A REPLICATED FILE SYSTEM FOR RESOURCE CONSTRAINED MOBILE DEVICES[1]

João Barreto and Paulo Ferreira[2]
INESC-ID/IST
Rua Alves Redol N.º 9, 1000-029 Lisboa, Portugal
[joao.barreto, paulo.ferreira]@inesc-id.pt

ABSTRACT

The emergence of more powerful and resourceful mobile devices, as well as new wireless communication technologies, is turning the concept of ad-hoc networking into a viable and promising possibility for ubiquitous information sharing. However, the inherent characteristics of ad-hoc networks bring up new challenges for which most conventional systems don't provide an appropriate response. Namely, the lack of a pre-existing infrastructure, the high topological dynamism of these networks, the relatively low bandwidth of wireless links, as well as the limited storage and energy resources of mobile devices are issues that strongly affect the efficiency of any distributed system intended to provide ubiquitous information sharing.

In this paper we describe Haddock-FS, a transparent replicated file system designed to support collaboration in the novel usage scenarios enabled by mobile ad-hoc environments. Haddock-FS is based on a highly available optimistic consistency protocol. In order to support co-present collaborative activities in mobile ad-hoc environments, it provides stronger consistency guarantees during operation within highly connected ad-hoc groups of mobile devices. In order to effectively cope with the network bandwidth and device memory constraints of these environments, Haddock-FS employs a limited size log truncation scheme and a cross-file, cross-version content similarity exploitation mechanism.

KEYWORDS

Distributed file systems, optimistic replication, mobile ad-hoc networks, ubiquitous computing.

1. INTRODUCTION

The evolution of the computational power and memory capacity of mobile devices, combined with their increasing portability, is creating computers that are more and more suited to support the concept of ubiquitous computation (Weiser 1991). As a result, users are progressively using mobile devices, such as handheld or palmtop PCs, not only to perform many of the tasks that, in the past, required a desktop PC, but also to support innovative ways of working that are now possible.

At the same time, novel wireless communication technologies have provided these portable devices with the ability of easily interacting with other devices through wireless network links. Inevitably, effective ubiquitous information access is a highly desirable goal.

Many real life situations already suggest that users could benefit substantially if allowed to cooperatively interact using their mobile devices and without the requirement of a pre-existing infrastructure. A face-to-face work meeting is an example of such a scenario. The meeting participants usually co-exist within a limited space, possibly for a short period of time and may not have access to any pre-existing fixed infrastructure. Under such co-present collaborative activities (Luff and Heath 1998), participants hold, manipulate and exchange documents that are relevant to the purposes of the meeting.

If each participant holds a mobile device with wireless capabilities, a spontaneously formed wireless network can serve the purposes of the meeting. This way, a report held at one participant's handheld device might be shared with the remaining meeting participants' devices, while its contents are analyzed and discussed. Furthermore, each participant might even update the shared report's contents, thus contributing on

[1] This work was partially funded by Microsoft Research.
[2] Partially funded by FCT/FEDER.

the ongoing collaborative activity. These wireless networks, possibly short-lived and formed just for the needs of the moment, without any assistance from a pre-existing infrastructure, are normally referred to as mobile ad-hoc networks (Corson and Macker 1999).

One interesting solution for ubiquitous information sharing and manipulation for mobile networks and, in particular, ad-hoc environments, is through the use of a distributed file system. This approach allows already existing applications to access shared files in a transparent manner, using the same programming interface as that of the local file system. However, the nature of the scenarios we are addressing entails significant challenges to an effective DFS solution to be devised.

The high topological dynamism of mobile ad-hoc networks entails frequent network partitions. On the other hand, the possible absence of a fixed infrastructure means that most situations will require the network services to be offered by mobile devices themselves. Such devices are typically severely energy constrained. As a result, the services they offer are susceptible of frequent suspension periods in order to save battery life of the server's device. From the client's viewpoint, such occurrences are similar to server failures.

