
“Task-Based Hoarding of Personal Data”

Gonçalo da Silva Araújo
IST
goncalo.araujo@ist.utl.pt

João Coelho Garcia
INESC ID Lisboa / IST
jog@gad.inesc-id.pt

December 2011
Task-Based Hoarding of Personal Data

Gonçalo da Silva Araújo

João Coelho Garcia

IST

goncalo.araujo@ist.utl.pt

INESC ID Lisboa / IST

jog@gsd.inesc-id.pt

Abstract. Over the past years, users have witness important changes and improvements on their information systems. With the advent of mobility and the increasing popularity of mobile computing, users could finally gain access to all non-local information they need on their portable devices. However, this improvement has brought new problems. In the absence of readily available high-quality communication, users are often forced to operate disconnected from the network, which can be a major drawback if a remote file is needed to complete a particular task. An interesting solution to this problem, named hoarding, can be found by applying the same idea that is used in traditional caching to mobile devices: caching non-local files to the mobile device’s disk prior to disconnection. This thesis addresses the impact that users’ personal information can have on solving the problems of the current hoarding algorithms. In particular, it proposes a hoarding algorithm that, by classifying and grouping documents into tasks, combines the information about the tasks performed by the user and the information on its personal calendar to determine which data should be stored on the mobile device. Finally, we present some results to understand the performance differences between the proposed algorithm and previous solutions.

1 Introduction

The increasing popularity of mobile devices such as laptops and smartphones and the expansion of wireless networks gave users the power to access the network without requiring a fixed computer station to do it. This new concept of mobility brought however new problems, related to the lack of consistency on accessing wireless networks. One of those is the problem of disconnected operation, which happens when the users are forced to disconnect from the network (either because of lack of wireless signal or because of the expensive costs of accessing to the network) and therefore cannot access non-local data. This can become a huge problem, as nowadays users are increasingly relying on distributed resources to perform their tasks, and for such users, loosing access to non-local data can, in the worst case, make a mobile device impossible to use.

One possible solution for this problem uses the existing concept of caching and tries to adapt it to mobile devices: by saving some of the local data to the mobile device’s local memory (which would be the data required to execute the applications the user would need in the future), when the user disconnects his mobile device from the wireless network he can keep access the non-local information and data he needs,
as it has been previously copied to his mobile device’s local memory. This solution uses selective data replication, and is named \textit{hoarding}.

The goal with hoarding is, just as in caching, to minimize the costs on accessing the data that is needed. Despite similarities, there is one important difference between this two techniques: (1) if in a system that uses caching a cache miss cost is low (when there is data that the system needs to access and it’s not in cache, the cache miss cost would be the cost of accessing that data on a slower storage device), (2) in a hoarding system a similar cache miss can have severe consequences because, without network access, it is not possible to access to that required non-local data, and therefore could result in an error that could stop the system’s execution. [KRP02].

This raises the need for a hoarding optimization, so that one could copy to the mobile device’s local memory all the data needed to avoid cache misses during the time the user is disconnected to the network, but without exceeding the mobile device memory size. This challenge has, however, no optimal solution, as it would require a system to predict the user’s future behavior.

Even with no optimal solutions available, there have been several solutions trying to predict an accurate hoarding set (the selected group of files that are copied to the mobile device) that can be as close as possible to the user’s future behavior. Some of those solutions, like the one used in Coda [KS92], choose to relegate to the user the task of selecting the files that will be copied to the mobile device. Others, as in AFS system [How88], don’t have any specific method to handle the problem, forcing the users to run the programs they intend to use during the wireless disconnection period so that the cache can be filled with those program’s data. Even those solutions that automatically build the hoarding set of files, as the one used in the SEER project [KP97] (in which the data selected to be copied to the mobile device is determined according to the semantic distance between them), are intended to use only a snapshot of the data accessed by the users, without taking into account other sources of information from the user that could help to build a more accurate hoarding set. Many existing solutions have precisely the problem of using only the set of past accesses to the user’s files as an information resource for building the hoarding set. In addition, the hoarding set is normally built based on the set of individual files that users will need, without taking into account the type of relationship that may exist between certain groups of files. This causes related files not to be kept together.

It is therefore necessary an alternative solution to the existing methods, one that could build the hoarding set automatically and accurately, using all the relevant sources of information it can get from the user. Likewise, it is important that this solution would take into account the relationships between files so that when you build the hoarding set, all the files related to the activity that the user will perform in the future can be copied to the mobile device. This is the problem addressed in this work.

