Sistemas de Computação Móvel e Ubíqua 2018/2019

DATA DISSEMINATION AND MANAGEMENT

Carmen Morgado, Hervé Paulino, Nuno Preguiça e Sérgio Duarte

Data Dissemination: Interaction Models

On-demand (pull) - Request-Reply
- Synchronous
- Pooling
  - Consumes energy
  - Burdens servers with update requests when there may be nothing to update
- One-to-one communication
- Not scalable

Push - Publish-subscribe
- Asynchronous: decouples clients from servers in time
- Information broadcasting when it is available
- One-to-many communication
- Resource efficiency
- Scalable

Data Dissemination: Interaction Models

In the *on-demand mode (also known as pull mode), the information* sink send an explicit query every time we need particular information to an information source (a server or a peer).

The information source sends back the latest version of the requested (or queried) information.

The access latency in the *on-demand* mode therefore is at least the round-trip latency. In addition, the sender has to expend resources to send the query every time it needs the information.

The *publish-subscribe mode (also known as push mode) is used for obtaining information* whenever it is available. For example, we can subscribe to a stock ticker, and whenever the stock information is updated, it will be sent to us.

Publish-subscribe mode

In mobile computing, the **publish-subscribe** mode is combined with the broadcast nature of wireless communication to provide resource-efficient and scalable information delivery.

Wireless transmission is characterized by problems as signal fading, path loss, interference, and time dispersion.

These lead to high error rates and signal distortion. A consequence is that wireless links have a lower bandwidth and a higher communication latency.

Providing information services to mobile clients is an important application of **publish-subscribe** mode of data dissemination.

Examples of information services include financial information services, airport information services, and emergency services for traffic information.

Publish-subscribe in Mobile

Publication Content: which items to publish?

Publication Frequency: when to publish? (How often?)

Bandwidth Allocation: How to distribute the spectrum to allow for both pull and push communication?

- How to adaptively allocate bandwidth between uplink and downlink channels to reflect the changing usage patterns in a cell?

Transparent use: How can mobile users access services transparently?

- A simple way to accomplish this is to provide the user with a directory channel that has information about all the broadcast channels available in the new cell.

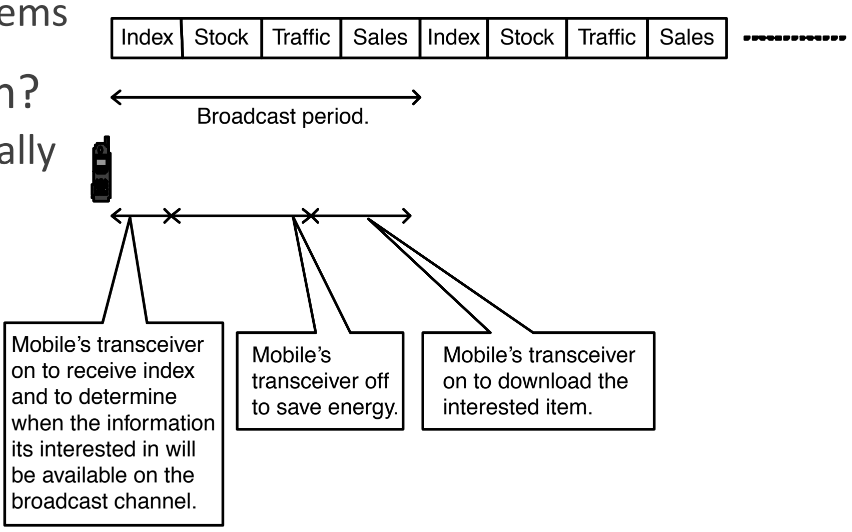
Publish-subscribe in Mobile

Which items to publish?

- Give priority to hot items

How often to publish?

- Disseminate periodically
  - With an index
- Different periods for different items


Index
Stock
Traffic
Sales
Index
Stock
Traffic
Sales
Broadcast period.
Mobile's transceiver on to receive index and to determine when the information its interested in will be available on the broadcast channel.
Mobile's transceiver off to save energy.
Mobile's transceiver on to download the interested item.

Taken from: Frank Adelstein, et. al. Fundamentals of Mobile and Pervasive Computing. McGraw-Hill. 2005

Publish-subscribe in Mobile

Broadcast scheduling: deals with determining how often to publish a certain data item.

View the broadcast channel as an extension to the memory hierarchy of the mobile node.

Channel structured as multiple virtual disks in the air, each spinning at different rates.