These aspects call for solutions that offer high availability, in spite of the expectedly frequent network partitions and device suspension periods. Pessimistic replication approaches, employed by conventional distributed file systems, such as NFS (Nowicki 1998) or AFS (Morris et al 1986), are too restrictive to fulfill such a requirement. On the other hand, optimistic replication strategies offer weak consistency guarantees, which may not reflect the expectations of users and applications, mainly in co-present collaborative activities. Hence, they ought to be complemented with stronger consistency guarantees in order to be adaptable to a wide spectrum of application domains, with differing correctness criteria and, consequently, distinct consistency requirements.

Whichever strategy is taken, it must take into account the important limitations in memory resources and processing power of typical mobile devices, as well as the reduced bandwidth of wireless links, when compared to other wired technologies.

In this paper we describe Haddock-FS, a transparent peer-to-peer replicated file system. Each mobile device is able to offer file system services upon the files it locally stores. The flexibility brought by a peer-to-peer model enables Haddock-FS to support a broad set of mobile ad-hoc network usage scenarios.

Namely, mobile peers may operate in isolation upon their local file replicas, when network connectivity to other mobile peers is weak or non-existent. Furthermore, multiple mobile devices may form, in an ad-hoc fashion, a group of peers that cooperatively share and manipulate common files, without any assistance from a fixed infrastructure.

Haddock-FS is based on a highly available optimistic consistency protocol. In order to support co-present collaborative activities in mobile ad-hoc environments, it provides stronger consistency guarantees during operation within highly connected ad-hoc groups of peers. Furthermore, in order to effectively cope with the resource constraints of mobile devices, Haddock-FS employs mechanisms to reduce the main memory and network usage needed; namely, the update log and update content propagation between peers. A limited size log truncation scheme and a cross-file, cross-version content similarity exploitation mechanism are used for such purposes.

The rest of the paper is organized as follows: Section 2 discusses the architecture, while Section 3 describes the actual implementation that was used to obtain the experimental results that are presented in Section 4. Finally, Section 5 describes related work and Section 6 draws some conclusions.

2. ARCHITECTURE

2.1. File System Consistency

Each Haddock-FS mobile peer constitutes a *replica manager*, which is able to receive file system requests and perform them upon its local replicas. The underlying replication mechanisms are transparent to applications, which may access Haddock-FS's services by using the same API as the one exported by the local file system. Provided the accessed files are locally replicated at a given Haddock-FS peer, the file system services will be available, independently of its current network connectivity. The following sections describe how consistency amongst file replicas is achieved.

Figure 1. Replica state maintained by Haddock-FS: stable value and update log, containing both stable and tentative updates, ordered by their DVVs.

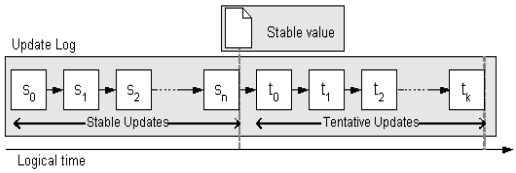
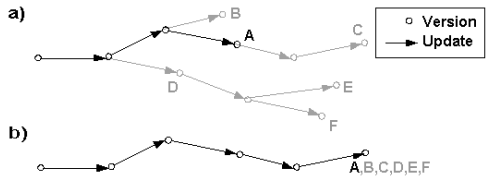


Figure 2. Example of optimistic evolution of replicas ***A***, *B,..,F* of the same file (a) towards an eventual common stable value (b). ***A*** is the current primary replica.



2.1.1. Optimistic consistency protocol

Haddock-FS's consistency protocol relies on dynamic version vectors (DVV) (Ratner, Reiher and Popek 1999) to identify replica versions and detect conflicts. A DVV is a variation of a version vector (Parker et al 1983) that allows the collection of replicas of each file to dynamically change over time. The DVVs of two replica versions, v_1 and v_2, can be compared to assert if v_1 causally precedes v_2, according to the happened-before relationship (Lamport 1978).

