Mobile Information Systems

Lecture 12: Mobile Algorithms

© 2015-23 Dr. Florian Echtler
Bauhaus-Universität Weimar
Aalborg University

Mobile algorithms

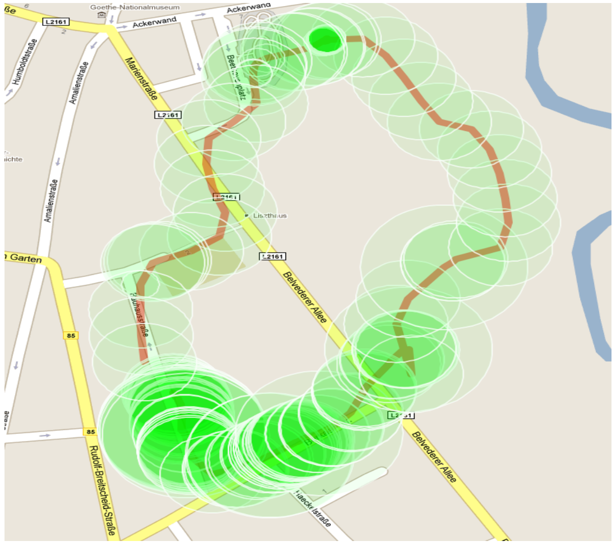
- Moving object databases
  - Queries for moving objects
  - Modelling of dynamic attributes
- Synchronization (Dropbox & Co.)
  - CAP theorem
  - Rsync algorithm
  - Trickle algorithm

Moving object queries

- Which restaurant is reachable within 30 min.?
- When will I arrive at the Marktplatz?
- How many ambulances are placed within a radius of 10 km?
- Show a warning if the distance between two airplanes will be less than 500 meters in the next 10 minutes!
- Which UPS driver will be closest to Bauhausstraße 11 at 17:10?

Moving object queries (2)

- Performing a query may or may not depend on the location of the querying device
- Query is related to the current, a past or a future location of an object
- Space and time might be independent
- Location information has vary- uncertainty
- Objects are (mostly) assumed dimensions)



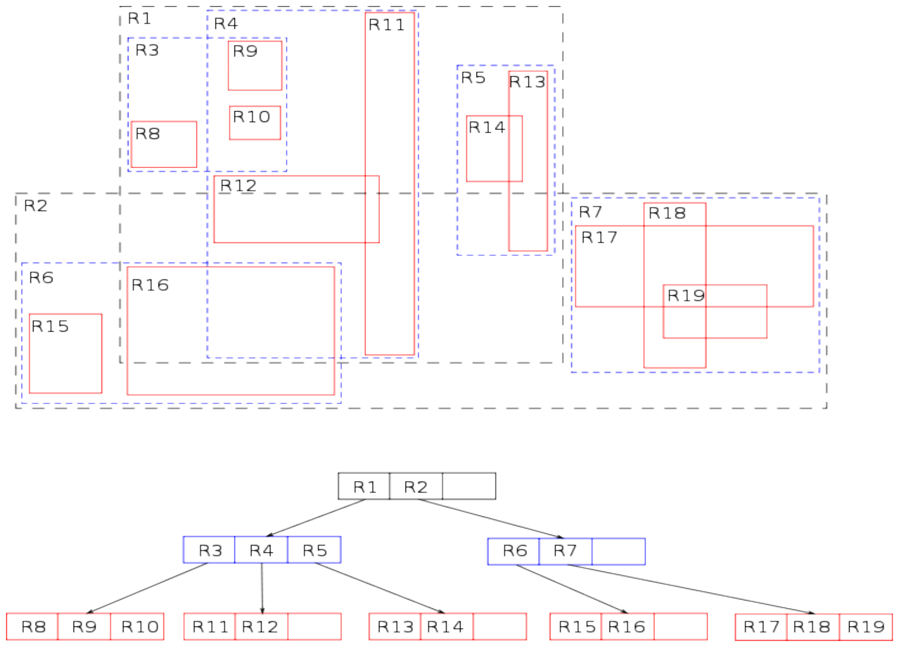
Issues with “traditional” DBMS

- Value is valid until it is changed
- Movement requires permanent updating
  - “Expensive” updates
  - High network traffic
  - Uncertainty
- Typically no history
- No possibility to forecast future locations

