Support for Recovery in Mobile Systems

Cris Pedregal-Martin, *Member*, *IEEE Computer Society*, and Krithi Ramamritham, *Fellow*, *IEEE*

Abstract—Mobile systems increasingly are being used for production-grade data-centered applications which require system support for transactional properties. For mobile applications, transactions can hide, to some extent, the infrastructure intrinsic to mobile systems, such as disconnection from the network, dozing, and storage limitations. In this paper, we introduce a framework to understand, specify, and reason about recovery support for transactional functionality, based on the notion of guarantees (promises one subsystem makes to another) and protocols (prescriptions for correct behavior). We apply our framework to a simple mobile system scenario, yielding an abstract specification that exposes the role of each component in achieving specific transactional semantics support, such as the redo-ability of committed updates that might be lost due to a failure; it also reveals unstated assumptions necessary for the correctness of recovery support. We also show how to reason about alternative ways of obtaining the desired transactional support and the requirements on the components to support recovery and transactions.

Index Terms—Recovery specification, database transactions, mobile systems.

1 Introduction

TRANSACTIONS mask concurrency and failure details from applications and, on mobile systems, can hide infrastructure shortcomings such as disconnection from the network, dozing, and limitations on local memory. However, providing recovery support, essential for transactions, is challenging, in general, because the semantic gap between the high-level properties of recovery (e.g., failure-atomicity, durability) and their implementation (buffer and log management, disk and network access, etc.) makes it difficult to reason about the low-level consequences of changes in the desired high-level properties and the performance consequences of changes in the underlying infrastructure. Thus, it is hard to both reuse and gain confidence on the correctness of recovery design and implementation. As distributed systems, specific challenges apply to the building of recovery for mobile systems which desire some variant of database-style transactions,[1] including limitations of communication bandwidth and range, power, storage, and computation. Also, the topology of the network changes dynamically as a mobile host connects with different base stations.

This paper is aimed at 1) bringing the benefits of abstraction, and the methodologies abstraction enables, to the characterization and crafting of recovery functionality and 2) studying how mobility affects the realization of recovery properties. We introduce a framework to: a) identify the relevant subsystems and their relationships; b) describe what each subsystem expects from others —*guarantees* one subsystem gets from another; and c) specify how subsystems behave in order to create the conditions for the guarantees to be useful—*protocols* that the subsystems follow. We propose several alternative ways of achieving mobile recovery and discuss the tradeoffs involved. Using the abstractions of guarantees and protocols, we analyze the recovery-related requirements imposed by these alternatives on the infrastructure.

1. In the sequel, we refer to these as simply *mobile systems*.

- *C. Pedregal-Martin is with the Department of Computer Science, University of New Mexico, Albuquerque, NM 87131. E-mail: cris@cs.unm.edu.*
- *K. Ramamritham is with the Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, India. E-mail: krithi@cse.iitb.ac.in.*

Manuscript received 15 July 2001; revised 15 May 2002; accepted 30 May 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 116661.

2 System Model and FL Framework

In this section, we first characterize the mobile systems under study with an informal system model and assumptions. We then introduce the elements of the Failure Liveness (FL) framework, including the formal definitions of guarantees and protocols.

We consider systems comprised of fixed hosts and mobile hosts. *Fixed* hosts are always connected via network; some or all of them are *base stations* which can communicate (e.g., wirelessly) with nearby mobile hosts. *Mobile* hosts move and may sequentially connect to any base station, but at most one at a time. We consider failures which cause the loss of volatile state (program state and data in volatile memory) in a host and we assume fail-stop behavior on all the subsystems. The recovery goal is supporting Failure Atomicity and Durability, regardless of where the mobile host was when it 1) issued operation(s) and 2) committed the corresponding transaction. The system should also be able to preserve the partial order of operations necessary to commit or abort transactions, e.g., via some variant of the WAL protocol [1], [2]. In this paper, "recoverable" means that there exists enough "recovery information" for an appropriate repair algorithm to restore state correctly.

We do not delve into concurrency control, instead making the simplifying assumption that, if the home server of a data item allows a data item to migrate to a mobile host, then the mobile host can perform operations on the item. We make no assumptions about who initiates or manages failure recovery and when: It could be the mobile host itself as part of its own recovery protocol.

We describe recovery using the FL framework to: 1) identify the relevant subsystems and their relationships; 2) describe (via *guarantees* and *forced protocols*) what each subsystem expects from another or accomplishes in spite of failures; and 3) specify (via *protocols*) how subsystems behave in order to create the conditions for the guarantees to be useful or applicable. Next, we formalize these concepts via histories.

