replic8: Location-aware data replication for high availability in ubiquitous environments

Evangelos Kotsovinos and Douglas McIlwraith

University of Cambridge Computer Laboratory
evangelos.kotsovinos@cl.cam.ac.uk

Abstract. File replication for uninterrupted availability is affected by the localised nature of network failures, particularly in ubiquitous, mobile environments; nearby nodes often get disconnected together, as a result of switching equipment faults, or of local wireless network unavailability – for instance, failure of a base station, or loss of network connectivity when a train enters a tunnel.
In this paper we propose replic8, a substrate for location-aware file replication, mitigating the effect of localised network failures by storing replicas at network locations selected for being far away. We demonstrate that, compared to storage of replicas at random network locations, replic8 achieves high data availability, and requires lower numbers of replicas to maintain that.

1 Introduction

Mobile and ubiquitous computing proposals envisage a world in which heterogeneous devices are interconnected to share data with ease and reliability. Connectivity of mobile devices such as mobile phones, PDAs and laptop computers is a commodity, due to the nature of the medium which is used.

The *transience* of such devices raises important research challenges in supporting uninterrupted presence of resources. Mobile devices often encounter network unavailability; this can be due to location, as when a user is passing through a valley where reception of her mobile phone network is weak, or when a train is going through a tunnel. It can also be due to failures of network software or hardware, such as network switches and transparent proxies, or base stations – for instance, as a result of poor weather conditions. *File replication* has been used for several years to provide significant benefits in terms of uninterrupted availability of files [1]. More recently its importance for mobile and ubiquitous environments has been realised [2].

In mobile networks we observe that such failures are usually *localised*; if a mobile phone cannot receive signal at a particular location it is highly unlikely that another one at the same location – connected to the same network – will, and if a PDA is not able to connect to a hotel lounge's wireless LAN it is improbable that other PDAs and laptops in the same lounge can. The probability of a mobile device being available, given that a second mobile device is unavailable, is higher if the two devices are a certain distance apart.

To address the localised nature of network failures, we propose *replic8*, a system for enhancing file availability in such environments. We use a location-based metric for determining which servers are "suitable" for hosting the replicas, based on the probability that they will fail together. Then, we replicate the file on nodes where that probability is low. In this paper we use physical distance between nodes as the aforementioned metric, there is no architectural restriction, however, to prevent other metrics from being used.

This paper presents the design and implementation of our system, and initial evaluation results obtained in an internal deployment. We present the system architecture, focusing on the components of the distributed architecture and the operations supported, in Section 2. The prototype deployment and evaluation setup is described in Section 3, along with our initial experimental results. Section 4 positions our system among related research work. Finally, Section 5 concludes and outlines our future plans for the deployment of our system in a mobile environment.

2 System Architecture

Servers participating in the replic8 substrate are termed *nodes*. Various types store replicas and run management software for communication with other nodes and distributed components. *Users* contact nodes for adding or retrieving files.

The node at which a data item is added to the system is called the *owner node* for that item, and the node that a user contacts for retrieving a file is called the *client node*. The owner node determines which nodes are suitable for hosting the replicas – based on their location – and passes replicas to other nodes. The nodes that store replicas of a data item are termed the *replica nodes* for that item. The set of nodes including both the owner node and replica node(s) for a given data item shall be known as the *data nodes* for that item.

A number of research challenges need to be addressed: as users need to be able to add, remove, modify, and retrieve files by contacting any of the replic8 nodes, all nodes need access to information about the availability and *position of other nodes*, as every time a file is inserted location-based decisions need to be made on where its replicas are to be stored. At the same time, nodes need to be able to obtain information about the *location of replicas*, to allow discovery of data items. *Consistency management* issues are also raised, as concurrent modifications of the same data item may be attempted from different nodes.

2.1 Overview

To distribute node position and replica location information we employ special groups of nodes – each one comprising nodes selected to be far from each other, for higher fault-tolerance; then we run each of the required services on a group of nodes in a replicated fashion. This is not dissimilar to super-peer nodes [3] on peer-to-peer networks, but where designers of such networks may strive to assign super-peer 'status' to nodes most capable of handling the load,

replic8 seeks nodes which are least the likely to fail together. We use three such groups of nodes:

- **Directory service nodes** participate in file discovery by storing and providing information about the files that exist in the system.
- **Group service nodes** identify data nodes for each data item in the system and keep track of the location of replicas for each file.
- **Position service nodes** store position information about the nodes in the system and assess the suitability of nodes to hold particular replicas.

