Distributed Systems
Principles and Paradigms

Christoph Dorn

Distributed Systems Group,
Vienna University of Technology

c.dorn@infosys.tuwien.ac.at
http://www.infosys.tuwien.ac.at/staff/dorn

Slides adapted from Maarten van Steen, VU Amsterdam, steen@cs.vu.nl

Chapter 07: Consistency & Replication

Distributed Systems Group

Contents

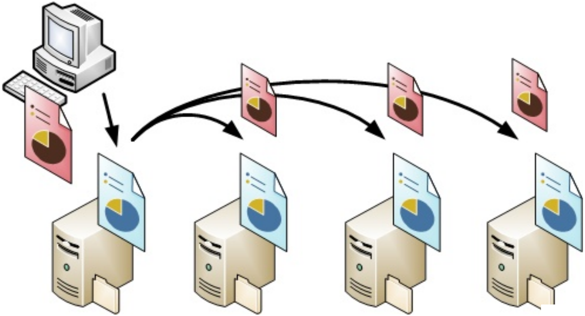
Consistency & replication

- Introduction (what's it all about)
- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols

What is Consistency and Replication?

Some good enough definitions

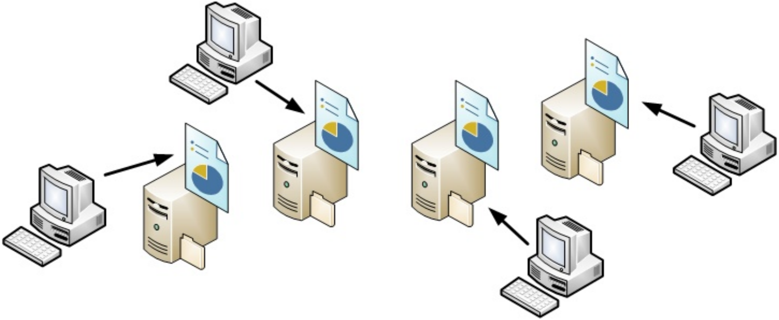
- Replication is the process of maintaining several copies of an data item at different locations.
- Consistency is the process of keeping data item copies the same when changes occur.



Performance vs Scalability Tradeoff

Benefits

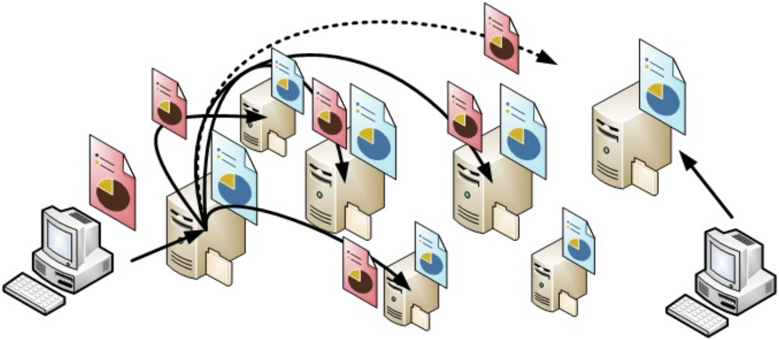
- More replicas can serve more client requests.
- Replicas close to the client improves response time/reduces bandwidth.



Performance vs Scalability Tradeoff

Drawbacks

- Keeping replicas up to date consumes bandwidth
- Updates to replicas are not immediately propagated (stale data)



A cure potentially worse than the disease

Replication - pick any two:

- Performance: low response time (for reading and writing)
- Scalability: support a lot of clients
- Consistency: any update should be reflected at all replicas else before any subsequent operation takes place

Synchronous Replication issue

Updates performed as a single atomic operation (transaction) requires agreement of all replicas when to perform the update. Becomes extremely costly very quickly.

Mitigation

Avoid (instantaneous) global synchronization

Maintaining Performance and Scalability

Main issue

To keep replicas consistent, we generally need to ensure that all conflicting operations are done in the the same order everywhere

Conflicting operations

From the world of transactions:

- Read–write conflict: a read operation and a write operation act concurrently
- Write–write conflict: two concurrent write operations

Issue

Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability
Solution: weaken consistency requirements so that hopefully global synchronization can be avoided

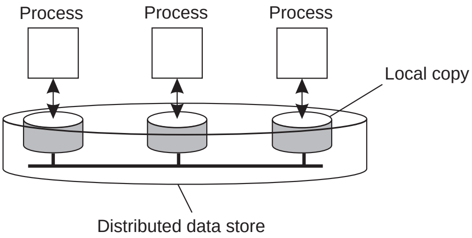
Data-centric consistency models

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.

Essential

A data store is a distributed collection of storages:



Continuous Consistency

Observation

We can actually talk a about a degree of consistency:

- replicas may differ in their numerical value
- replicas may differ in their relative staleness
- there may be differences with respect to (number and order) of performed update operations

Conit

Consistency unit ⇒ specifies the data unit over which consistency is to be measured.

Conit examples

webpage, table entry, entire table in DB, ...

