A Hash Based Archiving System

Anton Kos, Sašo Tomažič
University of Ljubljana, Faculty of Electrical Engineering, Ljubljana, Slovenia
E-mail: anton.kos@fe.uni-lj.si

Abstract - *The amount of data created annually is growing exponentially. Some of the newly created data is important enough to be kept for shorter or longer time, therefore the need for data archiving is growing proportionally to data growth. We designed an efficient and scalable archiving system that can easily be adapted to the needs of different archiving applications and scenarios. A Fast Hash-based File Existence Checking algorithm is the core of the system. It outperforms other algorithms for file uniqueness checking. The proposed archiving system is a distributed processing and storage system that works with metadata hashes and file hashes. We implemented the archiving system in a form of a backup-like application for files on personal computers in the laboratory. The application operational tests yielded very favourable results, for example, the need for archive storage almost halved just because of the elimination of files with exactly the same content. We expect the proposed archiving system will find many uses in the present and future data archiving efforts.*

1 Introduction

In the information society, reliance on data is continually growing and the amount of data being created worldwide is increasing exponentially.

According to the study [1], in 2011, more than 1.8 zettabytes (10^{21}) of data in more than 500 quadrillion (10^{15}) files were created and replicated to hard disks, DVDs, shared in the cloud, or stored in other data storage*. The amount of data created annually is reported to more than double in two years and is expected to grow to around 8 zettabytes in 2015. It is also expected that the number of files or containers that encapsulate data will grow even faster, by the factor of 8 in the next five years. The study states that the data growth continues to outpace the growth of storage capacity. The study predicts that in the next decade the number of servers (physical and virtual) will grow by the factor of 10, the amount of data managed by datacenters by the factor of 50, and the number of files in datacenters by the factor of 75.

*These numbers represent only the amount of data being stored. The amount of transient data is typically not stored (e.g., digital TV signals we watch but don't record, digital voice calls in the network backbone for the duration of a call) can be up to a few magnitudes bigger.

A certain portion of the newly created data is important enough to be kept for longer time periods. The storage that keeps such data is usually called *an archive* and the process that transfers data to the archive is called *archiving*. Let us briefly list two of the many definitions of the term archive that can be found on the Internet. In [2] it reads: "An archive is a collection of historical records, or the physical place they are located.". In [3] it reads: "A long-term storage area, often on magnetic tape, for backup copies of files or for files that are no longer in active use."

The need for archiving is in proportion to the volume of data created, which is increasing exponentially. Therefore a good archiving system should be prompt, efficient, capable, and above all future proof†.

The paper is organised as follows: in section 2 we present the motivation for our work, expose some general problems connected to archiving and propose the solutions for them. In section 3 the design of the proposed archiving section and descriptions of its main modules is given. Its implementation is presented in section 4, and some of the results of its operational analysis in section 5. We conclude with section 6.

2 Motivation, problems and proposed solutions

Our goal is to design an efficient and future proof archiving system that would satisfy as many archiving needs and scenarios as possible. We can get its main properties by answering the following questions.

Why do we need an archiving system? Data is archived for different reasons and for different purposes. For instance: some data is being kept for long periods of time, data from different sources must be kept in the same location, changes in data are being tracked, etc. *What kind of data do we archive?* We archive all data that we find important to keep for a certain period of time. *What do we expect from an archiving system?* Because of many different uses, an archiving system should be very flexible and should be offering many archiving options and scenarios. Since the volume of data is growing exponentially, the archiving system

†As archive is by the definition long-term data storage, the core part of the archiving system is expected to work relatively unchanged for the duration of the archive data validity.

should also be highly scalable. *What data (files) are eligible for archiving?* An efficient archiving system stores only *originals*. By that we mean only files with unique contents. For instance: two files with different names and exactly the same content should only be archived once. All versions of the same document with different content are all archived separately. *What are general archiving system requirements?* Above all a lot of expandable storage space for archived files. An excellent *archivist** that is fast, efficient and accurate. *What does a simple archiving process look like?* A client submits the file to the archiving system, the system checks file uniqueness, archives the file if necessary, and informs the client of archiving activities and results. All the necessary archiving information is kept by the archiving system.