For each file replica, an Haddock-FS replica manager maintains a collection of data structures, depicted in Figure 1: a *stable value*, which holds a version of the file's stable contents, as described in Section 2.1.2; and an *update log*, which records the data specifications of most recent update requests that have been accepted by the file replica. Each logged update is assigned a DVV that identifies the file version that results from the update. Within the log, updates are causally ordered by their DVVs. Furthermore, logged updates are labeled as stable or tentative, as described in Section 2.1.2.

Overall system consistency is eventually achieved by pair-wise reconciliation sessions between replica managers, where replica updates are epidemically propagated. Such reconciliation sessions occur whenever a pair of replica managers becomes mutually accessible.

Relying on an optimistic replication strategy, update conflicts may occur if the same file is updated at distinct replicas. Haddock-FS's replication protocol enforces each file update to be applied only on the context upon which it was initially issued. This means that, if a client issues a file update at a given replica, no file update that hasn't yet been applied to such replica will ever precede the issued update at any other replica of that file. Formally, this states that, given the update sequence that is applied at a given file replica, all updates must be causally related, according to the happened-before relationship defined by Lamport (1978). Hereafter, we shall designate such consistency guarantee as *strict causal consistency*.

Enforcing strict causal consistency requires that, in the presence of causally concurrent updates, only one of them will be selected at a given replica. This differs from some alternative approaches that resolve such conflicts by ordering the concurrent updates (Demers et al 1994, Ladin et al 1992). Haddock-FS's decision argues that, in the absence of any semantic knowledge to rely on, an acceptable ordering resolution cannot be guaranteed and, therefore, should not be made.

2.1.2. Reaching a common stable value

Given a particular logical file, one can consider the global state of Haddock-FS replicated system as a directed acyclic version graph, whose nodes represent replica versions that are linked by updates. The root node is the initial file version.

Due to the optimistic nature of Haddock-FS's consistency protocol, multiple terminal nodes may exist. Each one corresponds to a divergent tentative value of the file. The consistency protocol is responsible for reaching a consensus on which of such divergent values the system replicas should converge into. Such value is defined as stable.

Throughout time, the set of stable values evolves monotonically from the root node to a single terminal node, hence defining a non-decreasing *stable path* in the version graph (Figure 2.a). The consistency protocol ensures that all replicas of a given logical file will eventually hold the updates that are contained by the stable path in their logs, designated as *stable updates*. Hence, their values are guaranteed to eventually yield a common stable value that is sequentially consistent. Since replicas enforce strict causal consistency, evolving to a common stable value at all replicas entails that every path that diverges from the stable path will eventually be discarded (Figure 2.b).

The stable path is increased when one of the tentative updates whose source is the path's terminal node is consensually selected as a stable update. Haddock-FS uses a primary commit scheme (Terry et al 1995), in

which a single replica of each file, the *primary replica*, is responsible for selecting new stable updates and propagating such decision back to the remaining replicas.

Hence, each replica manager is able to offer two possibly distinct views upon a file replica: its stable and tentative values. The former is simply the same as stored in the stable replica value. The latter results from the application of the tentative updates included in the log to the stable value.

Each file is initially assigned a unique primary replica, at which it was originally created. After creation, primary replica rights may be transferred to other replicas, by exchanging a token that identifies the current primary replica.

2.1.3. Intra-group consistency

When high connectivity is available, even if temporarily, amongst a group of peers, *intra-group consistency mode* is used. A group of mutually accessible mobile peers working cooperatively on a set of shared files within an ad-hoc network is an example of such scenarios.

Under such scenarios, the expectations of mobile users may assume strong consistency guarantees due to the connectivity that exists within the group. For this case, Haddock-FS enforces a pessimistic single-writer multiple-readers (Li and Hudak 1986) token scheme on top of the base optimistic consistency protocol, similar to that proposed by (Boulkenafed and Issarny 2003).

