Parallel & Distributed System

Consistency & Replication

Md. Biplob Hosen

Lecturer, IIT-JU

Email: biplob.hosen@juniv.edu

Overview

- Data are replicated to **enhance reliability** or **improve performance**. One of the major problems is **keeping replicas consistent**.
- Consistency of replicated data can be achieved through various ways. In **consistency models**, it is assumed that multiple processes simultaneously access shared data.
- Consistency models for shared data are often hard to implement efficiently in large-scale distributed systems.
- One specific class is formed by **data-centric consistency models**.
- There are two issues we need to consider in consistency implementation.
- Firstly, in managing replicas, it takes into account not only the **placement of replica servers**, but also how **content is distributed** to these servers.
- The second issue is how **replicas are kept consistent**. In most cases, applications require a strong form of consistency, which means that updates are to be propagated immediately between replicas.

Reasons for Replication

Reliability:

1. **Mask failures:**

- If a file system has been replicated, it may be possible to continue working after one replica crashes .

2. **Mask corrupted data:**

- For example, let there be three copies of a file and every read and write operation is performed on each copy. We can safeguard ourselves against a single, failing write operation, by considering the value that is returned by at least two copies as being the correct one.

Reasons for Replication

Performance:

1. **Scalability (numbers and geographical size):**

- Scaling in numbers occurs, for example, when an increasing number of processes needs to access data that are managed by a single server. In that case, performance can be improved by replicating the server and subsequently dividing the work.
- Scaling with respect to the size of a geographical area may also require replication. The basic idea is that by placing a copy of data in the proximity of the process using them, the time to access the data decreases.

Cost of Replication

- Replicas must be kept consistent.

Dilemma:

- Having multiple copies may lead to consistency problem.
- Modification on one copy triggers modifications on all other replicas.
- Exactly when and how those modifications need to be carries out determines the price of replication.

Consistency Issues – Caching

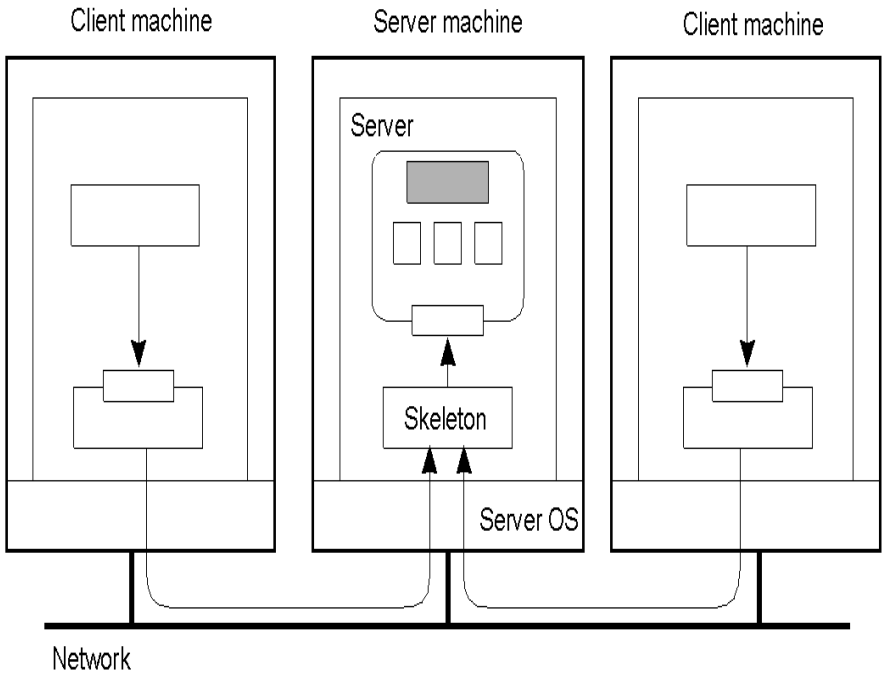
- To improve access times to web pages, web browsers often locally store a copy of a previously fetched web page (i.e., they cache a Web page).
- If a user requires that page again, the browser automatically returns the local copy. The access time as perceived by the user is excellent. However, if the user always wants to have the latest version of a page, he may be in for bad luck. The problem is that if the page has been modified in the meantime, modifications will not have been propagated to cached copies, making those copies out-of-date.
- One solution to the problem of returning a stale copy to the user is to forbid the browser to keep local copies in the first place, effectively letting the server be fully in charge of replication. However, this solution may still lead to poor access times if no replica is placed near the user.
- Another solution is to let the web server invalidate or update each cached copy, but this requires that the server keeps track of all caches and sending them messages.