From the above answers, it is evident that the single most critical operation in the archiving process is finding out if the submitted file is unique. This is especially important in archiving scenarios where a lot of unchanged files, or files having exactly the same content as files that have already been archived from other sources, is expected to be submitted to the archiving process, sometimes even on a periodical basis. An example of such a scenario is a periodical (daily, weekly, monthly, etc.) archiving of all the data of all computer systems in a certain environment, where a lot of unchanged files, that have already been archived, are repeatedly submitted to the archiving system.

Comparing each submitted file to all the files already in the archive is time consuming and would make an archiving system too slow. It is more elegant and sufficient to compare only the digests of a submitted file to digests of all already archived files. A file digest is a fixed-length bit pattern that uniquely identifies a file. Digests are usually much shorter than its files, what can cause a *collision*, an event where two or more different files have the same digest. The probability of a collision can be made low enough or negligible, if the algorithm for creating the file digest is chosen properly, and if the digest is long enough.

According to [4], cryptographic hashes as MD5, SHA-1, and SHA-2, are suitable candidates for file digests. They have good resistance to collisions and satisfactory probability distribution of hash values. The probability of collision is given in [4] as:

$$P_{coll} \doteq \frac{K(K-1)}{2 \cdot 2^n} \tag{1}$$

where n represents the length of the hash in bits and K the number of hashes (files) in the archiving system.

For an archiving system that holds a billion (10^9) of 160-bit hashes, the probability of collision is approximately 10^{-30}. If the number of hashes in the system increases to a quadrillion (10^{15}), then the probability of collision increases to approximately 10^{-18}. But if the length of a hash is then increased to 256, the collision probability drops to approximately 10^{-48}. It can be seen that with controlling the hash length, probability of collision can be kept low enough to be declared as negligible.

When a file is submitted to the archiving system, its hash is created first. The file hash is then compared to all the file hashes already stored in the system. The system archives the file itself only if the submitted hash does not yet exist in the system's hash table. At the time of archiving the file hash is added to the system's hash table.

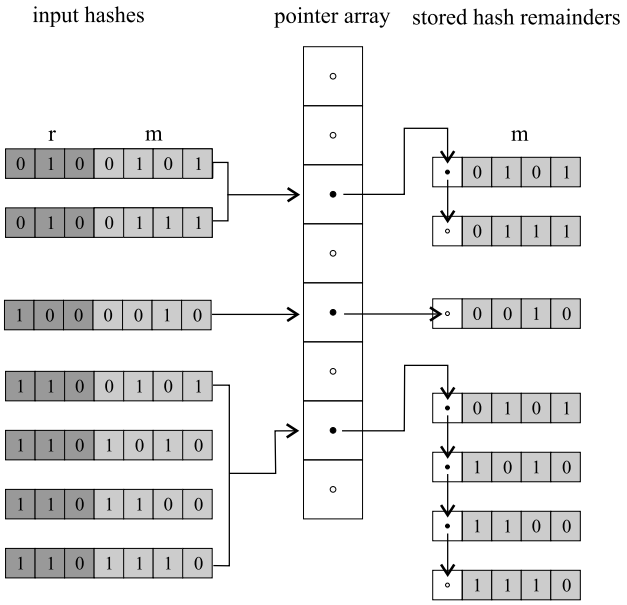

Figure 1. Hashes of n bits length are divided into r bits representing the region the hash belongs to, and m remaining bits specifying the exact hash in the region. Hash storage structure is based on a pointer array. The address of the pointer in the pointer array is defined by the first r bits of a hash. A null pointer (empty circle) in the pointer array denotes that no hash from that region exists in the system, other pointers (filled circle) point to the first stored remainder of the hash from that region. Each hash remainder in a non-empty region is accompanied by a pointer pointing to the next hash remainder in the same region. The last hash remainder in a region has a null pointer attached.

The hash comparison process would commonly use a Binary Index Search (BIS) algorithm, or alternatively a B+tree algorithm. BIS requires a sorted list (index) of hashes and does not need additional storage apart from that for the hashes themselves. It performs fast search in logarithmic time, but takes a lot of effort to insert a new hash into the sorted list. B+tree performs fast search and insertion in logarithmic time, but needs additional data storage for the tree structure. In [4] a new Fast Hash-based File Existence Checking (FHFEC) algorithm is proposed. It has been proven by theoretical analysis, simulation and implementation,

*An archivist is a person or in our case a process that makes sure the archiving is done in the right way.

that for hash (file) existence checking, FHFEC outperforms both of the above mentioned algorithms.

Hash existence checking is done by the following procedure (see figure 1 for help). When a file is submitted for archiving, its hash is calculated first. Then the pointer at the address defined by the first r bits of the hash is checked. If it contains a null value, the region is empty, meaning that the hash does not exist yet. It is stored into the system by writing m bits of the hash remainder to the available location. Null pointer of the region is updated to point to the location of the added remainder, and the added remainder is attached a null pointer denoting that this is the last remainder in this region. The file is archived. If the corresponding region is not empty, the linked list of hash remainders of the region is checked for the match. If found, the hash already exists, the file is not archived. If not found the remainder is added to the end of the linked list and corresponding pointers are updated accordingly. The file is archived. Further details and comparisons to BIS and B+tree can be found in [4].

Based on the study of above problems and solutions we have designed a simple lab-size archiving system.

3 Proposed archiving system

We designed our system to be a general purpose archiving system able to store archived files and the corresponding archiving history. Archived files are unique, meaning that no two files in the archive can have the same content. Archiving history includes all events and corresponding event data that relate to any archived file. Two most common events are a successful archiving of a file and a finding that the file with the same content has already been archived.

Such general purpose archiving system can be easily adapted to many specialised archiving tasks, scenarios, and applications. Among them are a classical file or document archiving, long term archiving, law enforced archiving of research or business documentation, computer system backups, and many others.

The proposed archiving system is a distributed computer system shown in figure 2. Its modules are interconnected through a data network, in our case IP network. The archive is generally a distributed, redundant, and expandable data storage with enough capacity to hold all the unique file instances expected to be submitted to the system during its lifetime. Directory and file servers are processing systems that process archiving submissions and hold a database of archiving history and logs. Metadata* and file FHFEC modules are storage/processing systems that hold archived metadata hash list and archived file hash list. They also perform the operation of hash uniqueness checking (and through it the file uniqueness), which is the most important archiving process in the archiving system. It relies on the proposed FHFEC algorithm that is briefly explained in the previous section and extensively in [4]. The most numerous and diverse are clients. They are processing systems that perform numerous and heterogeneous tasks required by the archiving scenarios or applications. Clients actually define the behaviour of the archiving system. For instance, the operation of a client for classical document archiving is very different from the operation of a client for system backup. The operation of other archiving system modules does not depend on a client operation, all the diversity is on the side of clients.

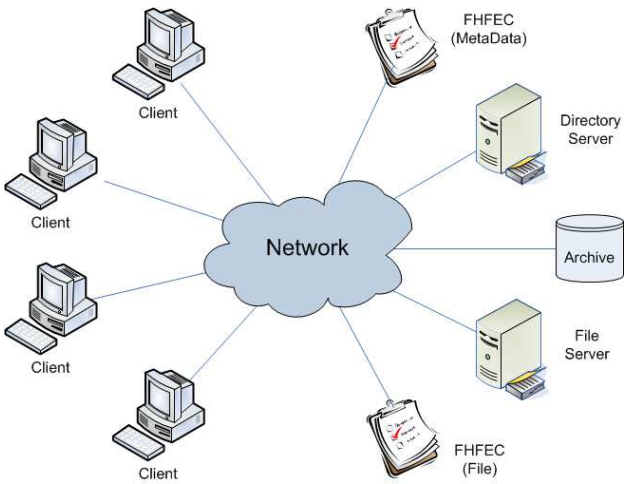

Figure 2. The proposed archiving system is a distributed computer system. It consists of several modules that communicate over the network. Archive is a data storage that holds archived files. FHFEC modules hold metadata and file hash storage structures based on pointer arrays and lists (see figure 1). Directory server performs the process of metadata hash checking and stores the metadata hash submission history. File server performs the process of file hash checking and stores the file hash submission history. Clients decide about file submissions and control the entire archiving process from metadata hash submission to possible file archiving.

The described architecture, where each module is an independent processing system, is most suitable for large-size systems with heavy archiving activity and good networking infrastructure. For small-size systems a certain degree of module and resource pooling is possible. For instance: directory server and metadata FHFEC modules can be combined, file server and file FHFEC modules can be combined, archive storage can be a part of any other module, or at the highest degree of pooling, all modules, except clients, can be one processing system.

The detailed process of a single file archiving is shown and briefly described in a flow diagram in figure 3. Let us emphasize here that the process of choosing which files are to be submitted to the archiving process

*Metadata is a collection of attributes of a file such as: filename, date and time of creation and/or last modification, size, location, etc. Metadata hash is a hash created from file's metadata.

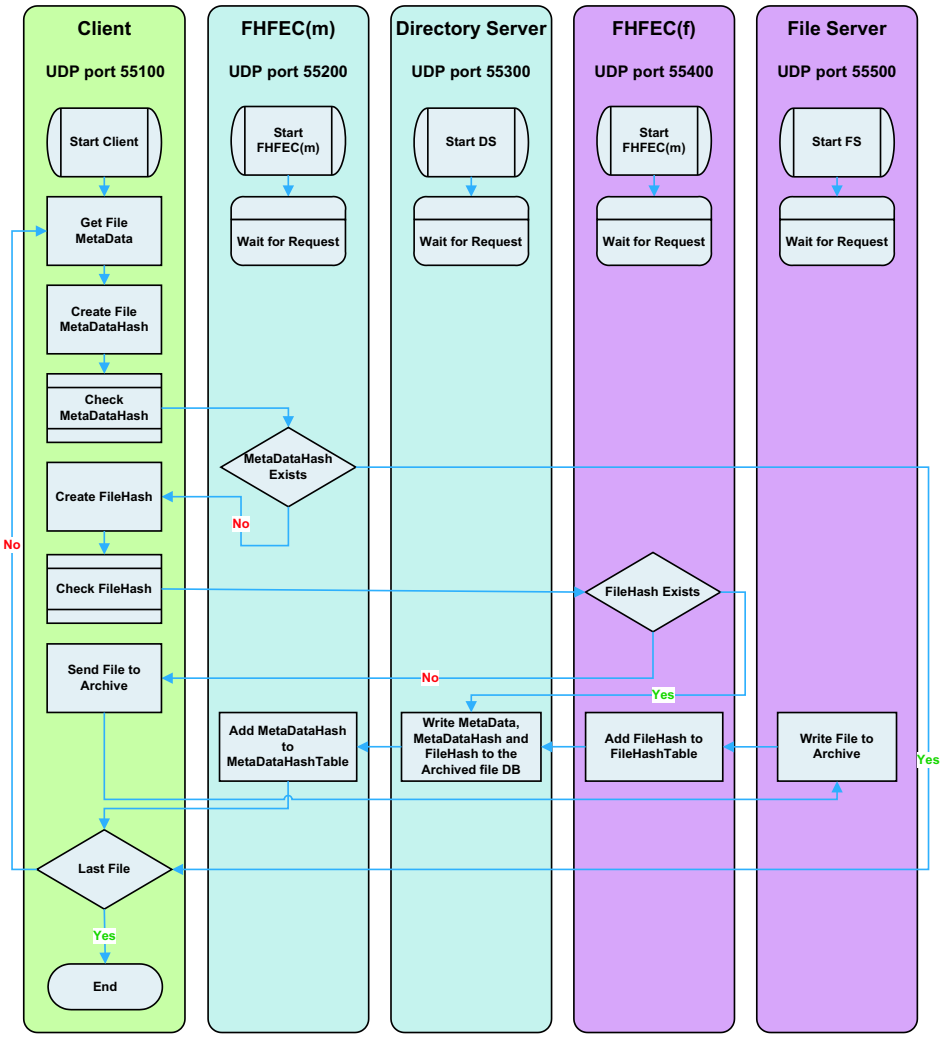

Figure 3. File archiving in the proposed system is a three-stage process. On the first stage the client decides, which file to submit for archiving, collects its metadata, calculates metadata hash and sends it to the metadata FHFEC module for existence checking. If the metadata hash exists, the client chooses the next file. If it does not exist, the client starts the second stage, calculates file hash and sends it to the file FHFEC module for existence checking. If the file hash does exist, the directory server updates archiving history and metadata FHFEC adds the metadata hash to its hash list. If it doesn't exist, the client start the third stage, the file is sent to the archive, directory and file server history is updated, metadata and file hashes are added to the corresponding hash lists in FHFEC modules.

depends on the archiving application using it and it is always done by the client.

The three-stage archiving process may not seem effective and in many parts redundant. This observation holds in the archiving scenarios where a great majority of submitted files are unique. In such cases the metadata hash creation and checking is superfluous. But in many scenarios, especially when only a small portion of the submitted files is unique, the three-stage process is very effective and reduces the processing load of core modules of the archiving system. Even the client benefits in such scenarios. Calculation of metadata hash is on average much faster than calculation of the file hash, and when the metadata hash already exists in the system, the file hash is not calculated at all, what saves processing time and speed up the client operation. Because of that we see the three-stage process more as an advantage as a disadvantage.

4 Archiving system implementation

In order to test the proposed archiving system, we developed an application for archiving files on personal computers. The application is intended to periodically back up all files, or any subset of files, on each of the PCs in our laboratory. We made all the modules of the system and tested them on the local area network inside the laboratory. We used IP protocol stack.

We dedicated two personal computers for hosting and running archiving system modules. We pooled the metadata and file FHFEC modules inside one dedicated PC, on the second dedicated PC we pooled the directory and file servers. Client is run separately on each of the test computers that have files to be archived. Despite pooling, each system module still works independently and has its own port number in the IP protocol stack (see figure 3).

5 System operation analysis

The first step of the system analysis was its validation. We conducted the attestation of the correctness of the message exchange protocols * , hash calculation exactness, calculation and recording of results, and archiving algorithm operation. The later we did partly with the help of many file duplicate finders that are freely available on the Internet. This test is probably the most important as it shows not only that the algorithm itself is able to find duplicate files, but also that the hash calculation is correct.

After successful system validation we defined archiving scenarios and expected archiving outcomes. Based on scenarios and outcomes we also defined the test archive parameters.

In the base scenario we start with an empty archive and consecutively archive personal computers in our laboratory. Since we only have Windows 7 and XP computers, it is to be expected that there will be many duplicate files during the archiving process. This is particularly true for the operating system files, and in smaller extent for user files. We followed many archiving parameters. The most interesting are: the number of new (original) metadata and file hashes, the number of old (duplicate) metadata and file hashes, the size of files being archived, and the size of duplicate files.

The number of old metadata hashes indicates all the files whose metadata has not changed. Basically it means that the file itself has not changed at all. The number of new metadata hashes indicates files whose metadata has changed. Only for these files the client calculates the file hash. The number of old file hashes indicates all the duplicate files, meaning that files with the exact same content have already been archived. The number of new file hashes indicates all the files that are being archived. On graphs below the horizontal axis determines the number of PCs archived.

Figures 4 to 6 show the archiving process for the base scenario. We see that for all the tests the number of old metadata hashes stays zero. This result was expected because we archive each personal computer for the first time on an empty archive. Because of that all the metadata, that among other parameters include also the host name, was new to the system, hence all metadata hashes were also new.

More interesting are the curves for old and new file hashes, where we can notice an interesting difference between Windows 7 (figure 4) and XP (figure 5) computers. Since XPs are older, they have accumulated a greater number of user files. Because of that the old (duplicate) file hashes, surpass the number of new

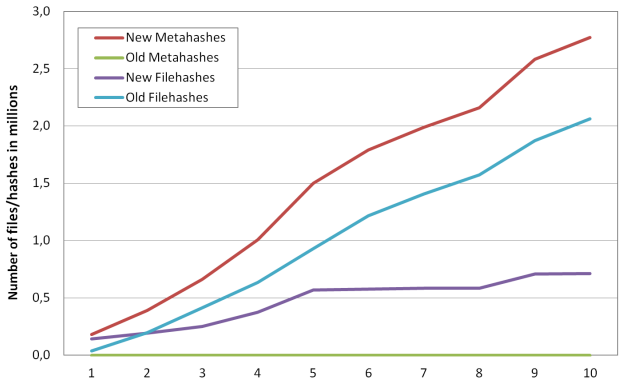

Figure 4. Archiving results for consecutive archiving of ten Windows 7 PCs. The graph shows the cumulative number of new and old metadata hashes together with new and old file hashes.

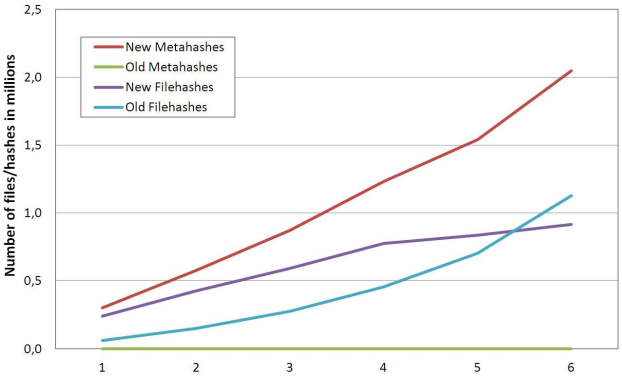

Figure 5. Archiving results for consecutive archiving of six XP PCs. The graph shows the cumulative number of new and old meta hashes together with new and old file hashes.

