

Hoarding Context Information with Context Clusters

Mylone Anandarajah
School of ITEE
University of Queensland
and
National ICT Australia
myilone@itee.uq.edu.au

Ricky Robinson
National ICT Australia
Queensland Research Laboratory
ricky.robinson@nicta.com.au

Jadwiga Indulska
School of ITEE
University of Queensland
and
National ICT Australia
jaga@itee.uq.edu.au

Abstract

The components of a context-aware system can often become disconnected because of the dynamic environments within which they are deployed. Hoarding context information on the client application side can improve the probability that the application will continue to behave correctly in the event of disconnection. While traditional approaches to caching and hoarding can be used to combat this problem, we contend that a cache management solution that uses the extra information captured by context modelling techniques will provide more robust operation. Specifically, we focus on the metadata provided by the Context Modelling Language (CML), which may enable smarter decisions to be made by a cache management system for context information.

1. Introduction

Context-aware applications are commonly deployed in highly dynamic and mobile computing environments. A well known characteristic of such environments is the propensity for disconnection due to the physical limitations of the underlying networks. In the absence of a suitable caching or hoarding algorithm, disconnection may result in application failure. In the case of context-aware applications, disconnection can mean application failure or incorrect application behaviour.

While it is possible to use existing caching techniques for context information, this paper shows that it is possible to develop cache management strategies that are optimised for context information. Specifically, we consider systems based on the Context Modelling Language (CML) [7], which includes features such as context fact quality, context histories, fact type classifications and fact type dependencies, all of which can be used by a hoarding or caching algorithm to decide which context facts to store on the client

side. The idea is to improve the chances that a context-aware application will continue to operate correctly during periods of disconnection.

This paper focuses on context-aware systems similar to that shown in Figure 1. That is, the system may consist of a centralised or distributed context management system, and a number of client devices upon which context-aware applications execute. The context-aware applications rely upon the context information stored by the context management system, which raises a problem when disconnection occurs.

The remainder of this paper is organised as follows. Section 2 defines what is meant by the terms hoarding, caching and prefetching within this paper. Section 3 provides an overview of some existing caching and hoarding techniques used in mobile environments and the web. In Section 4, we introduce *context clusters* and *context cluster* templates, our solution for enabling CML-based applications to continue operating during periods of disconnection. Finally, Section 5 summarises our contributions and discusses future work.

2. Caching, Hoarding and Prefetching

The terms caching, hoarding and prefetching have overloaded meanings in different fields of computer science research, and these meanings are usually closely tied to their use rather than their actual function. Moreover, the three terms are often confused or used as synonyms. For example, prefetching is similar in nature to hoarding, but in mobile computing, they are used to achieve different things.

For the purposes of this paper, we will differentiate between these terms in the following manner (also see Saygin's explanation of the differences between these concepts [18]):

Caching Storing a copy of an object retrieved by an application in a completely passive manner, for the purposes of reducing latency and bandwidth consumption.

Prefetching Proactively retrieving an object which is

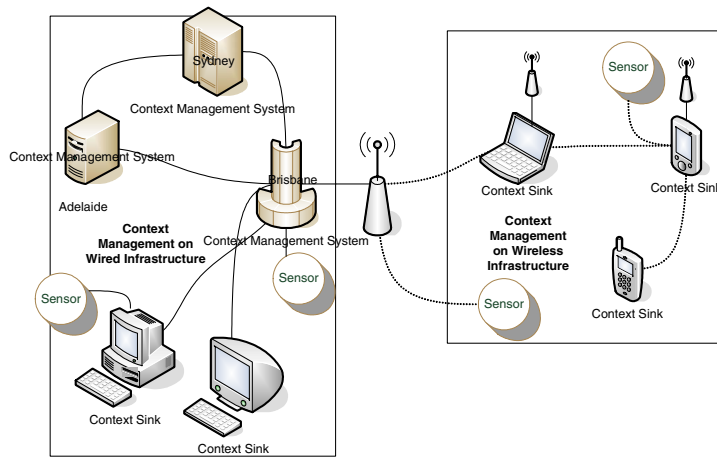


Figure 1. Context Aware System

likely to be used in the future for the purpose of reducing latency.

Hoarding Proactively retrieving all the objects likely to be used during disconnected periods.

Prefetching makes use of idle time (either of the network or of the CPU) or periods of high-bandwidth connectivity to retrieve objects that an application may require in the future. Hoarding on the other hand is usually a bulk operation, done to enable mobile clients to continue operating while they are disconnected. Hoarding algorithms often use historical knowledge and user preferences to decide which objects to fetch during connected periods. Typically, a hoarding algorithm will contain a reintegration phase which is entered immediately after a host becomes reconnected to the network. Reintegration involves synchronising changes with the server or servers. For example, in a distributed file system with hoarding functionality, a host that becomes reconnected would push any changes made to files during the period of disconnection back to the server, and any changed files on the server which are also in the client's hoard would be fetched by the client.

The focus of the work presented in this paper is to enable context-aware applications to continue operating during disconnected periods. Therefore, hoarding, by the definition above, is the appropriate technique for use by mobile devices that host context-aware applications.

3. Related Work

3.1. Overview

Caching involves temporarily keeping copies of information and using these to satisfy subsequent accesses. Caching management techniques can be used by nodes of a context-aware system to provide applications with context informa-

tion even if they are disconnected from parts of the network. Cache management consists of caching algorithms that determine which information should be cached and replacement algorithms that purge information. Cache replacement algorithms are used to decide which cached data should be purged when the cache is full and new data needs to be placed in the cache.

Client oriented caching solutions [5, 12] are implemented close to the client. There are three reasons for using this type of caching. The first reason is to improve access latency; the second is that they can reduce bandwidth on the network and the third is that if the user becomes disconnected from the server they may still be able to access information in a cache. This research is interested in the third reason and how to use caching techniques on the nodes of context-aware systems to help them handle disconnections.

Research has been carried out in cache management techniques to improve access latency [1]. There has also been research in the area of power efficient cache management systems [9, 19]. However our research is not concerned about these issues.

Several systems have also been developed to share context information [13].

Traditionally, information was removed from the cache based on expiry algorithms. These algorithms determine which information to remove based on age, size, and access history. Least Recently Used (LRU) and Least Frequently Used (LFU) are two common expiry algorithms whose functions are reflected in their names.

This literature survey covers current state of the art in cache management which have the potential to be taken advantage of to make context-aware systems more fault-tolerant. The cache management algorithms which aim to make clients fault tolerant can be grouped into the following categories:

- LRU and LFU
- data mining
- temporality (time based)
- locality (location based)
- hoarding techniques based on program trees
- user specified priority

Data mining techniques aim to find interesting patterns relating large collections of data [6]. The two data mining techniques used in hoarding are clustering and association rules.

Clustering is used by the Seers hoarding system [11, 18]. Clustering techniques are used in hoarding to discover groups of objects which are accessed in close semantic distance to each other. These groups are referred to as “projects”. The hoarding algorithms fill the cache with projects which currently have active objects. Once Seers has grouped objects into projects it uses a replication system such as Coda [10] to do the hoarding.

Association rules describe the association between sets of data. Users’ behavior is observed to extract association rules. These rules can be used to predict future client requests. The predicted request set is what is loaded into the cache prior to disconnection.

The simplest cache management algorithms based on access history are Least Recently Used (LRU) and Least Frequently Used (LFU). These common algorithms are used by web caches to determine which documents to remove. LRU removes the least-recently used documents, and LFU removes the least-frequently used documents. The expiry algorithms will affect the number of successful hits on the cache.

Program Trees are another caching technique based on access history [8]. Hoarding techniques based on program trees uses the history of accesses of programs to predict which files will be accessed when a program is running.

A temporality based caching algorithm [4] caches information associated with a period of time. This period of time is based on the user’s needs. Such an algorithm could be useful in an application used by a fieldworker who is usually out in the field between 11am and 3pm. She has a mobile device which will not be connected to a network while she is away. The algorithm could cache information on the device associated with the time the field worker will be away from her desk.

Other systems may know that in the mornings before the user sets out to work they like reading the tabloids so this information will be cached in the morning. Later on in the day the user may be more interested in financial news.

Location based caching algorithms cache information related to the client’s location. For example such an algorithm

may only cache information in the vicinity of the user. Others may only cache information associated with the direction the client is moving in.

Cache replacement strategies such as FAR [17] first replace the data that is furthest from the user and is not in the user’s moving direction.

Mishra et al. address the issue of caching QoS and security related context information on base stations. This research aims to reduce latency of handoffs of mobile clients between base stations. Neighbor graphs are utilised to provide a candidate set of potential new base stations [14].

One of the options supported by Coda allows the possibility of priority being obtained from the user’s preference. Usenet-on-the-fly [3] caches information a user will need each time it comes into contact with another mobile device. In prefetch mode the 7DS [15, 16] device proactively tries to cache information the user may need based on preference information set by the user. User’s needs are determined from priority values set by the user in Usenet-on-the-fly and 7DS.

3.2. Comparison

The effectiveness of the hoarding techniques discussed here vary depending on the system they will be used in [8]. There is also no existing data available which compares the effectiveness of these techniques to each other. How long the client is disconnected will also influence how effective the different caching algorithms are. A client may be disconnected from a few seconds to a few days. Some misses may not have much of an effect on the operation of a client; however, the lack of a critical piece of data may force the user to cease work until the miss can be serviced.

Other than user specified priority and temporality based caching techniques all the other methods are automated except for the user specifying some minor preferences. The automated algorithms use previous and current behavior of the client to predict future behavior. These algorithms hoard data based on these predictions. Temporality based caching techniques could be based on user provided preferences or on access history. Some data mining techniques are based on previous access history others hoard based on the type of the data. The user provided priority type algorithms are prone to error since the user may not update their preferences often enough for the algorithms to be effective.

Association rule based techniques are designed for voluntary disconnections. Coda can handle voluntary and involuntary disconnections well. The program tree and some time based methods are designed for voluntary disconnections. All the other caching techniques discussed can handle both voluntary and involuntary disconnections.

LRU is usually an adequate approach to hoarding [11]. Data mining and program tree approaches work better than

LRU when “attention shift” occurs from the user. A client using LRU must individually reference each file in the shift to have the same effect as data mining and program tree based techniques. It is possible to handle “attention shift” in Coda by a user changing the hoarding profile that they are using. This method however is tedious.

The effectiveness of the location based methods vary depending on the movement of the client and the location of information the context-aware application is interested in. For example the location based caching algorithms which cache based on the location the client is moving in will be effective assuming that the context-aware application is only interested in information located in the direction the client is moving in. This type of algorithm will also be ineffective if the client is always located around the same area or stationary. For example if the client is used by a user who only uses it in an office.

4. Hoarding Context Information

The context-aware systems that we develop are based on the CML design notation. CML is widely regarded as an excellent means for modelling context information, because it includes support for data quality metrics, context histories, fact type classifications and other features specific to context modelling. An example CML model is shown in Figure 2. The hoard management strategy outlined in this section makes use of several of these elements for the purposes of making more informed decisions about what to hoard.

4.1. The Hoard

The hoard is an area that is used for storing context facts. The size of the hoard is device dependent, and depending upon implementation and purpose, the hoard may be built on primary or secondary memory, or both.

On application initialisation, and after reconnection, the hoard is populated by the hoarding algorithm (see 4.2). All requests by the application are routed through the hoard, regardless of the current state of connectivity. If the request can be satisfied from the hoard, the appropriate context fact or facts are returned. If the request cannot be satisfied from the hoard, then the outcome depends upon the current state of connectivity. If the device is connected, then the request can be satisfied from the context management infrastructure (this may be a centralised context manager, or a distributed context manager, but for the purposes of this paper we ignore this detail). If the device is disconnected, then the application may fail, behave incorrectly or be partially unusable depending upon how critical the hoard miss is.

4.2. The Hoard Manager

Our hoard management strategy uses knowledge of the CML model or models to which the stored context facts conform and is inspired by the Seers hoarding system, [11, 18] described in section 3. In addition, the hoard manager uses additional information that cannot be represented by CML. This additional information pertains to *particular* context facts that are used by a *particular* instance of a context-aware application.

For example, Bob’s application may be interested only in information pertaining to Bob. Therefore, a hoarding mechanism must be able to hoard only fact instances that are associated with Bob. The application’s interests may be more specific. It may only be interested in information about Bob when he is at work. “Bob” and “Bob’s work” are termed first class objects in our solution.

These first class objects, or *keys*, identify a *cluster*. A cluster is a set of context facts that conform to an interconnected series of CML fact types, called a *cluster template*. An example cluster template is shown in Figure 3. Clusters and their keys may be identified by application designers, or they can be learned at runtime by monitoring the types of requests that are made by the application.

The hoard manager is constituted by four main algorithms. These algorithms are: Hoarding algorithm; purging algorithm; recording algorithm and resynchronization algorithm. Each cluster in the context management system is associated with a priority value. The priority value of a cluster is determined by a priority function. The priority function can be based on traditional cache management techniques such as LRU or LFU. The priority function can also make use of one or more CML factors identified in our previous paper [2]. For the purposes of this paper, we assume the use of LRU, in which the most recently accessed clusters obtain the highest priority. The hoarding, purging and resynchronization algorithms make use of the priority function. Figure 4 shows a logical view of the system.

The recording algorithm keeps track of the last time a cluster was accessed and by which application. This information will be input into the priority function. The hoarding algorithm determines if a cluster should be added to the hoard. The purging algorithm removes clusters from the hoard which should no longer be there. The resynchronization algorithm runs when the client reconnects to the context management system. This algorithm coordinates with the context management system and updates the hoard appropriately.

The hoard and the context management system are said to be unsynchronised if there is a cluster in the context management system which satisfies all the following three requirements: it has not been hoarded; has a higher priority than at least one cluster in the hoard; and satisfies the qual-

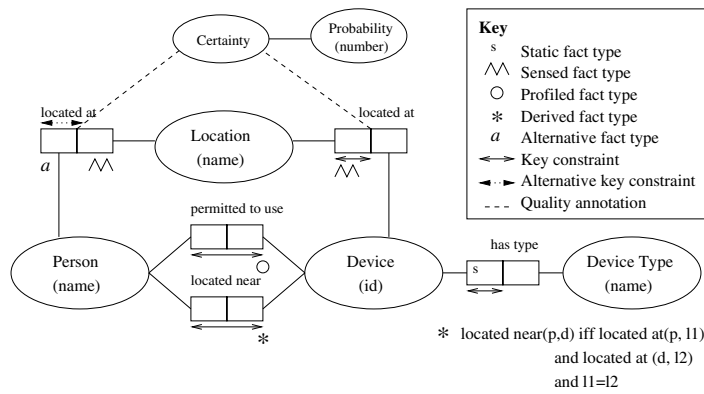


Figure 2. Example Context Model

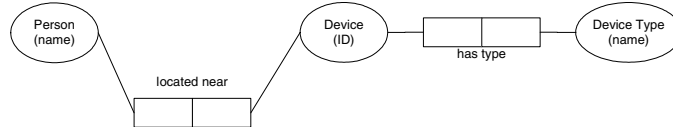


Figure 3. A cluster template extracted from the CML model shown in Figure 2

ity requirements. A change in priority values may cause the hoard and context management system to become unsynchronised. The hoard and context management system are also said to be unsynchronised if a hoarded cluster that has been updated in the context management system during the disconnected period has not been updated in the hoard. The resynchronization is carried out by the resynchronization algorithm.

The hoarding algorithm runs in the hoarding state each time the hoard manager is notified of a change in the state of the context management system. The hoarding algorithm must resynchronise the hoard and context management system if required. If a cluster in the context management system has a higher priority than what is already in the hoard and the cluster satisfies the quality requirements it will be placed in the hoard. If there is no room to place such a cluster the hoarding algorithm purges the cluster in the hoard with the lowest priority value. For alternative fact types (as defined in CML), which allow multiple conflicting facts to be stored, the hoard manager keeps only the fact with the higher priority.

The hoarding algorithm must also work out which fact instances belong to the different clusters. This information will be used by the hoarding algorithm to request the context management system to send a notification when an instance belonging to a hoarded cluster has changed.

When a request is made it must be determined which cluster the request expects results from. This cluster is now put on top of the list of clusters to hoard (since we are using an LRU system).

The hoarding algorithm must also constantly observe which context-aware applications are active. Context-aware applications which have stopped running should have their associated clusters purged from the hoard. The free space made available by the purge should be filled up with the most recently used cluster, associated with an active application, which has not yet been hoarded.

The purging algorithm is called at regular intervals and runs during the hoarding state. The purging algorithm replaces the purged clusters with new clusters until the hoard is full. These new clusters are the ones with the highest priority in the context management system which are not yet in the hoard.

If an instance of a hoarded cluster changes, the hoarding algorithm must determine if there are any other instances that need to be updated in the hoard. For example refer to the cluster template shown in Figure 3. If a new device is located near “Bob” the corresponding device type of that device must also be hoarded.