1.1 Goal

A possible solution to the problem described above could use tasks’ hoarding instead of files’ hoarding. A \textit{task} can be described as a set of documents that, according to the user's perspective, are all related and that are associated to a common idea, where this idea would label the set of documents [Gar09]. By building a hoarding set based on tasks, the hoarding process would decide on which tasks should be copied to the mobile device by calculating the semantic distance between the existing tasks, instead of the semantic distance between individual files. This semantic distance could be
calculated using not only a regular MRU\textsuperscript{1} method based on tasks (recording the most recently performed tasks), but also using the user’s personal information – like his personal calendar/agenda - whose future behavior can help in understanding what tasks he will perform. A user’s calendar can be used not only to gather information about which tasks the user has planned on his future calendar, but it can also be used to predict his future behavior based on his past calendar, determining behavior patterns on the tasks he previously performed. All this information could then be classified in tasks by using proper classification document’s algorithms.

In this document, we try to investigate the contribution of the user’s personal information (that is, his performed tasks and his personal calendar) to the optimization of hoarding. To evaluate it, a task-based automatic hoarding algorithm was implemented, described properly on chapter 3.

2 Related Work

The problem described in the last section is not, however, a newly discovered issue, and therefore there are many different existing approaches that try to address this problem. In this chapter we will describe some of those approaches, first by discussing systems that use other methods than hoarding to solve the problem of accessing non-local files during the disconnection period; then by describing an existing hoarding method; and finally describing an algorithm for documents’ classification, which is necessary for the proposed algorithm so it can classify the user’s documents into their proper tasks.

2.1 Personal Data Management Systems

Personal data management systems are not a new topic, there have been such systems since many years ago such as Microsoft Briefcase [Bri10], Rsync [TM96] or Data Staging[FSTS03]. A recent and popular approach to it is Dropbox [Dro10]. This system is a file storage service operated by Dropbox, Inc. using cloud computing techniques to allow users to store files and folders on the network, sharing them with other dropbox users. This service consists mainly on a client application that is installed on your computer or mobile device user, where it also creates a folder that is linked to the Dropbox client. The user can then place any file in the folder associated with the client application, which is then synchronized with the copy of the repository hosted on Dropbox’s servers, using Web Services [W3C10], as well as with any other copies of the repository hosted on other computers or mobile devices containing the same client of this application.

Despite being an excellent service for sharing and synchronizing files, it is not very interesting when the goal is to have an automatic solution to solve the problem to perform non-local operations when disconnected from the network. This is because the choice to synchronize the files must always be made by the user. Relaying to the user the task of building his hoarding set is not only time consuming for the user but also a not very effective method because the user is not always aware of the all the files that are essential for those applications that he needs during the disconnected period can be executed. Another major disadvantage is that, for many mobile

\textsuperscript{1} MRU – Most Recently Used
devices, the size of all content on Dropbox becomes eventually unaffordable, not only because of the limited mobile device’s local memory space, but also because of the high costs of current network to transfer large amounts of information.

2.2 Hoarding Algorithms

There are several existing algorithms nowadays that keep trying to optimize the hoarding process. Some examples are the ones used in AFS system (How88), in CODA (KS92), SEER (KP97), Transparent Analytical Spying (TLAC95) or COGENIA(RST05), and they all use different methods to build the hoarding set.

Another relevant method used by UBI DATA (ZHH03) behaves as shown in Fig. 1. This system uses an algorithm that by hooking into the operating system’s kernel, collects user file events (such as file open, close, write, etc) that are stored in a buffer of a pseudo-device driver. These files are then read by an application called Collector which filters the file access, discarding non-interest actions (e.g. files that were accessed because of an “ls” UNIX command), and piping the interest actions to another application, the Analyzer. It computes the hybrid priority (a method for ranking files based on recency of access, frequency and active period) for every interesting file. The list of files with highest priority during a period of time constitutes the candidates for future hoarding, and is published into the mobile server. Then the system, periodically or before anticipated disconnections, uses the hoarding list to hoard data to the mobile device.