Obtaining a token upon a replica requires the replica manager that requests the token to reconcile with its previous owner. This means that, in an intra-group scope, mobile users and applications benefit from sequential consistency guarantees (Davidson, Garcia-Molina and Skeen 1985). From a system-wide viewpoint, however, a group of peers working in *intra-group consistency mode* is regarded as a single optimistic replica manager. Therefore, in a system-wide scope, weak consistency guarantees still prevail.

2.2. Replica content storage and propagation

The inherent memory and bandwidth constraints of mobile devices and wireless links are severe limitations to the effectiveness of a distributed file system for ad-hoc environments. For this reason, Haddock-FS tries to reduce the size of update logs stored at each device, as well as of update data to be transferred during replica reconciliation.

This is achieved by exploiting the cross-file and cross-version similarities that exist within the replicated data held by Haddock-FS's mobile peers. Such approach is similar to that of the Low-Bandwidth File System (LBFS) (Muthitacharoen, Chen and Mazieres 2001).

Our solution differs from the latter because file content similarity is exploited, not just for reducing network usage when transferring file contents between replicas, as it happens with LBFS, but also for the storage of all replicated file content-related data structures. Namely, these are the update log and the stable value of each replicated file.

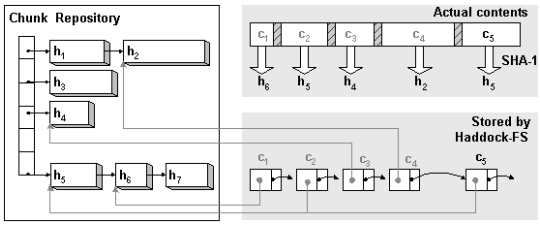
2.2.1. File Chunks

The basic idea of the content storage and transference scheme consists of applying the SHA-1 hash function to portions of each replica's contents; each portion is called a chunk. The probability of two distinct inputs to SHA-1 producing the same hash value is far lower than the probability of hardware bit errors (NIST 1995). Relying on this fact, the obtained hash values can be used to univocally identify their corresponding chunk contents. From this assumption, if two chunks produce the same output upon application of the SHA-1 hash function, then they are considered to have the same contents. If both chunks are to be stored locally at the same computer, then only the contents of one of them needs to be effectively stored. In a similar way, if one of the chunks is to be sent to a remote machine that is holding the other chunk, the actual transference of the contents over the network can be avoided.

A content-based approach is employed to divide replica contents into a set of non-overlapping chunks, based on Rabin's fingerprints (Rabin 1981). As a result, chunks may have variable sizes, depending on their contents. An important property of such chunk division approach is that it minimizes the consequences of *insert-and-shift* operations in the global chunk structure of a replica.

The expected average chunk size may, however, be parameterized by controlling the number of low-order bits from the fingerprint's output that are considered by the chunk division algorithm. Moreover, to prevent

Figure 3. Example of replica content storage using the chunk repository.



cases where abnormally sized chunks might be identified, a minimum and maximum chunk dimension is imposed.

2.2.2. Chunk Repository

Haddock-FS extends the use of LBFS's strategy to both local storage and network transference of replicated file data.

Our solution considers the existence, on each file system peer, of a common chunk repository which stores all data chunks, indexed by their hash value, that comprise the contents of the files that are locally replicated at that peer. The data structures associated with the content of locally replicated files simply store references to chunks in the chunk repository. This applies both to the update log and the stable value of each replicated file. Hence, the contents of an update or replica value consist of a sequence of references to data chunks, stored in the chunk repository (Figure 3).

When a data chunk is created or modified, either by a newly issued update or by applying some stable update to the stable value of a file, its hash value is calculated and the chunk repository is examined to determine if an equally hashed chunk is already stored. If not, a new entry corresponding to the new chunk is inserted in the repository and a reference to it is used. If a similar chunk already exists, then a new reference to that chunk is used. So, if different files or versions of the same file contain data chunks with similar contents, then they will share references to the same entry in the chunk repository, thus reducing memory usage by the file system.

