a) then B:read t’(x,a); t’>t

Sequential Consistency (weaker than strict 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” (Lamport 1979).
* A write to a variable does not have to be seen instantaneously, however, writes to variables by different processes have to be seen in same order by all processes.
* All processes see the same interleaving of operations.

Linearizability (weaker than strict consistency, stronger than sequential consistency)

* Interleaving of reads and writes into a single total order that respects the local ordering of the operations of every process. ○ A trace is consistent when every read returns the latest write preceding the read.
* A trace is linearizable when ○ It is consistent. ○ If t1 , t2 are the times at which pi and pj perform operations, and t1 < t2 , then the consistent trace must satisfy the condition that t1 < t2.
* The linearizable trace is A:W(x,1), B:W(y,1), A:R(y,1), B:R(x,1).

Causal Consistency

* All writes that are causally related must be seen by every process in the same order, and reads must be consistent with this order.
* Writes that are not causally related to one another (concurrent) can be seen in any order.
* No constraints on the order of values read by a process if writes are not causally related.
* For example, consider a chat between three people, where Attiya asks “shall we have lunch?”, and Barbarella & Cyrus respond with “yes”, and “no”, respectively. Causal consistency allows Attiya to observe “lunch?”, “yes”, “no”; and Barbarella to observe “lunch?”, “no”, “yes”. However, no participant ever observes “yes” or “no” prior to the question “lunch?”

**Security threats in Distributed System**s

**Issues**

Separation of

Security policies :Precise defifinition of which entities in the system can take what actions

Security mechanism: Means of enforcing that policy

Distributed system security

* Communication between users or processes that may be on different machines: Secure channel (authentication, message integrity, confidentiality)
* Authorization to ensure that a user or process performs only those actions that is allowed under the security policy: Access Control

Security services

* What any distributed system should provide as part of its infrastructure to enable the implementation of different policies.

**Basic Concepts**

* Security: attempt to protect the services and data it offers against security threats.
* Confidentiality: the property of a computer system whereby its information is disclosed only to authorized parties
* Integrity: the characteristic that alterations to a systemʼs assets can be made only in an authorized way.

**Types of Threats**

* Interception: An unauthorized party has gained access to a service or data
* Interruption: Services or data become unavailable, unusable, destroyed and so on.
* Modification: Unauthorized changing of data or tampering with a service so that it no longer adheres to its original specifications
* Fabrication: Refers to the situation in which additional data or activity are generated that would normally not exist

**Security Threats**

Consider a client and server connected by a network link. There are a number of

threats posed by such links:

* Passive Tap
* Active Tap
* Denial of Service
* Faking
* Replay
* Traffic Analysis

Accidental Access

The passive tap allows the interloper to observe traffic passing on the link. Whilst it cannot change the observed messages, it may be able to use the information to gain improper access, e.g. if a password is observed it can later be used to log in as someone else. Passive tapping is extremely easy on some networks, e.g. shared Ethernet.

An active tap is where the interloper can interpose a process between the original client and server. This may be purely for obtaining information such as passwords (“Trojan Horse”), or it may be as a means of changing messages, usually to the benefit of the interloper.

An often forgotten threat is denial of service. In some cases preventing access by a legitimate user can be a serious threat, so very persistent attempts to access a server, although they may be badly formed messages, may jam up the protocol handler and prevent legitimate access. There was reputed to be such a case some years ago where bookings for a Bruce Springsteen concert jammed all the Washington telephone exchanges for several hours. We were assured that there was no threat to key US Government lines.

A process may fake messages and send them to a server. These may be modeled on previous legitimate messages, with key fields changed. If the fields are not changed, it is termed replay, and this may be valuable to the fraud if it can get, say, multiple payments into its bank account. Audit trails are usually used in secure systems to record all transfers, but although the perpetrator may be known, it may be too late to recover the loss. In computer systems, audit trails need to record details of people, processes and machines, although the latter is very difficult on LANs.