Issues regarding indices

Image source (PD): https://en.wikipedia.org/wiki/R-tree#/media/File:R-tree.svg

- Possibly large number of moving objects
- Efficient access required
  - Index over location information
- Spatial index: R-tree
  - Each node is a rectangle covering all nodes beneath
  - Updates expensive → ill suited to moving objects



Issues regarding query languages

- Queries with spatial and temporal conditions:
  - Spatial-temporal range query: “objects that overlap with polygon P within next 3 minutes”
  - Spatial-temporal join query: “find airplanes that will have a distance less than 1000 m and return the regarding time point”
- Traditional query languages like SQL are not (well) suited
  - Extensions for spatial/temporal aspects exist, but not optimized for moving objects

Issues regarding uncertainty

- Implications for queries:
  - 2 different results
    - Objects that certainly fulfil the condition
    - Objects that possibly fulfil the condition
  - Or per object: possibility that a condition is fulfilled
- Questions:
  - When does uncertainty become more expensive than updates?
  - How can a DBMS „tell“ how uncertain a piece of information can be at most?

Location model for moving objects

- Traditional:
  - Stored value is valid until it is explicitly changed
- MOD (moving object databases):
  - Explicit updates unacceptable
  - Object might be temporarily disconnected
- Solution
  - Location (and other attributes) are represented as function over time

Dynamic attributes

- Dynamic attribute A consists of 3 sub-attribs:
  - updateValue, updateTime, function f(t)
- Value v of A:
  - t = updateTime: v = updateValue
  - t = updateTime + t0: v = updateValue + f(t0)
- Explicit update of updateValue and/or f(t):
  - updateTime = timestamp of last change

Dynamic attributes (2)

- Queries for same attribute at t1 and t2 may differ, even without intermediate updates
- Queries about past possible
  - Needs log of prior updates
  - Storage space?
- Queries about future possible
  - Prediction based on current function f
  - Changes to f will change query result
    (e.g. airplanes: deviation from flight plan?)

Location attributes

- Location of a moving object:
  - 2 (or 3) dynamic attributes L.x, L.y, (L.z)
  - f(t) = updateValue + (t – updateTime) * velocity
- Suitable for objects with straight movement (airplanes, ships at sea)
  - Many updates for objects that follow a route (e.g. street) and frequently change their direction
  - Each turn requires an update of the function sub-attribute

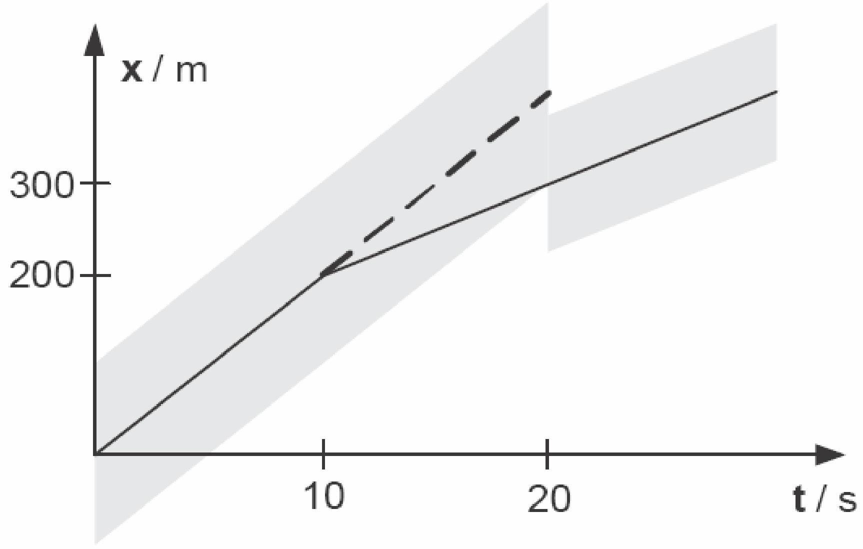
Location attributes (2)