Replication for Scaling – Access/Update Ratio

- Replication and caching are applied as scaling techniques. Placing copies of data close to the processes using them can improve performance through reduction of access time and thus solve scalability problems. A possible trade-off that needs to be made is that keeping copies up to date may require more network bandwidth.
- Consider a process P that accesses a local replica N times per second, whereas the replica itself is updated M times per second. Assume that an update completely refreshes the previous version of the local replica.
- If N << M, that is, the access-to-update ratio is very low, we have the situation where many updated versions of the local replica will never be accessed by P, rendering the network communication for those versions useless.
- In this case, it may have been better not to install a local replica close to P, or to apply a different strategy for updating the replica.
- A more serious problem, however, is that keeping multiple copies consistent may itself be subject to serious scalability problems.

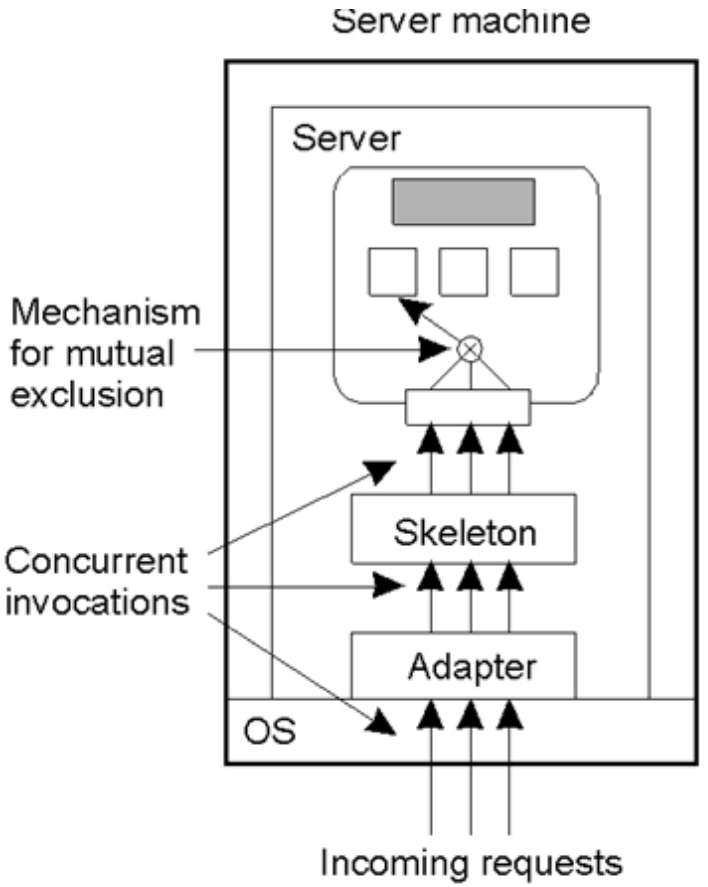
Object Replication

- Objects have the benefit of encapsulating data and operations on that data.
- Therefore, it is easier to draw the line between operations that are specific to some data and operations that are typically data independent.
- Consider the distributed remote object that is shared by multiple clients.
- First we need to solve the problem of how to protect the object against simultaneous access by multiple client.