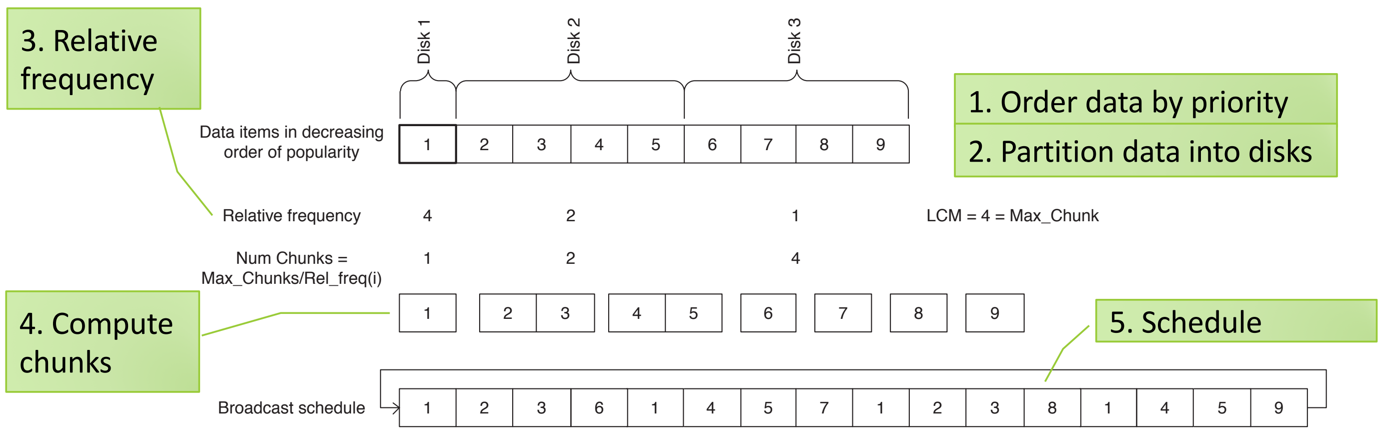
The hottest data is "allocated " to the fastest-spinning disk, the next hottest to the next fastest disk, and so on …

Publish-subscribe in Mobile

Broadcast Disks

- Virtual extension of the device's memory hierarchy
- Several disks, with different speeds, may be used

AAFZ algorithm for deriving the schedule for the broadcast channel



Publish-subscribe in Mobile

Channel allocation

Not all data can be provided on the broadcast channel.

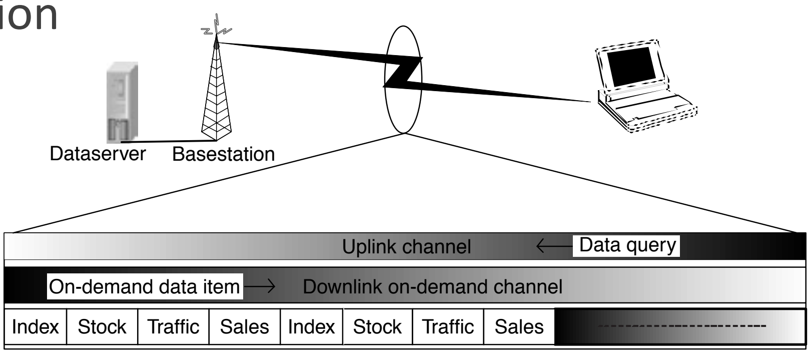
Wireless bandwidth bandwidth is divided into three logical channels:

- Uplink request channel – shared by all the mobile clients
- On-demand download channel – for the server to send on-demand request data items
- Broadcast download channel – broadcast hottest data items

Publish-Subscribe in Mobile

Bandwidth allocation

- On-demand
  - Request (uplink)
  - Reply (downlink)
- Broadcast



There are algorithms do decide how much bandwidth to allocate to which channel.

Data Dissemination: Bandwidth Allocation

Bandwidth Allocation

- Bandwidth for on-demand channel – B_0
- Bandwidth for broadcast channel – B_b
- Available bandwidth $B = B_0 + B_b$

Data Server

- N data items: D1, D2, …, Dn
  - D1 – the most popular data items, with popularity ratio p1 (between 0 and 1)
  - D2 – the next popular data item with popularity ratio p2 (between 0 and 1)
- Size for each data item – S
- Size of each data query - R

Each mobile node generates requests at an average rate of r

Data Dissemination: Allocate All Bandwidth

Compute Average Access Time T over all data items

- T = Tb + To
- Tb – average access time to access a data item from the broadcast channel
- To – average access time to access an on demand item

Data Dissemination: Allocate All Bandwidth

The average time to service an on-demand request

- $(S + R)/B_0$

If all data items are provided only on-demand, the average rate for all the on-demand items will be:

$M * r$ – queuing generation rate

- M – the number of mobile nodes in the wireless cell
- r – average request rate of a mobile node

Data Dissemination: Allocate All Bandwidth

Applying Queuing Theory to Analyze the Problem

- As the number of mobile users ↑ (increases),
  - the average queuing generation rate $(M * r)$ ↑
- As $(M * r)$ approaches → the service rate [B0/(S+R)],
  - -> the average service time (including queuing delay) ↑ rapidly
- What is acceptable server time threshold?
- Allocating all the bandwidth to the on-demand channels → Poor Scalability

Data Dissemination
Allocate All Bandwidth for Broadcast Channel

If all the data items are published on the broadcast channel with the same frequency (ignoring the popularity ratio)

Average waiting:

- n/2 data items before getting the data items from the broadcast channel

Average access time for a data item:

- $(n/2) \times (S/B_b)$
- Independent of number of mobile nodes in the cell
- Average access time proportional to the number of data items <u>n</u>
- Average access time ↑ as the number of data items to broadcast ↑

Data Dissemination
Bandwidth Allocation – A Simple Case

Two data items: D1 and D2

p1 of D1 >> p2 of D2 (D1 is much more popular than D2)

- Broadcast D1 all the time → cause D2 access time to be infinite (D2 is never available)

Broadcast frequency calculation to achieve **minimum average access time**

- $f1 = \sqrt{p1}/(\sqrt{p1} + \sqrt{p2})$
- $f2 = \sqrt{p2}/(\sqrt{p1} + \sqrt{p2})$
- An example: p1 = 0.9, p2 = 0.1
  - sqrt(0.9) = 0.9487, sqrt(0.1) = 0.3162
  - f1 = 0.75
  - f2 = 0.25
  - D1 broadcast 3-times more often than D2, even D1 is 9-times more popular than D2

Data Management - Motivation

Mobile devices tend to have (when compared with stationary computers):

- Limited bandwidth
- High latency
- Variable network conditions
- Periods of disconnection

Challenges for data management:

- Provide high availability (despite periods of disconnection)
- Provide responsive data access

Data Management - Challenges

Weak connectivity of clients in a mobile wireless computing platform creates a new challenge for data management

- how to ensure high data availability in mobile computing environments where frequent disconnections may occur because the clients and server may be weakly connected. Ex. distributed file system Coda uses data caching to improve availability at the expense of transparency.

Severe constraints on the availability of resources (such as battery power and processing capabilities)

- This characteristic leads to another challenge for data management in mobile computing environments, specifically, how to minimize resource consumption (e.g., energy and bandwidth) for data management while ensuring a desirable level of data consistency.

Data Management - Challenges

Asymmetric communication links

- For example, in architecture-based wireless networks, the downstream (base station to mobile nodes) communication link capacity is usually much higher than the upstream (mobile to base station) capacity. To make matters worse, mobile nodes may have to compete with several other mobile nodes to get access to an upstream channel (using some medium access techniques, such as ALOHA).

Data may be location and time (context) dependent

- For example, a mobile user may query various databases periodically to retrieve both location dependent and time-dependent information.

Context-dependent data impose another problem for cache management algorithms

- The decision to cache or replace a data item now also depends on the context (e.g., location) of the mobile node, in addition to the temporal or spatial locality in the reference pattern.

Disconnected operation

Disconnection

- Expected
  - Due to communication costs, energy saving, location changes, etc.
- Unexpected
  - Due to network unavailability, network congestion, server failures, etc.

Disconnected operation consists in allowing applications to continue operating during periods of disconnection

- Necessary to prepare for disconnection while connected

Ultimate goal: hide disconnection from users

- Can be provided by applications or by the system.

Caching and Replication

Key technique to improve performance and availability (and provide support for disconnected operation).

Caching (and replication) consist in creating and maintaining copies of data used by applications.

The goal is to attempt to guarantee that most data requests are serviced from particular data replicas. What for?

- Providing availability even for disconnected nodes.
- Improving performance by servicing requests closer to the clients.

Caching vs. Replication

Some systems have two levels of replication.

First-class replicas: **replica**

- Data replicas are long-lived
- The system actively maintains data replicas

Second-class replicas: **cache**

- Data replicas are short-lived – created and destroyed when necessary

Caching in Mobile Computing: problems

How to reduce client-side latency.

How to maintain cache consistency between various caches and the server.

How to ensure high data availability in the presence of frequent disconnections.

How to achieve high energy/bandwidth efficiency

How to determine the cost of a cache miss and how to incorporate this cost in the cache management scheme.

How to manage location-dependent data in the cache.

How to enable cooperation between multiple peer caches. (P2P)

Caching in Mobile Computing

There are many challenges, we highlight:

- Where to cache?
- What to cache (when do we cache a data item and for how long)?
- How to invalidate cache items?

Caching in Mobile Computing: Where to cache?

Server
- Cache for popular items → minimize cost of retrieving the same information multiple times

Device
- Cache data to improve performance and availability
- Provide support for disconnected operation

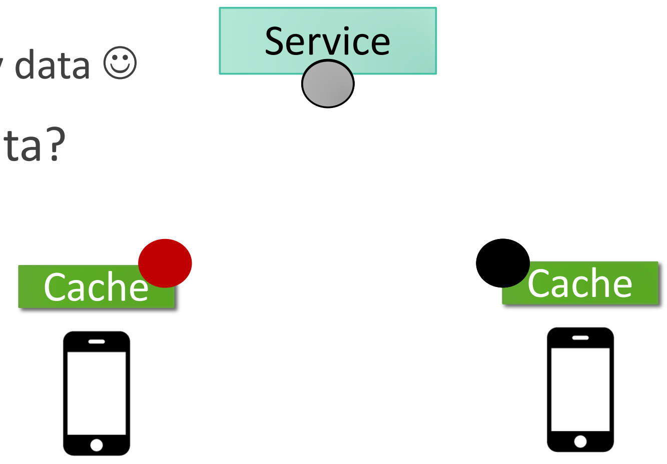
Proxy
- Cache data to improve performance and availability
- Remove storage burden from device

Device vs Proxy
- Access pattern, communication cost, update rate, available resources at the device

Caching in Mobile Computing: Maintenance

Cache contents must be kept synchronized with official versions (maintained in servers or other mobile computers)
- Not a problem for read-only data ☺

What about read-write data?



Caching in Mobile Computing: Maintenance

The features of mobile environments, coupled with the need to support seamless mobility, make maintaining a consistent cache at the mobile client a challenging task.

Reasons:

- The underlying cache maintenance protocol should not overburden wireless resources and the mobile device.
- Protocols should be energy-efficient, tolerant of dis-connections, and adaptive to varying the QoS provided by the wireless network.

Caching in Mobile Computing: Maintenance

There are two types of cache consistency requirement:

- Strong consistency
  - Cache is always up-to-date
- Weak consistency
  - Data may not always be up-to-date

Strong consistency precludes disconnected execution.
Polling generates more network traffic and introduces latency on data access.

Consistency maintenance

- Polling
  - On every data access → client polls the server to assess if data has changed **Strong**
- TTL (Time To Live)
  - TTL expires → Client polls the server to assess if data has changed **Weak**
- Invalidation
  - Stateless → invalidation reports on data modification or periodically (batch) **Weak**
  - Stateful → invalidation reports tailored for each client

Caching in Mobile Computing: Push-based Systems

Caching in Mobile Computing: Push-based Systems

Broadcast Invalidation Reports

An invalidation is a pair *(id, ts)*

- *ts* denotes the timestamp when object with identifier *id* was modified

Use a time window of *w* units

- Defines how long a client my be disconnected

When receiving a invalidation at time t

- For each object (id, ts) in the invalidation report

```
if (timestampInCache(id) < ts)
   Object is stale, remove it from the cache
else
   updateTimestamp(id, t)
```

Caching in Mobile Computing: Push-based Systems

Broadcast Invalidation Reports

The window _w_ “decides” how long the client can be disconnected and still be able to validate the cache entry of data item.

If client disconnected longer then _w_ time units has to discard all the items in its cache (or revalidate them before use)

Disconnected Operation

Disconnected operation - whenever some information is better than no information (when availability is more important than consistency)

Disconnected Operation Pre-fetching (hoarding)

Pre-fetching consists in the process of populating the cache with data before it is necessary

- Why is pre-fetching essential for mobile computers?

The term **hoarding** is often used for naming the pre-fetch process executed by mobile nodes prior to disconnection.

Disconnected Operation Pre-fetching (hoarding)

Main issues:

- What data items (files) do we hoard?
- When and how often do we perform hoarding?
- How to deal with cache misses?
- How reconcile the cache version of the data item with the version at the server.

Pre-fetching: what to pre-fetch?

User defined: the user explicitly specifies the data that should be cached

Automatic process: the system automatically infers the data that needs to be cached for each user

- Based on prior access history and access patterns
  - Recently used files
  - Groups of files that are usually accessed together.

Pre-fetching: when to pre-fetch?

Data must be pre-fetched during the connected period.

For supporting unexpected disconnection, mobile nodes need to maintain the cache populated at all times

- Add/remove items as users interests change
- Keep replicas synchronized

Pre-fetching: how to pre-fetch?

Pull-model: each client actively requests data to be cached

- The most common model

Push-model: servers push data that clients cache (or not)

- Interesting for scenarios where there is a broadcast channel that can be shared by multiple clients
  - E.g. broadcast disks

Mobile Cache Maintenance Schemes

Invalidating-Based Strategies - Server initiates the cache consistency verification

- Stateless Approach – server does not maintain info (how long) about the cache contents of the clients
  - Stateless Asynchronous approach
    - invalidation reports sent out on data modification
  - Stateless Synchronous approach
    - Server sends out invalidation reports periodically
- Stateful Approach – server keeps track of the cache contents of its clients
  - Stateful Asynchronous approach - use invalidation reports (callbacks) - Home Location Cache
  - Stateful Synchronous approach - periodic broadcast of invalidation reports (server keeps track of objects that are recently updated and broadcast information to clients periodically)

Disconnected Execution
Asynchronous Stateful Invalidation

Asynchronous invalidation reports when data changes.

For each mobile client a HLC is maintained by its HA

Home Agent (HA)
- Builds on the similar concept similar in Mobile IP
- Can be maintained at any thrusted static host
- May move closer to the mobile host for improving performance

Home Location Cache (HLC)
- Buffers invalidation messages
- Keeps the last invalidation timestamp of every object in cache

Mobile host
- Two modes: awake and asleep (disconnected)

Disconnected Execution
Asynchronous Stateful Invalidation

AS Scheme ensures that the data returned to a mobile client is at most t seconds old, where t is the max latency of forwarding an invalidation report from the server to the client via its HA.