Methods of Attack

* Eavesdropping: Obtaining copies of messages without authority
* Masquerading:Sending or receiving messages using the identity of another principal without their authority
* Message tampering: Intercepting messages and altering their contents before passing them on to the intended recipient:Man-in-the-middle attack
* Replaying: Storing intercepted messages and sending them out at a later time
* Denial of service:Flooding a communication channel or a system resource with messages in order to deny access for others

Security Mechanisms

Encryption

* transform data into something an attacker cannot understand
* provides a means to implement confidentiality
* provides support for integrity

Authentication

* verify the claimed identity of a user, client, server and so on

Authorization

* check whether the client is authorized to perform the action requested

Auditing

* auditing tools are used to trace which clients accessed what, and which way

**Secure Channels**

* Secure communication requires authentication of the communicating parties, but also ensuring message integrity and possibly confidentiality as well.
* A secure channel protects senders and receivers against interception, modification, and fabrication of messages.
* It does not necessarily protect against interruption.
* Protecting messages against interception is done by ensuring confidentiality
* Protecting messages against modification and fabrication is done through
* protocols for mutual authentication and message integrity.

Authentication

Authentication and message integrity cannot do without each other.

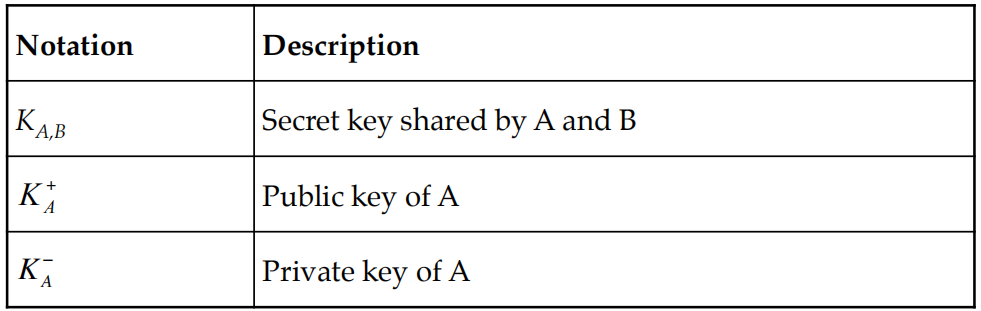
The combination works as follows:

Alice starts by sending a message to Bob to set up a channel

Once the channel has been set up, Alice knows for sure that she is talking to Bob,and Bob knows for sure that he is talking to Alice, they can exchange messages

To subsequently ensure integrity it is common practice to use secret-key

cryptography by means of session keys. + KA



**Message Integrity and Confidentiality**

Besides authentication, a secure channel should also provide guarantees for message integrity and confidentiality

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

Protecting a message against modifications is somewhat more complicated

Digital Signatures: Digitally sign a message in such a way that the signature is uniquely tied to its content

Several ways to place digital signatures:

* Use a public-key cryptosystem such as RSA
* Use a message digest

**Session Keys**

During the establishment of a secure channel, after the authentication phase has completed, the communicating parties generally use a unique shared session key for confidentiality.

The session key is safely discarded when the channel is no longer used.

Why not use the same keys for confidentiality as those that are used for setting up the secure channel?

* Cryptographic keys are subject to “wear and tear” just like ordinary keys.
* Protection against replay attacks
* If a key is compromised, only a single session is affected
* The combination of a long-lasting keys with the much cheaper and more temporary session keys is often a good choice for implementing secure channels for exchanging data

**Access Control**

In the client-server model, once a client and a server have set up a secure channel, the client can issue requests that are to be carried out by the server.

A request involve carrying out operations on resources that are controlled by the server.

Such a request can be carried out only if the client has sufficient access rights for that request.

Verifying access rights is referred to as access control, whereas authorization is about granting access rights.

**Role-Based Access Control**

Related to having groups as protection domains, it is also possible to implement protection domains as roles.

In role-based access control, a user always logs into the system with a specific role, which is often associated with a function the user has in an organization.

A user may have several functions.

Depending on the role the user takes when logging in, he may be assigned different privileges (i.e. his role determines the protection domain in which he will operate).