Updates sent to any of a group of service nodes are immediately *multicast* to other members of the group. This ensures that all members of the group maintain the same information. If acknowledgement is received from all members then the update is complete, else the original recipient of the update enters a phase of *continual transmission* to the unavailable node, until acknowledgement. This ensures the missing node receives the update soon after it becomes available again. Specific group node operations are discussed in Section 2.2.

In our prototype implementation service nodes are selected randomly – as long as they are further apart than a specified minimum distance, in a future implementation, however, we envisage using a distributed election algorithm appropriate for mobile and ubiquitous environments [4, 5].

Service nodes inform others of the same group when they are disconnecting. Each node also periodically multicasts a heartbeat message, to make sure others can find out if it disappears abnormally, without prior notification.

Since initial results are obtained in a trusted environment, we ignore the possibility of malicious intent. This will need to be re-addressed as we deploy on more large scale distributed networks, perhaps using our experience on trust and reputation management systems [6] and honesty detection mechanisms [7].

2.2 Operations

In this section we describe the internal actions that are taken every time a user calls one of replic8's interfaces to add, retrieve, modify, or remove a file. We also discuss two operations that take place periodically in the background, namely replica relocation and consistency management.

File Addition Users contact any replic8 node, which becomes the owner node for the file to be added. Additions to the system require generation of a file identifier, and replication on suitably remote nodes. The former is carried out with the help of the directory service, which generates a unique file ID, stores its mapping to the file's name, and returns the ID to the user – operation 2 in Figure 1. The latter is completed by communication with the position service, which recommends a list of servers according to physical proximity – operation 3. Nodes periodically submit advertisements to the position service specifying their current status and location – operation 1.

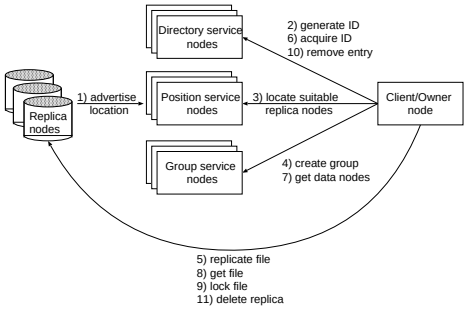

Fig. 1. Overview of the replic8 architecture.

The owner node decides, based on the position service's recommendations, on the set of replica nodes to be used and replicates the file there – operation 5. Successful completion triggers communication with the group service to commit the relationship between the file and its data nodes – operation 4.

If a proposed replica node is unable to receive the replica then the owner node enters a state of retransmit, trying to contact the unavailable node until it responds. If after a period of time no response is received the owner node attempts to find another suitable set of replica nodes. If there is no other suitable set of replica nodes then the data is replicated on the available nodes only and inconsistency is resolved at a later stage.

File Discovery and Retrieval Users request the retrieval of a file providing its human-readable filename. The client node first acquires the unique file ID for the requested file from the directory service – operation 6. On receipt of the ID, the client node contacts the group service to request the data nodes for this file – operation 7, then tries to retrieve the file from each one of the data nodes until the owner node or the first available replica node returns the file – operation 8.

File Modification Firstly, the client node that has been contacted retrieves the identifier for the file to be modified, which is carried out by communication with the directory service – operation 6. Then, support from the group service is required to provide the list of data nodes for the file – operation 7.

The client node then contacts all data nodes that store copies of the file. If all data nodes are available, the client requests an exclusive *write lock* on the given file on all those nodes – operation 9. If only a subset of the data nodes for that file can be contacted, updates go ahead as above, but only at the nodes contained in the subset. Resolution of the arising inconsistency is described below.

To prevent deadlock we use *soft-state* file locks; each lock is associated with an expiration time and cannot be retained by a node indefinitely. Once confirmed by all involved, updated copies are written to the data nodes and the locks released.

File Removal Removing a file from the system involves initially deleting the entry for the file in the directory service, which is carried out by the client node in communication with the directory service – operation 10. This ensures that any further searches for the file will not return any results.

Any replicas of the file stored in remote nodes then have to be deleted by submitting replica deletion requests – operation 11. We follow a "lazy" approach towards replica removal; nodes that receive such requests mark files as "to be deleted" but do not actively delete them unless the allocated space for storing replicas is filling up. This allows for file recovery and better file removal performance, without compromising consistency.

Replica relocation management Nodes advise the position service about changes in their position regularly – operation 1. As we are focusing on ubiquitous and mobile environments, it is necessary that our architecture is able to adapt to change of location and availability of mobile nodes. When a replica node gets closer to the owner node, replic8 needs to locate another, more suitable replica node, copy the file there, and remove the now nearby, old replica node from the data nodes group for the file.