2.1 History and Events

H is a partial order on events (actions) which denotes the complete history of the system, with $<$, the partial order predicate relation "happens before" on H. To state that an event must take place, we simply require that the event appear in H.

p, q, send, ... are events (actions) in the history H. (We use greek letters $\alpha, \beta, \ldots$ for *event variables*.) Events can be data events (e.g., write, update), transaction/recovery (commit) events, and ad hoc (handoff). When relevant, we indicate event parameters in italics and where it takes place with the subsystem's name subscripted (see Table 1).

2.2 Pre and Postactions

Let α, β be events in H:

$$\mathrm{PreAction}(\beta) = \alpha \equiv_{def} \beta \in H \Rightarrow \alpha \in H \wedge \alpha < \beta.$$

This says that if β occurs, then α must have occurred also and before β.

$$\mathrm{PostAction}(\alpha) = \beta \equiv_{def} \alpha \in H \Rightarrow \beta \in H \wedge \alpha < \beta.$$

Once α occurs, then eventually β will also occur. We also introduce ExclAction:

$$\mathrm{ExclAction}(\alpha) = \beta \equiv_{def} \alpha \in H \Rightarrow \neg\beta \in H.$$

TABLE 1
Example Mobile Events/Actions

notation	*name*	*description*
p_M	data operation	data operation p executed on host M
$\mathsf{send}_A(p_M, B)$	operation send	A sends operation p to host B to be executed on M
$\mathsf{inpt}_M(p)$	local input	user at M invokes operation p to be executed locally
$\mathsf{hndf}_A(M, B)$	mobile handoff	M leaves A's cell for B's cell
$\mathsf{slog}_A(p)$ / $R(p)$	stable log / rec. info	recovery info (R(p)) for operation p stable logged by A

The occurrence of α precludes that of β.

2.3 Guarantees

A *requestor* requests a guarantee from a *guarantor* subsystem which, in response, may enable the guarantee. If enabled, the requestor may later invoke an action that triggers the guarantee. Once the guarantee has been triggered, the guarantor *must* (eventually) perform the discharge action. Formally, a guarantee is a relation on a 4-tuple of events

$$\begin{aligned} &Guarantee(\text{Genable}, \text{Gdisable}, \text{Gtrigger}, \text{Gdischarge}) \\ &\equiv_{def} \text{PreAction}(\text{Gtrigger}) = \text{Genable} \wedge \text{ExclAction}(\text{Gtrigger}) \\ &= \text{Gdisable} \wedge \text{PostAction}(\text{Gtrigger}) = \text{Gdischarge}. \end{aligned}$$

With each of the actions we associate a predicate which becomes true if and only if the corresponding action takes place. Thus, for a guarantee G, $enabled_G$ is true only after the action Genable occurs and similarly for the other three predicates. We abuse notation and mix actions and their predicates when convenient.

2.4 Protocols and Forced Protocols

A protocol constrains the order of events and is defined: $\alpha \rightarrow \beta \equiv_{def} \text{PreAction}(\beta) = \alpha$. This says that if β takes place, α must have taken place before β. A forced protocol stipulates forward causality of events. Formally: $\alpha \mapsto \beta \equiv_{def} \text{PostAction}(\alpha) = \beta$. This statement says that once α takes place, then, eventually, β takes place after α.

With these elements, we specify a recovery scheme in the following sense: Stating a guarantee means that the subsystem that offers it supports its semantics, e.g., the ability to deliver the recovery information in spite of failures (when appropriately triggered and assuming the guarantee had been enabled). A protocol means that the scheduling component of the appropriate subsystem will allow only event sequences that conform to the protocol. Both guarantees and protocols are thus requirements for the subsystems that support them as much as assumptions for those that use them.

3 Recovery in Mobile Systems

We consider the following scenario: Mobile host M does some operations at fixed A, then is handed off to fixed B, and possibly needs recovery at B. M expects the same recovery support whether at A or at B. We specify this by stating that a handoff requires that B obtain a new guarantee from A, promising that A will supply B with the recovery information necessary to honor A's guarantee to M should M require it.