As a consequence of the variable length character of data chunks, seeking a given position within the contents of a file requires traversing the chunk sequence list to determine the chunk that contains such position. This operation takes *O(N)* time, where *N* is the number of chunks that compose the file contents. Therefore, the performance of random accesses to replicated files may be compromised should the number of chunks be significantly high. Nevertheless, this aspect shouldn't be important due to the typically small dimensions of the files that the resource constrained devices addressed by Haddock-FS are expected to hold.

Update propagation between peers also makes use of the chunk repositories of each peer. When a chunk has to be sent across the network to another peer, only its hash value is firstly sent. The receiving peer then looks up its chunk repository to see if that chunk is already stored locally. If so, it avoids the transference of that chunk's content and simply stores a reference to the already existing chunk. Otherwise, the chunk contents are sent and a new chunk is added to the repository.

To achieve acceptable chunk lookup times, a hash-table is used to implement the chunk repository, indexed by the chunk hash values. This way, obtaining a chunk given its hash value is performed in *O(lg N)* time, where *N* is the number of stored chunks. On the other hand, read accesses to a file's contents can be served by a single indirect memory access to the chunks referenced by the chunk references stored in the file's data structures.

Serving a write request upon a local replica is an expensive operation. First, the newly written data is sequentially fingerprinted in order to determine chunk boundaries. Then, a SHA-1 hash value is calculated for each found chunk. Finally, a chunk lookup is performed in order to determine if each chunk should be added to or already exists in the chunk repository. In contrast, read operations do not have any associated processing overhead for accessing data in the chunk repository.

Experimental results on file systems (Baker et al 1991) show that write accesses are significantly rare when compared to read accesses. Relying on this assumption, the overall performance of local file accesses should not be significantly affected by the processing overhead of write operations.

To deal with the deletion of unused chunks from the repository, each chunk maintains a reference counter that is incremented each time a new reference is set to that chunk. Conversely, that counter is decremented when a reference to it is removed from the file system's structures. This can occur when a previously replicated file is removed from the set of replicated files or as a result of update log truncation, described in Section 2.2.3.

2.2.3. Log truncation

A fundamental requirement imposed by the typically scarce memory resources of mobile devices is that the storage overhead associated with update logs must be kept low. Haddock-FS uses a log truncation scheme that ensures that the total memory requirements of update logs are limited by a configurable constant size. Depending on each device's memory resources, such maximum value should be set to an adequate amount.

In order to attain high availability, replicas must be able to continuously receive update requests, despite such log size limit. Thus, in order to accept new updates, the oldest logged updates may have to be discarded.

An update that has already been received at all replicas of the logical file may be safely discarded from their logs. Hereafter, we will designate such updates as *safe*. A *non-safe* stable update, in turn, is not guaranteed to have been received at all replicas but has been applied onto the stable value of, at least, the replica that holds it in its log. As a consequence, incremental reconciliation between replicas becomes disabled if the version delta includes the discarded update. Since the stable value contents can still be copied between the reconciling replicas, consistency is not affected.

In the case of a tentative update, however, a replica manager must ensure that it is applied to its stable replica value before discarding it. Otherwise, the update information might be lost, since that replica may be the only one holding the update in consideration. Moreover, the missing tentative update would disallow access to the replica's tentative value.

Upon application of the tentative update the replica's stable value ceases to be stable, since it is no longer guaranteed to reflect a stable value. Hence, the replica manager looses its ability to provide a stable view of the file. Such ability may be later regained if the replica manager receives information that every tentative update that was applied to the replica value has become stable.

When necessary, in order to bound the memory usage of update logs to a maximum amount, Haddock-FS automatically discards safe and stable updates. In the case of tentative updates, the decision is left to the user: allow tentative updates to be discarded to achieve high availability; or prohibit it and thus ensure that both stable and tentative values always remain accessible.