Continuous Consistency Example

Conit Example

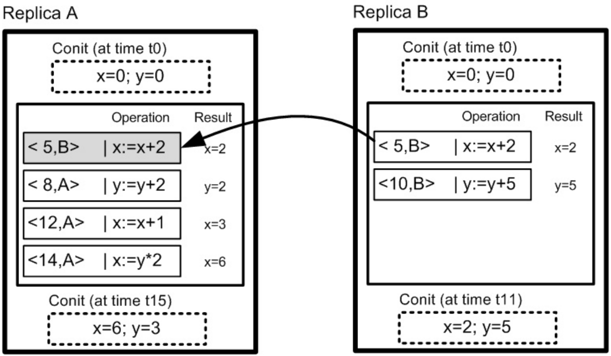
Consistency unit in our example is the price of a particular stock.

Example constraints

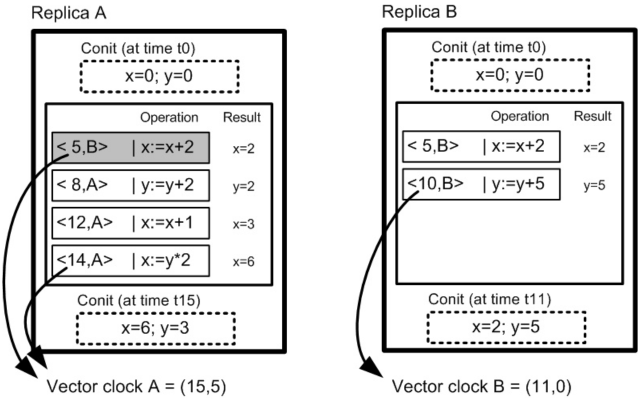
Specify degree of consistency for stock price:

- Local value may differ in numerical value from other replica by 10 cents
- Local value needs to be checked for staleness at least every 10 seconds
- There may be no more than 3 unseen performed update operations

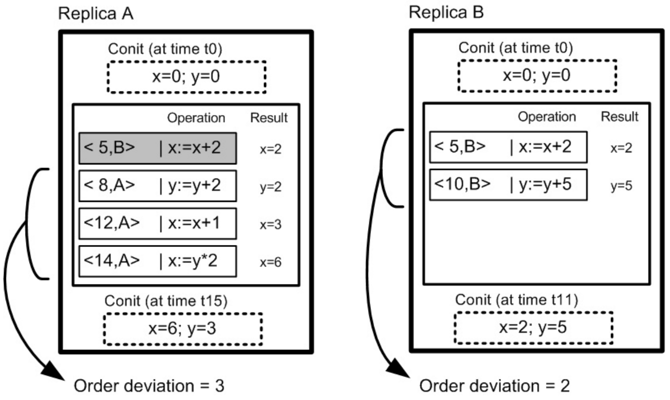
Example: Conit



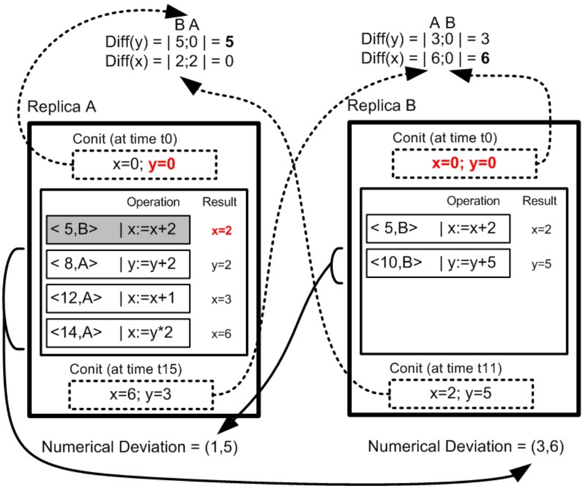
Example: Conit



Example: Conit



Example: Conit



Ordering of Operations

Desired Behavior

read returns result of most recent write

No global clock!

What is the most recent (last) write?

Relax timing

- consider intervals of R/W operations
- define precisely what are acceptable behavior for conflicting operations
- replicas need to agree on consistent global ordering of updates

Sequential consistency

Definition

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.

P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)b	R(x)a

(a)

P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)a	R(x)b

(b)

(a) sequentially consistent, (b) not consistent

Sequential consistency

Example

- Assume x is a shared social network timeline
- Peter posts: I'm going skiing, who's in?
- Paul posts: I'm going hiking, who's in?
- Petra reads: first Paul's, then Peter's post
- Pam reads: first Paul's, then Peter's post

Beware

To be sequentially consistent: every reader of the timeline needs to receive the updates in exactly the same order.

Sequential consistency

Example

- Assume x is a shared social network timeline
- Peter posts: I'm going skiing, who's in?
- Paul posts: I'm going hiking, who's in?
- Petra reads: first Paul's, then Peter's post
- Pam reads: first Paul's, then Peter's post

Beware

To be sequentially consistent: every reader of the timeline needs to receive the updates in exactly the same order.

Definition

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 by different processes.