Designed for applications that require strict data currency guarantees and in which access to stale data is undesirable.

Such applications include access to critical data such as Bank Account info, and Air Traffic Info.

Disconnected Execution Asynchronous Stateful Invalidation

Use
- Cache consistency maintenance through Asynchronous invalidation reports (callbacks)

Send
- Invalidation reports to mobile client (MH) only when some data changes

Home Agent
- Keep track of what data have been locally cached at its mobile hosts (cache state info of the mobile host)
- A HA can be maintained at any trusted static host
- Pass all messages between the Mobile Host and Data server
- To assist with handling disconnections

Disconnected Execution Asynchronous Stateful Invalidation

Home Location Cache
- HLC can be viewed as a Proxy
- Hold a list of records for Mobile Hosts

  (x, T, Invalid Tag)

  x – Identifier of a data item

  T – Timestamp of the last invalidation of x

  Invalid Tag – TRUE (invalidation has been sent to the host, but no ACK yet)

Disconnected Execution Asynchronous Stateful Invalidation

Assume the following Computing (Operation) Scenario

- The application program runs on the client and communicates with the data server through messages
- Client Request Data Item
  - Client → Uplink Request (Query) → Data Server
  - Client saves some data in its local memory to minimize the number of data request
- Data Server Replying
  - Data Server → Downlink Reply (data) → Client

Disconnected Execution Asynchronous Stateful Invalidation

Assumptions:

- No message is lost due to communication failure or otherwise in the wired network
- Data server update any data items → Send an Invalidation Message → all Home Agents via the wired network
- Home Agents forwards Invalidation Message to the relevant Mobile Host
- Mobile Host
  - Can detect if connected to the network or not
  - Receives the Invalidation Message (through roaming and is not disconnected)
  - Informs HA before it stores (or updates) any data item in its local cache

Disconnected Execution Asynchronous Stateful Invalidation

A Mobile Host (MH) operates in two modes

- In Sleep Mode
  - Unable to receive any invalidation messages
  - Suspends processing of any queries from the applications
  - Each Mobile Host maintains a Cache timestamp for holding the timestamp of the last message received from its Home Agent
  - The HA uses the cache timestamp to discard invalidations that ii no longer needs

Disconnected Execution Asynchronous Stateful Invalidation

- Wakeup after a Sleep
  - Send a probe message to its HA with its cache timestamp
  - The probe message is piggybacked on the 1st query after the wakeup to avoid unnecessary probing
- HA response to the probe message (invalidation check)
  - Sends an invalidation report
- MH determines which data items changed while disconnected
- Defers answering all queries that it received, until it has received all the invalidation report from its HA

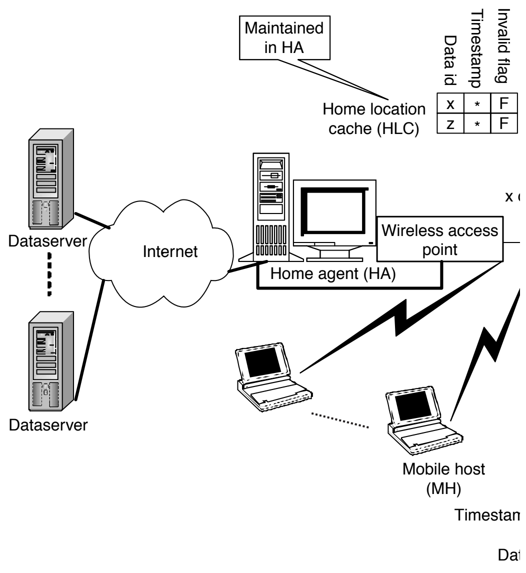
Disconnected Execution Asynchronous Stateful Invalidation

An Example Scenario

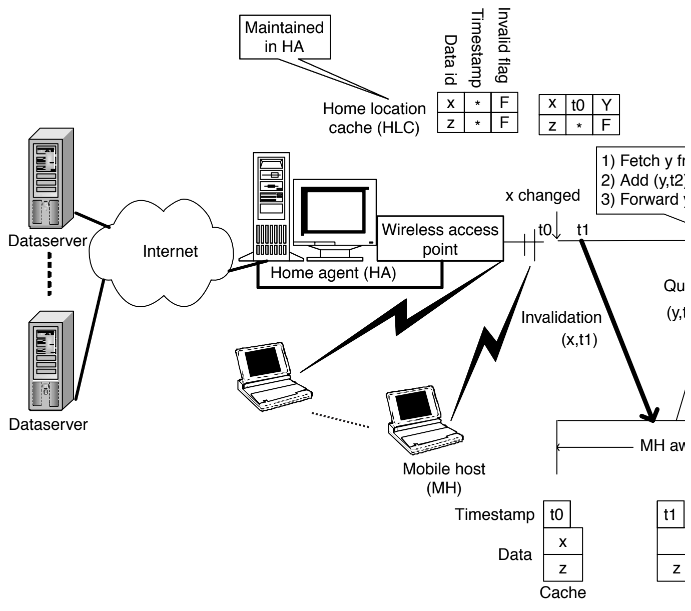
Cache Timestamp t0
- Two data items: IDs x and z

Timestamp t1
- The HLC received an Invalidation Message (IM) notifying that data item x changed at the server at time t1
- Adds IM to mobile host HLC and forward it to mobile host with data item ID and timestamp (x; t1)
- MH updates its cache timestamp to t1, and delete x from the its cache
- MH wants to access y it sends a data request (y;t1) to the HLC
- HLC responds, fetches y, and send it (y; t2)

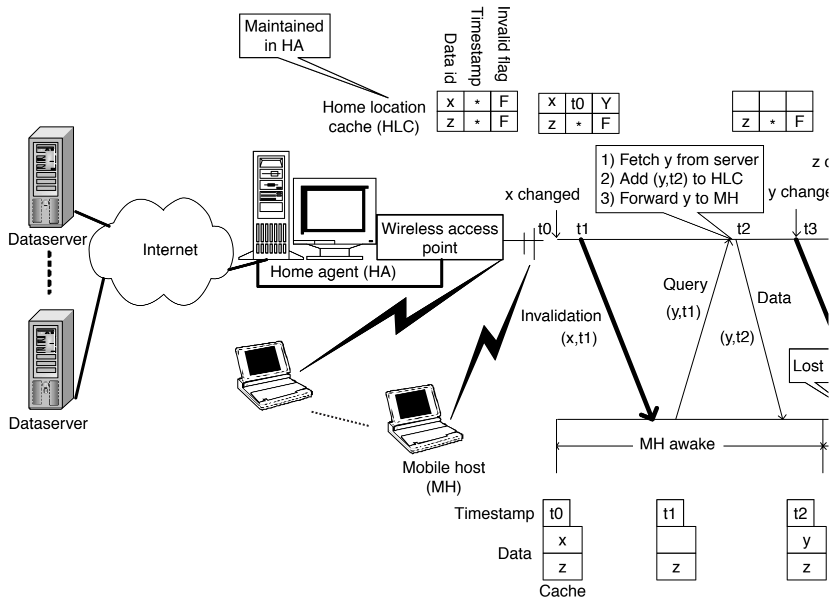
Disconnected Execution Asynchronous Stateful Invalidation



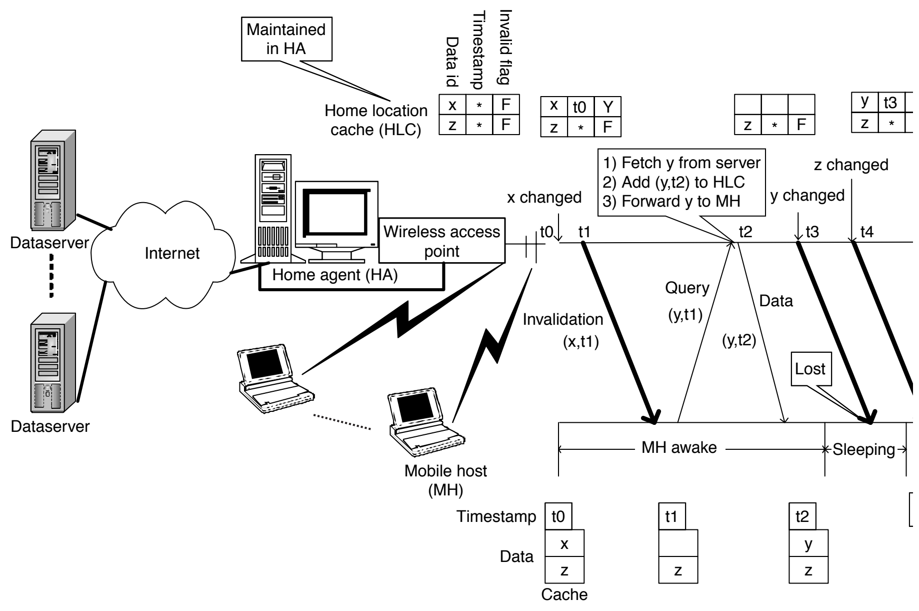
Disconnected Execution Asynchronous Stateful Invalidation