3.1 Eager Scheme: Recovery Information Follows Mobile

An obvious approach to satisfying the mobile recovery requirement is to make recovery information follow the mobile, thus all recovery information generated by the mobile at all base stations must be available at the current base station. We specify[2] this eager scheme via 1-4 (here, A, B are base stations and M a mobile):

1. A adheres to $\text{Pslogsend}(A, M)$, to ensure recovery information is stored at A and M adheres to $\text{Pslog}(M, A)$.
2. M obtains $\text{Gslog}(A, M)$ from A, which is all that is necessary to support recovery for M as long as M remains in A. This is enabled by $\text{Pslogsend}(A, M)$ and by $\text{Pslog}(M)$.
3. A adheres to $\text{Phndf}_E(A, M, B)$ to ensure recovery information is available at the destination base station of a handoff.
4. B obtains $\text{Ghndf}_E(A, B)$ from A, enabled by complying with $\text{Phndf}_E(A, M, B)$. This is the guarantee that the recovery information will be at B while M is at B. (See also Tables 1, 2, and 3).

We can demonstrate that mobile recovery is ensured by the above specifications by considering the cases of M at A (before the handoff) and M at B (after). Before the handoff, M counts on having $\text{Gslog}(A, M)$ enabled and can trigger it by starting recovery.

After the handoff, B can recover p because the handoff enabled $\text{Ghndf}_E(A, B)$. (The handoff is an atomic action, so this analysis suffices.) Notice that, in this "eager" approach, as soon as M leaves A for B (i.e., the handoff is complete), A no longer needs to keep any recovery information for M. In particular, if M disconnects and then reconnects, requesting recovery assistance, the system will have to locate the last base station M visited before failure as that is where the recovery information is.

Fig. 1 portrays how the recoverability of operation p is supported in the eager handoff scheme case. The relationships between subsystems and the protocols they follow and guarantees they offer and use are represented for the two situations, before and after a handoff. The top level indicates the events at the mobile's level of abstraction, namely the execution of p while at A and the subsequent migration (indicated by a handoff) to B. The next level indicates the respective recovery properties, i.e., that p be recoverable while M is at each of A and B. For each level down, we indicate how the current level's guarantee is supported by guarantees and protocols at the subsystem in the level below. (The type of subsystems is indicated by the leftmost column.)

For example, on the right side of the figure (below and to the right of the heading "M migrates to B"), we depict the top levels of the support for recoverability for the same operation p that took place in A, once M is at a different base station B. The handoff is done adhering to eager handoff protocol $\text{Phndf}_E(A, M, B)$. Recoverability is achieved by adherence to this protocol and the eager guarantee $\text{Ghndf}_E(A, B)$ which it enables.

We indicate in the next level how $\text{Ghndf}_E(A, B)$ is supported by the same guarantee $\text{Gslog}(A, M)$ at A, coupled with the two communication guarantees (to and fro A and B) and their protocols. All these are at the same mobile host/base station level as the guarantees they support, showing an example of composition of

2. We omit "well-formedness" protocols, see Section 3.2.1.

TABLE 2
Mobile System Guarantees

name	*enable*	*trigger*	*discharge*	*comments*
$\mathsf{Gcomm}(A,B)$	TRUE	$\mathsf{send}_A(x,B)$	$x \in B$	$\forall x, \forall A, B$ *any* hosts
$\mathsf{Gslog}(A)$	$\mathsf{slog}_A(p)$	A scans log / recovers	$R(p) \in A$	A is a *fixed* host
$\mathsf{Gslog}(A,M)$	$\mathsf{slog}_A(p_M)$	A starts recovery M	$R(p_M) \in M$	M at A for recovery
$\mathsf{Gvlog}(A,M)$	$\mathsf{vlog}_A(p_M)$	A starts recovery M	$R(p_M) \in M$	M at A for recovery
$\mathsf{Ghndf}_L(A,B)$ (lazy handoff)	$\mathsf{slog}_B(hndf_A(M,B)) \wedge$ $\mathsf{slog}_A(p_M)$	M up after failure, so B starts recovery M	$R(p_M) \in M$	M failed at $B \neq A$ B manages recovery
$\mathsf{Ghndf}_E(A,B)$ (eager handoff)	$\mathsf{send}_A(R(p_M),B) \wedge$ $\mathsf{slog}_B(p_M)$	M up after failure, so B starts recovery M	$R(p_M) \in M$	M failed at $B \neq A$ B manages recovery