This is carried out periodically by the owner node for the file, which requests recommendations from position service and determines whether there is a significantly better set of replica nodes that may be used. The period at which this operation is undertaken, as well as the threshold above which a change of replica nodes for a file is required, present two important trade-offs; higher frequency of this operation and low restructuring threshold lead to higher data availability – as the replica nodes are most of the time suitable – but incur higher network traffic and computational load for the replic8 nodes.

Consistency management As we are employing an optimistic replication scheme, it is necessary that replic8 performs internal management operations periodically to ensure consistency of replicas. Detection and resolution of inconsistencies for a file is ultimately a responsibility of the owner node for the file, and is achieved using Version Vectors [8]. Version Vectors are well understood and it is known that they cannot resolve consistency when copies of a file of the same version have been modified in isolation from the others. Any attempt at resolution will result in one or more modifications to the file being lost. If this situation occurs in replic8 manual intervention is required, and we provide methods to signal this to the file owner affected.

3 Evaluation

In this section, we present our initial results in assessing replic8's performance in terms of achieving higher data availability than a random replication strategy, where files are copied to arbitrary nodes, in an environment where localised failure is common. All experiments were carried out on an Intel Pentium III

Mobile CPU, clocked at 866MHz, with 128Mb of main memory. Java 1.4.1-02 was used on a Windows 2000 platform.

To measure replic8's effectiveness, we launched a system comprising a thousand replic8 nodes, and associated a random position in a 2D Cartesian coordinate space with each. The 20x20-sized coordinate space used was split in four hundred 1x1-sized sub-spaces called *quadrants*, each *contains* the nodes whose position coordinates fall inside that quadrant. In our experiments we do not consider varying network conditions, such as route changes or packet loss.

We also present results indicitave of replic8's *network overhead* for file maintainance compared to that of a system which is not location aware. Both systems implement functionality to maintain file consistency, enable file addition and file retrieval, but unlike replic8, the system incapable of location aware operation did not implement services to respond to file movement by relocating replicas.

The simulation started with *file addition*, where files were inserted in the system and replicated accordingly. After that, the simulation entered an iteration comprising *failure generation*, measurement of the number of *unavailable files* in the system, and *node relocation*, as explained below.

File addition. Suitable replica nodes for the files inserted were discovered, and the files were replicated on these nodes. File addition only happened once, and the files remained in the system throughout its operation after that point. All quadrants were initially set to active.

We added a thousand files, or one on each replic8 node. The *minimum distance* that replicas should be held at, based on which the position service recommends suitable replica holders to the owner nodes at file addition, had been set to seven.

Failure generation. The first step of the simulation iteration involved injecting localised network failures. We did so by making neighbouring quadrants fail together according to a simple localised failure generation algorithm, and compared the number of available files in the system using random replication to that achieved using replic8. A file is considered *available* if it can be found on at least one currently connected replic8 node.

The failure generation algorithm we used works as follows; each time the failure generation algorithm is called, a single quadrant is picked randomly from the coordinate space and set as disabled. This ensures that all nodes it contains are immediately marked as disconnected. At the same time, neighbouring quadrants may be disabled too, making up a disconnected area of a radius equal to the *localisation coefficient*. In our experiments, the coefficient has been varied from one to twenty.

Node relocation. Nodes move around the coordinate space in random directions by a fixed *walk distance* and register and unregister with quadrants accordingly. In the tests we carried out, to simulate a highly mobile environment nodes moved around by a unit – ensuring most change quadrant every iteration.

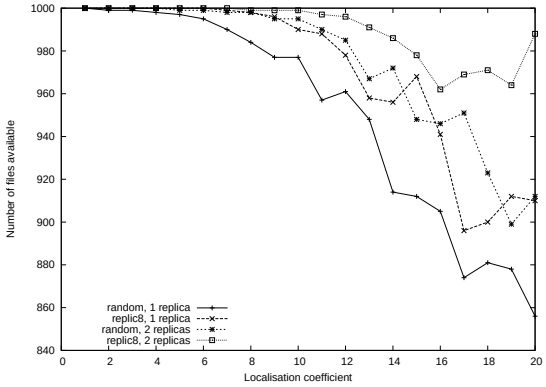

Fig. 2. File availability as the size of a disconnected area grows.

After each relocation step, each owner node ensures that all nodes previously used as replica holders for its file have not got prohibitively close, then a new failure generation step is invoked. This cycle is continued until a number of iterations is reached.