![Fig. 1 – Ubidata’s algorithm](image)

This is an interest method that makes the hoarding process almost transparent to the user but, like most of the existent hoarding algorithms, it builds its hoarding set by looking at the file’s semantic distance only, which is determined based only on user’s file access.
2.3 Documents’ Classification Algorithm

A hoarding algorithm that builds its hoarding set based on tasks needs to classify the user’s documents into tasks. This is a very crucial part of the algorithm as it is needed an accurate classification to avoid unnecessary attention from the user that would have to, otherwise, pay attention to check if his files were being classified in a correct way. Despite there being several documents’ classification algorithms, here we present the one that was used on our solution, Support Vector Machines (SVM) [Vap00, Seb02]. This was the chosen algorithm based on a document’s classification methods study made by Y. Yang and L. Lui in 1999 [YL99] which compared several algorithms and pointed at SVM as one of the most accurate methods.

SVM is a method that, given a set of training examples, each set belonging to a given category, tries to classify a new example into one of those categories. For this task it represents the examples as points in a given space, mapped so that the examples of the separate categories are divided by a clear line that divides both categories at their maximum distance. When a new point is intended to be classified, the algorithm maps the point into the same space as the already classified points, and classifies the new one into the category for which the position where the point was mapped belongs.

3 Hoarding using Personal Information

As described earlier, the goal of this work is to investigate the contribution of the user’s personal information (that is, he’s performed tasks and he’s personal calendar) to the optimization of hoarding, and to achieve this we design a hoarding algorithm so that it could later be submitted to users evaluation and we could get conclusions from the tests results.

This algorithm has uses the concept of task hoarding instead of single files hoarding. A task is the text representation of an idea of the user's work. That is, is a set of files that are used together and therefore are labeled with the name of the user activity they are being used for. By performing task hoarding we can the problems discussed before about the difficulty in detecting reliable dependencies and semantic relationships between individual files. Since each task is associated with a set of files that share the same idea concerning about the user's personal work, by choosing a particular task to be hoarded, it is guaranteed that all of the files needed to complete that task are hoarded to the mobile device.

An important part of the algorithm is, therefore, the mechanism by which the files are associated with the tasks they are being used on. To do this the algorithm records the user’s file accesses and, for each relevant file, classifies it into its given task. This classification is performed used the document’s classification algorithm described in section 2.3, SVM. The file classification is performed automatically by the SVM algorithm, although the user is allowed to change the task where the file was classified, if he thinks the classification performed incorrectly.

Having the user’s files classified in their associated tasks, the hoarding algorithm can then, when a disconnection period occurs, select the tasks that most likely will be required by the user and copy their files into the mobile device. This selection is made based in task prioritization. This means that the algorithm will create a task rank, by ordering the tasks in a list, in ascending order (from the most likely to be needed in the future to the least likely) depending on the probability of each task being able to
be used in the future. From this list the tasks are selected to be part of the hoarding set, starting with the task that is in the top of the hierarchy and continuing up to the task whose position indicates as being the least likely of being needed in the future. This process is repeated for as many tasks as the mobile device’s local memory can support.

Fig. 2 shows a diagram of the architecture of the algorithm, and explains the process of determining the rank of a given task in user tasks list.

![Diagram of the Algorithm Architecture](image)

**Fig. 2 - Arquitectura do algoritmo de hoarding**

In order to rank the user tasks, a metric is used that combines different prioritization sub-lists into one final task’s order list. Each of those sub-lists is built according to different type of information about the user’s behavior. Here are the indicators that are used to rank a given task:

- **The time since the user accessed any document associated with this task.** Using the same metric as in MRU algorithm, most recently accessed tasks will get higher ranks than the tasks that have been accessed before.

- **The probability of the task being executed in the coming future.** This indicator seeks to use information of the user’s calendar (both future and past events) to determine which tasks are most likely to be performed by the user. To create this rank, two sub-indicators are used:
  - **Future appointments of the calendar.** The future appointments on the user’s calendar indicate events that the user will perform in the near future in a given time. By classifying those
appointments into their corresponding tasks (by classifying the appointments’ name using SVM algorithm), we can determine which tasks will be performed at a given time;

- **Predicted appointments.** Because it is very unlikely that the user fills his calendar with appointments for all of his future events, the algorithm tries to predict future tasks that the user will perform. This prediction is made by checking possible task patterns that may exist in the user’s past calendar, classifying the calendar past assignments into tasks (using SVM algorithm to classify the assignment’s name). If a given task reveals a periodic pattern (for instance, is repeated daily, weekly or monthly) then it is likely that the same task may be performed by the user in a given period in the future. To increase the amount of information that can be used to find patterns, it is used not only the assignments from the user’s past calendar but also a user’s task history that records the times the user performed a given task.