3. IMPLEMENTATION

Haddock-FS is an Installable File System Driver (Murray 1998) for the MS Windows CE.Net 4.2 embedded operating system. The current version supports the replica consistency protocol, as well as cross-file and cross-version similarity exploitation optimizations for storage and network usage. Most of the file system functions were implemented, which enables using the office applications that are typically bundled with Windows CE to access the replicated file system. No modification to the application's code was necessary, due to the transparency provided by Haddock-FS's design. This is a very important aspect for portability reasons.

The distributed file system consists of a dynamic link library that exports all the function calls that exist in the generic Windows CE file system API. Examples of such functions are *CreateFile*, *CloseFile*, *ReadFile* or *WriteFile*. Using the *LoadFSD* function of the *FSD Manager* service of Windows CE, the file system can be mounted at run time.

The server side of each peer resides in a thread of *Device Manager* process, which is created when the file system is mounted. The server thread is continually waiting for remote procedure call requests from other peers across the network. Such requests are served upon access to the file system data structures stored in the address space of *Device Manager* process. On the other hand, the file system functions that are exported by the dynamic link library constitute the client side of each peer. Most of those functions access the shared data structures of the server thread. Interaction between peers is achieved using a remote procedure call library developed along with Haddock-FS.

4. EVALUATION

Haddock-FS was evaluated through several experiments. All measurements were obtained while running one or more Haddock-FS peers on the Windows CE.Net 4.2 Emulator on a Pentium III 1GHz computer. The

Figure 4. Local file access times.

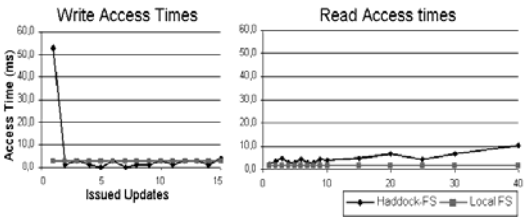
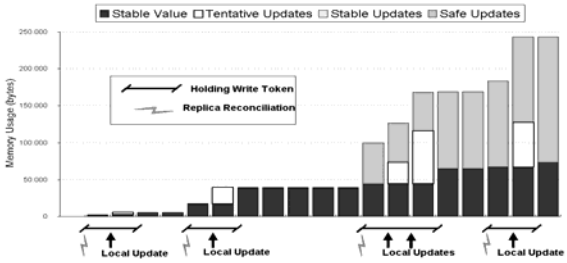


Figure 5. Log evolution of non-primary replica throughout collaborative intra-group consistency mode experiment.



emulated platform provided an equivalent to an x86 embedded device with 48MBytes RAM, running Windows CE.Net 4.2.[3]

To evaluate Haddock-FS's performance with practical workloads, we used an unmodified version of the MS WordPad word processing application to access replicated files. This application is typically bundled with Windows CE.Net devices.

The first experiment measured the effectiveness of local replica content storage, based on the use of a chunk repository. In order to obtain realistic measurements, we simulated the composition of the present paper using 19 different backup versions of its source text, ordered chronologically. Each version contents were individually applied to a local file replica by using the WordPad application to open, write and close such contents to the replica.

The results show that the optimal expected chunk size for the used workload is 512 bytes, which achieved a 47% reduction in memory usage by use of the chunk repository, in comparison to a non-optimized approach (that is, without cross-file, cross-version content similarity exploitation). On the other hand, the performance overhead on file access times were not significant when compared to Windows CE.Net native local file system counterpart. Read accesses to a file holding no more than eight updates in its log were, on average, 14% slower than if the local file system was used. Write accesses are strongly dependant on whether the written chunks are already stored in the chunk repository or not, as shown in Figure 4. Namely, the first update was significantly expensive (17 times slower than a similar access to the local file system), since every written chunk had to be added to the repository. From then on, file updates were, on average, 39% faster than the local file system, due to the previous existence of common chunks in the repository.