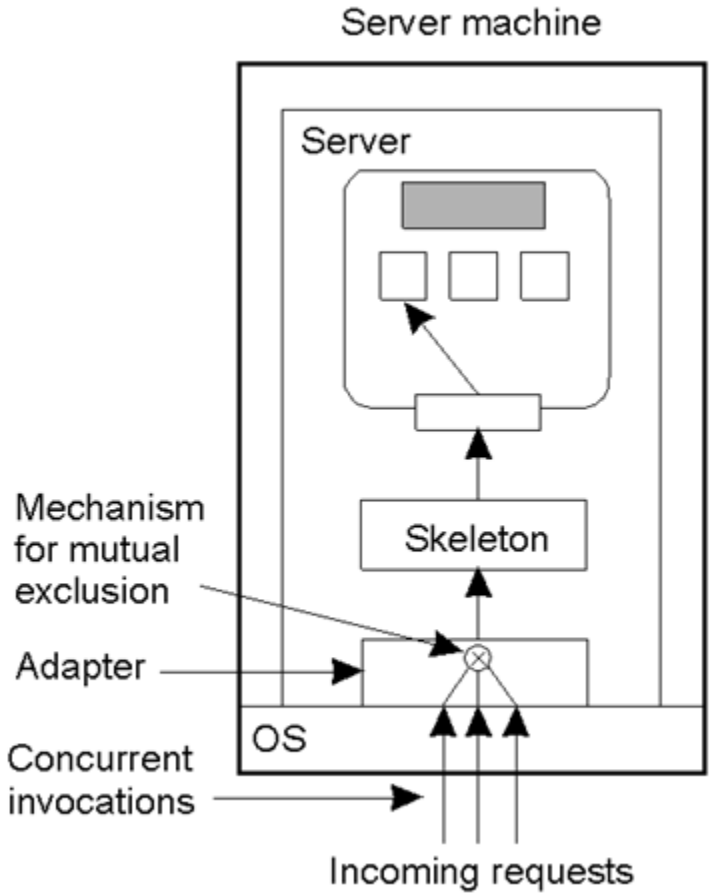
Continue…

- The first solution is that the object itself can handle concurrent invocations.
- Assume that two clients simultaneously invoke a method of the same object, leading to two concurrent threads at the server where the object resides.
- In Java, if the object's methods have been synchronized, only one of those two threads is allowed to proceed while the other is blocked until further notice.
- A remote object itself is capable of handling concurrent invocations on its own.



