Offloading file search operation for performance improvement of smart phones

Ashutosh Jain
CSE Dept.
IIT, Delhi
New Delhi, 110016, India
mcs112566@cse.iitd.ac.in

Vigya Sharma
CSE Dept.
IIT, Delhi
New Delhi, 110016, India
mcs112564@cse.iitd.ac.in

Shehbaz Jaffer
CSE Dept.
IIT, Delhi
New Delhi, 110016, India
mcs112578@cse.iitd.ac.in

Kolin Paul
CSE Dept.
IIT, Delhi
New Delhi, 110016, India
kolin@cse.iitd.ac.in

***Abstract*—The file search operation has always been a very time consuming and expensive operation in large secondary memory devices. As the size of secondary memory in smart phones is increasing this operation will take a lot of time and consume a lot of battery of the device. In this paper we implement offloaded file search operation in which the directory structure of the file system or the hash index is offloaded to a server and the server returns the result of the file search query of the user thereby reducing search time and improving performance.**

I. Introduction

Mobile devices such as cellphones and tablets come with a large amount of secondary memory. 32 GB secondary memory is very common in smart phones nowadays and this number is expected to grow by a large amount in the near future. A file search operation is a frequently used operation invoked by users and system tasks. With increase in the size of secondary memory and more number of files, time for searching a file will increase. Searching becomes an expensive operation as the file system size grows mainly for three reasons. First, as smart phones do not have very powerful processors, it takes a lot of time to search for a file in the file system. Second, the primary memory consumption to store the whole file system tree or a hash index in main memory is expensive as there is a limitation on the size of main memory. Third, the execution of computation intensive applications on processors or the use of very large amount of memory consumes a lot of energy and smart phones generally have very limited battery life. Whereas the secondary memory in smart phones is increasing rapidly, the amount of energy that can be stored in a battery is growing at only 5% annually [4]. Employing bigger batteries for increasing battery life is also not an attractive option for smart phones. Also without active cooling the power budget of small devices is limited to about three watts [5]. For these reasons file searching is generally not provided in less powerful phones.

With the growing number of cloud based smartphone applications, cloud computing can help improve the file search operation. As the synchronization of contacts and calendar data is widely used, users of smartphones are already adapted to store some part of their phone's data in the cloud. Therefore, we propose to offload the file search operation to a powerful server or to a cloud. Application of this type can easily be provided by a phone manufacturer to improve the file search performance of the phone.

Now, file searching is of two types: if the user searches in a particular directory, or the user searches in the home directory. For searching in a particular directory the whole tree of that directory has to be searched. For searching in the home directory a hash table can be maintained to improve the search time. In this paper we offloaded both types of searches i.e. the server can either maintain a tree or a hash table or both for the directory structure of the phone. We also compared the performances of both offloaded implementations with local searching and also with each other both in terms of time and energy.

The rest of the paper is organised as follows. Section 2 describes the related work done. Section 3 describes the model of implementation for the real world devices. Section 4 describes our implementation to evaluate the performance benefits of offloading the search operation. Section 5 describes the experimental results and observations. Section 6 concludes with the advantages and extensions of offloading the search operation and finally Section 7 discusses the future work.

II. Related Work

K Kumar et.al.[1] provide useful insights to when offloading computation can actually be beneficial. There work considers the power consumed by the device due to transmission of data while offloading the computation. They state that offloading can be beneficial if an application requires large amount of computation with relatively less data transfer, similar to the search operation application that we have offloaded. Issues and concerns about privacy and security have also been highlighted in their work. R Kemp et.al.[2] demonstrate Cukoo - a framework for offloading computation of compute intensive applications for the android platform. The framework decides at runtime whether to switch to cloud or to perform the computation locally. Their emphasis is to reduce the load on the thin client and let the fat server (cloud) perform the major compute intensive applications.

III. Offloaded file search

In our model of offloaded file search the first time when the phone user wants to use the offloaded file search, the phone obtains data set containing full paths of each of the files of the phone and it sends this data set to the server. The server maintains meta-data for each of the phones and it can distinguish between the phones either using their mobile numbers or the IMEI numbers. The server upon receiving the data set creates a tree and/or hash table corresponding to these files for that particular phone. Since the number of files are less when a phone is used for the first time, this will not involve very heavy data transfer. After this setup is done, the user can use the file search operation, the query string will be sent to the server and server replies with the file paths by searching in the tree or the hash table maintained for that particular phone. Now the thing that need to be handled here is that the user is free to 1)add, 2)delete, 3)move, or 4)rename files during the lifetime of the phone which will make the meta-data on the server obsolete. One way to solve this problem is to send the listing of full paths again every time user does any of the four above mentioned operations but that will involve very heavy data transfer as these operations are very frequent. Other is to send only the change that occured to the server but that will involve very frequent data transfer and will not work if the network is unavailable at that time. To efficiently solve this problem our model maintains a change log in the phone. The change log keeps a track of all changes done to the file structure. There will be a synchronization manager whose responsibility is to watch when any of the four operations are done and make an entry in the change log. Now the change log can contain four types of entries:

1) In case of addition or deletion of file the change log contains the file's path(the last name in the path will be the file name).
2) In case of renaming the change log contains the file's path and file's new name.
3) In case of movement of file the change log contains file's old path and file's new path.

Each entry has an indicator field to indicate what operation is done to the entry. Whenever the server receives the change log it updates its tree and/or hash table by the doing all the operations in the change log. This way the data structure on the server is synchronized with the directory structure of the phone. This sending of the change log to the server is the synchronization phase. The advantage of using this change log model is that the amount of data transfer used to synchronize the server with the phone is decreased. A typical change log with 100 changes sizes to only 10kB which will not take much time to transfer and for a high end server to process it.
Once the server does the requested operations in the change log the server sends an acknowledgement indicating that the update process is completed. The phone on receiving this acknowledgement resets its change log. Now this synchronization phase can occur at different points of time:

1) Whenever the user calls the search operation, the phone first sends the change log and after getting acknowledgement from the server sends the query string. This is the simplest approach but the disadvantage of this is that if the change log becomes large then the sending of the change log and updation at the server may take time and user may get slow down.
2) Whenever the size of the change log exceeds a certain limit. This will remove the disadvantage of change log becoming very large but the network has to be available at that time.
3) Whenever the network becomes available. This can be combined with the previous strategy to efficiently do synchronization.

Now because of the above strategies there can also be a scenario when search is to be performed but the change log is not empty, i.e. the file structure is not up to date. One of the following options can then be employed:

- The server can perform a synchronization before completing the query.
- If a local copy of the directory structure tree is also present, the search can be initiated locally, while the synchronization is being performed.
- Alternatively, the system can look up the change log for the queried file name. If it is not present in the change log, the system can proceed with offloaded search. If the change log indicates that a file of that name is added, it can return the new path from the change log, and also perform an offloaded search for other occurences of that file name. If the change log indicates that a file has been renamed to the query name, the query can be modified to include both the old and the new names before it has been sent to the server. In case the file has been deleted or moved, the offloaded search can proceed as usual, and the occurences for the deleted file can be removed from the output once it is received.

IV. Implementation

There are two parts of the implementation: implementing the search locally, and implementing the offloaded search. All implementations are written in C programming language.
The server used here is an Intel Core 2 Duo Processor with 2 GB RAM and 1.83 GHz clock speed. The processor used to simulate smart phone is Intel Atom processor with 512 MB RAM, 1 GHz processor, running Fedora 11 operating system. We assume a bandwidth availability of ethernet, 100 Mbps. This is inline with the assumption that with increasing availability of better networks in future, bandwidth will not be a bottleneck in data communication.

A. *Tree based search*

For implementing the local search we first need to have the full paths of all the files in the file system which we can convert to a tree. This is done in linux using find command. Then a program reads this file and converts this to a directory tree structure. The program then reads a filename and a directory name and searches for the file(s) in the directory and returns all the paths where the file is found. In the offloaded search program the file obtained using find command is sent to server and the program at server reads this file and converts it to directory tree structure. The server program then waits for input from the client side. The client program inputs the filename and the directory name and sends them to the server and waits for the reply from the server. The server program receives the filename and the directory name, searches for the filename in the given directory and whenever a file is found it sends the file path back and resumes searching. In this way the client receives the file paths incrementally.

B. Hash based search

Hash based search is used when user searches in the root directory i.e. irrespective of the location of directory. It is also used in the desktop search in windows where as the user types a filename it starts giving suggestions. So, our hash table consists of 27 locations and collision resolution with chaining is done. The hash function which we used is to put all the files which have their name starting from 'a' to the same location and similarly for other alphabets. All filenames which dont start from alphabet are put into 27th location. Local search and offloaded search programs are same as in tree based search except that searching is done in hash table instead of tree.

V. Observations and results

For comparing the performances of different implementations we have done, we generated sample data by running find command on different directories of our computers so that results for different number of files are obtained. These files are then input to the programs we have made and statistics are obtained.

A. Local tree based search vs offloaded tree based search

Figure 1 shows the times obtained for different number of files for local search and offloaded search. As the results show

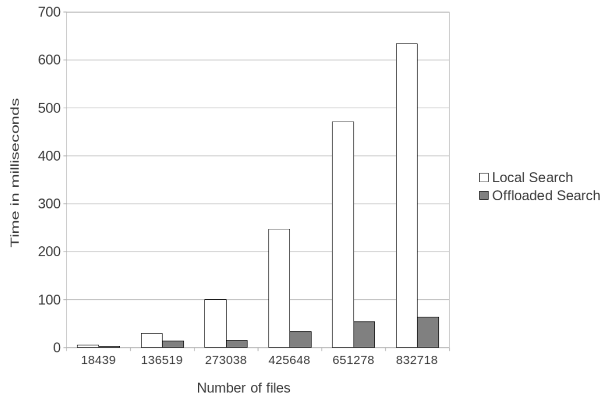

Fig. 1. Comparison of times taken for local tree search and offloaded tree search

offloaded searching is definitely performing better than local searching and as the number of files increase the benefit of offloading increases. But the difference in search time is not enough. For battery enabled devices we have to care of the power as well. So we next measured the difference in the energy required to perform local tree search and offloaded tree search. Figure 2 shows the results. The results clearly show that offloaded tree search consumes less energy than local tree search because none of the calculations occurs on the phone's processor. The client process job is to just send the search string and receive the search results. So the offloaded tree search will improve the battery life of the phone as well.

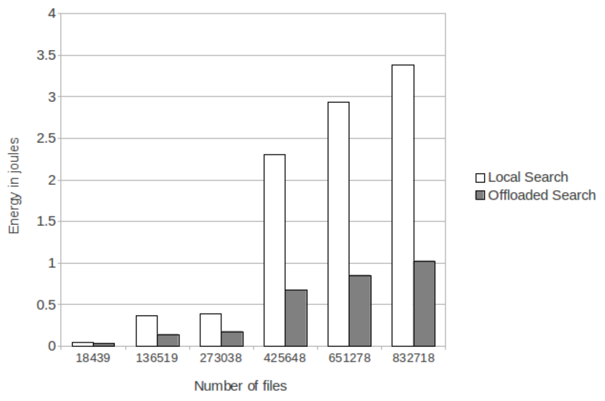

Fig. 2. Comparison of energy consumed in local tree search and offloaded tree search

B. Local hash based search vs offloaded hash based search

For using the local hash based search the processor of the smart phone has to build the hash table each time the user switches on his phone and also the hash table has to remain in main memory till the phone is on. So we first show the time to build the hash table and the memory requirements of hash table for different number of files. Table I shows the results. The results clearly show that hashing is not possible on smart

Number of files	Build time(in secs)	Memory usage(in MB)
20000	0.30	37.6
40000	0.59	78.2
60000	0.88	117
80000	1.21	156
100000	1.54	199.3
120000	2.40	276.8
136000	Process killed	Process killed

TABLE I
Time required to build the hash table and memory usage

phones. The reasons are:

1) Hashing requires a lot memory and such large amount of memory is not available on phones so the operating system kills the process.
2) Even if memory is available sometimes its not resource optimal to allocate such large amount of memory to one process.
3) The time it takes to build the hash table becomes very large. Users can't wait for so long when the phone is started.

Because of these reasons to take advantage of hashing, hash table has to be offloaded. Figure 3 shows the search time comparison and Figure 4 shows the energy consumption of offloaded hashing and offloaded tree based search.

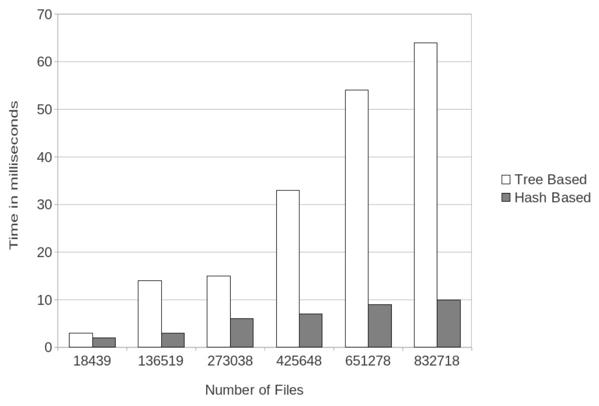

Fig. 3. Comparison of times taken for offloaded hash based search and offloaded tree based search

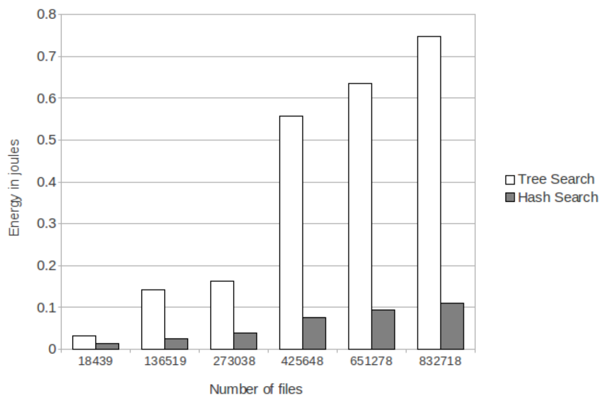

Fig. 4. Comparison of energy consumed in offloaded tree search and offloaded hash search

From the results it is clear that hashing outperforms tree based search not only in terms of time but also in terms of energy because the process of searching completes in less time. So if the user wants to perform a search on the root directory then hashing is the best option. Also if the user wants suggestions while searching as in desktop searching then hashing has to done because then it is easy to report all the items in the chain of the hash index if the user has only typed one character instead of searching the whole tree for the filenames starting with that character.

C. Is offloading always beneficial

We conducted a study to understand whether offloading is always beneficial. It was found that for large directory structures, offloading search was a viable option and yielded significant speedup. But if the number of files is less, a local search is found to provide speedup. Figure 5 indicates a threshold, after which offloaded search yielded speedup. As the number of files is less the current flow increases by a very small amount from the normal and according to the equation $Energy = Voltage \times Current \times Time$, the voltage is fixed so this time threshold works as the energy threshold as well. Using the threshold obtained, a local decision can be made. If

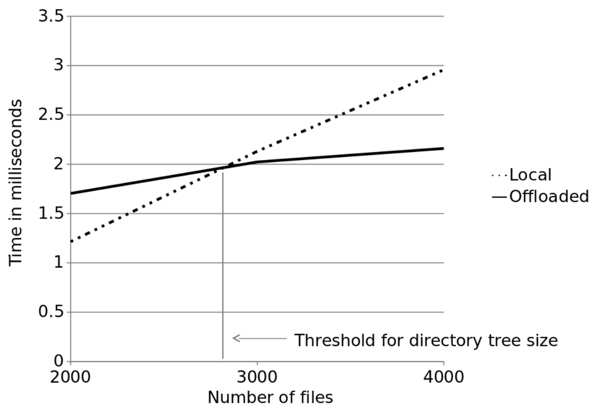

Fig. 5. Threshold after which offloading performs better than local search

the number of files is less than the threshold, a local search can be done, otherwise an offloaded search can be performed.

VI. Conclusions and Extensions

1) The study revealed that offloaded searching definitely performs better than local searching after a threshold.
2) Hash based searching is not feasible for smart phones because it takes a lot of memory.
3) If the user searches in the home directory then hashing is performing better than tree based search and this can be used in smart phones like the desktop search option.
4) Tree based searching is useful when user searches in a particular directory because it is difficult to maintain hash table for every directory.
5) Reducing the computation load from the smart phones increases its battery life.
6) Searching can be made context aware so that the results can be arranged based on context. e.g. if the user is in office then the results from the office directory will be displayed on the top. Changing of the context doesn't involve any change of directory structure in the phone. The phone only has to send a signal to the server whenever its context changes.
7) Users call and message records can also be offloaded to the server so that large call record histories can be maintained and the call records wont get lost when the user changes his sim card.

VII. Future work

We will implement this offloaded file search on android platform. The file manager of the operating system need to be modified to write to the change log whenever any of the four operations mentioned in Section II are done. Also like the desktop search option a client for android platform will be developed which will give file name suggestions as the user types the name and it will ease the user in opening files.

References

[1] K. Kumar and Y. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," in *IEEE Computer*, vol. 43, no. 4, pp. 51-56, 2010.

[2] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *3rd International Conference on Mobile Computing, Applications, and Services (MobiCASE)*, Santa Clara, CA, USA, 2010.

[3] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

[4] S. Robinson, "Cellphone Energy Gap: Desperately Seeking Solutions," *Tech. rep., 2009. Strategy Analytics*.

[5] Y. Neuvo, "Cellular phones as embedded systems," in *Digest of Technical Papers, IEEE Solid-State Circuits Conference (ISSCC)* vol. 1, pp. 3237, 2004.