- Better handling of road-based routes
- Location attribute with 4 sub-attributes:
  - updateValue – current index on route
  - updateTime – time point of last update
  - route – list of route points
  - speed – scalar speed
- f(t) = route[updateValue + (t – updateTime) * speed]
  - Requires interpolation for long straight segments

Uncertainty and inaccuracy

Source (FU): Mobile Datenbanken, Höpfner et al., dpunkt.verlag

- Position information is inaccurate
- Objects do not move exactly along the projected route but update only if …
  - Inaccuracy is above threshold
  - Time interval expires
- From the view point of a query: correctness of result is uncertain



Spatial Indices

- Frequent queries for dynamic attributes
  - Range queries → index on attributes required
- „Normal“ spatial index is not applicable
  - High update frequency
- Division of the problem in two sub-problems
  - Suitable geometric representation of dynamic attributes
    - Time as an additional space dimension
  - Efficient access via index structures
    - Classical spatial indexes

Synchronization

Several issues for Dropbox, OwnCloud & Co.:

- Problem: intermittent connectivity
  - Transfers may be interrupted
  - Updates may happen while offline
- Problem: bandwidth on mobile (still) limited
  - Impossible to just copy GBs of data
  - Adaptive synchronization required

Background: CAP theorem

Source (FU): http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

"The CAP theorem states that any networked shared-data system can have at most two of three desirable properties:

- *consistency* (C) equivalent to having a single up-to-date copy of the data;
- *high availability* (A) of that data (for updates);
- tolerance to network *partitions* (P)."

Note: network partitions = disjoint, unconnected sub-networks (single, disconnected device is its own network partition!)

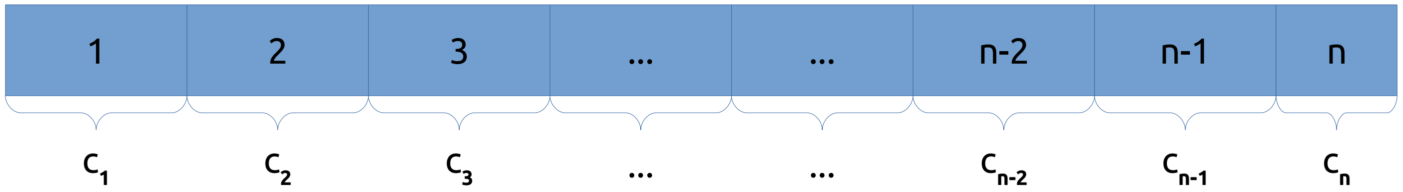
CAP theorem Dropbox?

- CAP theorem → 3 options: CA, AP, PC
  - Which combination applies to Dropbox etc.? (AP)
  - How is the missing feature handled?
- Examples for other combinations?
  - CA – Example: “classic” cluster database (MySQL, MariaDB, PostgreSQL, Oracle, …) → whole DB is offline when connections within cluster fail
  - CP – Examples: some distributed databases (not as widely used) → e.g. disable “minority” partitions