Finally, a more complete experiment was conducted, in which two Haddock-FS peers collaboratively issued updates to a shared replicated file using intra-group consistency mode. The considered set of updates was the same as the previous experiment, though distributed by both peers. Figure 5 shows the evolution of the update log of the non-primary replica throughout the experiment. The obtained results showed that, for a total updated content of 532Kbytes, the replica in consideration only had to store 237Kbytes using the chunk repository, which represents a 55% reduction in memory usage. Moreover, network usage also observed a significant optimization. From a total amount of 460Kbytes that needed to be transmitted during reconciliation sessions between peers upon acquisition of the write token, only 237Kbytes were effectively sent. Hence, a 48% optimization was accomplished.

5. RELATED WORK

The issue of optimistic data replication for loosely coupled environments has been addressed by a number of projects. Such previous work has the common goal of achieving high data availability, whether in the form of a file system, a database or a collection of objects.

However, most of the proposed solutions do not assume that their replicas will be held at devices doted with poor memory and network bandwidth resources. In particular, effective replica storage and transference optimization mechanisms are absent from most of the work that is presented below. Thus, their practical application to mobile ad-hoc scenarios is, in practice, inadequate.

Ladin et al. (1992) proposed a framework for providing highly available optimistic replication services for applications with weak consistency requirements. Adaptability to applications with stronger consistency

[3] It's worthy to note that the emulator provides testing conditions very similar to those found in a real world setting mainly w.r.t. the memory available.

needs is provided by two differentiated update types, at the cost of availability: a replica manager is required to belong to a majority partition in order to generate such update types. However, convergence to a common stable value is not guaranteed, which may be intolerable on most semantic domains. On the other hand, log truncation follows the rule that only safe updates may be discarded, which may require large memory resources if some replica managers are frequently unavailable.

The Bayou System (Demers et al 1994) is an optimistic database replication system, which relies on application-specific conflict detection and resolution procedures to attain adaptable consistency guarantees. The non-transparent character of Bayou's approach, however, prohibits the use of already existing applications, in contrast to Haddock-FS's solution.

Similarly to Haddock-FS, Bayou adopts a primary commit scheme to achieve eventual convergence to a common stable replica value. However, Bayou's replica managers are allowed to discard logged updates only when they become stable. Log truncation is, hence, dependent on the operation of a single point of failure (the primary replica manager), whose unavailability may imply log sizes that are impractical for resource constrained replica managers.

Keleher (Keleher 1999) proposes a replicated object protocol that eliminates the single point of failure of the primary commit scheme by employing an epidemic voting scheme for the purpose of reaching a consensus concerning a common stable value.

Roam (Ratner, Reiher and Popek 1999) is an optimistic replicated file system that provides a serverless service, intended for mobile networks. Its consistency protocol does not require replica managers to store an update log, which eliminates the significant memory overhead that is typically imposed by such a data structure. Nevertheless, Roam's consistency protocol does not regard any notion of a stable replica value. This important limitation restricts Roam's applicability to applications whose correctness criteria are sufficiently relaxed to tolerate dealing only with tentative data.

Another proposal for distributed file system for mobile networks is that of AdHocFS (Boulkenafed and Issarny 2003), which explicitly addresses the co-present collaborative scenarios that are enabled by mobile ad-hoc networks. AdHocFS exploits the high connectivity of such ad-hoc groups of replica managers by enforcing a pessimistic strategy amongst the group members.

Despite supporting such mobile ad-hoc scenarios, AdHocFS's architecture is still based on the existence of fixed server infrastructures, where the stable values of files are held. This means that, should that infrastructure be unavailable, mobile users and applications are restricted to accessing merely tentative data. This limitation may not be acceptable to many application domains.

Kang, Wilensky and Kubiatowicz (2003) have proposed a log truncation scheme that relies on an aging method based on roughly synchronized clocks. Haddock-FS' log truncation scheme resembles such method in the sense that any type of update, regardless of its stability state, may be discarded. However, Haddock-FS's solution is able to determine if a tentative update is about to be discarded, so as to notify the user that access to the stable value is no longer possible. In contrast, discarding updates based solely on their roughly synchronized age disallows providing such consistency statements.