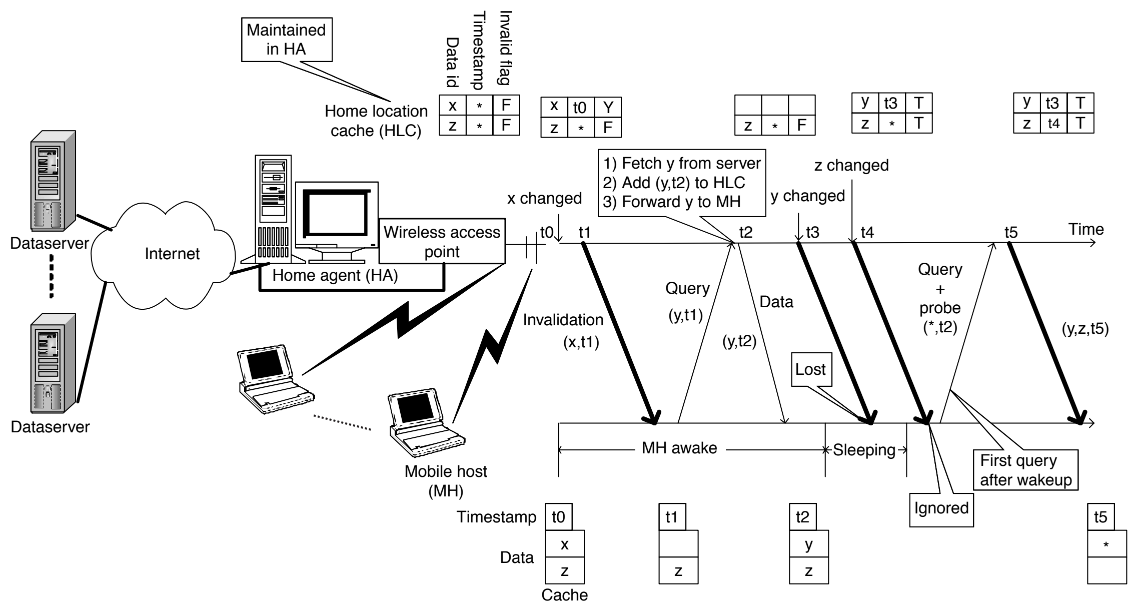
Disconnected Execution Asynchronous Stateful Invalidation



Disconnected Execution Asynchronous Stateful Invalidation



Disconnected Execution Asynchronous Stateful Invalidation



Disconnected Execution Asynchronous Stateful Invalidation

Timestamp t2
- MH updates its timestamp to t2, adds y to the cache

Timestamp t3
- MH get disconnected – sleep mode
- y changed, and Invalidation message for y is lost

Timestamp t4
- Wakeup
- Z changed, and invalidation message for z is sent
- Ignore all invalidation messages until the 1st query
- Query + Probe (*, t2)

Timestamp t5
- Invalidate message (y,x,t5)

Disconnected Execution Asynchronous Stateful Invalidation

"The AS scheme ensures, in absence of any loss of invalidation reports in wired networks, that the data returned to a mobile client is at most $\underline{t}$ seconds old, where $\underline{t}$ is the maximum latency of forwarding an invalidation report from the server to the client via its HA."

Data Management Beyond the Cache

Data Management in current mobile phones

Mobile OSs support
- SQL databases
- Key-value pairs
- Files

HTML 5 supports
- Session storage – key/value pairs maintained during one application session
- Local storage – key/value pairs maintained for an application across sessions

- IndexedDB – key/value object-oriented database for Web browsers
  - stores JSON objects with indexes
  - limited support on latter versions of most known Web browsers

Data management in current mobile phones

No default synchronization mechanism with server databases

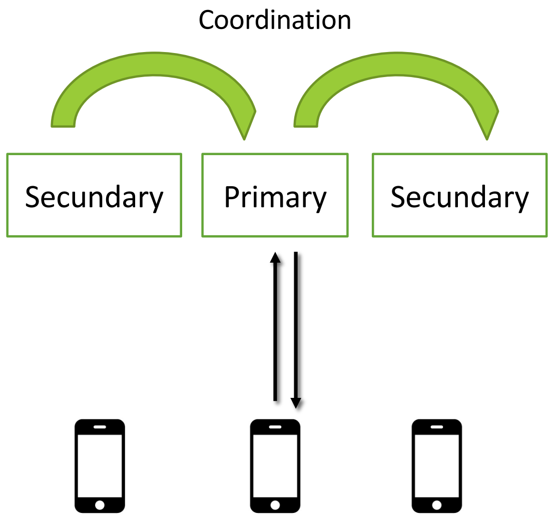
Several third party tools support this type of synchronization

Data management solutions

Single replica

Multiple replicas – pessimistic approach (e.g. single master)

- Updates are coordinated; only one update at a time
- Properties?
  - Simple synchronization process

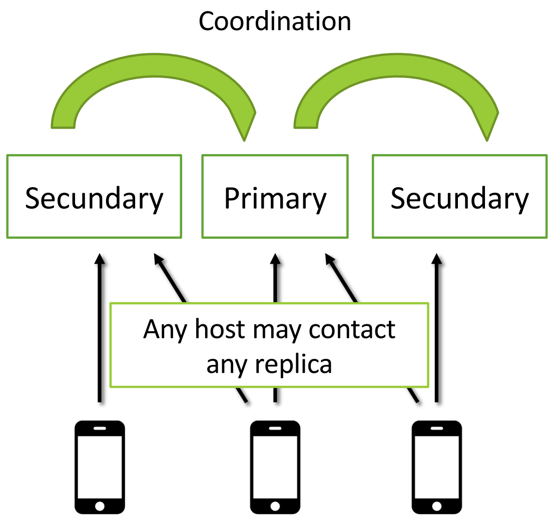


Data management solutions

Single replica

Multiple replicas – pessimistic approach (e.g. single master)

- Updates are coordinated; only one update at a time
- Properties?
  - Simple synchronization process

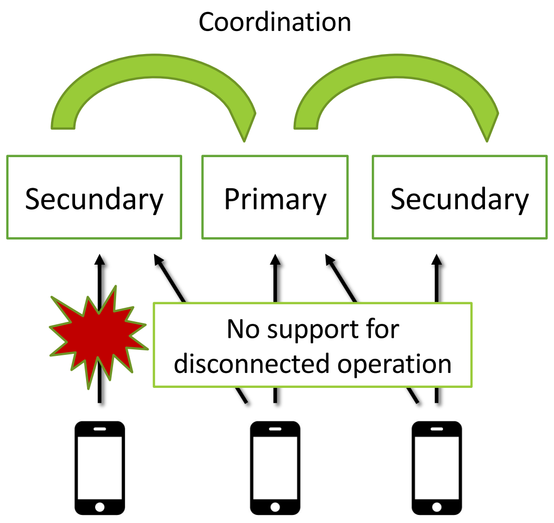


Data management solutions

Single replica

Multiple replicas – pessimistic approach (e.g. single master)

- Updates are coordinated; only one update at a time
- Properties?
  - Simple synchronization process
  - Bad (or no) support for disconnected operation

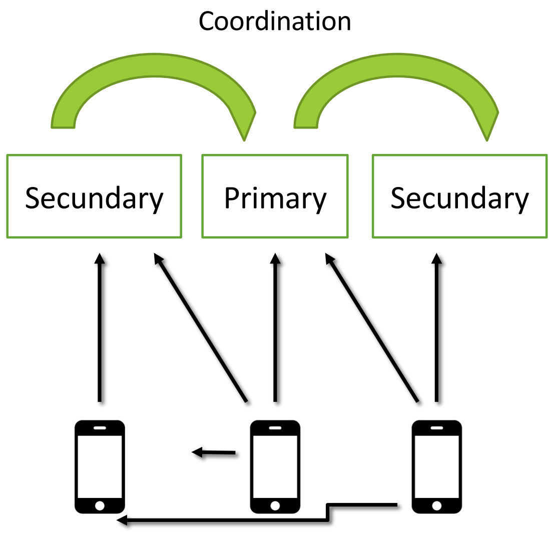


Data management solutions

Single replica

Multiple replicas – pessimistic approach (e.g. single master)

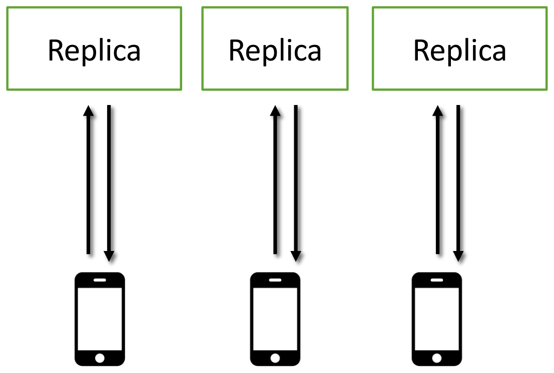
- Updates are coordinated; only one update at a time
- Properties?
  - Simple synchronization process
  - Bad (or no) support for disconnected operation
    - Unless one of the mobile hosts *hosts* the primary replica



Data management solutions

Multiple replicas – optimistic approach

- Uncoordinated updates; users can execute update at any time, any replica
- Properties?
  - Supports disconnected operation; cooperative work
  - May lead to replica divergence due to concurrent updates
    - **Need mechanism to detect conflicts and solve conflicts (reconciliation)**



Data management: Reconciliation

Desirable properties

- Eventual convergence
  - When the system is idle all copies converge to the same state.
- Integration of all contributions
- Intention preservation (whenever possible)
  - The effects of the operations must be preserved

Approaches

- State-based
  - Reconciliation mechanism uses different data version as the basis to create a reconciled version
- Operation-based
  - When users execute updates, the system creates a log of operations; these logs are used to merge concurrent updates

Bibliography

Books:

- Frank Adelstein, et. al. Fundamentals of Mobile and Pervasive Computing. McGraw-Hill. 2005. Chap 3.

Papers

- **Operational Transform (just the concept):** Imine A., Molli P., Oster G., and Rusinowitch M. (2003) Proving correctness of transformation functions in real-time groupware. In *Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work* (ECSCW'03), 277-293.
- **CRDTs (just the concept and examples):** Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, Mihai Leia. A commutative replicated data type for cooperative editing. 29th IEEE International Conference on Distributed Com- puting Systems (ICDCS 2009), IEEE Computer Society, pp.395-403, 2009.
- **Coda (section 4):** Kistler J. and Satyanarayanan M. (1992). Disconnected operation in the Coda File System. *ACM Trans. Comput. Syst.* 10, 1 (February 1992), 3-25.