P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

(a) causally consistent

P1:	W(x)a		W(x)c			
P2:		R(x)a		W(x)b		
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

(b) causally consistent

Causal consistency - more examples

P1: W(x)a				
P2:	R(x)a	W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(a)

P1: W(x)a				
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(b)

(a) causally inconsistent, (b) consistent

Causal consistency

Example

- Again, assume x is a shared social network timeline
- Peter posts: I had a car accident!
- Peter posts: But I'm ok!
- Paul read this and posts: Happy for you!
- Petra reads: Peter's first post, the Paul's, the Peter's second
- Pam reads: both Peter's, then Paul's post

Beware

How to determine causally related writes?

Causal consistency

Example

- Again, assume x is a shared social network timeline
- Peter posts: I had a car accident!
- Peter posts: But I'm ok!
- Paul read this and posts: Happy for you!
- Petra reads: Peter's first post, the Paul's, the Peter's second
- Pam reads: both Peter's, then Paul's post

Beware

How to determine causally related writes?

FIFO consistency

Definition

- 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.

P1:	W(x)a					
P2:	R(x)a	W(x)b	W(x)c			
P3:				R(x)b	R(x)a	R(x)c
P4:				R(x)a	R(x)b	R(x)c

Valid sequence of event of FIFO consistency

$R(x)b \rightarrow R(x)c$ only

FIFO consistency

Implications

- Easy to implement: Writes from different processes are always assumed ”concurrent”.
- Different processes may see the statements executed in different order
- Some results may be counterintuitive

Example

- Process P1: x := 1; if (y==0) kill (P2);
- Process P2: y := 1; if (x==0) kill (P1);

Effect

Two concurrent processes, both may be killed with FIFO (but not with sequential consistency).

Frequent questions to the audience

Get ready!

From time to time I will do Simultaneous Voting to check whether the presented concepts are clear to everyone.

Will you attend the lecture next Monday?

Your choices are:

- Sure, I just love Distributed Systems! (head)
- Not sure yet, do I really need to? (ear)
- No way, Garfield is my second name! (nose)

Know your Consistency Models

Question to the audience

Observe following read/write events. The sequence is only valid for one of the following consistency models, which one?

Your choices are:

- Sequential Consistency (head)
- Causal Consistency (ear)
- FIFO Consistency (nose)

P1:	W(x)a			W(x)c				
P2:		R(x)a	W(x)b					
P3:					R(x)a		R(x)c	R(x)b
P4:					R(x)c	R(x)a		R(x)b

Answer: causal consistency

Know your Consistency Models

Question to the audience

Observe following read/write events. The sequence is only valid for one of the following consistency models, which one?

Your choices are:

- Sequential Consistency (head)
- Causal Consistency (ear)
- FIFO Consistency (nose)

P1:	W(x)a			W(x)c				
P2:		R(x)a	W(x)b					
P3:					R(x)a		R(x)c	R(x)b
P4:					R(x)c	R(x)a		R(x)b

Answer: causal consistency

Grouping operations

Definition

- Accesses to synchronization variables are sequentially consistent.
- No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere.
- No data access is allowed to be performed until all previous accesses to synchronization variables have been performed.

Grouping operations

Definition

- Everyone has exactly the same view on a lock (a synchronization variable)
- Have a lock: Cannot unlock until data value is synchronized everywhere
- Grab a lock: then only allowed to proceed when everyone has the same view on the lock

Basic idea

You don't care that reads and writes of a series of operations are immediately known to other processes. You just want the effect of the series itself to be known.

Grouping operations

Definition

- Everyone has exactly the same view on a lock (a synchronization variable)
- Have a lock: Cannot unlock until data value is synchronized everywhere
- Grab a lock: then only allowed to proceed when everyone has the same view on the lock

Basic idea

You don't care that reads and writes of a series of operations are immediately known to other processes. You just want the effect of the series itself to be known.

Grouping operations

P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)

P2: Acq(Lx) R(x)a R(y) NIL

P3: Acq(Ly) R(y)b

Observation

Weak consistency implies that we need to lock and unlock data (implicitly or not).

Question

Why do we need a lock here?

The underlying distributed system might decide to push updates to all replicas after lock release OR not until a new lock is acquired.

Grouping operations

P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)

P2: Acq(Lx) R(x)a R(y) NIL

P3: Acq(Ly) R(y)b

Observation

Weak consistency implies that we need to lock and unlock data (implicitly or not).

Question

Why do we need a lock here?

The underlying distributed system might decide to push updates to all replicas after lock release OR not until a new lock is acquired.

Avoiding Data-centric Consistency

Concurrent processes

- So far required simultaneous updates of shared data
- Consistency and Isolation have to be maintained
- Synchronization required

Lack of concurrent processes

- Lack of simultaneous updates (or distinct update regions)
- Easy resolved or acceptable inconsistencies
- Focus on guarantees for a single (mobile) client (but not for concurrent access)!

Client-centric consistency models

Overview

- System model
- Monotonic reads
- Monotonic writes
- Read-your-writes
- Write-follows-reads

Goal

Show how we can perhaps avoid systemwide consistency, by concentrating on what specific clients want, instead of what should be maintained by servers.