5. Conclusion

Clusters represent fact instances which are related to each other and are influenced by the same first class objects. Traditional distributed systems methods of improving robustness in the face of disconnections can be applied to context-aware systems. However the additional metadata and information about clusters available to context-aware systems, may be leveraged to provide more effective hoarding algorithms.

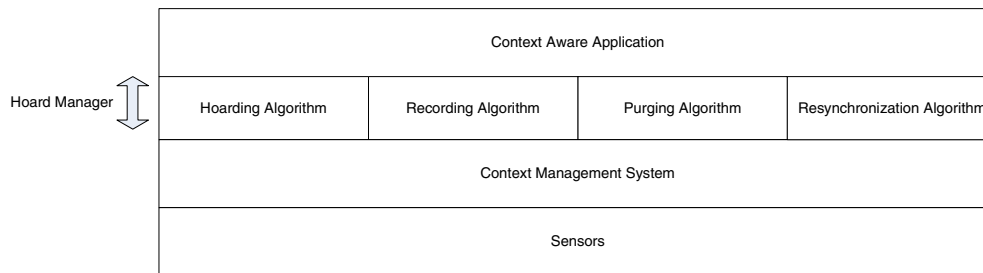


Figure 4. Logical View of System

6. Acknowledgments

National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts; the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs; and the Queensland Government.

References

- [1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communication environments. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 199–210, New York, NY, USA, 1995. ACM Press.
- [2] M. Anandarajah, J. Indulska, and R. Robinson. Caching context information in pervasive systems. In *MDS '06: Proceedings of the 3rd international Middleware doctoral symposium*, Melbourne, Australia, 2006. ACM Press.
- [3] C. Becker, M. Bauer, and J. Hhner. Usenet-on-the-fly – supporting locality of information in spontaneous networking environments. In *In R. Liscano and G. Kortuem, editors, Workshop on Ad Hoc Communications and Collaboration in Ubiquitous Computing Environments, New Orleans, USA*. ACM Press, 2002.
- [4] P. Brown and G. Jones. Researching issues in context-aware retrieval: Building a context-aware cache.
- [5] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Technical Conference*, pages 153–164, San Diego, CA, USA, 1996.
- [6] M.-S. Chen, J. Han, and P. S. Yu. Data mining: an overview from a database perspective. *Ieee Trans. On Knowledge And Data Engineering*, 8:866–883, 1996.
- [7] K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference*, pages 77–86, 2004.
- [8] M. I. Imad Mahgoub. *Mobile Computing Handbook*. CRC Press, Dec 2004.
- [9] M. S. Joseph, M. Kumar, H. Shen, and S. Das. Energy efficient data retrieval and caching in mobile peer-to-peer networks. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 50–54, Washington, DC, USA, 2005. IEEE Computer Society.
- [10] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *Thirteenth ACM Symposium on Operating Systems Principles*, volume 25, pages 213–225, Asilomar Conference Center, Pacific Grove, U.S., 1991. ACM Press.
- [11] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. *SIGOPS Oper. Syst. Rev.*, 31(5):264–275, 1997.
- [12] I. T. Kuz. *An Approach to A Scalable Wide-Area Web Service*. PhD thesis, Technische Universiteit, 2003.
- [13] M. Mahdavi, J. Shepherd, and B. Benatallah. A collaborative approach for caching dynamic data in portal applications. In *ADC '04: Proceedings of the fifteenth Australasian database conference*, pages 181–188, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.
- [14] A. Mishra, M. Shin, and W. Arbaugh. Context caching using neighbor graphs for fast handoffs in a wireless network, 2003.
- [15] M. Papadopouli, 2001. 7DS FAQ, <http://www.cs.unc.edu/maria/7ds/faq.html>.
- [16] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *Proceedings of the 2nd ACM international symposium on Mobile Ad Hoc Networking & Computing*, pages 117–127. ACM Press, Long Beach, CA, USA, 2001.
- [17] Q. Ren and M. H. Dunham. Using semantic caching to manage location dependent data in mobile computing. In *Mobile Computing and Networking*, pages 210–221, 2000.
- [18] Y. Saygin. Hoarding in mobile computing environments. In M. I. Imad Mahgoub, editor, *Mobile Computing Handbook*, page 200. CRC Press, 2004.
- [19] H. Shen, M. Kumar, S. Das, and Z. Wang. Energy-efficient caching and prefetching with data consistency in mobile distributed systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 67, April 2004.