When creating the final task list, that is, the list that contains the tasks ranking that will be used for hoarding, all the different created lists (described earlier) are joined as in Fig. 3, through two main steps:

1. Joining the ranking list of the tasks from the future calendar with the ranking list of the predicted tasks to create a “Calendar task list”;
2. Joining that Calendar task list with the ranking list of the most recently accessed tasks (MRU).

On the first step, the joining can be made using the **weight union**, and the rank of a given task \( t \) is defined according to the following formula:

\[
Cal(t) = \text{Fut}(t) \cdot i + \text{Prev}(t) \cdot j
\]

where there are given weights to give more emphasis to the future tasks list, as the tasks detected in the future assignments (created by the own user) are is more reliable than the predicted tasks. In this formula, \( \text{Fut}(t) \) is the rank of \( t \) in the list of tasks from the future calendar, \( \text{Prev}(t) \) is the rank of \( t \) in the list of the predicted tasks, and \( \text{Cal}(t) \) is the rank of \( t \) in the final calendar task list. By default \( i=0.6 \) and \( j=0.4 \).

Another method for joining those lists is the **data union**, where there aren’t weights to associate to the lists. Instead, for each day in a given future period, first all the future tasks scheduled for that day are added into the calendar task list, and only after that is when all of the predicted tasks scheduled for that day are added too.

Finally, in the second step, the joining of the calendar task list with the MRU task list is made according to the rank of the tasks, which is defined according to the following formula:

\[
\text{Hoarding}(t) = \text{Cal}(t) \cdot k + \text{MRU}(t) \cdot l
\]

where there are given weights to the lists just like in the weight union method described earlier. In this formula, \( \text{Cal}(t) \) is the rank of a task in the calendar task list, \( \text{MRU}(t) \) is the rank of a task in the MRU task list, and \( \text{Hoarding}(t) \) is the rank of a task in the final task list that will be used to hoard tasks into the mobile device.
4 Evaluation

The main in evaluating our algorithm was to know how accurate the hoarding set built by the system was, and to compare it with an MRU task based algorithm. Therefore, we have focus on two main indicators for this evaluation:

- The percentage of non accessed files from the list of files that are needed so that no single miss occurs during the disconnection period (miss-free hoard size).
- Total waste memory space, that is, the size of all of the non accessed files.

Another method that is regularly used when comparing hoarding algorithms is to know the miss-free hoard size [KP97], that is, the minimum amount of memory space that is required by a given hoarding algorithm so that no single miss occurs during the disconnection period. We have decided to use this method as well, together with the two methods described earlier.

To evaluate the algorithm we have implemented it on a task manager application called Taskoscope [Gar09], and asked the users to use it during 5 days. On the fourth day, the user would press a button to hoard the files, and then during day 5 the system would check which of the hoarded files were being accessed by the user.

In Graph 1 and Graph 2 the evaluation results of the test between an MRU task based algorithm and the hoarding algorithm described in this thesis are shown:

**Graph 1 - MRU vs. Hoarding using personal info (waste of files)**
As is shown on the results, there is a performance improvement in the developed algorithm comparing with the MRU. The justification can be that by having new sources of information about the user’s future behavior, the algorithm managed to build a more accurate hoarding set. However, the amount of files hoarded that are not accessed by the user is still big, as 50% of the files were not used. One of the problems that are helping creating this huge amount of waste is that users define very high-level tasks. This means that, for instance, a given user instead of having a task for “school documents”, “family documents” and “friends documents”, has only one task with the label “documents”. This has serious impact on the hoarding results, because if this same user has many school tasks scheduled for the next days, the hoarding algorithm will get all the files from the task documents to the mobile device instead of getting only those school documents the user will really need, causing unnecessary waste on his mobile device’s local memory.

5 Conclusions

During this work our goal was to focus on the new mobility problems that are being experience by the users, and to develop a new hoarding method that could help solve the problem of “disconnected operation”. We have presented an alternative algorithm that, by hoarding tasks instead of files, and by using different sources of information (the user’s personal calendar) would get more accurate results when hoarding. Finally we have presented some results that show improvements comparing to a MRU task based algorithm, making this work in a small step on searching for an optimized solution to the disconnected operation problem.
6 References