TABLE 3
Mobile System Protocols

name	*action* $\rightarrow$ *action*	*comments*
$\mathsf{Pinpt}(M)$	$\mathsf{inpt}_M(p) \rightarrow \mathsf{p}_M$	operation caused by input or
$\mathsf{Psend}(A,M)$	$\mathsf{send}_A(p_M,M) \rightarrow \mathsf{p}_M$	message from another host A
$\mathsf{Pslog}(M,A)$	$\mathsf{slog}_A(p_M) \rightarrow \mathsf{p}_M$	WAL on host M (log sent to A)
$\mathsf{Pvlog}(M,A)$	$\mathsf{vlog}_A(p_M) \rightarrow \mathsf{p}_M$	WAL on host M (log volatile in A)
$\mathsf{Pslogsend}(A,M)$	$\mathsf{slog}_A(p_M) \rightarrow \mathsf{send}_A(p_M,M)$	WAL base A to M (M at A)
$\mathsf{Phndf}_\mathsf{E}(A,M,B)$	$\mathsf{send}_A(R(p_M),B) \rightarrow \mathsf{hndf}_A(M,B)$	eager handoff
$\mathsf{Phndf}_\mathsf{E}(A,M,B,S)$	$\mathsf{send}_A(R(p_M),S) \rightarrow \mathsf{hndf}_A(M,B)$	eager central server handoff
$\mathsf{Phndf}_\mathsf{L}(A,M,B)$	$\mathsf{slog}_B(hndf_A(M,B)) \rightarrow \mathsf{hndf}_A(M,B)$	lazy handoff

guarantees and protocols. Guarantee $\mathrm{Ghndf}_E(A,B)$ is completed by guarantee $\mathrm{Gloghndf}_B$, which ensures the logged handoffs survive failures.

This figure also shows how handoffs present another example of *composition* of guarantees. To illustrate: A mobile M obtains a high-level ("central") guarantee while, at base station A's cell, and, while M stays within A, the guarantee may be supported locally. When M moves to B, the system must, in effect, construct a new guarantee to maintain its promise to M. This new guarantee, usable in B, is a composition of the original guarantee in A and guarantees about A's reachability from B (through the fixed network).

In this eager scheme, the cost of the handoff grows with the length of the mobile's trajectory in the system. The next section considers this and other dimensions that give rise to variants for the mobile systems.

3.2 Alternatives to the Eager Approach

We consider here whether the recovery information is stored in persistent storage at some node or nodes within the fixed network.

1. Lazy. Recovery information is made persistent at the base station where mobile was when the operation was carried out and propagated only if and when necessary.
2. Eager, to central server. Recovery information is always propagated from a base station to a central server S at handoff time at the latest.
3. Lazy, to central server. Recovery info is propagated on demand from the base station to the central server S. This admits several variants: a) Propagation is done in batches, under the control of a buffer-management subsystem at the base station (e.g., triggered by a transaction commit event); b) propagation is triggered by resource or time events, e.g., a low watermark for free memory at the base station; c) propagation is triggered by request for recovery information from other base stations (possibly via the central server S).

Here, the mobile communicates directly with the fixed network; however, recovery info may reach the fixed network via a second mobile host M'. M' in turn can do that in a lazy or eager manner. Once the data reaches the fixed network, the cases are as described in the preceding list.

3.2.1 Mobile Protocols and Guarantees

The scenarios outlined above share many characteristics. We show that they can be specified by different instantiations (for different subsystems) of a small set of guarantees and protocols, which we describe here (see also Tables 2 and 3).

$\mathrm{Gcomm}(A,B)$ expresses the expectation that communication between hosts is reliable. It includes mobile hosts, encoding the assumption that a mobile host will *eventually* communicate.

$\mathrm{Gslog}(A)$ means that host A is endowed with its own recovery support, e.g., stable storage. Stable logging is just a possible implementation.

$\mathrm{Gslog}(A,M)$ is similar to $\mathrm{Gslog(A)}$ but from A to M.

$\mathrm{Gvlog}(A,M)$ is the version of $\mathrm{Gslog(A)}$ for a host A lacking stable storage.[3] See Section 3.4.

$\mathrm{Ghndf}_L(A,B)$ is the lazy handoff guarantee, a promise from A that it will supply recovery information for operation p [4] if and