Consistency for mobile users

Example

Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database.

- At location *A* you access the database doing reads and updates.
- At location *B* you continue your work, but unless you access the same server as the one at location *A*, you may detect inconsistencies:
  - your updates at *A* may not have yet been propagated to *B*
  - you may be reading newer entries than the ones available at *A*
  - your updates at *B* may eventually conflict with those at *A*

Consistency for mobile users

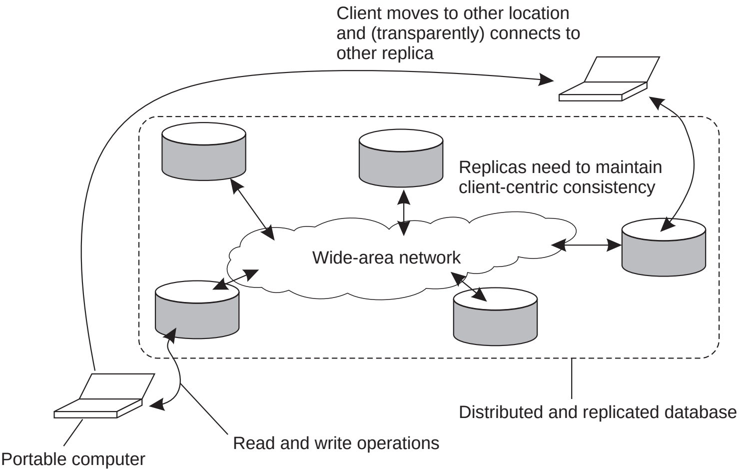
Note

The only thing you really want is that the entries you updated and/or read at *A*, are in *B* the way you left them in *A*. In that case, the database will appear to be consistent to you.

Eventual Consistency

- Update is performed at one replica (at a time)
- Propagation to other replicas is performed in a lazy fashion
- Eventually, all replicas will be updated
- I.e., replicas gradually become consistent if no update takes place for a while

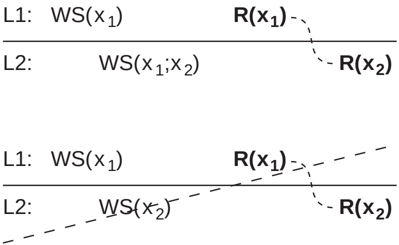
Basic architecture



Monotonic reads

Definition

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.



Client-centric consistency: notation

Notation

- $WS(x_i[t])$ is the set of write operations (at L_i) that lead to version x_i of x (at time t)
- $WS(x_i[t_1]; x_j[t_2])$ indicates that it is known that $WS(x_i[t_1])$ is part of $WS(x_j[t_2])$.
- Note: Parameter t is omitted from figures.

Monotonic reads

Example

Automatically reading your personal calendar updates from different servers. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

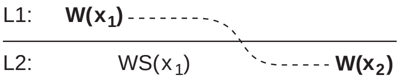
Example

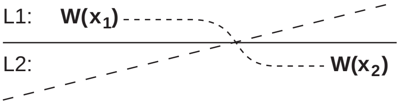
Reading (not modifying) incoming mail while you are on the move. Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.

Monotonic writes

Definition

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





Monotonic writes

Example

Updating a program at server S_2, and ensuring that all components on which compilation and linking depends, are also placed at S_2.

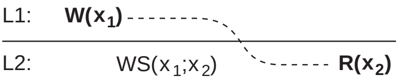
Example

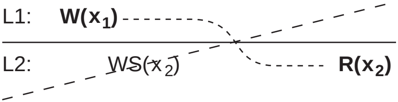
Maintaining versions of replicated files in the correct order everywhere (propagate the previous version to the server where the newest version is installed).

Read your writes

Definition

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.





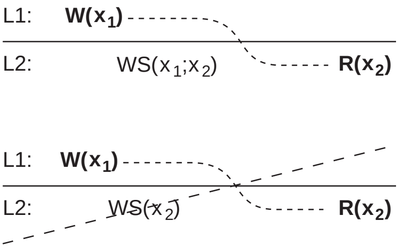
Example

Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

Read your writes

Definition

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.



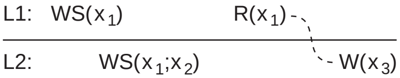
Example

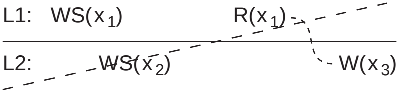
Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

Writes follow reads

Definition

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.





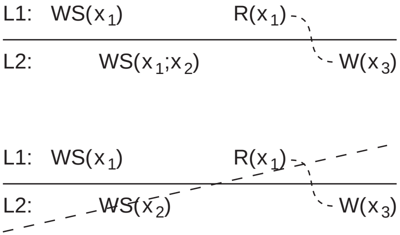
Example

See reactions to posted articles only if you have the original posting (a read "pulls in" the corresponding write operation).

Writes follow reads

Definition

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.



Example

See reactions to posted articles only if you have the original posting (a read “pulls in” the corresponding write operation).

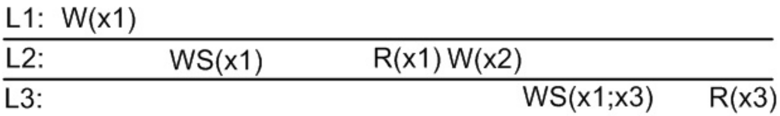
Know your Consistency Models

Question to the audience

Observe following read/write set. What changes are required to make this sequence correct with regard to Read your writes?

Your choices are:

- Nothing, it's correct (head)
- Add as first event for L3: R(x2) (ear)
- Replace L3: WS(x1;x3) with WS(x1;x2;x3) (nose)



Know your Consistency Models

Answer

Replace L3: WS(x1;x3) with WS(x1;x2;x3)

L1:	W(x1)				
L2:		WS(x1)	R(x1) W(x2)		
L3:				WS(x1;x2;x3)	R(x3)

Replica Consistency Concerns

From consistency models to management

- Replicas need to be kept consistent according to some model
- No update $\rightarrow$ no problem
- If access-to-update ratio is high, replication will help
- If updates-to-access ratio is high, updates will not be consumed
- Ideally, update only replicas that are going to be accessed
- In general, try to keep replicas in proximity to clients

Replica Management

Challenges

- Replica server placement
  - often a management or commercial issue
- Content replication and placement
- Content distribution
  - state vs. operation
  - push vs. pull vs. lease
  - blocking vs. non-blocking (eager vs lazy)
  - unicast vs multicast (group communication)

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N - K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N - K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N - K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N - K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

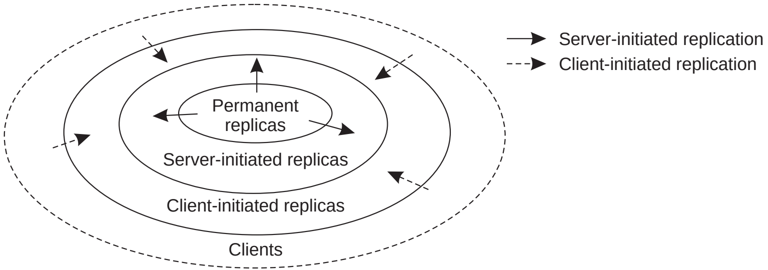
Content replication

Distinguish different processes

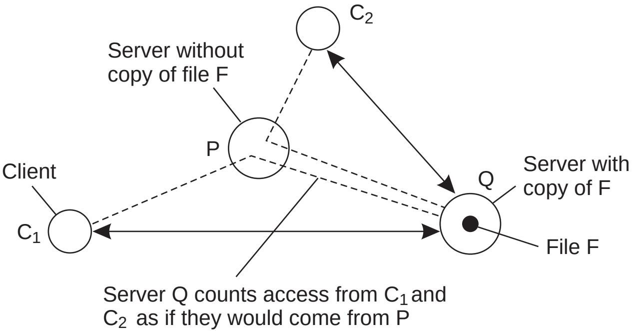
A process is capable of hosting a replica of an object or data:

- Permanent replicas: Process/machine always having a replica (i.e. origin server)
  - initial set (small)
  - LAN; e.g., Web server cluster or database cluster
  - geographically; e.g., Web mirror or federated database
- Server-initiated replica: Process that can dynamically host a replica on request of another server in the data store
  - performance, e.g., push cache or Web hosting service
  - reduce server load and replicate to server placed in the proximity of requesting clients
- Client-initiated replica: Process that can dynamically host a replica on request of a client (client cache)

Content replication



Server-initiated replicas



- Keep track of access counts per file, aggregated by considering server closest to requesting clients
- Number of accesses drops below threshold $D \Rightarrow$ drop file
- Number of accesses exceeds threshold $R \Rightarrow$ replicate file
- Number of access between D and $R \Rightarrow$ migrate file

Content distribution

Model

Consider only a client-server combination:

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

Note

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

Content distribution

Push (server)-based protocols

- Updates are propagated to other replicas without those replicas asking for updates
- Used by permanent and server-initiated replicas, but also by some client caches
- High degree of consistency (consistent data can be made available faster)
- If server keeps track of clients that have cached the data, we have a stateful server: limited scalability and less fault tolerant
- Often, multicasting is more efficient

Content distribution

Pull (client)-based protocols

- A replica requests another replica to send it any updates it has at the moment
- Often used by client caches
- I.e. client polls server if updates are available
- E.g. Web modified since
- Response time increases in case of a cache miss
- Unicasting instead of multicasting

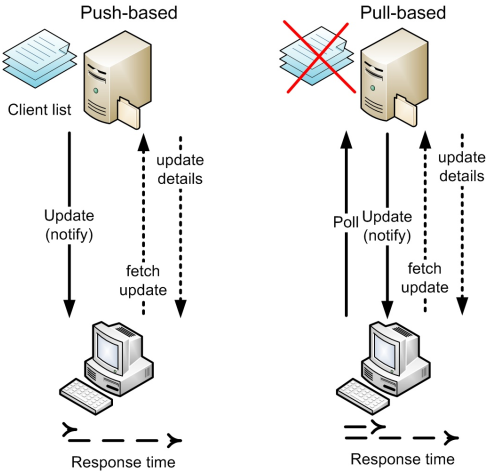
Content distribution: client/server system

- Pushing updates: server-initiated approach, in which update is propagated regardless whether target asked for it.
- Pulling updates: client-initiated approach, in which client requests to be updated.

Issue	Push-based	Pull-based
1:	List of client caches	None
2:	Update (and possibly fetch update)	Poll and update
3:	Immediate (or fetch-update time)	Fetch-update time
1: State at server *2: Messages to be exchanged* *3: Response time at the client*		

Content distribution: client/server system

Push-based vs Pull-based Updates



Observation

We can dynamically switch between pulling and pushing using leases: A contract in which the server promises to push updates to the client until the lease expires.

Content distribution

Issue

Make lease expiration time dependent on system's behavior (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

Question

Why are we doing all this?

Trying to reduce the server's state as much as possible while providing strong consistency.

Content distribution

Issue

Make lease expiration time dependent on system's behavior (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

Question

Why are we doing all this?

Trying to reduce the server's state as much as possible while providing strong consistency.

Content distribution

Issue

Make lease expiration time dependent on system's behavior (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

Question

Why are we doing all this?

Trying to reduce the server's state as much as possible while providing strong consistency.

Issue

Make lease expiration time dependent on system's behavior (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

Question

Why are we doing all this?

Trying to reduce the server's state as much as possible while providing strong consistency.

Content distribution

Issue

Make lease expiration time dependent on system's behavior (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

Question

Why are we doing all this?

Trying to reduce the server's state as much as possible while providing strong consistency.

Content distribution

Issue

Make lease expiration time dependent on system's behavior (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

Question

Why are we doing all this?

Trying to reduce the server's state as much as possible while providing strong consistency.

Blocking vs Non-Blocking

When are (push) updates propagated?

- Synchronous (blocking, eager):
  All replicas are updated immediately, then reply to client (that issued the update)
- Asynchronous (non-blocking, lazy):
  Update is applied to one copy, then reply to client, propagation to other replicas afterwards

Consistency protocols

Consistency protocol

Describes the implementation of a specific consistency model.

- Continuous consistency
- Primary-based protocols
- Replicated-write protocols

Continuous consistency: Numerical errors

Principal operation

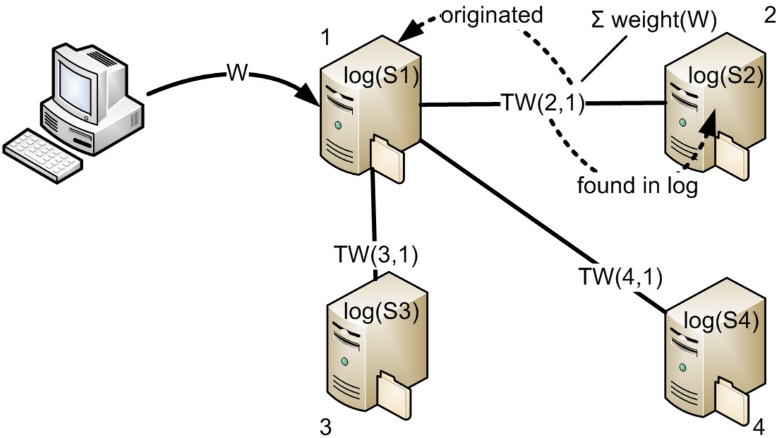
- Every server S_i has a log, denoted as $log(S_i)$.
- Consider a data item x and let $weight(W)$ denote the numerical change in its value after a write operation W. Assume that

$$\forall W : weight(W) > 0$$

- W is initially forwarded to one of the N replicas, denoted as $origin(W)$. $TW[i,j]$ are the writes executed by server S_i that originated from S_j:

$$TW[i,j] = \sum\{weight(W) | origin(W) = S_j \; \& \; W \in log(S_i)\}$$

Continuous consistency: Numerical errors



Continuous consistency: Numerical errors

Note

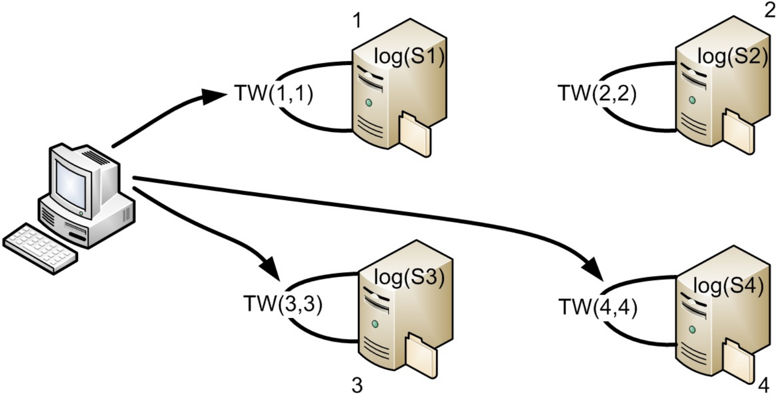
Actual value $v(t)$ of x:

$$v(t) = v_{init} + \sum_{k=1}^{N} TW[k,k]$$

value v_i of x at replica i:

$$v_i = v_{init} + \sum_{k=1}^{N} TW[i,k]$$

Continuous consistency: Numerical errors



Continuous consistency: Numerical errors

Note

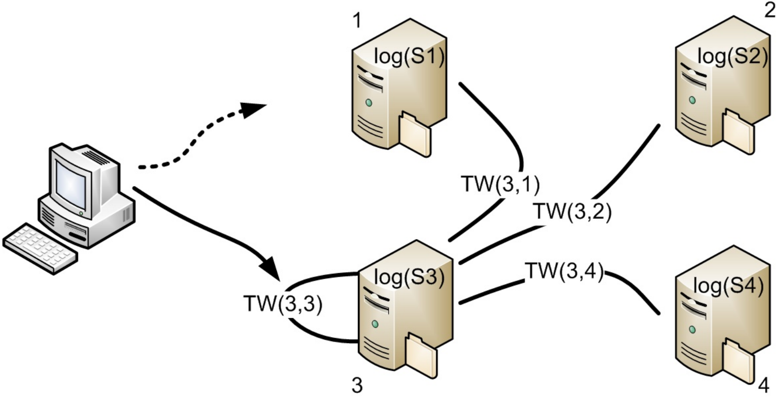
Actual value $v(t)$ of x:

$$v(t) = v_{init} + \sum_{k=1}^{N} TW[k,k]$$

value v_i of x at replica i:

$$v_i = v_{init} + \sum_{k=1}^{N} TW[i,k]$$

Continuous consistency: Numerical errors



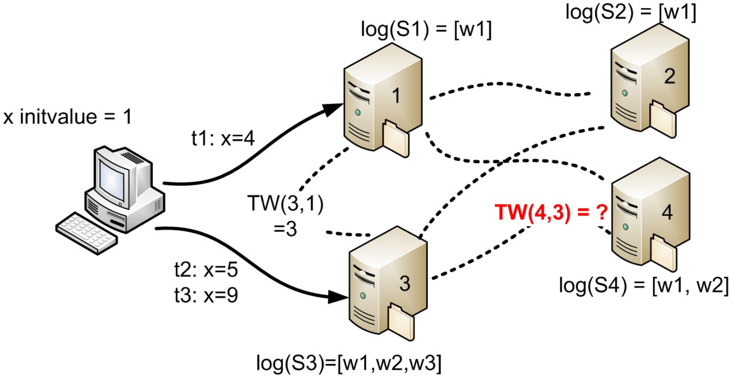
Know your Consistency Models

Question to the audience

What is the value of TW(4,3)?

Your choices are:

1 (head) ; 4 (ear); 8 (nose); Answer = 1



Continuous consistency: Numerical errors

Problem

We need to ensure that $v(t) - v_i < \delta_i$ for every server S_i.

Approach

Let every server S_k maintain a view $TW_k[i,j]$ of what it believes is the value of $TW[i,j]$. This information can be gossiped when an update is propagated.

Note

$$0 \leq TW_k[i,j] \leq TW[i,j] \leq TW[j,j]$$

Continuous consistency: Numerical errors

Problem

We need to ensure that $v(t) - v_i < \delta_i$ for every server S_i.

Approach

Let every server S_k maintain a view $TW_k[i,j]$ of what it believes is the value of $TW[i,j]$. This information can be gossiped when an update is propagated.

Note

$$0 \leq TW_k[i,j] \leq TW[i,j] \leq TW[j,j]$$

Continuous consistency: Numerical errors

Problem

We need to ensure that $v(t) - v_i < \delta_i$ for every server S_i.

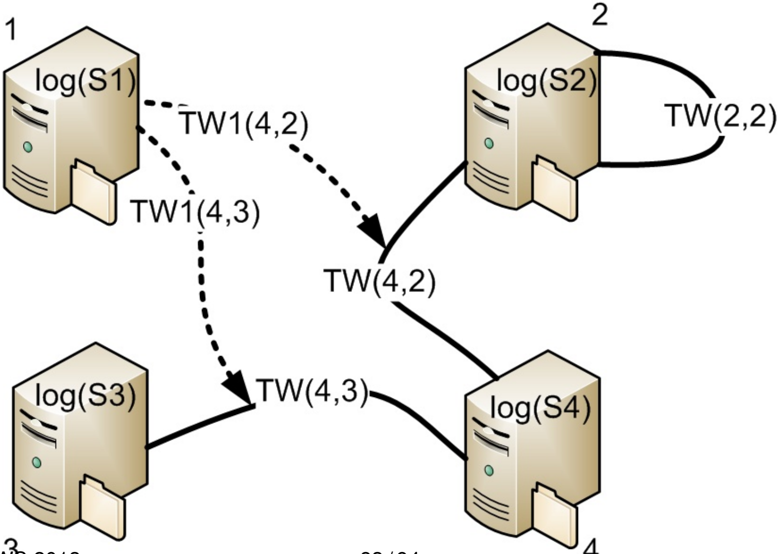
Approach

Let every server S_k maintain a view $TW_k[i,j]$ of what it believes is the value of $TW[i,j]$. This information can be gossiped when an update is propagated.

Note

$$0 \leq TW_k[i,j] \leq TW[i,j] \leq TW[j,j]$$

Continuous consistency: Numerical errors



Continuous consistency: Numerical errors

Solution

S_k sends operations from its log to S_i when it sees that $TW_k[i,k]$ is getting too far from $TW[k,k]$, in particular, when

$$TW[k,k] - TW_k[i,k] > \delta_i/(N-1)$$

Question

To what extent are we being pessimistic here: where does $\delta_i/(N-1)$ come from?

Note

Staleness can be done analogously, by essentially keeping track of what has been seen last from S_i (see book)

Continuous consistency: Numerical errors

Solution

S_k sends operations from its log to S_i when it sees that $TW_k[i,k]$ is getting too far from $TW[k,k]$, in particular, when

$$TW[k,k] - TW_k[i,k] > \delta_i/(N-1)$$

Question

To what extent are we being pessimistic here: where does $\delta_i/(N-1)$ come from?

Note

Staleness can be done analogously, by essentially keeping track of what has been seen last from S_i (see book)

Continuous consistency: Numerical errors

Solution

S_k sends operations from its log to S_i when it sees that $TW_k[i,k]$ is getting too far from $TW[k,k]$, in particular, when

$$TW[k,k] - TW_k[i,k] > \delta_i/(N-1)$$

Question

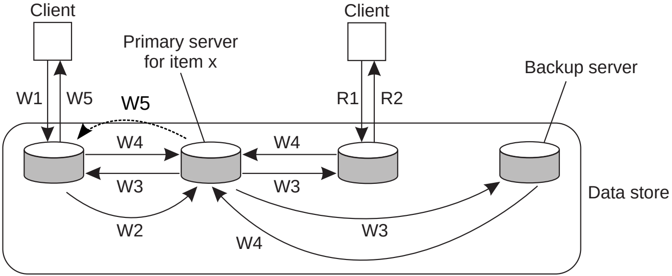
To what extent are we being pessimistic here: where does $\delta_i/(N-1)$ come from?

Note

Staleness can be done analogously, by essentially keeping track of what has been seen last from S_i (see book)

Primary-based protocols

Primary-backup protocol



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

Primary-based Synchronous Replication

Advantages

- No inconsistencies (identical copies)
- Reading the local copy yields the most up-to-date value
- Changes are atomic

Disadvantages

A write operation has to update all sites

- slow
- not resilient against network or node failure

Primary-based Asynchronous Replication

Advantages

- Fast, since only primary replica is updated immediately
- Resilient against node and link failure

Disadvantages

- Data inconsistencies can occur
- a local read does not always return the most up-to-date value

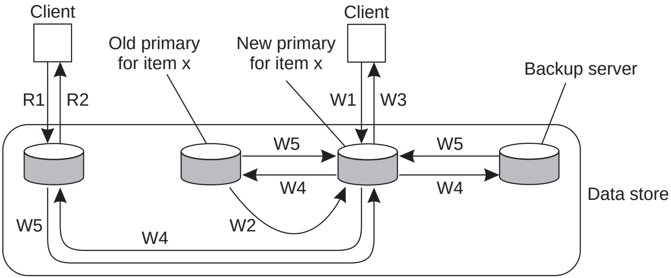
Primary-based protocols

Example primary-backup protocol

Traditionally applied in distributed databases and file systems that require a high degree of fault tolerance. Replicas are often placed on same LAN.

Primary-based protocols

Primary-backup protocol with local writes



W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

Primary-based (passive) Replication

Advantages

- At least one node exists which has all updates
- ordering guarantees are relatively easy to achieve (no inter-site synchronization necessary)

Disadvantages

- Primary is bottleneck and single point of failure
- High reconfiguration costs when primary fails

Primary-based protocols

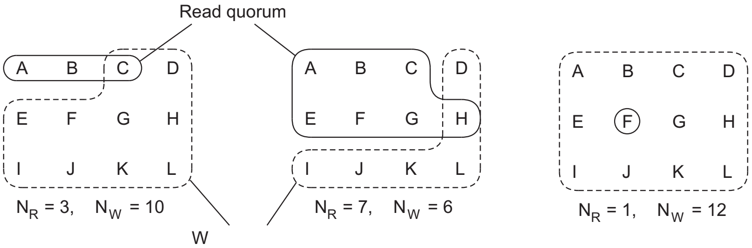
Example primary-backup protocol with local writes

Mobile computing in disconnected mode (ship all relevant files to user before disconnecting, and update later on).

Replicated-write protocols

Quorum-based protocols

Ensure that each operation is carried out in such a way that a majority vote is established: distinguish read quorum and write quorum:



required: $N_R + N_W > N$ and $N_W > N/2$