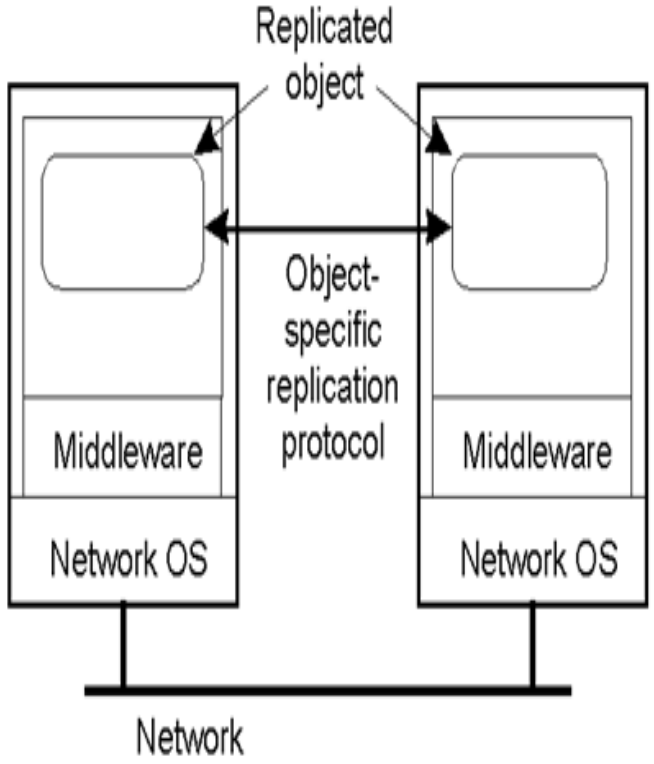
Continue…

- The second solution is that the object is completely unprotected against concurrent invocations, but the server in which the object resides is made responsible for concurrency control.
- In particular, by using an appropriate object adapter, it becomes possible to ensure that concurrent invocation will not leave the object in corrupted state.
- For example, such an object adapter is one that uses a single thread per object, effectively serializing all accesses to each object it manage.



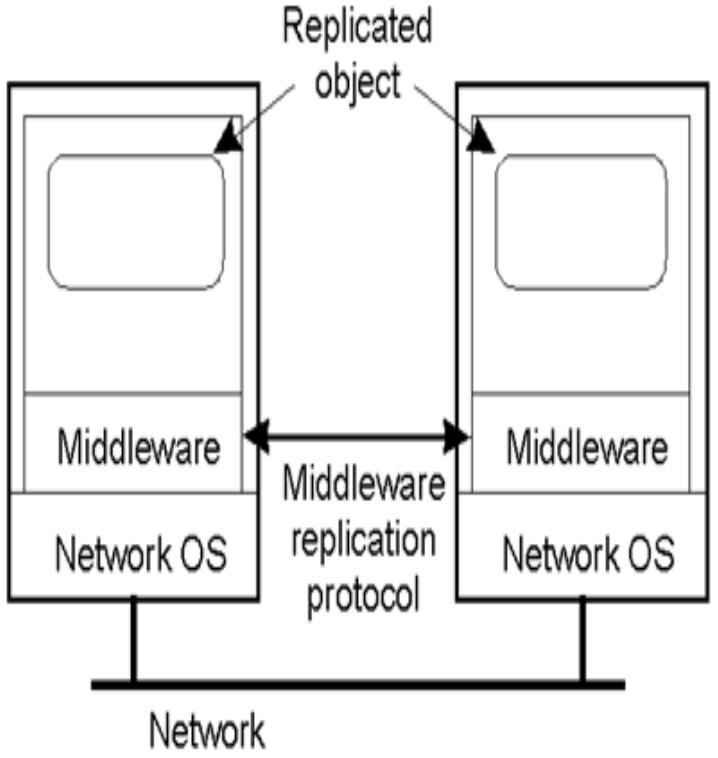
Continue…

- Replication share remote object without taking any special measure regarding the handling of concurrency control may lead to consistency problems.
- There are essentially only two approach to solve this problem.
- First approach: object is aware of the fact that it can be replicated. Therefore, the object is responsible for ensuring that its replicas stay consistence in the presence of concurrent invocation.
- The distributed system underlying such objects essentially need not provide any general support for replication.