3. A need not have persistent storage, it suffices that A not fail when M fails.

4. To reduce clutter we do not list operation p as a parameter.

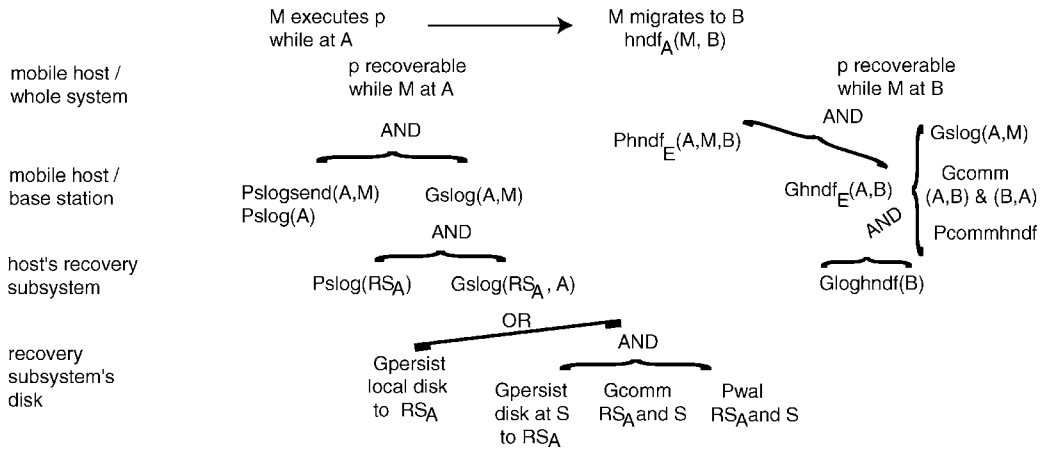

Fig. 1. Operation recoverability in terms of underlying guarantees and protocols.

when B assumes M's recovery (see Section 3.3). Thus, A remains obligated to keep the recovery information around.

$\mathrm{Ghndf}_E(A, B)$ is the eager handoff guarantee, which ensures that the recovery information has been furnished to the base station (B). Contrast to $\mathrm{Ghndf}_L(A, B)$.

To avail themselves of the guarantees, subsystems must follow protocols. The classic database case is WAL, which mandates logging an update (to obtain the recovery guarantee from the recovery system) before applying the update. Although obvious, we include the first two protocols below (Psend, Pinpt) to show their role in prescribing well-formedness. The remaining protocols are related to guarantees in that they ensure that certain operations only happen after the enable action of some appropriate guarantee (see also Table 3):

$\mathrm{Psend}(M), \mathrm{Pinpt}(A, M)$. These are "well-formedness" protocols: They prescribe where an operation (p at M) may originate: from an input or from a message from another host.

$\mathrm{Pslog}(M, A)$. Companion to $\mathrm{Gslog}(A, M)$. Ensures log is stably written before an operation (WAL).

$\mathrm{Pvlog}(M, A)$. Like the previous protocol, this one (which pairs with $\mathrm{Gvlog}(A, M)$) ensures that a volatile log is written before an operation.

$\mathrm{Pslogsend}(A, M)$. Specific to some mobile system schemes in which messages containing operations are logged by the base station before sending them on to their destination mobile host (see [3]).

$\mathrm{Phndf}_E(A, M, B)$. Ensures that M's (p) recovery information is sent from A to B eagerly at the handoff, which contributes to enabling $\mathrm{Ghndf}_E(A, B)$.

$\mathrm{Phndf}_E(A, M, B, S)$. A variant of the eager handoff protocol (see the previous one) in which the information is sent to a central server S instead.

$\mathrm{Phndf}_L(A, M, B)$. Prescribes that the handoff itself be logged at destination host B so that B knows later to ask A for the needed recovery information. See $\mathrm{Ghndf}_L(A, B)$ above and Section 3.3.

3.3 Lazy Scheme: Recovery Information Remains at Original Base Stations

In Section 3.1, we specified an eager approach in which recovery information was always propagated at handoff. The lazy variant leaves the recovery information at the base stations where it originated (say, A), relying on them later to support recovery if and when it is necessary. We assume here that recovery is managed by the base station last associated with a Mobile host prior to the initiation of recovery. Instead of receiving the mobile's recovery information at handoff, the destination host (say, B) obtains a guarantee that it will be available from A. Of course, B must be able to communicate with A after the handoff, but that is a given in our system model (see Section 2 and Table 2). The specification for this lazy case is given by (cf. Section 3.1) 1-5:

1. A follows $\mathrm{Pslogsend}(A, M)$, and M to $\mathrm{Pslog}(M, A)$, ensuring that recovery information is stored at A.
2. M adheres to $\mathrm{Pslog}(M)$, write-ahead logging.
3. A grants M $\mathrm{Gslog}(A, M)$, enough to support recovery for M as long as M remains in A. This is enabled by $\mathrm{Pslogsend}(A, M)$ and by $\mathrm{Pslog}(M)$.
4. Those protocols also enable $\mathrm{Gslog}(A)$, guaranteeing that A will have access to the recovery information (with the goal, in this case, of passing it to B).
5. B can get recovery information $R(p)$ from A because B and A can always communicate. This is given by the two instances, $\mathrm{Gcomm}(B, A)$ for B and $\mathrm{Gcomm}(A, B)$ for A, which hold for all the hosts.