Synchronization: rsync algorithm

Source (FU): https://rsync.samba.org/tech_report/node2.html

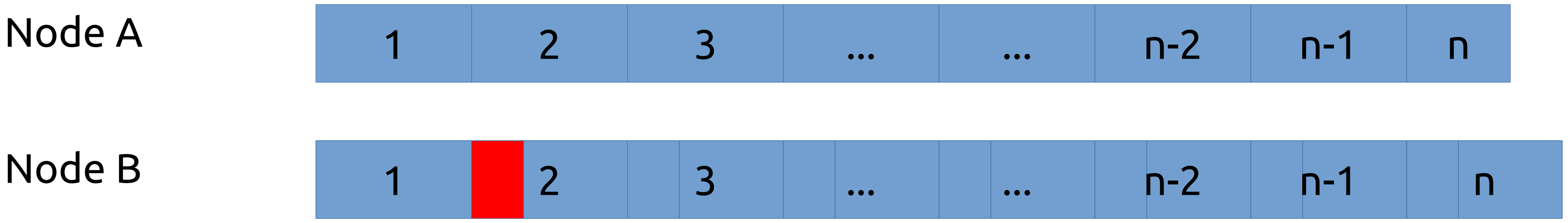
- rsync = Unix tool, created 1996 by A. Tridgell
- Goal: synchronize two *similar* files
  - Designed for slow communication links
  - Core idea: split file into fixed-size blocks, calculate checksum for each block
    - Send checksums to synchronization partner
    - Decide which blocks to transmit



Rsync algorithm (2)

Source (FU): https://rsync.samba.org/tech_report/node3.html

- Works fine if blocks stay in place
- Problem: what if offsets differ?

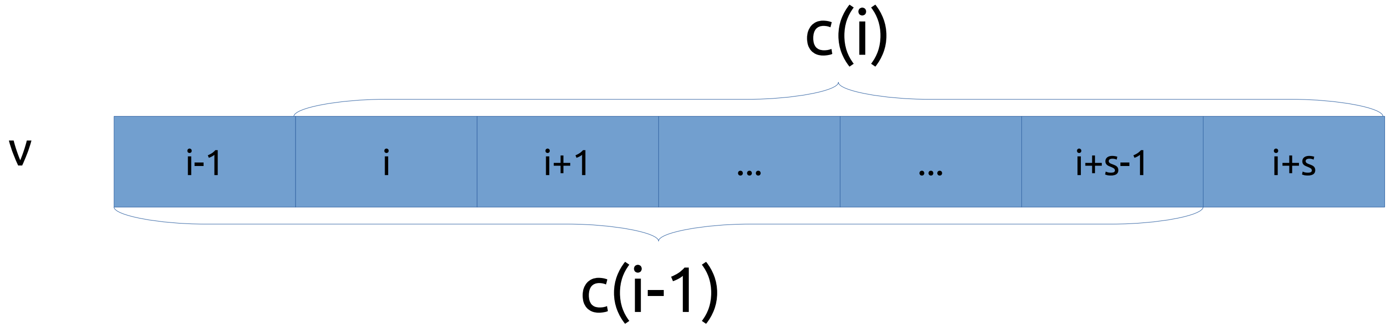


- Inserted data (red) → all following c_i change
- Solution: *rolling* checksum

Rsync algorithm (3)

Source (FU): https://rsync.samba.org/tech_report/node3.html

- Rolling checksum: can be computed for *every possible block offset* in one pass
  - "Sliding window" approach (cf. signal processing)
  - Blocksize s, values v, checksum c, offset i
  - Simplified: c(i) = (c(i-1) - v[i-1] + v[i+s]) mod M



Rsync algorithm (4)

Source (FU): https://rsync.samba.org/tech_report/node3.html

- During calculation of rolling checksum:
  - Search checksum list received from sync partner
    - Note: list only contains only 1 checksum per block
    - Fast search using hash table possible
  - If match found: verify with *strong checksum* (MD5)
    - If match verified: current block exists on both sides
    - If no match: block needs to be transmitted

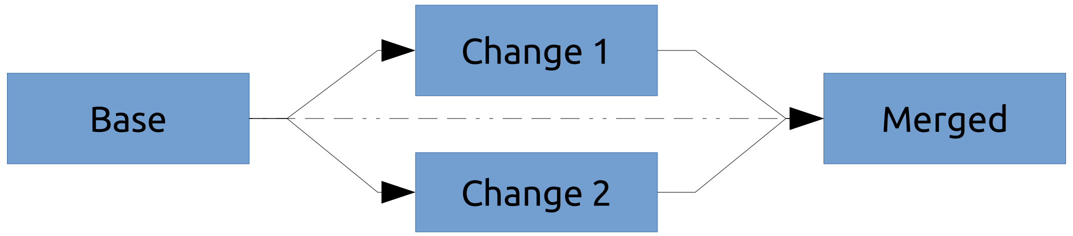
- Rsync algorithm is widely used (Dropbox!)
- Also suitable for backups etc. → space-saving

Synchronization: issues?