3.1 Results

File Availability In the aforementioned setting, we have executed the simulation iteration twenty times for each combination of values for number of replicas and localisation coefficient. We have calculated the average number of available files out of the one thousand files initially added to the system as the size of the disconnected area increases. Figure 2 compares the availability achieved using replic8 to that using a scheme where replica nodes were selected at random.

We have plotted, along the x-axis, the *localisation coefficient*, representing the maximum size of the disconnected area around a randomly selected quadrant. The y-axis displays the *availability* of files – the number of files accessible through at least one of their data nodes. Availability has been measured in two cases; in the first case, only one replica of each file was being held – so two copies of the file existed in the system, including the original one. In the second case, each file was copied twice on replica nodes.

The graph shows that using replic8, for localisation coefficient 15, only 32 files are unavailable, compared to 88 using random replication. Even by maintaining an extra copy of the file, random replication does not surpass replic8, leading to 52 files being unavailable; replic8 achieves availability of all files but 22 using two replicas per file. In general, *replic8 using one replica performs almost as well as random replication using two replicas* – in some cases even better. This is an important result given that storage space can be restricted on mobile devices.

The graph shows that file availability decreases as the size of the disconnected area grows for both file replication schemes – replic8 and random replication.

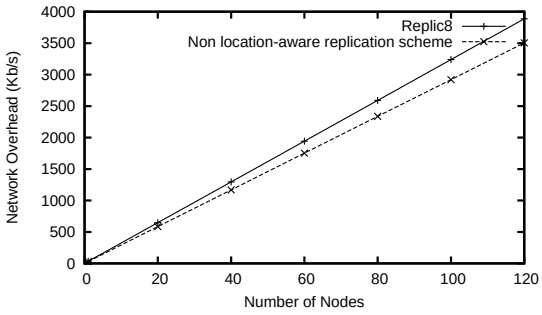

Fig. 3. Network overhead as the number of nodes increase.

This is inevitable, as increasing the size of the disconnected area causes more nodes to be disconnected. Furthermore, the frequently changing direction of the plot is due to the randomness present in the simulation; since nodes move arbitrarily and failure is random, convergence to an average availability for a given localisation factor is not guaranteed. We believe it is exaggerated by the comparatively large size of the quadrants to the entire area.

Overall, replic8 provides significant file availability benefits in environments where node disconnection is highly localised, such as – potentially – in ubiquitous, mobile, and wireless networks.

Network Overhead Figure 3 compares *average* network overhead with replic8, and with a non location-aware replication scheme which does not respond to the movement of nodes. The overhead illustrated is due to file maintainance when the system is populated with a varying number of nodes. Each node owns a 500K file and is responsible for its consistency with replicas as well as, in replic8's case, file movement between nodes when node positions become unsuitable.

Both plots exhibit linear behaviour as the number of nodes increase. This is due to increased load on the network from consistency checking and, in replic8, movement of files. These trends are linear since both aforementioned operations occur with a fixed period.

The regularity of node failure also effects this load as, upon service failure, service nodes enter a state of permanent retransmit until all members of a service group are available. Node failure frequency affects data holders for a file also, since the more often data holders for a file are unavailable, the more likely updates will occur optimisitically, at a subset of the data nodes, requiring expensive additional clean-up operations to regain consistency. These figures were based on individual nodes failing with an *average* probability of 0.001.

With replic8, network overhead grows more rapidly as the number of nodes increase. This is due to additional overhead in maintaining an extra service, namely regular publication of position from nodes, plus the load exhibited from

file relocation as nodes move around the system. For 120 nodes we calculate an increased network overhead of around 11%

While Figure 3 demonstrates load for file maintainance, it does not illustrate the cost of individual operations. We calculate that replic8 creates a load of 1099Kb on the network introducing a single 500K file to the system, in comparison to a load of 1008Kb for a *non-location-aware* system – an increased load of around 9%. File removal and retrieval in replic8 costs the same whether or not the solution is location aware.

Replic8 provides, with minor network overhead, significant availability improvements over random replication – allowing the use of fewer replicas. This lets us trade storage overhead for a small network overhead in environments where storage is a commodity.

4 Research Context

Distributed storage systems have used replication to improve data availability in unreliable network conditions. Ficus [9] adopts optimistic replication and periodic consistency checks – like replic8 – for reliable access to files on transient devices, and delegates the decision as to how many file replicas are to exist and where, to the clients. Bayou [10] targets heterogeneous environments and provides high availability via a read-any/write-any scheme of data access. A different approach for data replication, based on an economic model for managing the allocation of storage and queries to servers, is employed in Mariposa [11].

All above systems have presented important solutions to problems such as disconnected operation, replica discovery, and conflict resolution. Our work specifically targets the issue of improving file availability in *localised network failure* conditions. replic8 addresses the problem by basing the decisions on where replicas are to be stored on location, or other application-specific metrics.

Fluid Replication [12] employs location-based replication in static environments, ensuring that replicas are stored near the original for reduced latency. Replic8 differs by targeting environments where localised replication would be disastrous; a single failure may either bring down an entire mobile service cluster or a client and its local replicas.

Replic8's service nodes are inspired by super-peer nodes found in peer to peer systems [3]. The simple replication strategy followed is sufficient to validate our approach, as shown by the evaluation results, and avoids requiring the storage and processing of excessive amounts of data on mobile devices.

5 Conclusions and future work

In this paper we have presented replic8, a system for intelligent file replication to achieve high data availability in environments where device disconnections are non-arbitrary. We have shown replic8 to operate more effectively than random replication in an environment simulating highly localised failures of a mobile

network – while requiring fewer replicas. We have also demonstrated that this can be achieved with small network overhead above a *non-location-aware* system.

We plan to deploy replic8 on our COMS[1] and XenoServers [13] and obtain more accurate results and experience on the performance and fault-tolerance of our system. Furthermore, we plan to augment distributed storage systems such as CODA [14] with replic8 to achieve higher file availability with fewer replicas under conditions of non-arbitrary node disconnection.

References

1. Walker, B., Popek, G., English, R., Kline, C., Thiel, G.: The LOCUS Distributed Operating System. In: Proc. of the 9th ACM Symposium on Operating Systems Principles. (1983)
2. Ratner, D., Reiher, P., Popek, G.J., Kuenning, G.H.: Replication Requirements in Mobile Environments. Mobile Network Applications **6** (2001) 525–533
3. Yang, B., Garcia-Molina, H.: Designing a Super-Peer Network. In: Proc. of the 19th Intl. Conf. on Data Engineering. (2003)
4. Malpani, N., Welch, J.L., Vaidya, N.: Leader Election Algorithms for Mobile Ad Hoc Networks. In: Proc. of the 4th Intl. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications. (2000)
5. Vasudevan, S., DeCleene, B., Immerman, N., Kurose, J., Towsley, D.: Leader Election Algorithms for Wireless Ad Hoc Networks. In: Proc. of the 3rd DARPA Inf. Survivability Conf. and Exposition (DISCEX- III). (2003)
6. Dragovic, B., Hand, S., Harris, T., Kotsovinos, E., Twigg, A.: Managing Trust and Reputation in the XenoServer Open Platform. In: Proc. of the 1st International Conference on Trust Management (iTrust 2003). (2003)
7. Fernandes, A., Kotsovinos, E., Ostring, S., Dragovic, B.: Pinocchio: Incentives for Honest Participation in Distributed Trust Management. In: Proc. of the 2nd International Conference on Trust Management (iTrust 2004). (2004)
8. Parker Jr., D., Popek, G., Rudisin, G., Stoughton, A., Walker, B., Walton, E., Chow, J., Edwards, D., Kiser, S., Kline, C.: Detection of Mutual Inconsistency in Distributed Systems. IEEE Transactions on Software Engineering SE-9, (1983)
9. Guy, R.G., Heidemann, J.S., Page, Jr., T.W.: The Ficus Replicated File System. SIGOPS Oper. Syst. Rev. **26** (1992) 26
10. Terry, D.B., Theimer, M.M., Petersen, K., Demers, A.J., Spreitzer, M.J., Hauser, C.: Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In: Proc. of the 15th Symp. on Oper. Sys. Principles (SOSP-15). (1995)
11. Sidell, J., Aoki, P.M., Sah, A., Staelin, C., Stonebraker, M., Yu, A.: Data replication in mariposa. In: Proc. of the 12th International Conference on Data Engineering, IEEE Computer Society (1996) 485–494
12. Noble, B., Fleis, B., Kim, M., Zajkowski, J.: Fluid replication. In: Proc. of Netstore '99, the Network Storage Symposium. (1999)
13. Hand, S., Harris, T.L., Kotsovinos, E., Pratt, I.: Controlling the XenoServer Open Platform. In: Proc. of the 6th International Conference on Open Architectures and Network Programming (OPENARCH). (2003)
14. Kistler, J.J., Satyanarayanan, M.: Disconnected operation in the coda file system. In: Proc. of the 13th ACM Symp. on Oper. Sys. Principles (SOSP-13). Volume 25., Asilomar Conference Center, Pacific Grove, U.S., ACM Press (1991) 213–225

[1] Cambridge Open Mobile System, `http://www.cl.cam.ac.uk/coms/`