Mobile Distributed Systems I

Carlos Baquero

Grupo de Sistemas Distribuídos
Departamento de Informática
Universidade do Minho

2005/2006

Background

Before mobility there were classical *distributed systems*.

Background

Before mobility there were classical *distributed systems*.

A definition [Coulouris and Dollimore 88] can distinguish those from multi-processor systems and parallel architectures.

> *Shared resources needed to provide an integrated computing service are provided by some of the computers in the network and are accessed by system software that runs in all of the computers, using the network to coordinate their work and to transfer data between them.*

Background

Before mobility there were classical *distributed systems*.
A definition [Coulouris and Dollimore 88] can distinguish those from multi-processor systems and parallel architectures.

> *Shared resources needed to provide an integrated computing service are provided by some of the computers in the network and are accessed by system software that runs in all of the computers, using the network to coordinate their work and to transfer data between them.*

The key is *independent failure*.

Mobile Distributed Systems

- In the 90s technological progress made Distributed Systems go mobile.

Mobile Distributed Systems

- In the 90s technological progress made Distributed Systems go mobile.
- Machine hardware became transportable and then truly portable.

Mobile Distributed Systems

- In the 90s technological progress made Distributed Systems go mobile.
- Machine hardware became transportable and then truly portable.
- Communication methods proliferated and became ubiquitous.

Mobile Distributed Systems

- In the 90s technological progress made Distributed Systems go mobile.
- Machine hardware became transportable and then truly portable.
- Communication methods proliferated and became ubiquitous.
- Is a Mobile Distributed System just another kind of Distributed System ?

Worst Case Distributed Systems

Mobile systems take distributed systems to extreme scenarios.

Worst Case Distributed Systems

Mobile systems take distributed systems to extreme scenarios.
Citing [Pitoura and Samaras]

Worst Case Distributed Systems

Mobile systems take distributed systems to extreme scenarios.
Citing [Pitoura and Samaras]

> *In a sense, mobile computing is the worst case of distributed computing since fundamental assumptions about connectivity, immobility and scale are no longer valid.*

Worst Case Distributed Systems

Mobile systems take distributed systems to extreme scenarios. Citing [Pitoura and Samaras]

> *In a sense, mobile computing is the worst case of distributed computing since fundamental assumptions about connectivity, immobility and scale are no longer valid.*

But mobile telephony already masks many difficulties . . .

Worst Case Distributed Systems

Mobile systems take distributed systems to extreme scenarios. Citing [Pitoura and Samaras]

> *In a sense, mobile computing is the worst case of distributed computing since fundamental assumptions about connectivity, immobility and scale are no longer valid.*

But mobile telephony already masks many difficulties . . .
Would it be enough to use GPRS/UMTS and traditional client/server technology ?

Mirages of Connectivity

- Connectivity is not really ubiquitous.
- Pricing used to be a major limitation (before Kanguru).

Mirages of Connectivity

- Connectivity is not really ubiquitous.
- Pricing used to be a major limitation (before Kanguru).
- GPRS/UMTS in PT: Megabyte for 5 to 1.2 euros (¿50Mb).
- 5Mb $\equiv$ 2 min medium quality video $\equiv$ 25 to 6 euros.
- 5Mb $\equiv$ 15 photos at 1.2 MegaPixel.

Mirages of Connectivity

- Connectivity is not really ubiquitous.
- Pricing used to be a major limitation (before Kanguru).
- GPRS/UMTS in PT: Megabyte for 5 to 1.2 euros (¿50Mb).
- 5Mb $\equiv$ 2 min medium quality video $\equiv$ 25 to 6 euros.
- 5Mb $\equiv$ 15 photos at 1.2 MegaPixel.
- Cheaper UMTS $\Longrightarrow$ Sponsored content.

Mirages of Connectivity

- Connectivity is not really ubiquitous.
- Pricing used to be a major limitation (before Kanguru).
- GPRS/UMTS in PT: Megabyte for 5 to 1.2 euros (¿50Mb).
- 5Mb $\equiv$ 2 min medium quality video $\equiv$ 25 to 6 euros.
- 5Mb $\equiv$ 15 photos at 1.2 MegaPixel.
- Cheaper UMTS $\Longrightarrow$ Sponsored content.
- Synchronous connections used to be cheaper.
- With synchronous GSM connections 1Mb by 0.65 euros

Mirages of Connectivity

- Connectivity is not really ubiquitous.
- Pricing used to be a major limitation (before Kanguru).
- GPRS/UMTS in PT: Megabyte for 5 to 1.2 euros (¿50Mb).
- 5Mb $\equiv$ 2 min medium quality video $\equiv$ 25 to 6 euros.
- 5Mb $\equiv$ 15 photos at 1.2 MegaPixel.
- Cheaper UMTS $\Longrightarrow$ Sponsored content.
- Synchronous connections used to be cheaper.
- With synchronous GSM connections 1Mb by 0.65 euros
- In 2005 Kanguru droped prices by 3 orders of magnitude.
- WIFI in PT: Hour near 5 euros.
- Cybercafes are usually cheaper and should not be.

In search of sensible solutions

- Ads are almost always misleading.

In search of sensible solutions

- Ads are almost always misleading.
- Technology and business models are moving targets.

In search of sensible solutions

- Ads are almost always misleading.
- Technology and business models are moving targets.
- Some relations seldom change:
  - Fixed vs Portable memory and CPU power.
  - Wired vs Wireless resources.
  - Power lines vs Batteries.

In search of sensible solutions

- Ads are almost always misleading.
- Technology and business models are moving targets.
- Some relations seldom change:
  - Fixed vs Portable memory and CPU power.
  - Wired vs Wireless resources.
  - Power lines vs Batteries.
- Fundamental results are forever.
  - Causality.
  - Atomicity.
  - Data convergence.

Outline

- Introduction to mobile data management: Concepts, assumptions, motivations, modeling of mobile distributed systems
- Caching/Stashing: Single writers, invalidation, update dissemination, prefetching. Case study.
- Coordinated Replication: Locks, conflicts, state and log propagation. Mobile file systems. Case study.
- Consistency Models: Strong and weak consistency. Uses of weak consistency. Divergence detection and quantification, reconciliation. Case study.
- Data Bases: Data snapshots, data reservations, transactions, operation re-integration. Case study.

Bibliography

- Data Management for Mobile Computing. Evaggelia Pitoura, George Samaras, 1998, Kluwer.
- Mobility: Processes, Computers and Agents. Dejan Milojicic, Fred Douglis, Richard Wheeler, 1999, ACM press.
- Technical articles (Pointers to be provided in http://gsd.di.uminho.pt).

Mobile Computing Context and Challenges

Sources

- The Challenges of Mobile Computing. G. Forman, J. Zahorjan, April 1994, IEEE Computer.
- Fundamental Challenges in Mobile Computing. M. Satyanarayanan, 1996, ACM PODC.
- Environnements Mobiles: Etude et Synthèse Bibliographique, A. Baggio, 1995, Tech-report INRIA.

These papers survey the intrinsic characteristics of Mobile Computing.

Constraints [Satya 96]

Resources: *Mobile elements are resource-poor relative to static elements*

> *For a given cost and level of technology, considerations of weight, power, size and ergonomics will exact a penalty in computational resources ... While mobile elements will improve in absolute ability, they will always be resource-poor relative to static elements.*

Constraints [Satya 96]

Vulnerability: *Mobility is inherently hazardous*

> *A Wall Street stockbroker is more likely to be mugged on the streets of Manhattan and have his laptop stolen than to have his workstation in a locked office physically subverted. In addition to security concerns portable computers are more vulnerable to loss or damage.*

Constraints [Satya 96]

Vulnerability: *Mobility is inherently hazardous*

> *A Wall Street stockbroker is more likely to be mugged on the streets of Manhattan and have his laptop stolen than to have his workstation in a locked office physically subverted. In addition to security concerns portable computers are more vulnerable to loss or damage.*

In addition: Some PDAs are less vulnerable to intrusion and data logging.

Constraints [Satya 96]

Connectivity: *Mobile connectivity is highly variable in performance and reliability*

> *Some buildings may offer reliable, high-bandwidth wireless connectivity while others may only offer low-bandwidth connectivity. Outdoors, a mobile client may have to rely on a low-bandwidth network with gaps in coverage.*

Constraints [Satya 96]

Connectivity: *Mobile connectivity is highly variable in performance and reliability*

> *Some buildings may offer reliable, high-bandwidth wireless connectivity while others may only offer low-bandwidth connectivity. Outdoors, a mobile client may have to rely on a low-bandwidth network with gaps in coverage.*

In addition: Connectivity costs are also highly variable.

Constraints [Satya 96]

Energy: *Mobile elements rely on a finite energy source*

> *While battery technology will undoubtedly improve over time, the need to be sensitive to power consumption will not diminish. Concern for power consumption must span many levels of hardware and software to be fully effective.*

Constraints [Satya 96]

Energy: *Mobile elements rely on a finite energy source*

> *While battery technology will undoubtedly improve over time, the need to be sensitive to power consumption will not diminish. Concern for power consumption must span many levels of hardware and software to be fully effective.*

In addition: Energy scavenging does not dispense power concerns.

Design Issues [FH 94, Baggio 95]

Communication Modes

Communication modes

- Connected - Low cost and high bandwidth

Design Issues [FH 94, Baggio 95]

Communication Modes

Communication modes

- Connected - Low cost and high bandwidth
- Partially Connected / Semi-Connected - High cost or low bandwidth

Design Issues [FH 94, Baggio 95]

Communication Modes

Communication modes

- Connected - Low cost and high bandwidth
- Partially Connected / Semi-Connected - High cost or low bandwidth
- Disconnected - Null bandwidth

Design Issues [FH 94, Baggio 95]

Communication Modes

Communication modes

- Connected - Low cost and high bandwidth
- Partially Connected / Semi-Connected - High cost or low bandwidth
- Disconnected - Null bandwidth

Special cases

Design Issues [FH 94, Baggio 95]

Communication Modes

Communication modes

- Connected - Low cost and high bandwidth
- Partially Connected / Semi-Connected - High cost or low bandwidth
- Disconnected - Null bandwidth

Special cases

- Ad-hoc networks - Restrict connectivity horizons

Design Issues [FH 94, Baggio 95]
Communication Modes

Communication modes

- Connected - Low cost and high bandwidth
- Partially Connected / Semi-Connected - High cost or low bandwidth
- Disconnected - Null bandwidth

Special cases

- Ad-hoc networks - Restrict connectivity horizons
- Broadcast Disks - Asymmetric connectivity.

Design Issues [FH 94, Baggio 95]

Communication Modes

Communication modes

- Connected - Low cost and high bandwidth
- Partially Connected / Semi-Connected - High cost or low bandwidth
- Disconnected - Null bandwidth

Special cases

- Ad-hoc networks - Restrict connectivity horizons
- Broadcast Disks - Asymmetric connectivity.

In addition: power conservation influences connection modes.

Design Issues [FH 94, Baggio 95]
Communication Modes

When fully connected mobile systems operate like classical distributed systems. Full connections introduce opportunities for data re-integration, system updates and preparation for future mobility.

Design Issues [FH 94, Baggio 95]
Communication Modes

When semi-connected the use of available bandwidth must be under scrutiny. Compression, deltas shipping, aggregation, digests and content distillation can help on reducing communication. High latency and low bandwidth have strong impacts on “synchronous” interactions in client/server models.

Design Issues [FH 94, Baggio 95]

Communication Modes

When disconnected mobile nodes are restricted to local data, calling for disconnection preparation, replication, operation logging. Optimistic techniques allow operation on shared data at the expense of global consistency.

Design Issues [FH 94, Baggio 95]

Mobility

Machine Mobility

Mobility leads to contacts with heterogeneous networks and changes of identity. Mobility crosses administrative and security domains. Modern tools like DHCP, SSH tunnels and VPNs alleviate some of the problems. Mobile-IP (covered elsewhere) also addresses migration.

Design Issues [FH 94, Baggio 95]

Mobility

Several paradigms come to the support of machine mobility.

Design Issues [FH 94, Baggio 95]

Mobility

Several paradigms come to the support of machine mobility.

- Home Bases keep track of mobile machines/users and establish fixed contact points. Mobile nodes register with their home station as they roam.
- Networks of Mobile Support Stations, sometimes with hand-off procedures, mediate connectivity and storage requirements for mobile nodes.

Design Issues [FH 94, Baggio 95]

Mobility

User Mobility

Mobility of users introduces demands for access transparency, portable authentication methods. User mobility is often a source of unintended replication and with negative impacts on global consistency.

Design Issues [FH 94, Baggio 95]

Mobility

Application Mobility

Application mobility is here a consequence of user mobility. Active applications associated to a user can follow it and dynamically associate to its new location. Session management, migration of profiles and locks are issues of concern. Mail applications are notable examples of user initiated mobility.

Design Issues [FH 94, Baggio 95]

Portability

Power issues

Power conservation is a basic concern on the design and operation of mobile hardware. Design factors include CPU speeds, backlighting, memory size, communication activity and wireless medium protocols. For instance WIFI power demands, unlike bluetooth, are important strain on PDA batteries.

Design Issues [FH 94, Baggio 95]

Portability

Risks to data

Making computers portable heightens their risk of physical damage, unauthorized access, loss, and theft.

The risks can be reduced by using cryptographic techniques, avoiding the storage of sensible data, and easing backup procedures.

Design Issues [FH 94, Baggio 95]

Portability

Interface issues

The different interface models introduce both restrictions and enhancements. Some issues concern, screen and font sizes, input models (pen, buttons, wheels, . . .). With specialized hardware other I/O opportunities can be taken into account when devising solutions: accelerometers, temperature, light and pressure sensors, cameras, microphones, etc.

These issues will be covered elsewhere.

Design Issues [FH 94, Baggio 95]
Portability

Small storage and memory

Persistent storage and memory introduces important constraints both on its capacity and on the access models, in particular on PDAs where some operating systems abstractions are simplified. These constrains often lead to tradeoffs among memory, computation and consumption.

Mobile WWW browsing

Sources

- MobiScape: WWW Browsing under Disconnected and Semi-Connected Operation. Baquero, Fonte, Moura, Oliveira, 1995, CAN3.
- Optimizing World-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach. Liljeber, Alanko, Kojo, Laamanen, Raatikainen, 1995, SDNE.
- WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. Barron, Lindquist, 1996 ACM Mobicom.
- Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation. Fox, Brewer, 1996, 5*th* Int. WWW Conference.
- TeleWeb: Loosely Connected Access to the World Wide Web. Schilit, Douglis, Kristol, Krzyzanowski, Sienicki, Trotter, 1996, 5*th* Int. WWW Conference.
- The Operation of the WWW in Wireless Environments. Hadjiefthymiades, Merakos, 1999, Tech-report University of Athens.

WWW Ancient History

- First accesses by telnet to `info.cern.ch` and emacs WWW clients.

WWW Ancient History

- First accesses by telnet to `info.cern.ch` and emacs WWW clients.
- Mosaic introduced graphical browsing, but made sequential fetches.

WWW Ancient History

- First accesses by telnet to `info.cern.ch` and emacs WWW clients.
- Mosaic introduced graphical browsing, but made sequential fetches.
- Netscape escaped sequentiality by making parallel fetches of page inline contents.

Mobiscape
Context

1995

- Mobile laptops.
- First browsers with proxy support.
- Expensive dial-up connections over wired lines.
- Slow Internet connectivity in LAN networks or slow HTTP servers.
- Caching for reference locality in users groups.

Mobiscape
Model

- Mobiscape installs proxies both at the Mobile Host and Support Station.

Mobiscape
Model

- Mobiscape installs proxies both at the Mobile Host and Support Station.
- Proxies mediate access to LRU caches.

Mobiscape
Model

- Mobiscape installs proxies both at the Mobile Host and Support Station.
- Proxies mediate access to LRU caches.
- Caches are updated both by user activity and profile agents.
- Profilers follow a used-defined script of "should-always-be-in-cache" documents.

Mobiscape
Model

- Mobiscape installs proxies both at the Mobile Host and Support Station.
- Proxies mediate access to LRU caches.
- Caches are updated both by user activity and profile agents.
- Profilers follow a used-defined script of “should-always-be-in-cache” documents.
- SS to MH communication is compressed.

Mobiscape
Caching in Mobiscape

- Profiling is present both in the SS and MH, but the MH profiling scripts must be more conservative.
- Profiling specifies recycling periods and fetches start by comparing headers.

Mobiscape

Caching in Mobiscape

- Profiling is present both in the SS and MH, but the MH profiling scripts must be more conservative.
- Profiling specifies recycling periods and fetches start by comparing headers.
- User activity leads to new insertions in the cache.

Mobiscape
Caching in Mobiscape

- Profiling is present both in the SS and MH, but the MH profiling scripts must be more conservative.
- Profiling specifies recycling periods and fetches start by comparing headers.
- User activity leads to new insertions in the cache.
- Interrupted fetches continue at SS side.

Mobiscape

Issues

- Connections may break so cache updates are only effective after full fetch.

Mobiscape
Issues

- Connections may break so cache updates are only effective after full fetch.
- How to define which links to descend on prefetching ?
- How to tune prefetch aggressiveness to available bandwidth ?

Mobiscape
Issues

- Connections may break so cache updates are only effective after full fetch.
- How to define which links to descend on prefetching ?
- How to tune prefetch aggressiveness to available bandwidth ?
- How to deal with images ?

Mowgli
Model and context

- Proxies at both ends.
- Targets wireless links.
- Long round-trip time, latency.
- Time based accounting.

Mowgli
Performance Improvements

- Prefetching of page contents after parsing in proxies.

Mowgli
Performance Improvements

- Prefetching of page contents after parsing in proxies.
- Aggressive DNS resolve at proxies.

Mowgli
Performance Improvements

- Prefetching of page contents after parsing in proxies.
- Aggressive DNS resolve at proxies.
- Lossless and Lossy compression of same data types (e.g. images).

Mowgli
Performance Improvements

- Prefetching of page contents after parsing in proxies.
- Aggressive DNS resolve at proxies.
- Lossless and Lossy compression of same data types (e.g. images).
- Size limits on some contents.

Mowgli

Performance Improvements

- Prefetching of page contents after parsing in proxies.
- Aggressive DNS resolve at proxies.
- Lossless and Lossy compression of same data types (e.g. images).
- Size limits on some contents.
- Aggressive prefetching of potential links that keeps link use optimal.

WebExpress
Model and context

- Proxies at both ends.
- Assumes high cost in a per byte accounting.
- High latency.
- Low bandwidth.

WebExpress

Protocol Reductions

All requests are routed over a single TCP/IP connection to avoid repeating the costly connection establishment overheads. Since HTTP is stateless there is redundancy among browser capabilities headers, this can be reduced at the proxy layer.

WebExpress
Caching

Caching is similar to Mobiscape. Is present at both ends. Objects have a coherency interval (CI) measured in minutes, that triggers the need to refresh objects. Coherency checks are only made on user fetches, while Mobiscape profilers are more aggressive on the SS side.

WebExpress

Differencing

The concept of differencing is introduced with motivation on CGI based content. Diffs act on a underlying base object that is subject to base changes when contents drift to much from the active base. Diffs are checked with CRCs.

TeleWeb
Model

- Client side proxy only.
- Loose connectivity.
- Adaptation to link changes.

TeleWeb
Model

- Client side proxy only.
- Loose connectivity.
- Adaptation to link changes.

TeleWeb advances the issue of monetary cost control to a prime concern.

TeleWeb
Caching

- Consistency in terms of staleness checks should depend on connectivity.
- Empty memory on MHs is useless so cache should fill it and adapt to demands.
- Users should be asked on what they demand keeping.

TeleWeb
Costs

- Costs should be exposed to the users.

TeleWeb
Costs

- Costs should be exposed to the users. No transparency here.

TeleWeb
Costs

- Costs should be exposed to the users. No transparency here.
- Postpone operations until high connectivity.

TeleWeb
Costs

- Costs should be exposed to the users. No transparency here.
- Postpone operations until high connectivity.
- Maximize use of pay-per-minute channels by batching.

TeleWeb
Dynamics

- Adapt to changing network interfaces and security boundaries.
- Select the most appropriate of multiple net interfaces.

TeleWeb

Dynamics

- Adapt to changing network interfaces and security boundaries.
- Select the most appropriate of multiple net interfaces.
- Session mobility follows user mobility: host-lists, history, cached pages, etc.

Distillation

Distillation is a highly lossy, real-time, datatype-specific compression that preserves most of the semantic content of the document.

Distillation

Distillation is a highly lossy, real-time, datatype-specific compression that preserves most of the semantic content of the document.

- Examples include images, postscript documents, and, why not, audio.

Distillation

Distillation is a highly lossy, real-time, datatype-specific compression that preserves most of the semantic content of the document.

- Examples include images, postscript documents, and, why not, audio.
- Target device capabilities can drive distillation.

Distillation

Distillation is a highly lossy, real-time, datatype-specific compression that preserves most of the semantic content of the document.

- Examples include images, postscript documents, and, why not, audio.
- Target device capabilities can drive distillation.
- Can be achieved with only a server side proxy.

Distillation
Refinement

A distilled content can be subject to selective refinement. Images can be partially enlarged, colors augmented, lossy compression reduced.

Distillation
Refinement

A distilled content can be subject to selective refinement. Images can be partially enlarged, colors augmented, lossy compression reduced.

The same concepts can be applied to text summarization.

Distillation

Cycles, Bandwidth and Battery

Distillation trades off CPU cycles at the SS for bandwidth in the loosely connect channel.

Distillation

Cycles, Bandwidth and Battery

Distillation trades off CPU cycles at the SS for bandwidth in the loosely connect channel.

The impact on the MH side is small but some complex lossy compression formats might need extra CPU in MHs before reconstruction.

Work-Package

Mobile WWW

Review these papers and follow the subsequent literature. After getting up to date on the subject the target is to present for evaluation an abstract that analysis Mobile WWW in the present context and proposes useful techniques and possibly some new ideas.

Tools: start with Google and CiteSeer.
Teams: Two to three co-authors.
Format: Max 5 pages abstract.

Delta Propagation

Sources

- Efficient Algorithms for Sorting and Synchronization. Andrew Tridgell, PhD Thesis, 1999.
- Algorithms for Delta Compression and Remote File Synchronization. Torsten Suel and Nasir Memon.
- xdelta. `http://www.xdelta.org/`

Delta Propagation

rsync [Tridgell 99]

Aims

- Work on binary data, not just text.
- Size on the order of a compressed diff.
- Fast for large files and large collections.
- No prior knowledge on files to sync, use similarities.
- High latencies so reduce round trips on protocol.
- Computationally cheap, is possible.

Delta Propagation

rsync [Tridgell 99]

The aim is for B to sync b_i from a a_i in A.

1. B send some data S based on b_i to A.
2. A matches this against a_i and sends soma data D to B.
3. B constructs b_i' using b_i, S and D.

> *. . . the algorithm requires a probabilistic basis to be useful. The data S that B sends to A will need to be much smaller than the complete files . . . unless links are asymmetric, and fast from B to A.*

Delta Propagation

rsync [Tridgell 99]

First Attempt

1. B divides b_i into N equally sized blocks b_i^j and computes a signature S^j on each block. These signatures are sent to A.
2. A divides a_i into N blocks a_i^k and computes S^k for each block.
3. A searches for S^j matching S^k for all k.
4. for each k, A send to B either a matching block index j or a literal block a_i^k.
5. B constructs a_i using blocks from b_i or literal blocks from a_i.

Delta Propagation

rsync [Tridgell 99]

First Attempt

1. B divides b_i into N equally sized blocks b_i^j and computes a signature S^j on each block. These signatures are sent to A.
2. A divides a_i into N blocks a_i^k and computes S^k for each block.
3. A searches for S^j matching S^k for all k.
4. for each k, A send to B either a matching block index j or a literal block a_i^k.
5. B constructs a_i using blocks from b_i or literal blocks from a_i.

Question

What is the weakness in this algorithm ?

Delta Propagation

rsync [Tridgell 99]

First Attempt

1. B divides b_i into N equally sized blocks b_i^j and computes a signature S^j on each block. These signatures are sent to A.
2. A divides a_i into N blocks a_i^k and computes S^k for each block.
3. A searches for S^j matching S^k for all k.
4. for each k, A send to B either a matching block index j or a literal block a_i^k.
5. B constructs a_i using blocks from b_i or literal blocks from a_i.

Question

What is the weakness in this algorithm ?

Answer

One single byte insertion ruins it.

Delta Propagation

rsync [Tridgell 99]

To solve the problem *A* needs to generate signatures not only at the block boundary but at each byte boundary to check matches with the received signatures. This allows arbitrary length insertions and deletions.

However the computational cost would demand a easy/weak signature and such signature would lead to unaffordable false positive matches.

Delta Propagation

rsync [Tridgell 99]

To solve the problem *A* needs to generate signatures not only at the block boundary but at each byte boundary to check matches with the received signatures. This allows arbitrary length insertions and deletions.

However the computational cost would demand a easy/weak signature and such signature would lead to unaffordable false positive matches.

Question

How to solve this dilemma and choose between a weak or a strong signature ?

Delta Propagation

rsync [Tridgell 99]

To solve the problem *A* needs to generate signatures not only at the block boundary but at each byte boundary to check matches with the received signatures. This allows arbitrary length insertions and deletions.

However the computational cost would demand a easy/weak signature and such signature would lead to unaffordable false positive matches.

Question

How to solve this dilemma and choose between a weak or a strong signature ?

Answer

Don't choose, use both !

Delta Propagation

rsync [Tridgell 99]

The solution (and the key to the rsync algorithm) is to use not one signature per block, both. The first signature needs to be very cheap to compute for all byte offsets and the second needs to have a very low probability of collision.

The second signature is only computed to confirm positive matches on the first.

Delta Propagation

rsync [Tridgell 99]

Two signatures algorithm

1. B divides b_i into N equally sized blocks b_i^j and computes signatures R^j and H^j on each block. These signatures are sent to A.
2. For each byte offset o in a_i A computes R^o for the block starting in o.
3. A compares R^o to each R^j received form B.
4. for each o where R^o matches R^j, A computes H^o and compares with H^j.
5. for each position o, A send to B either a matching block index j or a literal byte.
6. B constructs a_i using blocks from b_i and literal bytes from a_i.

Delta Propagation

rsync [Tridgell 99]

Strong Signature

Selection of the strong signature *H* is fairly simple, a cryptographically strength signature (MD4 in rsync 99, MD5, SHA1) will suffice and ”overkill”for the present needs.

Delta Propagation

rsync [Tridgell 99]

Strong Signature

Selection of the strong signature H is fairly simple, a cryptographically strength signature (MD4 in rsync 99, MD5, SHA1) will suffice and "overkill"for the present needs.

For a b bits signature:

- The probability that a randomly generated block has the same signature than a given block is $O(2^{-b})$.
- The computational difficulty of finding a second block that has the same signature of a given block is roughly $O(2^b)$.
- The individual bits in the signature are uncorrelated and have a uniform distribution.

Delta Propagation

rsync [Tridgell 99]

Fast Signature

The fast signature acts as a filter that prevents excessive use of the strong one. The first one tested in rsync was just a concatenation of the first 4 and last 4 bytes of each block. This was poor and lead to common false positives. It was important to depend on all the block bytes.

Delta Propagation

rsync [Tridgell 99]

Fast Signature

The fast signature acts as a filter that prevents excessive use of the strong one. The first one tested in rsync was just a concatenation of the first 4 and last 4 bytes of each block. This was poor and lead to common false positives. It was important to depend on all the block bytes.

With $R(a) = \sum a_i$ the signature depends on all block bytes and can be computed in a ”sliding fashion”by adding and subtracting when incrementing the offset.

Delta Propagation

rsync [Tridgell 99]

Fast Signature

The fast signature acts as a filter that prevents excessive use of the strong one. The first one tested in rsync was just a concatenation of the first 4 and last 4 bytes of each block. This was poor and lead to common false positives. It was important to depend on all the block bytes.

With $R(a) = \sum a_i$ the signature depends on all block bytes and can be computed in a "sliding fashion"by adding and subtracting when incrementing the offset.

> However this signature is independent on the order of bytes. rsync uses a signature that is dependent on the order and can be incrementally computed.

Delta Propagation

rsync [Tridgell 99]

Candidate signatures are found by a hash table with 16 bits index on the fast signature and a linear search in each hash position.

Delta Propagation

rsync [Tridgell 99]

Candidate signatures are found by a hash table with 16 bits index on the fast signature and a linear search in each hash position. There is a final signature “checksum” on the whole file to avoid strength dependent on the number of blocks.

Delta Propagation

rsync [Tridgell 99]

Candidate signatures are found by a hash table with 16 bits index on the fast signature and a linear search in each hash position. There is a final signature “checksum” on the whole file to avoid strength dependent on the number of blocks. Multiple files are pipelined for latency reduction.

The choice of block sizes is governed by:

- Block size must be larger than the combined size of R and H.
- A larger block size reduces the size of sent signature information from B to A.
- A smaller size is likely to allow more matches and reduce the number of bytes transmitted from A to B.

Delta Propagation

rsync [Tridgell 99]

Candidate signatures are found by a hash table with 16 bits index on the fast signature and a linear search in each hash position. There is a final signature "checksum" on the whole file to avoid strength dependent on the number of blocks. Multiple files are pipelined for latency reduction.

The choice of block sizes is governed by:

- Block size must be larger than the combined size of R and H.
- A larger block size reduces the size of sent signature information from B to A.
- A smaller size is likely to allow more matches and reduce the number of bytes transmitted from A to B.

Links might not always be symmetrical

Delta Propagation
xdelta [MacDonald 98]

- xdelta was based on rsync but optimized to take advantage on the presence of both files. Consequently the cost of sending signatures could be ignored and the produced deltas optimized.
- xdelta optimization allowed much smaller block sizes with respect to rsync.

Delta Propagation

xdelta [MacDonald 98]

- xdelta was based on rsync but optimized to take advantage on the presence of both files. Consequently the cost of sending signatures could be ignored and the produced deltas optimized.
- xdelta optimization allowed much smaller block sizes with respect to rsync.
- Unlike the text based “diff” algorithms xdelta and rsync can only be applied to the original files.

Delta Propagation

xdelta [MacDonald 98]

- xdelta was based on rsync but optimized to take advantage on the presence of both files. Consequently the cost of sending signatures could be ignored and the produced deltas optimized.
- xdelta optimization allowed much smaller block sizes with respect to rsync.
- Unlike the text based “diff” algorithms xdelta and rsync can only be applied to the original files.

HTTP

Both algorithms can be used for HTTP reduction and both drive distinct Web proxy prototypes.

Broadcast Disks

Broadcast Disks exploits communication asymmetry by treating a broadcast stream of data that are repeatedly and cyclicly transmitted as a storage device. The broadcast disk technique has two main components. First, multiple broadcast programs (or "disks") with different latencies are superimposed on a single broadcast channel, in order to provide improved performance for non-uniform data access patterns and increased availability for critical data. Second, the technique integrates the use of client storage resources for caching and prefetching data that is delivered over the broadcast.

Papers

http://www.cs.umd.edu/projects/bdisk/

Causality

Sources

- Time, Clocks, and the Ordering of Events in Distributed Systems, Leslie Lamport, Communications of ACM, 1978.
- Detecting Causal Relationships in Distriuted Computations: In search of the Holy Grail. Schwarz and Mattern, Distributed Computing, 1994.
- Detection of mutual inconsistency in distributed systems. Parker et al, IEEE Transactions on Software Engineering, 1983.
- Advanced Concepts in Operating Systems, Singhal and Shivaratri, MIT Press and Mc Graw Hill, Chapter 5.
- Version stamps: Decentralized Version Vectors. Almeida, Baquero and Fonte, IEEE ICDCS, 2002.
- The Hash History Approach for Reconciling Mutual Inconsistency. Hoon et al, IEEE ICDCS, 2003.

Limitations of Distributed Systems [Singhal]

Absence of a Global Clock

In a distributed system there exists no systemwide common clock (global clock) . . . the notion of global time does not exist.

Absence of Shared Memory

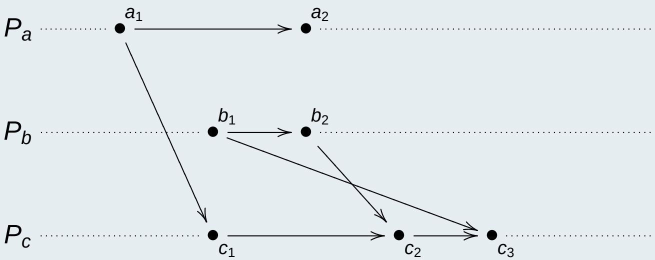
Since the computers in a distributed system do not share common memory, an up-to-date state of the entire system is not available to any individual process.

In asynchronous distributed systems, processes communicate by exchanging messages over communication channels. Both are subject to arbitrary delays. respectivelly, in computation and transmition time.

Lamport's Causality [Lamport 78]

Lamport defines a "happened before" relation betwen events in a distributed computation. Events related under this notion are connected by one or more directed paths in a time diagram for a given computation.

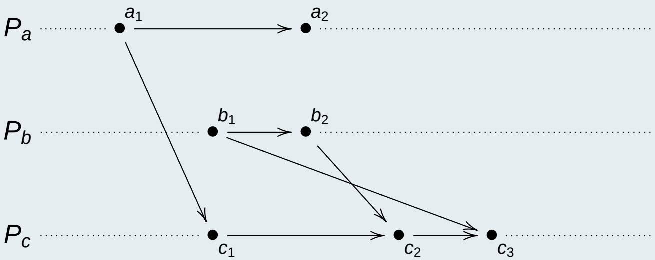
A time diagram of a distributed computation



Lamport's Causality [Lamport 78]

Lamport defines a "happened before" relation betwen events in a distributed computation. Events related under this notion are connected by one or more directed paths in a time diagram for a given computation.

A time diagram of a distributed computation



a_1 and b_1 are related in real time but are not causaly related.

A Causality Definiton

Definition

$\rightarrow$ is the smallest transitive relation satisfying:

- $a \rightarrow b$, if a and b are events in the same activity and a occurred before b.
- $a \rightarrow b$, if a is the event of sending a message and a is the corresponding event of receiving that message.

A Causality Definiton

Definition

$\rightarrow$ is the smallest transitive relation satisfying:

- $a \rightarrow b$, if a and b are events in the same activity and a occurred before b.
- $a \rightarrow b$, if a is the event of sending a message and a is the corresponding event of receiving that message.

Causality defines a partial order relation $(E, \rightarrow)$ on the set of events E.

A Causality Definiton

Definition

$\rightarrow$ is the smallest transitive relation satisfying:

- $a \rightarrow b$, if a and b are events in the same activity and a occurred before b.
- $a \rightarrow b$, if a is the event of sending a message and a is the corresponding event of receiving that message.

Causality defines a partial order relation $(E, \rightarrow)$ on the set of events E.

In addition: Two events a and b are concurrent ($a \parallel b$) if and only if $\neg(a \rightarrow b) \wedge \neg(b \rightarrow a)$.

Some Order Theoretic Tools [Mattern]

Let E denote the set of events in a distributed computation, and let $(S, <)$ denote an arbitrary partially ordered set. Let $\phi : E \longrightarrow S$ denote a mapping.

1. $(\phi, <)$ is said to be consistent with causality, if for all $a, b \in E$. $\phi(a) < \phi(b)$ if $a \rightarrow b$.

Some Order Theoretic Tools [Mattern]

Let E denote the set of events in a distributed computation, and let $(S, <)$ denote an arbitrary partially ordered set. Let $\phi : E \longrightarrow S$ denote a mapping.

1. $(\phi, <)$ is said to be consistent with causality, if for all $a, b \in E$. $\phi(a) < \phi(b)$ if $a \rightarrow b$.
2. $(\phi, <)$ is said to characterize causality, if for all $a, b \in E$. $\phi(a) < \phi(b)$ iff $a \rightarrow b$.

Some Order Theoretic Tools [Mattern]

Let E denote the set of events in a distributed computation, and let $(S, <)$ denote an arbitrary partially ordered set. Let $\phi : E \longrightarrow S$ denote a mapping.

1. $(\phi, <)$ is said to be consistent with causality, if for all $a, b \in E$. $\phi(a) < \phi(b)$ if $a \rightarrow b$.
2. $(\phi, <)$ is said to characterize causality, if for all $a, b \in E$. $\phi(a) < \phi(b)$ iff $a \rightarrow b$.

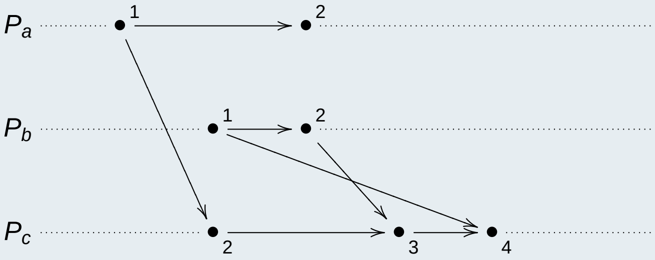
Real time and causality

Real time is consistent with causality, but does not characterize it. *Real time is a total order and total orders do not characterize partial orders.*

Lamport's Logical Time [Lamport 78]

Lamport defines logical time as total order consistent with causality.

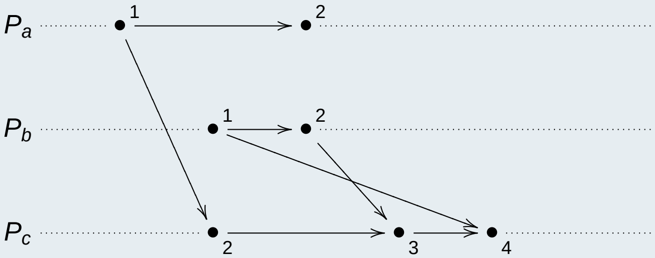
Logical Time: Lamport's Logical Clocks



Lamport's Logical Time [Lamport 78]

Lamport defines logical time as total order consistent with causality.

Logical Time: Lamport's Logical Clocks

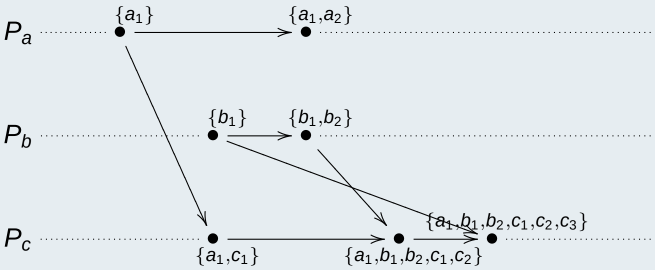


Relations

Logical time is consistent with causality.

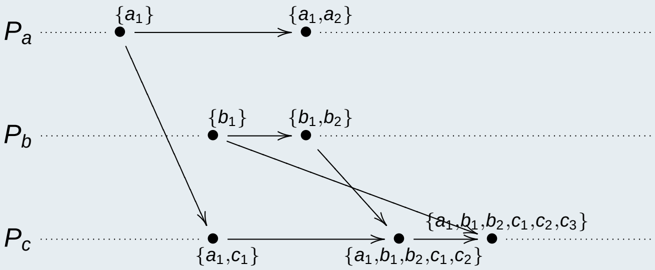
Causality Characterization

Causal Histories



Causality Characterization

Causal Histories

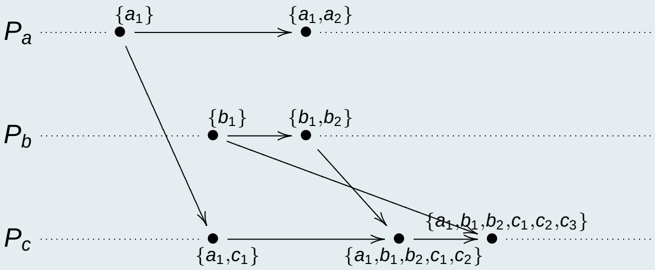


Relations

Causal histories characterize causality.

Causality Characterization

Causal Histories



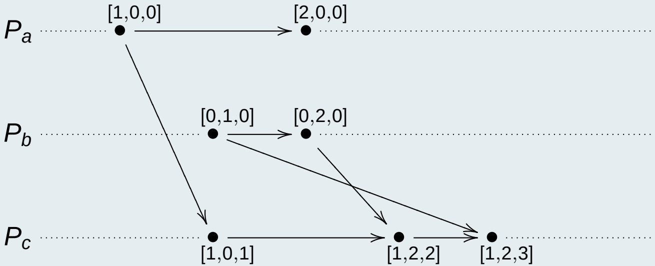
Relations

Causal histories characterize causality.

There are ample oportunities for compression, by taking the last event index from each site.

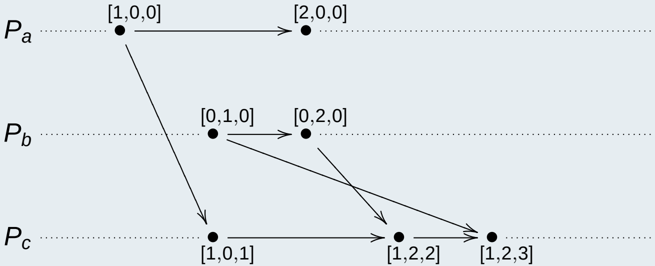
Vector Clocks: [Mattern 89][Fidge 88]

Vector Clocks



Vector Clocks: [Mattern 89][Fidge 88]

Vector Clocks



Relations

Vector clocks characterize causality (and causal histories).
All these relations are order isomorphic.

Charron-Bost's minimality result

Concerning the size of logical clocks in distributed systems. Charron-Bosts, Information Processing Letters, 1991.

Minimality

The minimality result by Charron-Bost, indicates that vector clocks are the most concise characterization of causality among process events.

Charron-Bost's minimality result

Concerning the size of logical clocks in distributed systems. Charron-Bosts, Information Processing Letters, 1991.

Minimality

The minimality result by Charron-Bost, indicates that vector clocks are the most concise characterization of causality among process events.

However, not all causality is process causality.

Data Causality

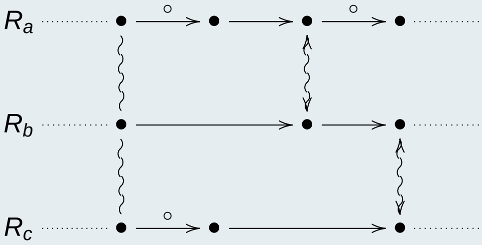
Consider a set of data replicas subject to two operations

- Local updates on the state of a replica.
- Pairwise synchronizations among two replicas bringing them to a common state.

Such model englobes several classes of systems, ranging from partitioned replicated system with strong consistency in each partition, to replicated file systems, databases and some classes of code version control systems.

Data Causality

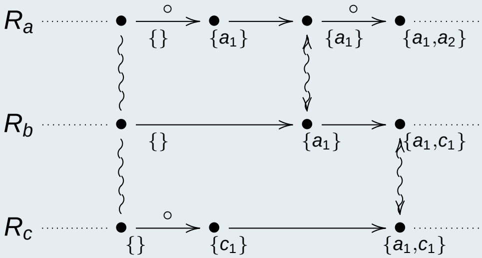
A sample run



The sign ○ represents updates. Synchronizations are depicted by vertical arrows.

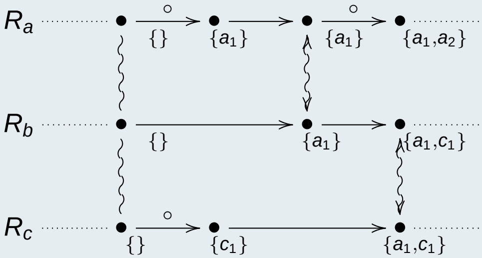
Data Causality

Tagging with causal histories



Data Causality

Tagging with causal histories

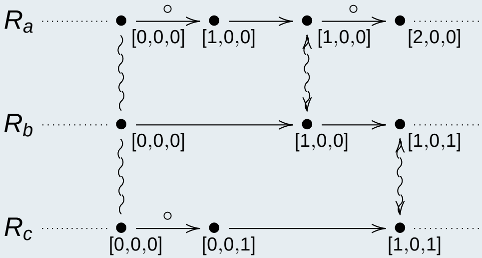


Again, there are ample oportunities for compression, by taking the last event index from each site.

Version Vectors [Parker 83]

Parker et al. Detection of mutual inconsistency in distributed systems. IEEE TSE 1983.

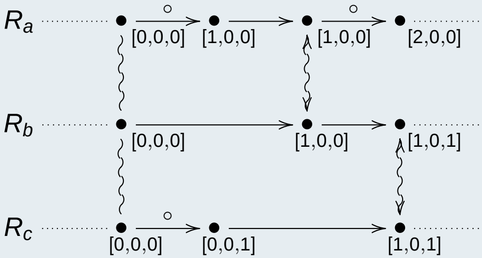
Tagging with version vectors



Version Vectors [Parker 83]

Parker et al. Detection of mutual inconsistency in distributed systems. IEEE TSE 1983.

Tagging with version vectors



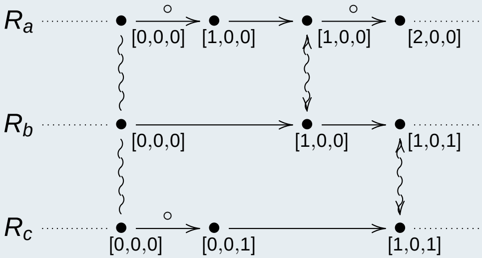
Question

Can we extrapolate that version vectors are minimal characterizations for data causality ?

Version Vectors [Parker 83]

Parker et al. Detection of mutual inconsistency in distributed systems. IEEE TSE 1983.

Tagging with version vectors



Question

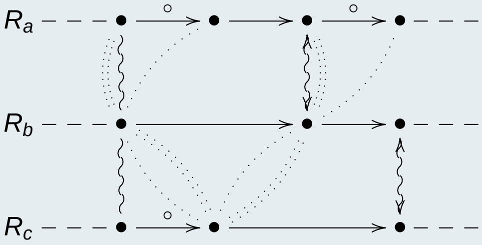
Can we extrapolate that version vectors are minimal characterizations for data causality ?

In fact, the problems addressed are different.

Frontiers

Version stamps - decentralized version vectors. Almeida, Baquero, Fonte. IEEE ICDCS 2002

At any given time only 3 replicas exist.



Version Vectors order limited sets

Bounded Version Vectors. Almeida$\times$2, Baquero. DISC 2004

> *If we consider the role of version vectors, data causality, there is always a limit to the number of possible relations that can be established on the set of replicas. This limit is independent on the number of update events that are considered on any given run. For example, in a two replica system $\{r_a, r_b\}$ only four cases can occur: $r_a = r_b$, $r_a < r_b$, $r_b > r_a$ and $r_a \parallel r_b$. If the two replicas are already divergent, the inclusion of new update events on any of the replicas does not change their mutual divergence and the corresponding relation between them.*

Version Vectors vs Vector Clocks

Vector Clocks

Vector clocks order an unlimited number of events occurring in a given number of processes.

Version Vectors vs Vector Clocks

Vector Clocks

Vector clocks order an unlimited number of events occurring in a given number of processes.

Version Vectors

Version vectors order a given number of replicas, according to an unlimited number of update events.

Version Vectors vs Vector Clocks

Vector Clocks

Vector clocks order an unlimited number of events occurring in a given number of processes.

Version Vectors

Version vectors order a given number of replicas, according to an unlimited number of update events.

This distinctions opened the path to bounded version vectors without contradicting Charron-Bost's minimality.

Bounded Version Vectors [Almeida, Baquero 04]

Bounded version vectors can characterize data causality with a state that is independent on the number of updates. The required state is polynomial with respect to the number of replicas.

Bounded Version Vectors [Almeida, Baquero 04]

Bounded version vectors can characterize data causality with a state that is independent on the number of updates. The required state is polynomial with respect to the number of replicas.

Let U be the number of updates, and N the number of replicas.

- Traditional version vectors have scale $O(N \log_2(U))$
- Bounded version vectors have scale $O(N^3 \log_2(N))$

Bounded Version Vectors [Almeida, Baquero 04]

Bounded version vectors can characterize data causality with a state that is independent on the number of updates. The required state is polynomial with respect to the number of replicas.

Let U be the number of updates, and N the number of replicas.

- Traditional version vectors have scale $O(N \log_2(U))$
- Bounded version vectors have scale $O(N^3 \log_2(N))$

Consequently, the bounded approach can only be efficient for very small numbers of replicas or extremely high update rates.

Identity Management

All the previous techniques (causal histories, vector clocks, version vectors, bounded version vectors) depend on the ability to name participant replicas.

Identity Management

All the previous techniques (causal histories, vector clocks, version vectors, bounded version vectors) depend on the ability to name participant replicas.

- When the number of replicas is known in advance they are given a total order and depicted in a vector. We have $\{1, 2, \ldots, N\} \rightarrow \mathbb{X}$
- In general, a mapping from replicas ids to "counters" is used. $ID \hookrightarrow \mathbb{X}$

being $\mathbb{X}$ an integer, a set of unique events, or a bounded stamp.

Identity Management

In order to establish unique identities we need centralized configuration (eventually hierarchical) or consistent distributed approaches (envolving consensus).
Alternativelly, there are random id approaches, but those are subject to the statistics of the birthday problem.

Identity Management

In order to establish unique identities we need centralized configuration (eventually hierarchical) or consistent distributed approaches (envolving consensus).
Alternativelly, there are random id approaches, but those are subject to the statistics of the birthday problem.

Birthday problem

The probabibility of two of more colisions when using n ids from a universe of d distinct ids is: $P_2(n, d) = 1 - \frac{d!}{(d-n)!d^n}$.
A classical case is: $P(23, 365) \approx 0.507297$
See: http://mathworld.wolfram.com/BirthdayProblem.html

Identity Management

In order to establish unique identities we need centralized configuration (eventually hierarchical) or consistent distributed approaches (envolving consensus).
Alternativelly, there are random id approaches, but those are subject to the statistics of the birthday problem.

Birthday problem

The probabibility of two of more colisions when using n ids from a universe of d distinct ids is: $P_2(n, d) = 1 - \frac{d!}{(d-n)!d^n}$.
A classical case is: $P(23, 365) \approx 0.507297$
See: http://mathworld.wolfram.com/BirthdayProblem.html

Mobile systems and ad-hoc networking call for autonomous data causality.

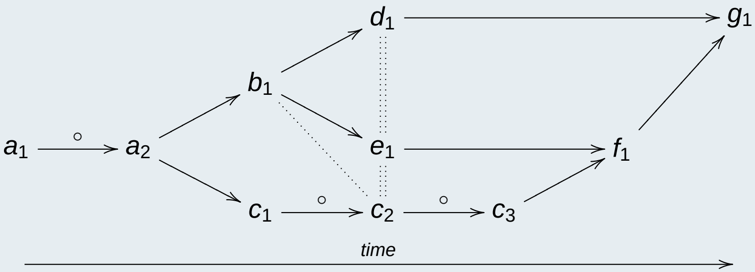
Autonomous Causality

Under autonomous causality we have a variable number of replicas: Each replica can register updates and create new ones; Any two replicas can merge/synchronize.

Autonomous Causality

Under autonomous causality we have a variable number of replicas: Each replica can register updates and create new ones; Any two replicas can merge/synchronize. With autonomous causality we can model standard data causality.

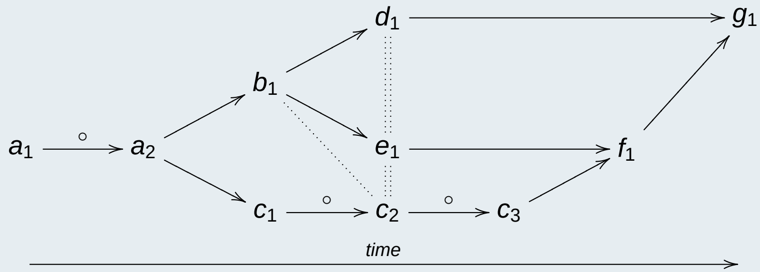
Sample run, with frontiers



Autonomous Causality

Under autonomous causality we have a variable number of replicas: Each replica can register updates and create new ones; Any two replicas can merge/synchronize. With autonomous causality we can model standard data causality.

Sample run, with frontiers

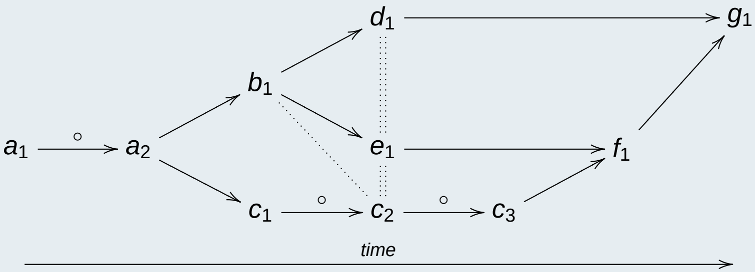


In order to characterize this type of run, first we must handle globally unique ids.

Autonomous Causality

Under autonomous causality we have a variable number of replicas: Each replica can register updates and create new ones; Any two replicas can merge/synchronize. With autonomous causality we can model standard data causality.

Sample run, with frontiers



In order to characterize this type of run, first we must handle globally unique ids. At least unique in each frontier.

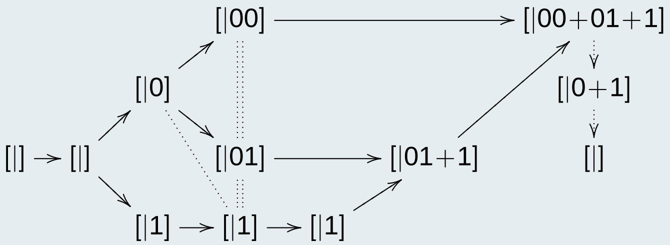
Version Stamps [Almeida, Baquero, Fonte 02]

The strategy to unique ids relies on a recursive proceedure that partitions the id space into separate zones. In order to achive this, individual ids must change.

Version Stamps [Almeida, Baquero, Fonte 02]

The strategy to unique ids relies on a recursive proceedure that partitions the id space into separate zones. In order to achive this, individual ids must change.

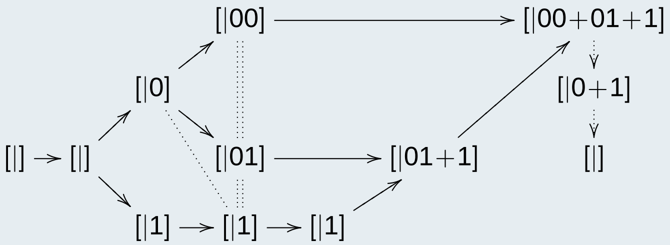
Version stamps ids



Version Stamps [Almeida, Baquero, Fonte 02]

The strategy to unique ids relies on a recursive proceedure that partitions the id space into separate zones. In order to achive this, individual ids must change.

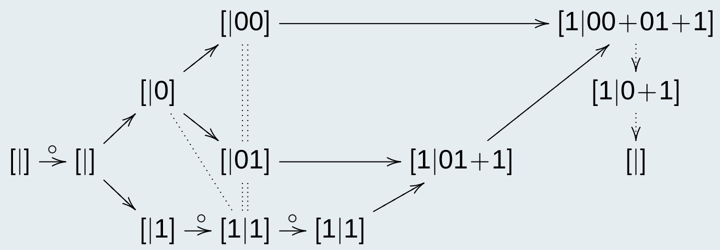
Version stamps ids



Now we need to correctly anotate updates.

Version Stamps [Almeida, Baquero, Fonte 02]

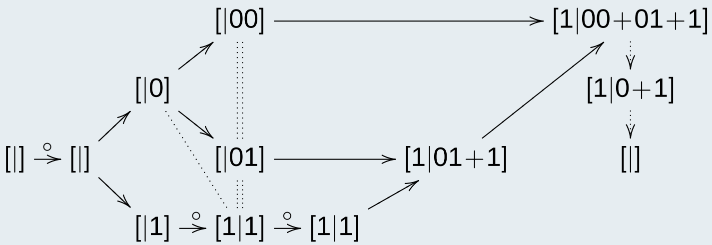
Version stamps



Consecutive updates are compressed and the mechanisms simplifies after joining “adjacent” replicas.

Version Stamps [Almeida, Baquero, Fonte 02]

Version stamps



Consecutive updates are compressed and the mechanisms simplifies after joining “adjacent” replicas.
The week point of this techinique is that some patterns of runs lead to very long identifiers, if the number of replicas is kept high all the time.

Hash Histories [Hoon 03]

The Hash History Approach for Reconciling Mutual Inconsistency. Brent ByungHoon Kang, Robert Wilensky and John Kubiatowicz. IEEE ICDCS 2003.

Quoting:

> *Hash histories,consisting of a directed graph of version hashes, are independent of the number of active nodes but dependent on the rate and number of modifications.*

Hash Histories [Hoon 03]

The Hash History Approach for Reconciling Mutual Inconsistency. Brent ByungHoon Kang, Robert Wilensky and John Kubiatowicz. IEEE ICDCS 2003.

Quoting:

> *Hash histories,consisting of a directed graph of version hashes, are independent of the number of active nodes but dependent on the rate and number of modifications.*

This approach shows that using hashes of replica contents we can track causality. While the authors keep a partial order on the hashes, a similar result can be obtained by a causal history whose ids are based on hashes.

Hash Histories [Hoon 03]

The Hash History Approach for Reconciling Mutual Inconsistency. Brent ByungHoon Kang, Robert Wilensky and John Kubiatowicz. IEEE ICDCS 2003.
Quoting:

> *Hash histories,consisting of a directed graph of version hashes, are independent of the number of active nodes but dependent on the rate and number of modifications.*

This approach shows that using hashes of replica contents we can track causality. While the authors keep a partial order on the hashes, a similar result can be obtained by a causal history whose ids are based on hashes.
The correctness of these techniques is vulnerable to statistical errors, and in some cases to the birthday problem.

File System Mobility

Sources

- Disconnected Operation in the Coda File System, Kistler, Satyanarayanan, ACM Transactions on Computer Systems, 1992.
- The Coda Distributed File System, Braam, Linux Journal, 1998.
- Primarily Disconnected Operation: Experiences with Ficus, Popek et al, 2nd Workshop on Management of Replicated Data, 1992.
- Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication, Popek et al, 1998.
- AdHocFS: Sharing Files in WLANs. Boulkenafed, Issarny. 2nd IEEE International Symposium on Network Computing and Applications. 2003.
- **Aditional sources:** MioNFS, Panasync, Ivy, Intermezzo, FEW.

Coda [Satya 92]

Coda was probabaly the first distributed file system that addressed mobility. Its genesis,in the early 90s, had a major influence in the mobile computing field.

Being a distributed file system, Coda is related to NFS and in particular to AFS (Andrew File System). In these systems, users *mount* into their local directory structures remote file systems. Failure of a server presented serious inconvenience to users.

In the AFS model a global hierarchical namespace is provided to client nodes by the federation of servers. In Coda this namespace starts under `/coda`.

Coda [Satya 92][Braam 98]

> *Clients view Coda as a single, location transparent shared Unix file system. The Coda namespace is mapped to individual file servers at the granularity of subtrees called volumes. At each client a cache manager (Venus) dynamically obtains and caches vlume mappings.*

A volume has a name and an unique ID in the federation of servers. It is possible to mount a volume anyware under `/coda`.

Mounting in Coda

```
cfs makemount u.braan /coda/user/braam
```

Coda [Satya 92][Braam 98]

Coda uses two distinct mechanims to achieve high availability: Server Replication and Disconnected Operation.

> *Server replication allows volumes to have read-write replicas at more than one server. The set of replication sites for a volume is its volume storage group (VSG). The subset of volumes that is currently accessible is a client's accessiblei VSG (AVSG). . . . Venus uses a cache coherence protocal based on callbacks to guarantee that an open of a files yields its latest copy in the AVSG. . . . Modifications in Coda are propagated in parallel to all AVSG sites, and eventually to missing VSG sites.*

Coda [Satya 92][Braam 98]

Disconnected operation takes effect when the AVSG becomes empty. While disconnected, Venus services file requests by relying solely on the contents of its cache. When disconnection ends, Venus propagates modifications and reverts to server replication.

Coda [Satya 92][Braam 98]

Disconnected operation takes effect when the AVSG becomes empty. While disconnected, Venus services file requests by relying solely on the contents of its cache. When disconnection ends, Venus propagates modifications and reverts to server replication.
Check run on [Satya 92] paper.

Coda [Satya 92][Braam 98]

First vs Second CLass Replication

Coda advocates two classes of replicas. They notice that diferences between clients and servers justify a distinguished treatment of replicas.

> *It is appropriate to distinguish between first class replicas on servers, and second class replicas (i.e. cache copies) on clients. . . . Whereas server replication preserves the quality of data in face of failures, disconnected operation forsakes quality for availability. . . . Server replication is expensive because it requires additional hardware. Disconnected operation, in contrast, costs little.*

Coda [Satya 92][Braam 98]

Optimistic vs Pessimistic Replica Control

Coda opted for optimistic replication, one of the reasons was their observation that the degree of write-sharing in typical Unix workloads is very small.

Coda [Satya 92][Braam 98]

Optimistic vs Pessimistic Replica Control

Coda opted for optimistic replication, one of the reasons was their observation that the degree of write-sharing in typical Unix workloads is very small.

Always the case?

In CVS workloads there is a much larger amount of write sharing.

Coda [Satya 92][Braam 98]

Optimistic vs Pessimistic Replica Control

Coda opted for optimistic replication, one of the reasons was their observation that the degree of write-sharing in typical Unix workloads is very small.

Always the case?

In CVS workloads there is a much larger amount of write sharing.

A pessimistic strategy would prevent updates when primary replicas are partitioned (or only allow in a single partition). Updates in disconnected operation should depend on the possession of a write token, that could have a given time-lease. However this has impacts on availability.

Coda [Satya 92][Braam 98]

Optimistic vs Pessimistic Replica Control

Coda opted for optimistic replication, one of the reasons was their observation that the degree of write-sharing in typical Unix workloads is very small.

Always the case?

In CVS workloads there is a much larger amount of write sharing.

A pessimistic strategy would prevent updates when primary replicas are partitioned (or only allow in a single partition). Updates in disconnected operation should depend on the possession of a write token, that could have a given time-lease. However this has impacts on availability.

Low write sharing

Low write sharing could also serve as an argument for pessimistic approaches, since availability would rarely be affected if token movement is efficient. As explored in MioNFS [Guedes, Moura].

Coda [Satya 92][Braam 98]

Implementation

In Linux Coda installs a kernal module that handles calls within the Coda sub-tree in the file system. This module is minimalistic, it keeps a cache of recently handled requests and conveys missed hits to a user level cache manager (Venus). This process checks a client disk cache and mediates contacts with volume servers. Updates are propagated upon file closure.

Coda [Satya 92][Braam 98]

Implementation

Communication with servers is “read-one, write-many”. Files are read from a singleserver in the AVSG and updates are propagated to all available servers.

Coda identifies files by a 32 bit FID, this FID is unique in a cluster of servers. The unit of replication here is the volume.

Coda [Satya 92][Braam 98]

Conflicts

Conflicts can occur between disconected replicas and the AVSG or among VSG partitions. There is no direct communication between mobile clients.

Coda tries to apply automatic conflict resolution and when that fails flags conflicts and asks for user intervention. Version vectors are used for conflict detection.

Ficus e Rumor [Popek et al]

In Ficus/Rumor all replicas are first class. Consequently, mobile nodes can do direct sychronizations without mediation from a base station.

> *While Coda explicitly changes its state between connected, disconnected, and reintrgrating, the Ficus model does not distinguish between connected and disconnected modes. Peers are dynamically connected to various degrees.*

Ficus and Rumor [Popek et al]

Ficus Overview

Ficus is a distributed file system featuring optimistic replication. The default synchronization policy provides single copy availability; so long as any copy of a data item is accessible, it may be updated. Once a single replica has been updated, the system makes a best effort to notify all accessible replicas that a new version of the object exists. Those replicas then attempt to pull over the new version. Ficus guarantees no lost update semantics despite this optimistic concurrency control. Conflicting updates are guaranteed to be detected, allowing recovery after the fact.

Ficus and Rumor [Popek et al]

Ficus Overview

Update propagation is the best-effort attempt to inform other replicas immediately of the presence of changes. In addition, a background process known as reconciliation runs on behalf of each replica after each reboot and periodically during normal operation. It compares all files and directories of a local volume replica with a corresponding remote replica, pulling over any missed updates and detecting any concurrent update conflicts. In the case of directories, most conflicts are repaired automatically by reconciliation, while for files, conflicting versions are marked as such and their owner notified.

Ficus and Rumor [Popek et al]

Reconciliation

Early Ficus designs called for all-pairs reconciliation, but the cost of $O(n^2)$ message exchanges proved too expensive. The alternative currently in use is to reconcile in a ring, each site pulling from the previous. To make this reconciliation topology resilient to failure, the ring skips sites which are inaccessible.

Ficus and Rumor [Popek et al]

Reconciliation

The usage patterns of Ficus confirmed the a low occurrence of write conflicts. They identify an important factor, the human write token. This occurs when users replicate essentially read-only files and files whose updates are done by a single user. In such case the user will issue updates in sequence in real time and conflicts can be avoided if connectivity is re-established as the user moves.

Ficus and Rumor [Popek et al]

Rumor

Rumor is an evolution of Ficus that allows oportunistic update propagation among sites. Unlike Ficus, rumour operates entirely at the application level. Both Ficus and Rumor use version vectors.

Ficus and Rumor [Popek et al]

Rumor

Rumor is an evolution of Ficus that allows oportunistic update propagation among sites. Unlike Ficus, rumour operates entirely at the application level. Both Ficus and Rumor use version vectors.

Rumor Reconciliation

Reconciliation operates between a pair of communicating replicas in a one-way, pull-oriented fashion. A one-way mode is more general than a two-way model, and lends support for inherently uni-directional forms of communication, such as floppy-disk transfer.

Ficus and Rumor [Popek et al]

Rumor Reconciliation

Reconciliation involves only pairs of replicas, rather than all replicas, because there is no guarantee that more than two will ever be simultaneously available. For example, mobile computers operating in a portable workgroup mode may only be connected to a single other computer. Additionally, operating in a point-to-point mode, with an underlying gossip-based transfer mechanism, allows a more flexible and dynamically changeable network configuration in terms of the machines' accessibility from each other.

AdHocFS [Boulkenafed, Issarny]

From this system we can highlight their hybrid solution to consistency management.

AdHocFS [Boulkenafed, Issarny]

From this system we can highlight their hybrid solution to consistency management.

Here they consider partitionable groups of mobile machines.

Optimism is used accross partitions, with divergence detection and flaging of conflicts. However, within each partition write-locks are used to avoid divergence, while using update propagation to keep all machines in sync.

AdHocFS [Boulkenafed, Issarny]

From this system we can highlight their hybrid solution to consistency management.

Here they consider partitionable groups of mobile machines. Optimism is used accross partitions, with divergence detection and flaging of conflicts. However, within each partition write-locks are used to avoid divergence, while using update propagation to keep all machines in sync.

Loss of availability, such as when editing a common file accross machines, is suggested to be overcome by a finer grain fragmentation of the document (a commmon approach for source code and LaTeXdocs).

Design options in Mobile File Systems

Explore

Existing systems have explored some design options. Not all these options are compatible and while some can be left to the users, at some point some decisions must be made. Building on the analysed systems we can explore some alternative designs.

Merge and Reconciliation

Sources

- What is Unision? A formal specification and reference implementation of a file synchronizer, Pierce, Vouillon, 2004.
- Resolving File Conflicts in the Ficus File System, Reiher, Heidemann, Ratner, Skinner, Popek, Usenix 1994.
- Using Structural Characteristics for Autonomous Operation, Baquero, Moura, ACM SIGOPS Review 1999.
- (IceCube) Semantics-based reconciliation for collaborative and mobile systems. Preguia, Shapiro, Matheson, COOPIS 2003.
- See also: Bayou, Xerox Parc.

Reconciliation Overview

- Divergence under optimistic replication $\Rightarrow$ reconciliation.
- Replicas can be opaque or structured elements (files vs filesystems).

Reconciliation Overview

- Divergence under optimistic replication $\Rightarrow$ reconciliation.
- Replicas can be opaque or structured elements (files vs filesystems).
- Detection of non-conflicting divergence (dominating replicas) simplifies reconciliation.

Reconciliation Overview

- Divergence under optimistic replication $\Rightarrow$ reconciliation.
- Replicas can be opaque or structured elements (files vs filesystems).
- Detection of non-conflicting divergence (dominating replicas) simplifies reconciliation.
- Pairwise reconciliation may not suffice for n-ary reconciliation (Unison).

Reconciliation Overview

- Divergence under optimistic replication $\Rightarrow$ reconciliation.
- Replicas can be opaque or structured elements (files vs filesystems).
- Detection of non-conflicting divergence (dominating replicas) simplifies reconciliation.
- Pairwise reconciliation may not suffice for n-ary reconciliation (Unison).
- After reconciliation two replicas might be identical or merely closer.

Reconciliation Overview

- Divergence under optimistic replication $\Rightarrow$ reconciliation.
- Replicas can be opaque or structured elements (files vs filesystems).
- Detection of non-conflicting divergence (dominating replicas) simplifies reconciliation.
- Pairwise reconciliation may not suffice for n-ary reconciliation (Unison).
- After reconciliation two replicas might be identical or merely closer.
- Reconciliation can be log based or state based.

File System Reconciliation

- Reconciliation of directory trees is a special case of structured reconciliation.

File System Reconciliation

- Reconciliation of directory trees is a special case of structured reconciliation.
- The Unison file synchronizer reconciles directory trees between two replicas. This system has been formaly specified.

File System Reconciliation

- Reconciliation of directory trees is a special case of structured reconciliation.
- The Unison file synchronizer reconciles directory trees between two replicas. This system has been formaly specified.
- Unison is state based and keeps the last synchronized state between the two replicas. This state is used to infer state evolution in both repplicas.

File System Reconciliation

- Reconciliation of directory trees is a special case of structured reconciliation.
- The Unison file synchronizer reconciles directory trees between two replicas. This system has been formaly specified.
- Unison is state based and keeps the last synchronized state between the two replicas. This state is used to infer state evolution in both repplicas.
- One relevant design choice in Unison is to keep replicas divergent (partially synchronized) when automated reconciliation fails.

Unison: Create/Remove Conflicts

Consider the following state in two divergent replicas:

A DIR{ f.FILE("Ag") g.FILE("Cl") }

B DIR{ g.FILE("Cl") }

Unison: Create/Remove Conflicts

Consider the following state in two divergent replicas:

A DIR{ f.FILE("Ag") g.FILE("Cl") }

B DIR{ g.FILE("Cl") }

What would be the desired reconciliation ? Preserving user intentions.

Unison: Create/Remove Conflicts

Consider the following state in two divergent replicas:

A DIR{ f.FILE(“Ag”) g.FILE(“Cl”) }

B DIR{ g.FILE(“Cl”) }

What would be the desired reconciliation ? Preserving user intentions.

We need to know the last common state:

O DIR{ f.FILE(“Ag”) g.FILE(“Cl”) }

Unison: Reconciliation 1

Input:

O DIR{ f.FILE("Ag") g.FILE("Cl") }

A DIR{ f.FILE("Ag") g.FILE("Cl") }

B DIR{ g.FILE("Cl") }

Output:

A' DIR{ g.FILE("Cl") }

B' DIR{ g.FILE("Cl") }

Unison: Preserve User Changes

Input:

A DIR{ f.DIR{} g.FILE("Cl") }

B . . .

O DIR{ f.FILE("Ag") g.FILE("Cl") }

Output:

A' DIR{ f.DIR{} g.FILE("Cl") }

B' . . .

Since f changes from O to A then f in A' must be preserved from A.

Unison: Propagate Only User Changes

No new changes can be created – is must never “make anything up”.
Input:

A DIR{ f.FILE(“Ag”) g.FILE(“Cl”) }

B DIR{ f.DIR{} g.FILE(“Cl”) }

O DIR{ f.FILE(“Ag”) g.FILE(“Cl”) }

Output:

A’ DIR{ f.DIR{} g.FILE(“Cl”) }

B’ . . .

Since f in A’ is diferent from f in A it must have come from B.

Unison: Modify/Modify Conflicts

Upon conflicting divergence Unison chooses two keep both versions.
Input:

A DIR{ f.FILE("Fe") g.FILE("Cl") }

B DIR{ f.FILE("Cu") g.FILE("Ni") }

O DIR{ f.FILE("Ag") g.FILE("Cl") }

Output:

Unison: Modify/Modify Conflicts

Upon conflicting divergence Unison chooses two keep both versions.
Input:

A DIR{ f.FILE("Fe") g.FILE("Cl") }

B DIR{ f.FILE("Cu") g.FILE("Ni") }

O DIR{ f.FILE("Ag") g.FILE("Cl") }

Output:

A' DIR{ f.FILE("Fe") g.FILE("Ni") }

B' DIR{ f.FILE("Cu") g.FILE("Ni") }

Unison: Modify/Modify Conflicts

Upon conflicting divergence Unison chooses two keep both versions.
Input:

A DIR{ f.FILE("Fe") g.FILE("Cl") }

B DIR{ f.FILE("Cu") g.FILE("Ni") }

O DIR{ f.FILE("Ag") g.FILE("Cl") }

Output:

A' DIR{ f.FILE("Fe") g.FILE("Ni") }

B' DIR{ f.FILE("Cu") g.FILE("Ni") }

And the new last common sate ?

The new O can come integrate non-conflicting changes:

O' DIR{ f.FILE("Ag") g.FILE("Ni") }

Unison: Remove/Modify Conflicts

Input:

A DIR{ g.FILE("Cl") }

B DIR{ f.FILE("Cu") g.FILE("Cl") }

O DIR{ f.FILE("Ag") g.FILE("Cl") }

Unison: Remove/Modify Conflicts

Input:

A DIR{ g.FILE("Cl") }

B DIR{ f.FILE("Cu") g.FILE("Cl") }

O DIR{ f.FILE("Ag") g.FILE("Cl") }

Output:

A' DIR{ g.FILE("Cl") }

B' DIR{ f.FILE("Cu") g.FILE("Cl") }

In this case O is unchanged.

Unison: Create/Create Conflicts

Input:

A DIR{ f.FILE("Fe") g.FILE("Cl") }

B DIR{ f.FILE("Cu") g.FILE("Cl") }

O DIR{ g.FILE("Cl") }

Unison: Create/Create Conflicts

Input:

A DIR{ f.FILE(“Fe”) g.FILE(“Cl”) }

B DIR{ f.FILE(“Cu”) g.FILE(“Cl”) }

O DIR{ g.FILE(“Cl”) }

Output:

A’ DIR{ f.FILE(“Fe”) g.FILE(“Cl”) }

B’ DIR{ f.FILE(“Cu”) g.FILE(“Cl”) }

In this case O is also unchanged.

Unison: Conflicts across paths

Now we have a delete/modify conflict on diferent levels of the file tree.
Input:

A DIR{ d.DIR{f.FILE("Fe") g.FILE("Cl")} }

B DIR{ }

O DIR{ d.DIR{f.FILE("Ag") g.FILE("Cl")} }

Unison: Conflicts across paths

Now we have a delete/modify conflict on diferent levels of the file tree.
Input:

A DIR{ d.DIR{f.FILE(“Fe”) g.FILE(“Cl”)} }

B DIR{ }

O DIR{ d.DIR{f.FILE(“Ag”) g.FILE(“Cl”)} }

Output:

A’ DIR{ d.DIR{f.FILE(“Fe”) g.FILE(“Cl”)} }

B’ DIR{ }

Unison: Conflicts across paths

Now we have a delete/modify conflict on diferent levels of the file tree.
Input:

A DIR{ d.DIR{f.FILE("Fe") g.FILE("Cl")} }

B DIR{ }

O DIR{ d.DIR{f.FILE("Ag") g.FILE("Cl")} }

Output:

A' DIR{ d.DIR{f.FILE("Fe") g.FILE("Cl")} }

B' DIR{ }

Alternative

A' DIR{ d.DIR{f.FILE("Fe") } }

B' DIR{ }

Here the change would only lock the branch and not the whole sub-tree.

Unison: Wrap up

Unison will try to keep the stated properties and make as much reconciliation as possible, bringing the two replicas to a closer state. Recall that since A’ can de distinct from B’ a trivial reconciliator that makes A’=A and B’=B would satisfy the correctness properties. In this sense Unison defines a Maximality property that leads to the propagation of non-conflicting changes.

Ficus: Resolving Conflicts

The ficus paper "Resolving File Conflicts in The Ficus File System" deals both with the notion of structural conflicts in directory trees and with file conflicts. In the later case, several file conflict resolvers are discussed.

Ficus: Conflict Types

Name Conflicts

Identical names for autonomously created files are solved by appending distinct sufixes to file names.

Ficus: Conflict Types

Name Conflicts

Identical names for autonomously created files are solved by appending distinct sufixes to file names.

Remove/Update

Here the choosen solution is to move removed files (updated elseware) to a *orphan* section of the volume.

Ficus: Conflict Types

Name Conflicts

Identical names for autonomously created files are solved by appending distinct sufixes to file names.

Remove/Update

Here the choosen solution is to move removed files (updated elseware) to a *orphan* section of the volume.

Update/Update

- DIR Update/Update: Creation of clashing file names leads to Name Conflicts. Removals can lead do Remove/Update conflicts.
- FILE Update/Update: Divergent file contents found to be in conflict trigger file specific conflict resolvers.

Ficus: Conflict Types

Ficus allows users to enable automated conflict resolution for some file types and conflict types.

Automatic Backup Files

Some editors create automatic backup files. The resolver can arbitrarily select one of the conflicting backup files. More conservative options can be adopted.

"Junk" Files

Some files can usually be deteled without harm. `core` dumps, . . .

Ficus: Conflict Types

Reconstructable/Dependent files

Some files often exhibit dependencies. This can lead to file reconstruction in the makefile tradition or to obsolete files, such as intermediate files in LaTeXcompilations.

Structured Files

Files such as `.newsrc`, where USENET newsgroups selection is stored are easy to reconcile in a deterministic way. Another case are ordered lists of game scores.

Since Ficus does not rely on comparision with last common states, nothing prevents n-ary reconciliation by iterating pairs of reconciliations.

Ficus: Conflict Statistics

Some statistics collected during 9 months of Ficus operation:

- 14,142,241 FILE Updates
  - 14,141,752 Non-Conflicting
  - 489 Conflicting
    - 162 Automatic
    - 176 Potentially Automatic
    - 151 Manual
- 98 DIR Update/Remove conflicts
- 708,780 DIR Name Creations
  - 708,652 Non-Conflicting
  - 128 Conflicting

CADT: Convergent Abstract Data Types

Some structured file types (unfortunatelly not all) can be reconciled by automated means without compromising divergent user changes, thus preserving intentions. This can be formally captured by representing such types as abstract data types, exhibiting a state and operations that change the state.

CADT: Convergent Abstract Data Types

Some structured file types (unfortunatelly not all) can be reconciled by automated means without compromising divergent user changes, thus preserving intentions. This can be formally captured by representing such types as abstract data types, exhibiting a state and operations that change the state.

Newsrc: A convergent file type

alt.elvis.sighting:33,45,60-200,356
alt.emulators.ibmpc.apple2:
comp.object:1-2628

This file type is a hierarchical composition of simpler convergent data types: Maps, Sets and Growing Sequencies.

CADT: Enshuring Convergence

Some properties of convergent data types:

- They must define an order of evolution (in fact a pre-order). If a user updates replica b to b' then $b \leq b'$. Not all user operations need to advance the replica.
- Copying a replica a into a new replica b should create equivalent replicas, thus $a \cong b$. In fact, $a \cong b$ is equivalent to $a \leq b$ and $b \leq a$.
- A reconciliation m from a and b, should be such that both $a \leq m$ and $b \leq m$. In addition there should be no m' distinct from m such that $m' \leq m \wedge a \leq m' \wedge b \leq m'$.

CADT: Enshuring Convergence

Removing non-determinism:

- **Idempotent** The reconciliation of a replica with herself should produce an unchanged replica.
- **Commutative** The order in which two replicas are supplied to the reconciliation procedure should not be relevant.
- **Associative** Pairwise reconciliation of three or more replicas should derive the same final result, independently of the actual order that was applied in the reconciliation.

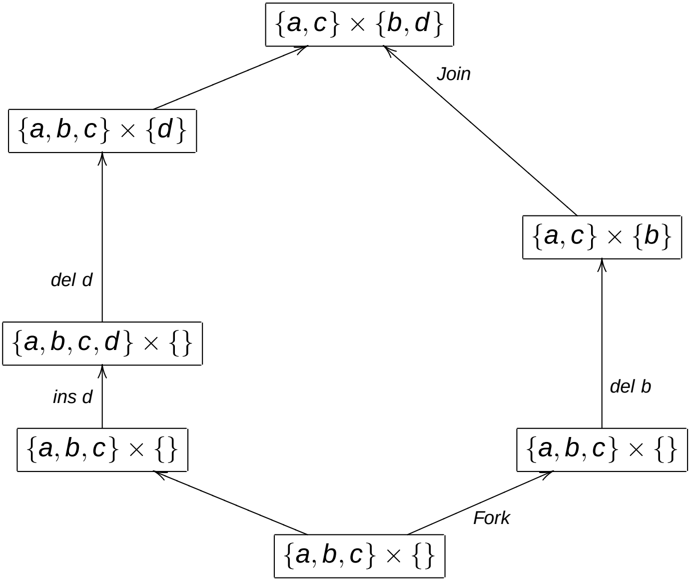
CADT: IncSet

Type: INCSET **extends** BASIC	
Write : *Storage* *Insert* : *Elem* *Find* : *Elem* $\rightarrow$ *Bool*	
Init, *Fork*, *Join*, *Leq*	
$\Sigma = \mathbf{2}^X$	$I = Y$
Init()	
$\sigma := \{\}$	$\iota := \perp$
Insert(*e*)	
$\sigma := \sigma \cup e$	
Find(*e*) $\rightarrow$ *b*	
$b := \begin{cases} true & \text{if } e \in \sigma \\ false & \text{if } e \notin \sigma \end{cases}$	
$Join(\sigma\iota', \sigma\iota'') \rightarrow \sigma\iota$	
$\sigma := \sigma' \cup \sigma''$	$\iota := \iota' \vee \iota := \iota''$
$Leq(\sigma\iota', \sigma\iota'') \rightarrow b$	
$b := \begin{cases} true & \text{if } \sigma' \subseteq \sigma'' \\ false & \text{if } \sigma' \not\subseteq \sigma'' \end{cases}$	

CADT: IncDecSet

Type: IncDecSet **extends** Basic	
$Write : Storage, Insert : Elem, Delete : Elem$ $Find : Elem \rightarrow Bool$	
$Init, Fork, Join, Leq$	
$\Sigma = \mathbf{2}^X \times \mathbf{2}^X$	$I = Y$
$Init()$	
$\sigma_{in} \times \sigma_{del} := \{\} \times \{\}$	$\iota := \perp$
$Insert(e)$	
$\sigma_{in} := \sigma_{in} \cup a$	
$Delete(e)$ **pre?** $e \in \sigma_{in}$	
$\sigma_{in} \times \sigma_{del} := (\sigma_{in} \setminus \{e\}) \times (\sigma_{del} \cup \{e\})$	
$Find(e) \rightarrow b$	
$b := \begin{cases} true & \text{if } e \in \sigma_{in} \\ false & \text{if } e \notin \sigma_{in} \end{cases}$	
$Join(\sigma\iota', \sigma\iota'') \rightarrow \sigma\iota$	
$\sigma_{in} \times \sigma_{del} := ((\sigma'_{in} \setminus \sigma''_{del}) \cup (\sigma''_{in} \setminus \sigma'_{del})) \times (\sigma'_{del} \cup \sigma''_{del})$	$\iota := \iota'$ $\vee\ \iota := \iota''$
$Leq(\sigma\iota', \sigma\iota'') \rightarrow b$	
$b := \begin{cases} true & \text{if } \sigma'_{del} \subseteq \sigma''_{del} \wedge (\sigma'_{in} \setminus \sigma''_{del}) \subseteq (\sigma''_{in} \setminus \sigma'_{del}) \\ false & \text{otherwise} \end{cases}$	

CADT: IncDecSet Run



CADT: IncNat

Type: IncNat	
Inc : *Count* $:\to \mathbb{N}$	
Init, *Fork*, *Join*, *Leq*	
$\Sigma = \mathbf{2}^* \hookrightarrow \mathbb{N}$	$I = \mathbf{2}^*$
Init()	
$\sigma := (\langle\rangle \mapsto 0)$	$\iota := \langle\rangle$
Inc()	
$\sigma := \sigma \dagger (\iota \mapsto \sigma(\iota) + 1)$	
$Count() \to n$	
$n := \sum_{j \in dom(\sigma)} \sigma(j)$	
$Fork(\sigma\iota) \to \sigma\iota', \sigma\iota''$	
$\sigma' := \sigma \cup (\iota' \mapsto 0), \sigma'' := \sigma \cup (\iota'' \mapsto 0)$	$\iota' := \iota + \langle 0\rangle$ $\iota'' := \iota + \langle 1\rangle$
$Leq(\sigma\iota', \sigma\iota'') \to b$	
let $Delta0(\sigma', \sigma'')$ be $\bigwedge_{e \in dom(\sigma' \setminus (\sigma' \cap \sigma''))} \sigma'(e) = 0$ in $b := \begin{cases} true & \text{if } (\iota' \neq \iota'' \wedge dom(\sigma') \subseteq dom(\sigma'')) \\ & \vee (\iota' = \iota'' \wedge \sigma'(\iota') \leq \sigma''(\iota'')) \\ & \vee (\iota' \neq \iota'' \wedge Delta0(\sigma', \sigma'')) \\ false & \text{otherwise} \end{cases}$	

CADT: IncNat Run

$$
\begin{tikzcd}
\begin{matrix} \langle 010 \rangle \mapsto 0 \\ \langle 01 \rangle \mapsto 0 \\ \langle 00 \rangle \mapsto 0 \\ \langle 0 \rangle \mapsto 3 \end{matrix} & & & \\
& \begin{matrix} \langle 010 \rangle \mapsto 0 \\ \langle 01 \rangle \mapsto 0 \\ \langle 0 \rangle \mapsto 3 \end{matrix} \arrow[ul, "\cong"'] & & \begin{matrix} \langle 011 \rangle \mapsto 2 \\ \langle 01 \rangle \mapsto 0 \\ \langle 0 \rangle \mapsto 3 \end{matrix} \\
\begin{matrix} \langle 00 \rangle \mapsto 0 \\ \langle 0 \rangle \mapsto 3 \end{matrix} \arrow[uu, "\cong"'] & & \begin{matrix} \langle 01 \rangle \mapsto 0 \\ \langle 0 \rangle \mapsto 3 \end{matrix} \arrow[ul, "\cong"'] \arrow[ur, "\preceq"] & \\
& \langle 0 \rangle \mapsto 3 \arrow[ul, "\cong"'] \arrow[ur, "\cong"] & &
\end{tikzcd}
$$

IceCube: Log Based Reconciliation

In the state based, and data type based, reconcilation approaches described earlier the treatment of conflicts addopted two policies: No automated reconciliation of conflicting divergence; Conflict avoidance by constrains on allowed data type behaviours.

IceCube: Log Based Reconciliation

In the state based, and data type based, reconcilation approaches described earlier the treatment of conflicts addopted two policies: No automated reconciliation of conflicting divergence; Conflict avoidance by constrains on allowed data type behaviours.

IceCube, uses a diferent approach: Operations collected under disconnection are kept in a log and, later on, logs are collected and a reconciliation schedule is calculated. The reconciliation schedule is choosen among alternative schedules. Some of the collected operations from the input logs might be dropped.

IceCube: Log Based Reconciliation

The reconciliation process is centralized and is only triggered after collecting all replica logs in a given node. The common schedule strives to minimize the number of dropped operations, thus maximizing the preservation of user intentions. The compatibility and grouping requirements of operations are collected by a number of semantic relations between operations.

IceCube: Constraints

Static Constraints Relates two actions unconditionally, e.g. Two appointment requests for Joe at 10:00 in distinct places.

Dynamic Constraints Success or failure of a single operation depending on the current state. e.g. An overdraft in a money account.

IceCube: Primitive Static Constraints

Before Noted as $\rightarrow$. For all actions $\alpha, \beta \in$ *s*, if $\alpha \rightarrow \beta$ then α comes before β in the schedule (not necessarelly immediatly before).

mustHave Noted as $\triangleright$. For any $\alpha \in$ *s*, every action β such that $\alpha \triangleright \beta$ is also in *s* (but not necessarely in that order nor contiguosly).

These primitive constraints are composed to provide composite primitives, used in log constraints and object constraints.

IceCube: Log Constraints

preSucc predSucc(α, β) is $\alpha \rightarrow \beta \wedge \beta \triangleright \alpha$. Meaning that action β executes only after α as succeeded. e.g. A user updates a file and copies the new version, whishing that the copy is only made if the update makes it to the common log.

parcel

alternative

IceCube: Log Constraints

preSucc

parcel parcel(α, β) is $\alpha \triangleright \beta \wedge \beta \triangleright \alpha$. Meaning an all-or-nothing grouping between two actions. Exhibiting a sub-set of transactional properties. e.g. A user tentativley copies two whole directory into a third directory as a parcel, and any of the individual copies might fail.

alternative

IceCube: Log Constraints

preSucc

parcel

alternative parcel(α, β) is $\alpha \rightarrow \beta \wedge \beta \rightarrow \alpha$. Meaning only one of the two actions can be choosen. Otherwise the constraint would create a cycle. e.g. A user specifices a meeting that can either occur at 10.00 or at 11:00.

IceCube Example: Reconcilable Mail Folders

A layer is interposed between a mail client and a mail server running IMAP. And semantics are captured by constraints:

- Mailbox creation is an idempotent action.
- Renaming a folder is a parcel linking into the new location and unlinking the old one.
- Renaming the same mail folder with diferent names is a conflict.
- Changing a message while concurrently removing it is a conflict.
- Concurrently copying and deleting the same message is allowed.

The constraints, driven by application semantics, will lead to diferent proprosed common log schedules that can be ultimalty choosen by a user and commited as the new common state.

Divergence Control

Divergence can lead to conflicts and consequently to the need to reconcile (or keep divergent state) or to drop some user actions. There are several techniques that try to prevent or provide warnings when the amount of divergence reaches predefined tresholds.

Divergence Control

Work on quasi-copies/quasi-caching [Alonso,Molina,Barbara] introduces some intuitive metrics on divergence:

Temporal Validity periods or indications of critical times, such as stock markets closing hours. Some ammount of clock sinchrony may be required.

Version Limits can be imposed to the number of diferent versions that separate disconnected states, this implies knowing individual modifications.

Aritmetic Data fields that can be interpreted as values can base metrics that limit aritmetic distance under disconnection. For instance, stocks may be constrained to only vary 10%.

Composite Composite criterias can be formed by applying logical composition of base criterias.

Divergence Control

Work on the line of Epsilon-serializability [Caltan Pu et al] can use knowledge on operation types to maximise future reconciliation.

Semantic Knowledge

In general, semantic knowledge of operation types can be explored to flag portentially conflicting divergence and this can take into account dynamic constraints. An example is limiting disconnected sales when critical stock levels are approached.

MobiSnap

- Mobile Transaction Management in Mobisnap, N. Preguiça, C. Baquero et al. ADBIS-DASFAA 2000.

Adaptation of transactional system to mobility lead to the definition of notions of tentative or weak transactions when accomodating disconnected transactions. Compatibility of transactions was tipically verified by the analysis of read and write sets.
MobiSnap extended this early work by defining a set of transaction classes and extending prebious escrow techniques in order to provides additional guaranties for mobile transactions.

MobiSnap: Problems to Handle

As expected, optimistic replication is used and fragments of shared data are present on the database users mobile computers. Several problems can occur:

MobiSnap: Problems to Handle

As expected, optimistic replication is used and fragments of shared data are present on the database users mobile computers. Several problems can occur:

- Updates performed by diferent users may raise mutual conflicts.

MobiSnap: Problems to Handle

As expected, optimistic replication is used and fragments of shared data are present on the database users mobile computers. Several problems can occur:

- Updates performed by diferent users may raise mutual conflicts.
- Due to this it is often impossible do imediatly determine the result of an update.

MobiSnap: Problems to Handle

As expected, optimistic replication is used and fragments of shared data are present on the database users mobile computers. Several problems can occur:

- Updates performed by diferent users may raise mutual conflicts.
- Due to this it is often impossible do imediatly determine the result of an update.
- Mobile users may wish to update data not presently available on their machine.

Mobisnap: Synopsis

- A central database is hosted in a traditional server and will hold the primary replica of all data items.

Mobisnap: Synopsis

- A central database is hosted in a traditional server and will hold the primary replica of all data items.
- Mobile clients replicate subsets of the database items.

Mobisnap: Synopsis

- A central database is hosted in a traditional server and will hold the primary replica of all data items.
- Mobile clients replicate subsets of the database items.
- Mobile clients do updates by “mobile transactions” that are specified in an extended subset of PL/SQL (Oracle).

Mobisnap: Synopsis

- A central database is hosted in a traditional server and will hold the primary replica of all data items.
- Mobile clients replicate subsets of the database items.
- Mobile clients do updates by “mobile transactions” that are specified in an extended subset of PL/SQL (Oracle).
- Transactions state semantics by pre-conditions, post-conditions and alternatives.

Mobisnap: Synopsis

- A central database is hosted in a traditional server and will hold the primary replica of all data items.
- Mobile clients replicate subsets of the database items.
- Mobile clients do updates by “mobile transactions” that are specified in an extended subset of PL/SQL (Oracle).
- Transactions state semantics by pre-conditions, post-conditions and alternatives.
- Final result of mobile transactions is determined after commiting in the primary replica.

Mobisnap: Synopsis

- A central database is hosted in a traditional server and will hold the primary replica of all data items.
- Mobile clients replicate subsets of the database items.
- Mobile clients do updates by “mobile transactions” that are specified in an extended subset of PL/SQL (Oracle).
- Transactions state semantics by pre-conditions, post-conditions and alternatives.
- Final result of mobile transactions is determined after commiting in the primary replica.
- System support for user notification is provided (email,sms).

MobiSnap: Mobile Transaction

See Fig. 1 on paper.

MobiSnap: Interaction

In a connected environment users have “imediate” feedback on the result of the transactions they issue. On failure, alternatives can be issued in a subsequent transactions.

MobiSnap: Interaction

In a connected environment users have “imediate” feedback on the result of the transactions they issue. On failure, alternatives can be issued in a subsequent transactions.

In disconnected mode users should provide the alternatives in the tentative transaction.

MobiSnap: Interaction

In a connected environment users have “imediate” feedback on the result of the transactions they issue. On failure, alternatives can be issued in a subsequent transactions.

In disconnected mode users should provide the alternatives in the tentative transaction.

Later on, when transactions reach the primary users can be notified of the final outcome.

MobiSnap: Reservations

Reservations in MobiSnap are used to provide guaranties to mobile/tentative transactions.

Escrow A partitionable resource can be devided and allocated to diferent users. E.g. stocks.

MobiSnap: Reservations

Reservations in MobiSnap are used to provide guaranties to mobile/tentative transactions.

Escrow A partitionable resource can be devided and allocated to diferent users. E.g. stocks.

Slot Reserves the right to insert a record with pre-defined values. E.g. Scheduling a meeting in a defined period.

MobiSnap: Reservations

Reservations in MobiSnap are used to provide guaranties to mobile/tentative transactions.

Escrow A partitionable resource can be devided and allocated to diferent users. E.g. stocks.

Slot Reserves the right to insert a record with pre-defined values. E.g. Scheduling a meeting in a defined period.

Value-change Reserve the right to change some values in the database. E.g. The right to change a description of some product.

MobiSnap: Reservations

Reservations in MobiSnap are used to provide guaranties to mobile/tentative transactions.

Escrow A partitionable resource can be devided and allocated to diferent users. E.g. stocks.

Slot Reserves the right to insert a record with pre-defined values. E.g. Scheduling a meeting in a defined period.

Value-change Reserve the right to change some values in the database. E.g. The right to change a description of some product.

Value-use Reserve the right to perform a value that use a given value for some field. E.g. A salesperson reserves the right to sell some product for a given price.

Leases

Reservations held by mobile user are limited in time.

MobiSnap: System Model

- Clients keep two copies of data: Commited and Tentative. Commited versions have be confirmed by an execution in the server (but may be outdated).

MobiSnap: System Model

- Clients keep two copies of data: Commited and Tentative. Commited versions have be confirmed by an execution in the server (but may be outdated).
- When disconnected, applications can use both versions. They should reflect to users the potential weak consistency (by color codes, etc).

MobiSnap: System Model

- Clients keep two copies of data: Commited and Tentative. Commited versions have be confirmed by an execution in the server (but may be outdated).
- When disconnected, applications can use both versions. They should reflect to users the potential weak consistency (by color codes, etc).
- When connected clients gather reservations by interacting with server side reservation scripts.

MobiSnap: System Model

- Clients keep two copies of data: Commited and Tentative. Commited versions have be confirmed by an execution in the server (but may be outdated).
- When disconnected, applications can use both versions. They should reflect to users the potential weak consistency (by color codes, etc).
- When connected clients gather reservations by interacting with server side reservation scripts.
- Servers use a standard SQL database and can have non-mobile clients. Reservations must be actually enforced in the database.

MobiSnap: System Model

- Clients keep two copies of data: Commited and Tentative. Commited versions have be confirmed by an execution in the server (but may be outdated).
- When disconnected, applications can use both versions. They should reflect to users the potential weak consistency (by color codes, etc).
- When connected clients gather reservations by interacting with server side reservation scripts.
- Servers use a standard SQL database and can have non-mobile clients. Reservations must be actually enforced in the database.
- Applications have mobile transaction templates that are instantiated with user values to create actual mobile transactions.

MobiSnap: Mobile Executions

Depending on data and reservations on the mobile machine, tentative transactions can have diferent outcomes while disconnected:

Reservation commit The code has executed until commit and all steps are backed up by granted reservations. The transaction will succedd if propagated in time.

MobiSnap: Mobile Executions

Depending on data and reservations on the mobile machine, tentative transactions can have diferent outcomes while disconnected:

Reservation commit The code has executed until commit and all steps are backed up by granted reservations. The transaction will succedd if propagated in time.

Tentative commit The code has executed until commit but only some steps are backed up by granted reservations.

MobiSnap: Mobile Executions

Depending on data and reservations on the mobile machine, tentative transactions can have diferent outcomes while disconnected:

Reservation commit The code has executed until commit and all steps are backed up by granted reservations. The transaction will succedd if propagated in time.

Tentative commit The code has executed until commit but only some steps are backed up by granted reservations.

Tentative abort The code execution lead to an abort while using the tentative database state. It might suceed later if the actual state changes.

MobiSnap: Mobile Executions

Depending on data and reservations on the mobile machine, tentative transactions can have diferent outcomes while disconnected:

Reservation commit The code has executed until commit and all steps are backed up by granted reservations. The transaction will succedd if propagated in time.

Tentative commit The code has executed until commit but only some steps are backed up by granted reservations.

Tentative abort The code execution lead to an abort while using the tentative database state. It might suceed later if the actual state changes.

Unknown Currently cached data is not enough to evaluate the result of the transaction. Some fields can be absent due to partial replication.

Ad-hoc Networking and Sensor Networks

To be continued . . .