- Rsync algorithm requires stable connection
  - What if device temporarily offline?
  - What if 2 nodes make changes while offline?

→ (Merge) conflicts

  - Resolved using 3-way merge algorithm
  - Problem: only applicable to unstructured files, e.g. source code, plain text



Mesh networks: Trickle algorithm

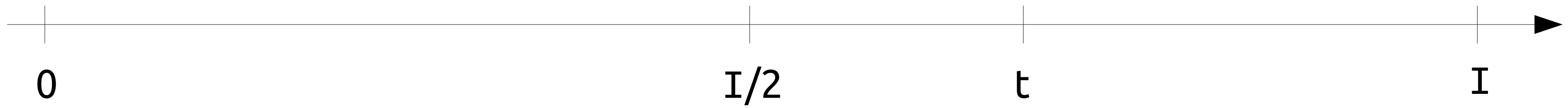
Source (BSD): https://tools.ietf.org/html/rfc6206

- Goal: distribute shared state across mesh
  - Examples: sensor configuration, routing tables, software updates, ...
  - Avoid “broadcast storms”, keep packet rate low
- Basic concept for each node: broadcast state, unless other transmissions suggest consistency
  - A broadcasts version V
    → B already has version V+1
    → B knows A needs update
  - B broadcasts version V+1
    → A has new data

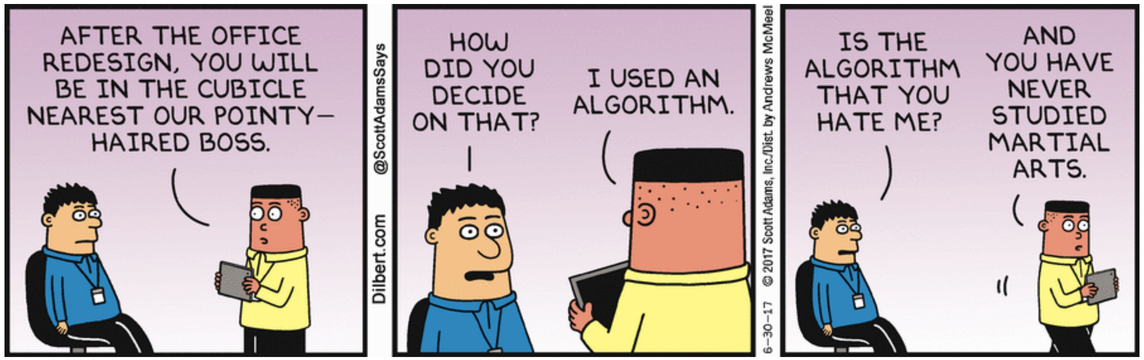
Mesh networks: Trickle algorithm

Source (BSD): https://tools.ietf.org/html/rfc6206

- Interval $[I_{min}, I_{max}]$, redundancy constant k
- Set random initial timer interval $I := [I_{min}, I_{max}]$
- Start timer (interval I), set $c := 0$, $t := [I/2, I]$
  - Receiving "consistent" transmission → c++
  - Receiving "inconsistent" TX → $I = I_{min}$, restart timer
  - Interval time = t && c < k → transmit own state
  - Interval time = I (timer expired) → $I = \min(2*I, I_{max})$



The End


AFTER THE OFFICE REDESIGN, YOU WILL BE IN THE CUBICLE NEAREST OUR POINTY-HAIRED BOSS.
HOW DID YOU DECIDE ON THAT?
I USED AN ALGORITHM.
IS THE ALGORITHM THAT YOU HATE ME?
AND YOU HAVE NEVER STUDIED MARTIAL ARTS.
Dilbert.com
@ScottAdamsSays
6-30-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel