Mobile Search and the Cloud: The Benefits of Offloading

Eemil Lagerspetz and Sasu Tarkoma
Helsinki Institute for Information Technology HIIT
University of Helsinki
P.O. Box 68 FI-00014 University of Helsinki, Finland
Email: firstname.lastname@hiit.fi

***Abstract*—This paper presents the benefits and drawbacks of mobile desktop search coupled with cloud-assisted operations, such as operation offloading, cloud storage, and cloud-assisted search. The energy trade-off when offloading a task is analyzed and measured in several different scenarios. An example case of offloading indexing is presented. The problem of cloud-assisted mobile desktop search is introduced and a possible solution outlined. This paper argues that the synergy between mobile platforms and cloud computing is under-utilized and should be explored further, particularly in the search and synchronization use case. Our measurements support offloading (parts of) search related tasks to a cloud service.**

I. Introduction

Internet services in the form of smartphone applications or so called widgets have become commonplace. The contemporary smartphone does not rely on local applications, but rather lets data centers and Internet servers handle part of the user's daily data processing tasks. Services such as weather, instant messaging, social media, Internet TV, email and even document processing are delivered through Web-based applications that run entirely or partially on a remote server.

It should therefore come as no surprise that offloading some tasks to remote processing facilities is clearly advantageous for the mobile device. Its processing power and memory capacity are limited compared to contemporary desktop computers. The data storage capacities of smartphones are increasing at a rapid pace, but their battery lifetime seems to even decrease compared to older, less capable models. Therefore energy conservation should be a priority on mobile devices. Local storage is expendable, but local processing and memory capacity should be conserved where possible. It is worth noting that offloading a task requires communication; the service that is to handle the offloaded task must at least know the nature of the task in question. In most cases, the task requires data that is available either on the phone or on the Internet. Communicating the task details and the required data to the service costs energy just like local processing. It is therefore always a trade-off; one cannot offload all of the phone's tasks to the cloud since communicating them costs precious energy.

In the rest of this paper, We analyze the energy trade-off between local computation and offloading the task to a remote service and present measurements in Section II. We describe a scenario of cloud-assisted desktop search on mobile devices in Section III. Section IV discusses the prototype implementation. Section V discusses related work, and Section VI concludes the paper.

II. Task Offloading: Energy considerations

The offloading of a mobile computing task is a trade-off between the energy used for local processing and the energy required for offloading the task, uploading its data, and downloading the result, if necessary. One can express the offloading energy trade-off with the formula

$$E_{trade} = E_{local} - E_{delegate} > 0,$$

where E_{local} is the energy used for complete local execution, and $E_{delegate}$ is the energy used from the perspective of the mobile device if the task is offloaded. If E_{trade} is greater than zero, then there is an energy benefit for delegating the task to the cloud. We can break down E_{local} into the task complexity C (number of instructions to be executed), the speed of local execution X_{local}, and the power used for local execution P_{exec}. E_{local} then becomes the energy used when executing C instructions at X_{local} instructions per second with the power P_{exec}:

$$E_{local} = \frac{P_{exec} \times C}{X_{local}}.$$

$E_{delegate}$, the cost of delegating the task to the cloud, combines the energy spent sending the task and data to the cloud E_s, idly waiting for the cloud to complete the task E_{idle}, and receiving the result of the task E_r:

$$E_{delegate} = E_s + E_{idle} + E_r.$$

Since $E = P \times T$, each energy variable can be broken down to corresponding average power P and time duration T. This is shown below:

$$E_{delegate} = P_s \times T_s + P_i \times T_i + P_r \times T_r.$$

In an optimal case, T_i could be minimized by spending the idle time performing other scheduled tasks, but this may not always be the case. The energy spent for data transfers could also be mitigated by scheduling offloading to coincide with other tasks that require network connectivity, such as VoIP or SIP calls. Here we consider the unmitigated case.

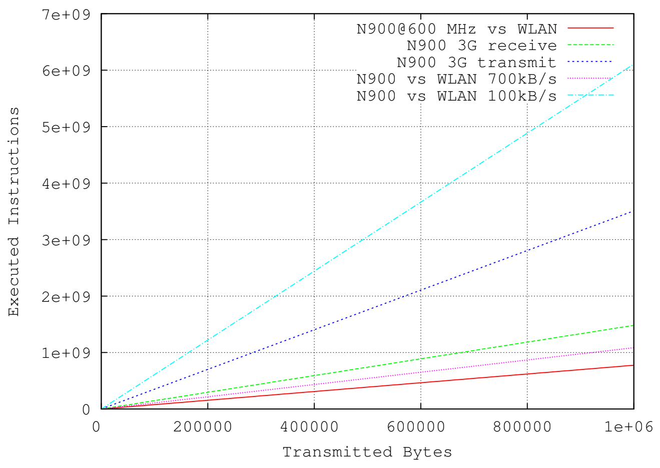

Fig. 1. Transmission Energy Versus Computation Energy.

We can further break down T_i if we take into account the execution speed of the cloud, X_{cloud}. Then T_i becomes the time the cloud uses for executing the task. Finally, when we take into account the amount of data to be sent and received D_s and D_r, and the sending and reception bandwidths B_s and B_r, the trade-off formula becomes

$$E_{trade} = \frac{P_{exec} \times C}{X_{local}} - \frac{P_i \times C}{X_{cloud}} - \frac{P_s \times D_s}{B_s} - \frac{P_r \times D_r}{B_r} > 0$$

which can be written as

$$E_{trade} = C(\frac{P_{exec}}{X_{local}} - \frac{P_i}{X_{cloud}}) - P_s\frac{D_s}{B_s} - P_r\frac{D_r}{B_r} > 0$$

where $P_{exec}/X_{local} - P_i/X_{cloud}$, P_s and P_r depend on the local and cloud platforms. It can now be seen that a higher task complexity increases the benefits of offloading the task, while a smaller bandwidth and higher amount of data to be transferred result in offloading becoming less attractive.

Note that the network and data transfer latency and other delays are included in the bandwidth in this formula. In this paper, we do not consider such delays separately.

Notice that the trade-off is positive when

$$C\frac{P_{exec}}{X_{local}} > C\frac{P_i}{X_{cloud}} + P_s\frac{D_s}{B_s} + P_r\frac{D_r}{B_r}$$

or

$$C\frac{P_{exec}}{X_{local}} > C\frac{P_i}{X_{cloud}} + \frac{D_s}{B_s}P_s \qquad (1)$$

if no data needs to be received by the mobile device, and

$$C\frac{P_{exec}}{X_{local}} > \frac{D_s}{B_s}P_s \qquad (2)$$

if we assume the idle time is negligible or that the cloud platform is fast compared to the mobile device.

Based on this analysis, we have made measurements on a Nokia N900 device, and we can match the amount of energy required for local computation against that required for data transmission, yielding a basic threshold for offloading.

For offloading to be beneficial, the amount of energy used for computation required for the task must exceed the energy required for transmitting the task's prerequisites, such as operation specifics and data, to the cloud. Figure 1 shows the thresholds for offloading tasks to the cloud. Using Formula 2, the values P_{exec} and P_s (in watts), X_{local} (in Hz), and B_s (in B/s) are measured for each scenario. C is calculated for each D_s up to 1 000 000 using the formula. The x–axis shows the amount of bytes transferred, and the y–axis shows the number of instructions executed. Each point on the lines matches the energy used for data transmission with the number of instructions that could be carried out with that amount of energy. In the Figure, we have not considered the time that the mobile client will spend in idle state waiting for the remote task to complete; we assume that the cloud platform is much faster in execution and/or the local platform uses much less energy in idle mode. For a given task with a fixed number of instructions and data to be transmitted, Figure 1 shows whether offloading the task is beneficial for a given communications medium on the Nokia N900, assuming that executing the task remotely is quick.

Language	I/LOC	LOC/Word Required
Assembly	1	6000
C	5	1200
C++	7	857
Java	10	600

TABLE I
LINES OF CODE PER WORD IN A 1MB FILE REQUIRED TO FILL ONE BILLION INSTRUCTIONS.

Each of the lines is a different scenario on the Nokia N900. In all scenarios, the N900 is running at 600 MHz for computation purposes, and consumes 0.9 W of power per second, as measured by Miettinen et al. [1]. This yields 667 million instructions per joule, i.e. $X_{local}/P_{exec} = 667$ MI/J. In *N900@600 MHz vs WLAN*, *N900 3G receive*, and *N900 3G transmit*, the Bytes per joule values are taken from the same paper [1]. In *N900 vs WLAN 700kB/s*, The Nokia N900 was benchmarked using Wireless LAN at the University of Helsinki, and downloading at 700 kB/s yielding $B_s/P_s =$ 614 kB/J. With a lower transfer rate of 100 kB/s, the energy efficiency is 109.2 kB/J, shown as *N900 vs WLAN 100kB/s*.

In effect, one megabyte of data transmitted via WLAN requires the task complexity to be around one billion instructions for it to be beneficial for offloading. This has several implications.

One billion instructions for a text file of one megabyte can be easily spent by basic indexing. If we assume that the text file is in UTF-8 and that the average word length is 5 characters, as is often cited as the average for the English language, there will be around 170 000 words in the file if one accounts for a space between the words occupying one extra byte. In his white paper on scripting languages, J. Ousterhout[1] places Java at 10 machine instructions per line of code, C at 5, and C++ at around 7. Based on these estimates, Table I shows how many lines of code run for each word in the

[1]http://home.pacbell.net/ouster/scripting.html

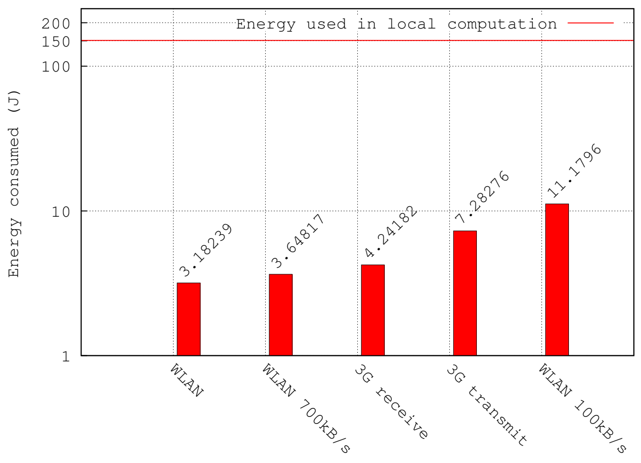

Fig. 2. Energy consumed when offloading versus local execution.

file in each programming language are required to fill up one billion instructions. In the table, the first column shows the programming languages. The second column shows the estimated number of machine instructions each line of code in that language translates to. The final column shows the number of lines of code of processing required for each word in the 1MB file to reach one billion instructions. It is not an easy task to develop an indexer that would with all of its libraries amount to less than 600 lines of processing for each word in Java or less than 1200 in C. Therefore indexing for text files in English can be offloaded. The measurements below clearly show this.

To verify the reasoning above and give an indication of the savings offloading can bestow, we carried out an indexing benchmark on an English text file on the Nokia N900. The results of the benchmark are shown in Figure 2. The y-axis has a logarithmic scale. Each bar shows the energy used when the communication medium marked under it is used for offloading. The line above the bars shows the energy that would be used if the indexing was done locally without offloading. The benchmark setup and its results are described in detail below.

The text file used in the benchmark was a concatenation of Jane Austen's Pride and Prejudice and Grimms' Fairy Tales from Project Gutenberg[2] in UTF-8 text. Extra front matter and back matter such as license and publication details was removed from both books before concatenation. Afterwards, chapters from Grimms' fairy tales were dropped until the file was 1000510 bytes long. The resulting file had 181 692 words, with average word length 4.59 characters. The Dessy desktop search system, described in Section IV was run on the N900 to index the benchmark file. The benchmark was repeated five times with five second pauses between runs. The indexing took 167.654 on average, with standard deviation $\sigma = 2.315$ seconds. The indexing process is CPU bound for the N900, since any delays caused by reading the file would be negated by caching when running the benchmark multiple times in succession, and such a drop in execution time was not measured. The benchmark was run at 600 MHz. This means the indexing took $600\,000\,000 \times 167.654 = 100.6$ billion instructions. The number of instructions may be compounded by the reduced instruction set of the ARM processor used in the N900. For comparison, on a dual-core desktop machine running at 3.16 GHz the same operation took 2.244 seconds with $\sigma = 28.33$ If we follow Formula 1, use the desktop machine as the cloud service, and take the idle time into account, even if the idle power P_i was equal to full execution power P_{exec}, the trade-off would be:

$$\frac{100.6GI}{667MI/J} > 0.9W \times 2.244s + \frac{1MB}{614035.09kB/J},$$

or $150.82J > 3.65J$. Here 614035.09 kB/J is the transfer efficiency for the *WLAN 700kB/s* scenario described before. The energy savings in this scenario when offloading are therefore $1 - 3.65J/150.82J = 97.6\%$. If the file was transferred with the least efficient method shown in Figure 1 (with 109.2kB/J transfer efficiency), the energy savings would still be 92.6%. The energy consumption measured in the benchmark is summarized in Figure 2.

With the analysis and measurements above, we arrive at the following guidelines:

1) tasks with little or no data should always be offloaded.
2) tasks with heavy computation requirements can be offloaded.
3) tasks that require secondary data transfers of greater length than required by offloading should be offloaded.

In the above, point three refers to tasks such as the indexing of HTML documents or PDF's. These files contain references and links to other documents of the same type and other resources on the Internet. While it would require a lot of energy and time for the mobile device to download, index and rank these documents, the cloud is well connected to the Internet and has a virtually unlimited energy supply. Therefore this type of task is best offloaded to the cloud. If the number of bytes to be downloaded to obtain linked documents is greater than the length of the original document, offloading is beneficial. In this case one can use heuristics such as the average size of a PDF file and the number of links and references. Also, one can use web search engines to find and connect to the HTTP servers hosting linked documents, and examine the length of the document reported by the HTTP server. This results in some communication, but may be preferable to offloading a large file with few linked documents.

The next section considers a cloud-assisted search scenario and examines the case of cloud-assisted index maintenance.

III. Global Cloud-Assisted Search

Cloud-based search and indexing can alleviate mobile device processing and energy concerns in several ways. One promising offloading scenario involves maintaining a virtual replica of the mobile device's documents at the cloud. This enables cloud-based indexing and search, and the mobile can easily update the virtual document storage by communicating

[2]http://www.gutenberg.org

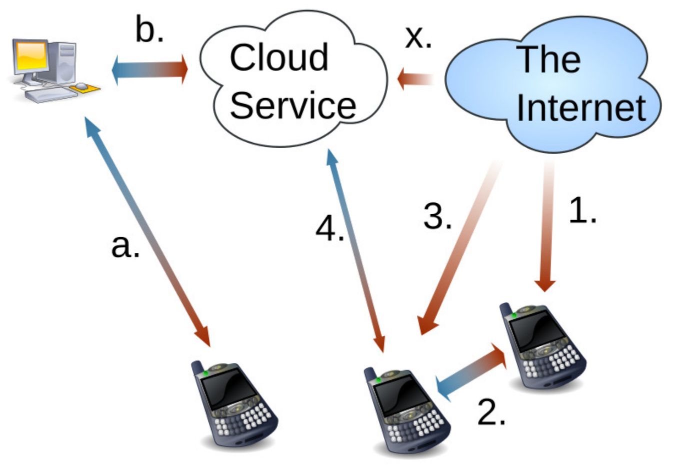

Fig. 3. Mobile global search scenario.

the URLs of downloaded documents (if the documents are available on the Web). The cloud platform should consist of different components, for example indexing component, virtual document storage, search component, and a security component.

In this section, we present the problem of global search and synchronization, and some proposed solutions. The problem is what an increasing number of computing users face: The need to find and obtain a certain piece of data which may reside on any of the many devices that the user has used in the past. To solve this problem, some basic functionality is required. Any device that the user has used must be able to respond to a query that consists of a description of data with a number of result descriptions. For devices that are off-line or inaccessible at a given point in time, it must be possible to retain a summary of their contents at an accessible location. Finally, it must be possible to obtain the latest copy of the queried piece of data based on the descriptions given by devices. If a device is inaccessible and has not synchronized the piece of data with any of the accessible devices, it may be considered sufficient for other devices to know which device the piece of data is on.

This scenario is shown in Figure 3. In the Figure, devices synchronize summaries or *index data* and files with each other in order to achieve global search. This is denoted with a double-ended arrow. Notice that devices may synchronize index data and files with any other device, not only the cloud service. In particular devices may synchronize data with more than one other device, which makes versioning more challenging. The unidirectional arrows denote download, or unidirectional sync; no new version is pushed to the origin of the file. While this is not necessary with read-write devices, it is required for read-only destinations such as HTTP servers. For read-write services such as Flickr, Picasa and Facebook, bidirectional synchronization can be used if suitable plug-ins to the global search system are developed.

In the Figure, operations are not strictly ordered. However, if devices synchronize index data and files with the cloud service after adding new files or downloading them from the Internet, the global search system can find the new files. In the Figure, the rightmost mobile device downloads some data from the Internet (1.) and then synchronizes its data with another mobile device (2.). The other device later obtains new data as well (3.) and then synchronizes with the cloud service (4.). In an independent event sequence, the leftmost mobile device synchronizes locally created files with a desktop computer (a.) which later synchronizes with the cloud service (b.). Finally, the cloud service may obtain files related to those obtained from synchronizations to improve the completeness of the index. This is shown as the arrow from The Internet to the Cloud Service (x.). It is also possible that the cloud service obtains files that are present on the other devices by URL from the Internet, thus avoiding an energy-expensive upload. This is also encompassed by the arrow (x.).

To enable truly global search, all devices should regularly synchronize at least their index data with a number of common destinations, such as the Cloud Service in the Figure. Then result files could be obtained through synchronizing them with the device that has the latest versions. This way the maintenance of the index can be offloaded to an online cloud service, allowing mobile clients to save in battery life and performance resources. This also allows both mobile clients and online services such as social networks keep up to date with the user's data. To keep indexes on the clients up to date, differential updates and Bloom filters [2] of the index could be transmitted to clients. These updates may be incorporated later or accessed through differential Bloom filters [3]. A hierarchical Bloom filter [4] can be used for predictive search suggestions and real-time search as the user types search terms. The downloading of Bloom filters and index updates allows clients to search for remote files in offline mode.

Security and privacy are current challenges for distributed search systems. A secure desktop search system needs to support secure indexes and configurable access policies. Privacy enhanced desktop search needs to prevent the leakage of personal information and preferences. A number of privacy-enhancing technologies can be adopted for searches that allow matching with only partially disclosed index data. With cloud-assisted desktop search, part of the custom security and privacy mechanisms need to be implemented in the mobile device if the cloud platform is not fully trusted. Given that the cloud platform is trusted, the security and privacy mechanisms can be fully cloud-based.

IV. Implementation

We have developed a search and synchronization application for mobile devices in the *Dessy project*[3]. Dessy was first introduced in 2007 [5]. It was extended in 2008 and 2009 to function on modern smartphones, such as Symbian phones and the Nokia N900 [6]. Its newer additions have recently been

[3]http://dessy.sourceforge.net

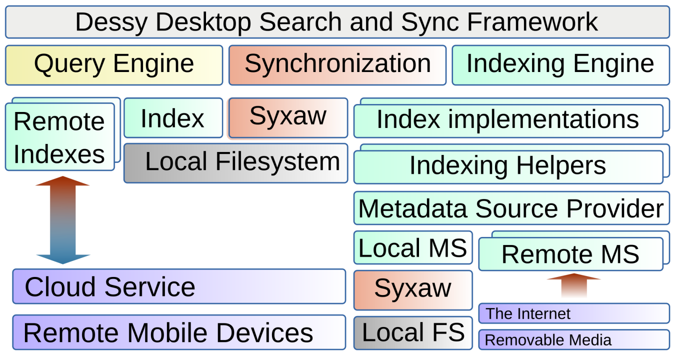

Fig. 4. Diagram of the Dessy architecture for global search.

demonstrated [7][4]. The application reads through or *indexes* a collection of files and allows finding them using words in their content, metadata, context information, and custom property–value pairs. Files may be found on remote devices running Dessy and on services such as Flickr, or through a web search plug-in. Files found with Dessy may be synchronized between two devices running Dessy, or a Dessy device and an online service. The synchronization uses the Syxaw XML-Aware File Synchronizer [8] developed at the Helsinki Institute for Information Technology.

A proposed solution to the global search problem could use Dessy described above. To enable global search and synchronization in Dessy, some modifications to the architecture are required. The modified architecture is shown in Figure 4. The query engine has been abbreviated in this Figure. The most prominent modification to the architecture is the addition of *Remote Indexes* that function side by side with the local Index. These are connection points to directly connected remote mobile devices, and the always accessible cloud service. With the remote indexes, policies can be defined for updating remote indexes. This could be done as soon as new data is available, only when strongly connected to the network, or only when charging the device. Also read-write services with indexing capabilities could appear as remote indexes. Read-only locations or locations without indexing capabilities are accessed through the *Remote Metadata Source* interface, which reads remote files and adds their metadata to the local index. For more capable and strongly connected devices running Dessy, data from read-only locations could also be locally mirrored.

In terms of the Dessy global search prototype, global search could be performed by first querying Bloom filters downloaded from known destinations. If a Bloom filter reports positive for a query, it is likely that the file exists at the destination. If it reports negative, the file does not exist. Therefore we can save needless queries by not querying the destination when the Bloom filter reports negative. In case of a positive, the query is sent to the destination and possible result descriptions received. This functionality is sufficient for individual destinations with separate data sets. However, if there are many destinations this becomes slow. Also, this does not handle offline destinations or inaccessible devices well.

Other interesting problems include the versioning of files when multiple synchronization endpoints are used and the partitioning of index data when parts of it are synchronized. In a multi-user scenario the maintenance of file access rights along with the breakdown of the index for different users is interesting. Some of these are currently being explored at the Department of Computer Science, University of Helsinki.

The use of the cloud for indexing allows better, faster and more in–depth indexing than would be possible otherwise (See end of Section II). Especially the indexing and context discovery of hyperlinked and cross-referencing documents is made possible by the better Internet connectivity and virtually limitless energy supply of the cloud.

The next section discusses related work.

V. Related Work

Wolski et al. [9] have examined various decision strategies for offloading. They predict the time taken for local and remote execution and use offloading to minimize total execution time. The bandwidth between the local and remote endpoints is taken as the bottleneck. The authors do not consider the energy aspect of offloading, as they consider a grid computing case instead of a mobile client offloading scenario.

Kumar et al. [10] consider the energy aspect and develop equations to see when offloading is beneficial. They leave the formula at a fairly simple stage and present one set of real device parameters. Their offloading scenario is similar to the one in this paper. They consider chess as an example task scenario. Chess has little data to be transmitted (piece positions) and requires a lot of computing, since for a perfect computer opponent, all possible move sequences starting from the current state to either side winning must be calculated. In reality computer opponents calculate up to a limited number of moves ahead to keep the amount of memory and computation required manageable. The authors also consider content-based image retrieval. This requires a single image to be transmitted, substantial computation for its processing and matching with a stored database of images, and then reception of a number of result images. The computation can be offloaded with relatively low cost, and storing and processing a large image database on a smartphone may not be energy-efficient in the first place.

In [1], the authors examine scenarios that would benefit from offloading. They compare the energy of transferring data with the amount of local computation possible with the same amount of energy on Nokia N810 and N900 devices, and arrive at the rule of thumb that the task must require more than 1000 cycles of computation for each transferred byte of data to be beneficial. They present some basic tasks with their computation to transfer ratios.

The offloading trade-off analysis presented in this paper is inspired by related work discussed above [1], [10].

[4]A demo paper of Dessy was also presented in MDM 2010. The paper can be found here: http://www.cs.helsinki.fi/u/lagerspe/publications/lagerspetz-demo-mdm-10.pdf

The Dessy mobile search and synchronization framework is inspired by desktop search applications such as the Tracker project[5], Beagle[6], Copernic Desktop Search[7], and Apple's Spotlight[8]. Its tagging capability and virtual file system are inspired by phlat [11] and Semantic File Systems [12]. Semantic File Systems [12] allowed devices to attach directories on a server to the local directory structure. Applications could then access these directories and their files based on their type or semantics. For example, John's device would attach the directory `/home` and only see John's files. Applications could access their data through similar directories, such as having all John's documents, be they PDF, text or formatted text documents, automatically collected at `/home/Documents`.

In Dessy, these *virtual directories* are used on the local system itself to collect the results of queries. Each virtual directory path represents a series of property–value pairs, and files found in such a path are results that match the entire query.

VI. Conclusions and Future work

Developing cloud computing platforms and mobile software in parallel is an intriguing prospect. In this paper, we have analysed the trade-off between local and remote processing from the viewpoint of energy use, taking into account the energy used for transferring the task from the local system to a remote service. When weakly connected to the network, the mobile device should offload computationally expensive tasks and delay offloading data intensive tasks. When a WLAN hotspot with sufficient bandwidth available is detected, data intensive tasks may be offloaded. When offloading, communication latency and possible user–perceived delays caused by it should also be considered.

We have described the idea of cloud-assisted search and synchronization on mobile devices. We have outlined its components and presented a proposed solution — the Dessy desktop search and synchronization system, and remaining research challenges. For cloud-assisted search and retrieval of data items, these include in particular global index updating and management, multipoint synchronization, and security and privacy aspects in a multi-user environment.

The desktop search system can be significantly improved with cloud-based technologies. Cloud-assisted search supports the offloading of the search and indexing functions from the mobile device to the cloud thus significantly alleviating energy and performance issues. Our experimental results indicate 97.6% improvement in the basic indexing scenario with English text files. In addition to basic offloading capability, cloud assisted search can be enhanced with various social networking and collaborative filtering solutions.

We have discussed related work that is useful in the context of multiple device search and indexing, and energy analysis. This paper argues that cloud-assisted search and synchronization on mobile devices is possible and realizable in the near future.

[5]http://projects.gnome.org/tracker/

[6]http://beagle-project.org/Main_Page

[7]http://www.copernic.com/en/products/desktop-search/

[8]http://www.apple.com/macosx/what-is-macosx/spotlight.html

References

[1] 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*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 4–4. [Online]. Available: http://portal.acm.org/citation.cfm?id=1863103.1863107

[2] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[3] L. L. Gremillion, "Designing a Bloom filter for differential file access," *Communications of the ACM*, vol. 25, no. 9, pp. 600–604, 1982.

[4] K. Shanmugasundaram, H. Brönnimann, and N. Memon, "Payload attribution via hierarchical Bloom filters," in *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2004, pp. 31–41.

[5] E. Lagerspetz, T. Lindholm, and S. Tarkoma, "Dessy: Towards flexible mobile desktop search," in *Proceedings of the Fourth ACM SIGACT-SIGOPS International Workshop on Foundations of Mobile Computing*.

[6] E. Lagerspetz, "Dessy: desktop search and synchronization," Master's thesis, University of Helsinki, Department of Computer Science, Helsinki, Finland, Nov. 2009. [Online]. Available: http://www.cs.helsinki.fi/u/lagerspe/publications/lagerspe-gradu.pdf

[7] E. Lagerspetz, T. Lindholm, and S. Tarkoma, "Dessy: search and synchronization on the move," in *MDM 2010*.

[8] T. Lindholm, J. Kangasharju, and S. Tarkoma, "Syxaw: Data synchronization middleware for the mobile web," *Mobile Networks and Applications*, vol. 14, no. 5, pp. 661–676, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11036-008-0146-1

[9] R. Wolski, S. Gurun, R. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *in Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2008), High-Performance Grid Computing Workshop*, 2008.

[10] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, pp. 51–56, 2010.

[11] E. Cutrell, D. Robbins, S. Dumais, and R. Sarin, "Fast, flexible filtering with phlat," in *SIGCHI '06*, pp. 261–270.

[12] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. James W. O'Toole, "Semantic file systems," in *SOSP '91*, pp. 16–25.