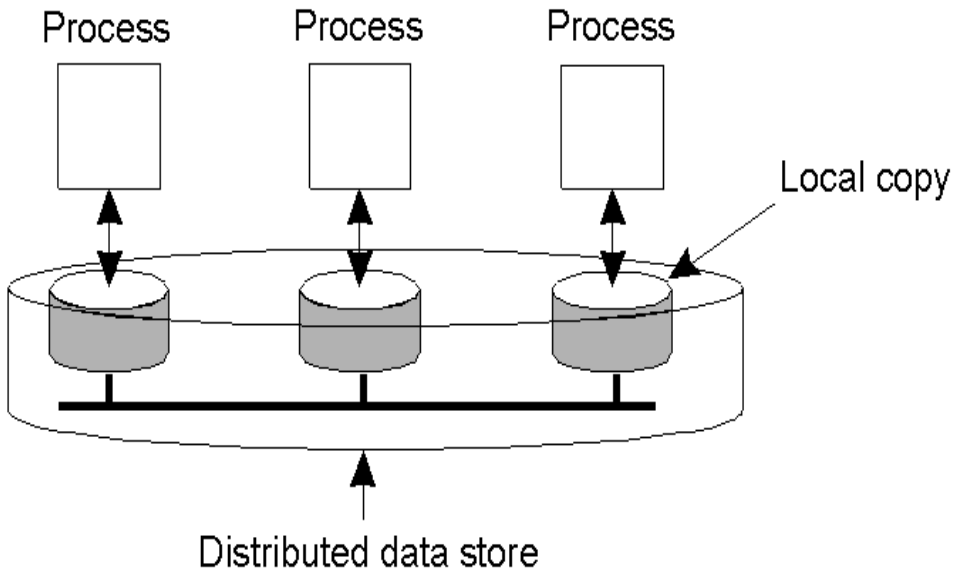
Continue…

- The second, more common approach for handling consistency with concurrent objects is to make the distributed system responsible for managing replication.
- In particular, the distributed system ensures that concurrent invocations are passed to the various replicas in the correct order.
- The advantage of letting the distributed system take care of replica management is the simplicity for application developers.



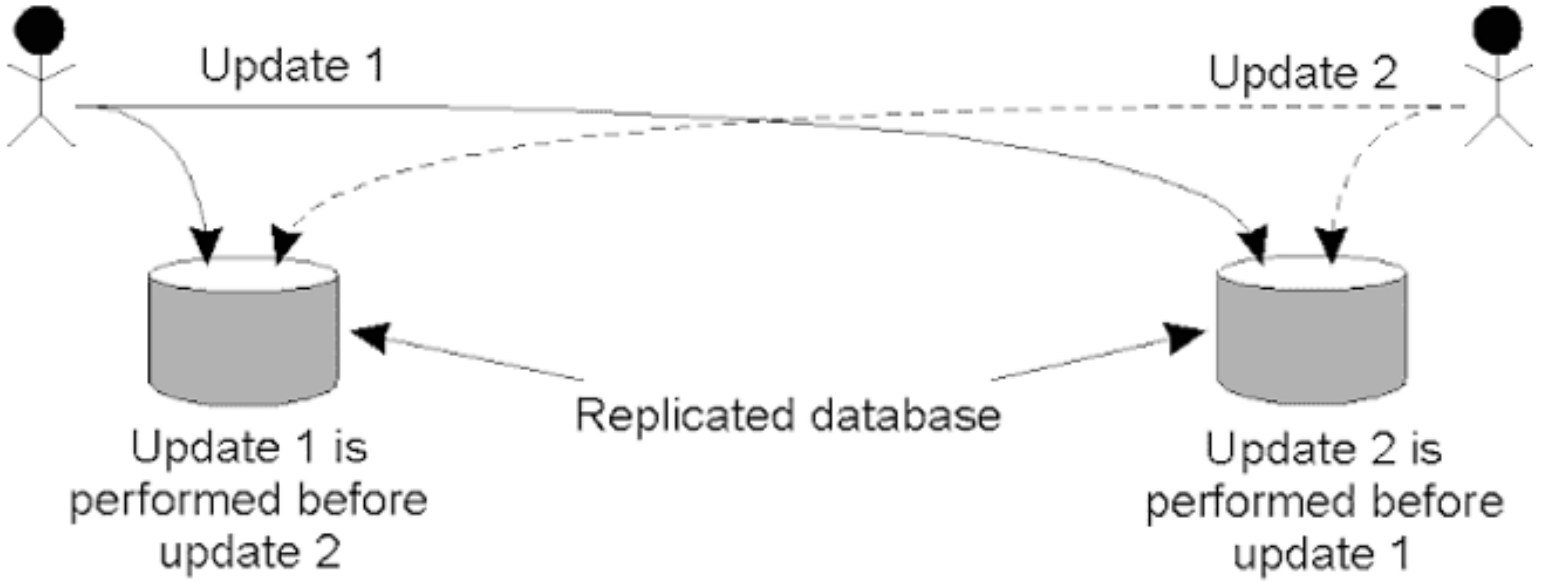
Data-centric Consistency Models

- A **data store** is a data repository of a set of integrated objects.
- A data store may be physically distributed across multiple machines.
- In particular, each process that can access data from the store is assumed to have a local (or nearby) copy available of the entire store.
- A **write operation** of one data store is propagated to other copies.
- A data operation is classified as a **write operation** when it changes the data and is otherwise classified as a **read operation**.
- A consistency model is essentially a contract between processes and the data store. If processes agree to obey certain rules, the store promises to work correctly.
- Following is the general organization of a logical data store, physically distributed and replicated across multiple processes:



Continue…

- A process performs a read operation on a data item, expects the operation to return a value that shows the result of the last write operation on that data.
- No global clock ⇒ difficult to define the last write operation.
- Different consistency models have different restrictions on the values that a read operation can return.



Strict Consistency

Condition:

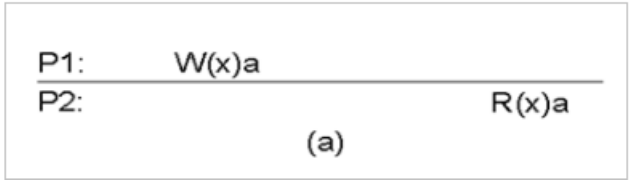
- Any read on a data item x returns a value corresponding to the result of the most recent write on x.

Disadvantage: Assume the existence of absolute global time.

- Behavior of two processes, operating on the same data item.
  1. A strictly consistent store.
  2. A store that is not strictly consistent.

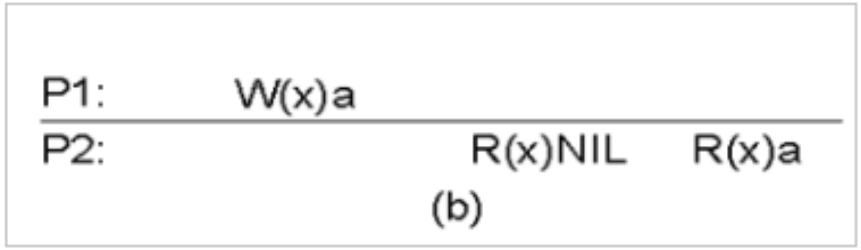
Example of Strict Consistency

- P1 does a write to a data item x, modifying its value to a(previous value of x was NIL).
- In principle, this operation W1(x)a is first performed on a copy of the data store that is local to P1, and is then subsequently propagated to the other copies.
- P2 later reads x from its local copy of the store and sees value a.
- This behavior is correct for a strictly consistent data store.



Continue…

- In contrast, in the case of a data store which is not strictly consistent, P2 does a read operation after the write operation (only a nanosecond after it, but still after it) and gets NIL.
- A subsequent read returns a.



- When a data store is **strictly consistent**, all writes are instantaneously visible to all processes and an absolute global time order is maintained.
- If a data item is changed, all subsequent reads operations performed on that data returns the new value, no matter after the change the reads are done and no matter which process is doing the reading and where they are located.
- Similarly, if a read is done, it gets the then-current value, no matter how quickly the next write is done.