Recovery in the pre-handoff portion (while M at A) is the same as in the eager case, so we omit it from the discussion. After the handoff, we rely on the communication guarantees Gcomm and A's own persistence guarantee $\mathrm{Gslog}(A)$ to show that B can help recover M. Unlike the eager case, because no transfer of recovery information takes place at handoff, we do not specify a protocol on handoffs (i.e., with handoff as its righthand side (cf. $\mathrm{Phndf}_E(A, M, B)$ in Table 3).

Let us now compose $\mathrm{Gslog}(A)$ and Gcomm to transfer $R(p)$ from A to B and attempt to prove that B can always obtain $R(p)$. Suppose there is a failure of B after the handoff and before M's recovery. It is possible that B may forget that M had visited A earlier and that A was holding $R(p)$. Without this knowledge, the recovery engineered by B will be incomplete, so we add the following protocol and guarantee, defined in Tables 2 and 3:

6. B adheres to the handoff protocol $\mathrm{Phndf}_L(A, M, B)$ to ensure handoff information is available (see next item).
7. B guarantees M's recovery via $\mathrm{Ghndf}_L(A, B)$, enabled by $\mathrm{Phndf}_L(A, M, B)$.

With this addition, the lazy scheme specification is complete in that it ensures recovery in spite of failures. Strictly, Gcomm and Gslog are no longer necessary. Ghndf_L subsumes Gcomm and Gslog, which suggests that the two can implement it, which is exactly the misleading intuition that overlooks the need for stabilizing the handoff chain. Of course, Gcomm and Gslog are

indeed useful to achieve Ghndf_L, but that more is needed is apparent by examining their respective enables (see Table 2).

3.4 Storing Recovery Information at a Central Server

Previously, we examined two schemes (eager and lazy) in which the base stations have the capability to store recovery information and manage recovery. In some mobile systems, base station hosts are little more than conduits between a wireless and a wired network protocol, lacking much by way of storage and computing power. We consider here a scenario in which a separate central server host S is in charge of managing recovery.[5]

Because base stations and server are connected, recovery information can propagate from base stations to server at handoff. Before handoff, the propagation is lazy (e.g., driven by resources or transaction events). The specification is similar to the eager case (see Section 3.1):

1. M adheres to $\mathrm{Gvlog}(A, M)$ (volatile logging at A). $\mathrm{Pvlog}(M, A)$ makes sense because of the guarantee A offers to M, which requires A to stabilize its log (next item).
2. $\mathrm{Gvlog}(A, M)$ guarantees M recovery at A. This imposes a requirement on A of preserving the recovery information, but leaves undefined how A satisfies it, e.g., by using its own disk or by shipping it elsewhere.
3. A adheres to $\mathrm{Phndf}_E(A, M, B, S)$ to ensure recovery information is available at server S.
4. B obtains $\mathrm{Ghndf}_E(A, S)$ from A, enabled by following $\mathrm{Phndf}_E(A, M, B, S)$. This guarantees that the recovery information will be at B while M is at B.

Before the handoff, M has guarantee $\mathrm{Gvlog}(A, M)$ from A which is enabled by the WAL variant $\mathrm{Gvlog}(A, M)$. After the handoff, M would request recovery assistance to B, which relies on $\mathrm{Ghndf}_E(A, S)$, i.e., B will go to S for the recovery information.

This scheme is similar to the lazy scheme of Section 3.3 in that the recovery information is not at the base station, but it is simpler in that its location is well-known.

3.5 Transferring Recovery Information to Another Mobile

We outline here a scenario in which mobile host M may transfer recovery information to another mobile host M', which in turn will forward it to a base station. We assume M sends its recovery information to M' and we ignore the specification of M's own operations so that we can treat the recovery information passed on by M as the operations M' is keeping recoverable. We only show here (1-4) how the recovery is maintained while at the original base station A, omitting the treatment of handoffs, which can be done in any of the ways developed in the previous examples.

1. M obtains guarantee $\mathrm{Gvlog}(M', M)$ ensuring volatile logging of its operations at M'. $\mathrm{Pvlog}(M, M')$ makes M' responsible for stabilizing its log (see next item).
2. M adheres to $\mathrm{Pslog}(M, M')$, write-ahead logging, to avail itself of the guarantee $\mathrm{Gvlog}(M', M)$.
3. M' obtains $\mathrm{Gslog}(A, M)$ from A, which is all that is necessary to support recovery for as long as M remains in A. This is enabled by $\mathrm{Pslogsend}(A, M)$ and by $\mathrm{Pslog}(M')$.
4. M' adheres to $\mathrm{Pslog}(M, A)$, and treats $R(p_M)$ as its own $p_{M'}$ to ensure recovery info from M is logged at A.

This outline suffices to indicate the similarities and differences with the previous examples. A complete specification requires adding events for powerdown, fade, etc.

5. We assume there is a single server to keep the treatment simple.

4 Related Work and Conclusions

Building recovery in transactional database systems is well-studied [1], [4], [5], [2], with much less work in both extending it to other domains and formalizing and abstracting its well-understood underlying principles. Examples of extending recovery beyond databases include [6], [7] (applications), [8], [9] (workflows). For mobile systems specifically, work includes [10], [11] (transactional support), [12], [13], [14] (database consistency), [15], [16] (concurrency control and correctness). The scenarios in this paper follow [3] (mobile recovery) and [17] (logging messages). There is little work on formalizing recovery [18], [19] and it does not address introducing abstraction.

In this paper, we discussed and specified mobile system recovery, showing how operations by a mobile host are recoverable under different strategies. The variants depend on where the data and the recovery information are stored, how they are propagated through the system, where operations on data, normal, and repair recovery actions take place. The solutions varied according to these dimensions, which are, in fact, implementation dimensions. The central requirement of recoverability was assured by different combinations of lower-level guarantees and protocols.

By describing the mobile system and its behavior in terms of guarantees and protocols, we obtained the following benefits: First, we used abstraction to separate what a component can expect from another, e.g., the mobile host's (recovery) expectations of the fixed hosts. This also shows what the system and each component must provide to others and documents precisely the challenges of providing recovery under mobility. For example, in showing handoff handling in terms of protocols and guarantees, we exposed assumptions that were implicit in the discussion of recovery of [3].

Second, in describing the alternatives for handoffs, we used abstraction to characterize the broad requirement (a more abstract guarantee) that all solutions must satisfy and then precisely showed how that requirement is satisfied by the eager and the lazy approach.

Third, we briefly showed how the guarantees and protocols used to decompose and synthesize the higher-level recovery properties can themselves be decomposed into guarantees and protocols cast in terms of the lower levels of implementation (Fig. 1). A complete unraveling of guarantees and protocols down to the infrastructure can be found in [20], [21].

References

[1] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques.* San Mateo, Calif.: Morgan Kaufmann, 1993.

[2] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Trans. Database Systems,* vol. 17, no. 1, pp. 94-162, Mar. 1992.

[3] D.K. Pradhan, P. Krishna, and N.H. Vaidya, "Recoverable Mobile Environments: Design and Trade-Off Analysis," Technical Report 95-053, Dept. of Computer Science, Texas A&M Univ., College Station, 1995.

[4] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems.* Reading, Mass.: Addison-Wesley, 1987.

[5] L.-F. Cabrera, J.A. McPherson, P.M. Schwarz, and J.C. Wyllie, "Implementing Atomicity in Two Systems: Techniques, Tradeoffs and Experience," *IEEE Trans. Software Eng.,* vol. 19, no. 10, pp. 950-961, Oct. 1993.

[6] D.B. Lomet and M.R. Tuttle, "Logical Logging to Extend Recovery to New Domains," *Proc. ACM SIGMOD Conf.,* pp. 73-84, June 1999.

[7] R.S. Barga and D.B. Lomet, "Phoenix: Making Applications Robust," *Proc. ACM SIGMOD Conf.,* pp. 562-564, June 1999.

[8] M.U. Kamath and K. Ramamritham, "Failure Handling and Coordinated Execution of Concurrent Workflows," *Proc. 14th IEEE Int'l Conf. Design Eng. (ICDE),* Feb. 1998.

[9] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and Implementation of Exceptions in Workflow Management Systems," *ACM Trans. Database Systems,* pp. 405-451, Sept. 1999.

[10] R. Alonso and H.A. Korth, "Database System Issues in Nomadic Computing," *Proc. 1993 ACM SIGMOD Conf.,* pp. 388-392, 1993.

[11] D. Barbara, "Mobile Computing and Databases—A Survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, pp. 108–117 Jan./Feb. 1999.
[12] P.K. Chrysanthis, "Transaction Processing in Mobile Computing Environment," *Proc. IEEE Workshop Advances in Parallel and Distributed Systems*, pp. 77-83, 1993.
[13] G.D. Walborn and P.K. Chrysanthis, "Supporting Semantics-Based Transaction Processing in Mobile Database Applications," *Proc. Symp. Reliable Distributed Systems*, pp. 31-40, 1995.
[14] S. Mazumdar and P.K. Chrysanthis, "Achieving Consistency in Mobile Databases through Localization in PRO-MOTION," *Proc. Int'l Conf. and Workshop Database and Expert Systems Applications (DEXA)*, pp. 82-89, 1999.
[15] S.K. Madria and B.K. Bhargava, "A Transaction Model for Mobile Computing," *Proc. Int'l Database Eng. and Application Symp.*, pp. 92-102, 1998.
[16] S.K. Madria and B.K. Bhargava, "On the Correctness of a Transaction Model for Mobile Computing," *Proc. Int'l Conf. and Workshop Database and Expert Systems Applications*, pp. 573-583, 1998.
[17] B. Yao, K.-F. Ssu, and W.K. Fuchs, "Message Logging in Mobile Computing," *Proc. Symp. Fault-Tolerant Computing*, pp. 294-301, 1999.
[18] D. Kuo, "Model and Verification of a Data Manager Based on Aries," *ACM Trans. Database Systems*, vol. 21, no. 4, pp. 427-479, Dec. 1997.
[19] C. Wallace, Y. Gurevich, and N. Soparkar, "A Formal Approach to Recovery in Transaction-Oriented Database Systems," *Springer J. Universal Computer Science*, vol. 3, no. 4, pp. 320-340, Apr. 1997.
[20] C. Pedregal-Martin and K. Ramamritham, "Guaranteeing Recoverability in Electronic Commerce," *Proc. Third Int'l Workshop Advanced Issues of E-Commerce and Web-Based Information Systems*, pp. 144-155, June 2001.
[21] C. Pedregal-Martin, "Transaction Recovery in Databases and Beyond," PhD thesis, Univ. of Massachusetts, Amherst, 2001.

Multiversion Data Broadcast

Evaggelia Pitoura, *Member*, *IEEE Computer Society*, and Panos K. Chrysanthis, *Member*, *IEEE*

Abstract—Recently, broadcasting has attracted considerable attention as a means of disseminating information to large client populations in both wired and wireless settings. In this paper, we consider broadcasting multiple versions of data items to increase the concurrency of client transactions in the presence of updates. We introduce various techniques for organizing multiple versions on the broadcast channel. Performance results show that the overhead of supporting multiple versions can be kept low while providing a considerable increase in concurrency. Besides increasing the concurrency of client transactions, multiversion broadcast provides clients with the possibility of accessing multiple server states in a single broadcast cycle. Furthermore, multiversioning increases the tolerance of client transactions of disconnections from the broadcast channel.

Index Terms—Mobile computing, broadcast, transaction management, versioning, consistency.

1 Introduction

ALTHOUGH the concept of broadcast delivery is not new, recently, data dissemination by broadcast has attracted considerable attention due to the physical support for broadcast provided by an increasingly important class of networked environments such as by most wireless computing infrastructures, including cellular architectures and satellite networks [10]. The use of broadcast for disseminating information to large client populations is also motivated by the explosion of data intensive applications created by the dramatic improvements in global connectivity and the popularity of the Internet. In such a setting, the server repetitively broadcasts data to a number of clients without any specific data request. Clients monitor the broadcast channel and retrieve the data items that they may need as they appear on the broadcast channel. Applications typically involve a small number of servers and a much larger number of clients with similar interests, operating in read-only mode. Examples include stock trading, electronic commerce applications, such as auction and electronic tendering, and networks of sensors.

As broadcast-based systems continue to evolve, more and more sophisticated client applications will require reading current and consistent data, despite updates at the server. In most related research, updates are mainly treated in the context of caching at the client (e.g., [4], [2]). In this case, the focus is on cache coherency; there are no transactional semantics. Transactions and broadcast were first discussed in the Datacycle project [5], where special hardware was used to detect changes of values read by transactions and thus ensure consistency. Recent work involves the development of new correctness criteria for transactions in broadcast environments [12], as well as the deployment of the broadcast medium for transmitting concurrency control related information to clients so that part of transaction management can be undertaken by them [3]. In our previous work [8], we proposed and comparatively studied a suite of invalidation-based techniques to ensure the consistency of client read-only transactions.

- *E. Pitoura is with the Department of Computer Science, University of Ioannina, GR 45110 Ioannina, Greece. E-mail: pitoura@cs.uoi.gr.*
- *P.K. Chrysanthis is with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: panos@cs.pitt.edu.*

Manuscript received 15 July 2001; revised 15 May 2002; accepted 4 June 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 116692.