6. CONCLUSIONS

In this paper we present the architecture and implementation of Haddock-FS, a replicated file system for resource constrained mobile devices. Our proposal is designed to meet the information sharing requirements imposed by mobile ad-hoc scenarios, in order to provide a viable support for co-present collaborative activities.

Namely, such requirements are the lack of a pre-existing infrastructure, the high topological dynamism of these networks, the relatively low bandwidth of wireless links, as well as the limited storage and energy resources of mobile devices.

Haddock-FS is based on an optimistic consistency protocol adapted to the network bandwidth and device memory constraints of these environments. Namely, through the use of an adaptable log truncation scheme and a cross-file, cross-version content similarity exploitation mechanism. Experimental results obtained from expected workloads show that Haddock-FS accomplishes significant network and memory usage optimizations when compared to traditional solutions.

As future work, we intend to further evaluate Haddock-FS by deploying it on real handheld mobile devices communicating through Bluetooth[4] and WiFi[5] wireless networks. Moreover, we plan to extend Haddock-FS's consistency model in order to support more adaptable consistency policies that may be better suited to potential mobile usage scenarios.

REFERENCES

Baker, M. et al, 1991. Measurements of a distributed file system. *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 198–212. Association for Computing Machinery SIGOPS.

Boulkenafed, M. and Issarny, V., 2003. Adhocfs: Sharing files in wlans. *Proceeding of the 2nd IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, USA.

Corson, S. and Macker, J., 1999. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. *Internet Request for Comment RFC 2501, Internet Engineering Task Force*.

Davidson, S, Garcia-Molina, H. and Skeen, D., 1985. Consistency in a partitioned network: a survey. *ACM Computing Surveys (CSUR)*, 17(3):341–370.

Demers, A. et al, 1994. The bayou architecture: Support for data sharing among mobile users. *Proceedings IEEE Workshop on Mobile Computing Systems & Applications*, Santa Cruz, California, pages 2–7.

Kang, B., Wilensky, R. and Kubiatowicz, J., 2003. Hash history approach for reconciling mutual inconsistency in optimistic replication. *23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03)*.

Keleher, P., 1999. Decentralized replicated-object protocols. *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC'99)*, 1999.

Ladin, R. et al, 1992. Providing high availability using lazy replication. *ACM Transactions on Computer Systems (TOCS)*, 10(4):360–391.

Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.

Li, K. and Hudak, P., 1986. Memory coherence in shared virtual memory systems. *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 229–239.

Luff, P. and Heath, C., 1998. Mobility in collaboration. *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pages 305–314. ACM Press, 1998.

Morris, J. et al, 1986. Andrew: a distributed personal computing environment. *Communications of the ACM*, 29(3):184–201.

Murray, J., 1998. *Inside Microsoft Windows CE*. Microsoft Press.

Muthitacharoen, A., Chen, B. and Mazieres, D., 2001. A low-bandwidth network file system. *Symposium on Operating Systems Principles*, pages 174–187.

National Institute of Standards and Technology, 1995. FIPS PUB 180-1: Secure Hash Standard. National Institute for Standards and Technology, Gaithersburg, MD, USA.

Nowicki, B., 1989. Nfs: Network file system protocol specification. *Internet Request for Comment RFC 1094, Internet Engineering Task Force*.

Parker, D. et al, 1983. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Transactions on Software Engineering*, 9(3):240-247.

Rabin, M., 1981. Fingerprinting by random polynomials. *Technical Report TR-15-81*, Center for Research in Computing Technology, Harvard University.

Ratner, D., Reiher, P. and Popek, G.,1999. Roam: A scalable replication system for mobile computing. *Mobility in Databases and Distributed Systems*.

Terry, D. et al, 1995. Managing update conflicts in bayou, a weakly connected replicated storage system. *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 172–182.

Weiser, M., 1991. The computer for the twenty-first century. *Scientific American*, 265:94–104.

[4] The official Bluetooth SIG website: http://www.bluetooth.com;
[5] IEEE 802.11 WLAN Working Group: http://grouper.ieee.org/groups/802/11/index.html