Distribution Protocols

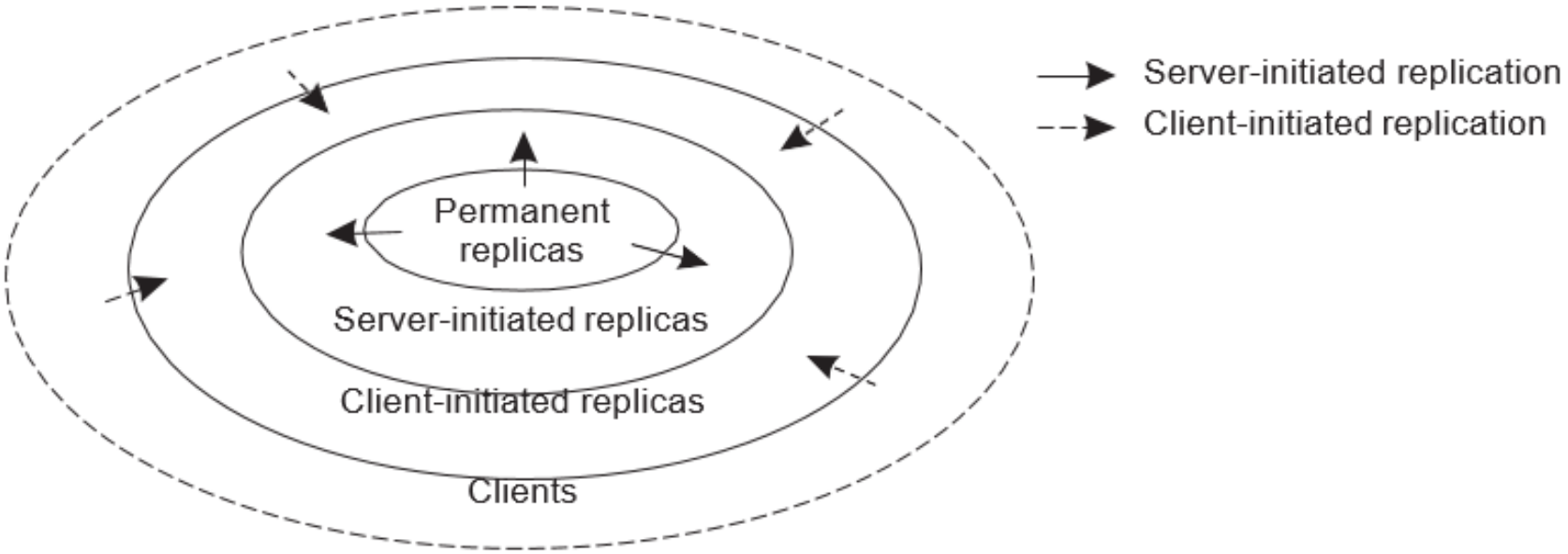
Purpose: Solve the following problem

- What is exactly propagated?
- Where updates are propagated?
- By whom propagation is initiated?

- Distribution Protocols:
  1. Replica Placement
  2. Update Propagation

Replica Placement

- A major design issue for distributed data stores is deciding where, when and by whom copies of the data store are to be placed.
- Three different types of copies can be distinguished, logically organized as shown in the figure.



Permanent Replicas

- Initial set of replicas that constitutes a distributed data store.
- Typically, the number of it is small.

Example - Web site: Distribution of a Web site generally comes in one of the two forms.

- The first kind of distribution is one in which the files of the Web site are replicated across a limited number of servers on a single local-area network. Whenever a request comes in, it is forwarded to one of the servers, using a round-robin strategy.
- The second form of distributed Web site is that of what is called mirroring. A web site is copied to a limited number of servers, called mirror sites, which are geographically spread across the Internet.

Server-Initiated Replicas

- Created at the initiative of the owner of the data store.
- Exist to enhance performance.
- For example, consider a Web server that is placed in New York.
- Normally, this server can handle incoming requests quite easily, but it
- It may happen that over a couple of days a sudden burst of requests come in from an unexpected location far from the server.
- In that case it may be worthwhile to install a number of temporary replicas in region where requests are coming from.
- Such replicas are also known as push caches.

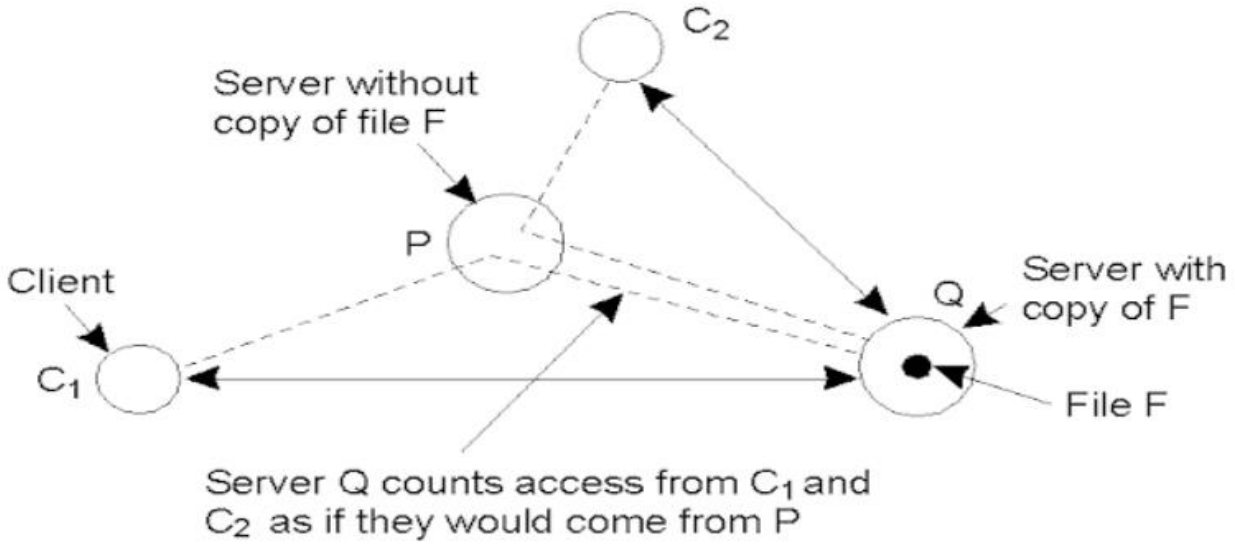
Continue…

How to decide where and when replicas should be created or deleted?

- Each server keeps track of access counts per file, and where access requests come from.
- In particular, it is assumed that given a client C, each server can determine which of the servers in the Web hosting service is closest to C (such information can be obtained from routing database).
- If client C1 and C2 share the same “closest” server P, all access requests for file F at server Q from C1 and C2 are jointly registered at Q as a single access count $cnt_Q(P,F)$.
- If for some server P, $cnt_Q(P,F)$ exceeds more than half of the total requests for F at Q, server P is requested to take over the copy of F. In other word, server Q will attempt to migrate F to P.
- Migration of F to server P may not always succeed, for example, because P is already heavily loaded or is out of disk space. In that case, Q will attempt to replicate F on other servers.

Continue…

- When a server Q decides to reevaluate the placement of the files it stores, it checks the access count for each file.
- If the total number of access requests for F at Q drops below the deletion threshold del (Q,F), it will delete F unless it is the last copy.



Counting access requests from different clients.

- A replication threshold rep(S,F), which is always higher than the deletion threshold, indicates that the number of requests for a specific file is so high that it may be worthwhile replicating it on another server.
- If the number of requests lie somewhere between the deletion and replication threshold, the file is allowed only to be migrated.

Continue…

- Server-initiated replication is gradually increasing in popularity, especially in the context of Web hosting services.
- As long as guarantees can be given that each data item is hosted by at least one server, it may suffice to use only server-initiated replication and not have any permanent replication.
- Permanent replicas are still often useful as a back-up facility or to be used as the only replicas that are allowed to be changed to guarantee consistency.

Continue…

Client-Initiated Replicas:

- Process that can dynamically host a replica on request of a client (client cache).

Update Propagation:

- Propagate only notification/invalidation of update (used for caches).
- Transfer data from one copy to another (distributed databases).
- Propagate the update operation to other copies (active replication).

- **Observation:** No single approach is the best, but depends highly on available bandwidth and read-to-write ratio at replicas.

Thank You ☺