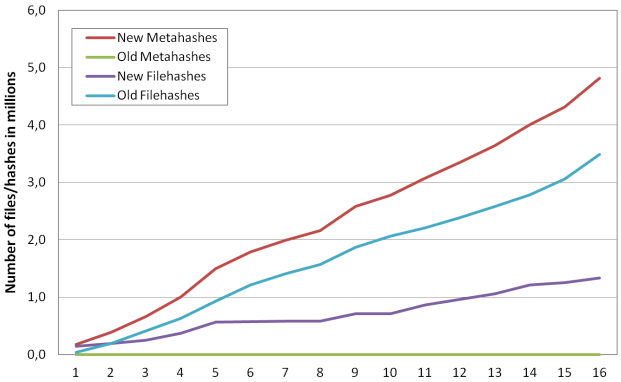

Figure 6. Archiving results for consecutive archiving of ten Windows 7, followed by six XP PCs. The graph shows the cumulative number of new and old meta hashes together with new and old file hashes.

(original) file hashes only at the end of the curve, while with Windows 7 PCs that happens very early. This is a very encouraging result indicating there are more duplicate files, than there are originals. Consequently, the required archive size is smaller.

We can expect that with higher number of computers archived, the number of old files would surpass the number of new files even by even greater extent.

*Messages are being exchanged between the system modules. Most of the modules act as a client and server, only the *client module* acts only as a client.

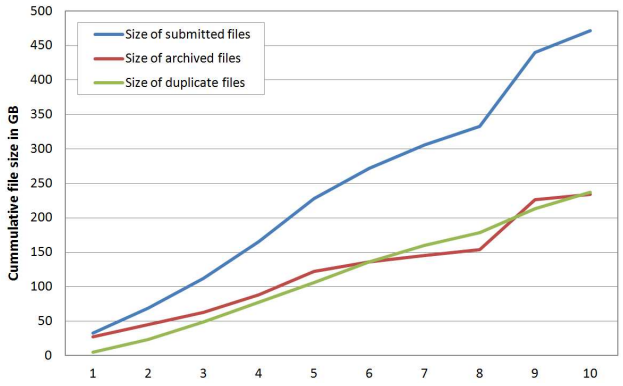

Figure 7. Archiving results for consecutive archiving of ten Windows 7 PCs. The graph shows the cumulative values for the size of all files submitted for archiving, the size of files that have been archived, and the size of duplicate files.

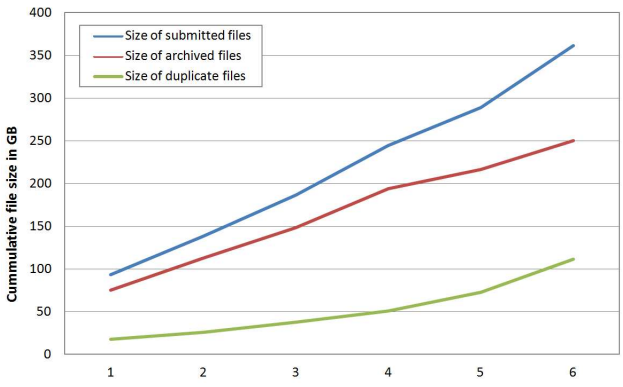

Figure 8. Archiving results for consecutive archiving of six XP PCs. The graph shows the cumulative values for the size of all files submitted for archiving, the size of files that have been archived, and the size of duplicate files.

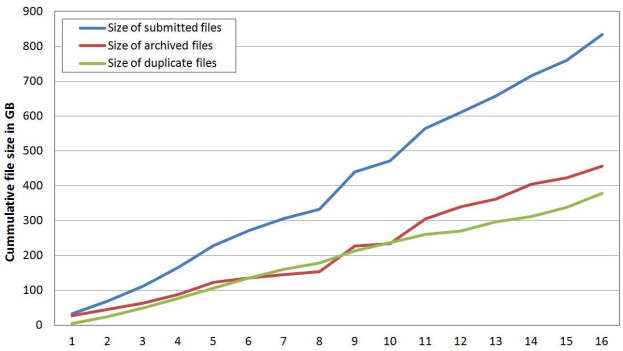

Figure 9. Archiving results for consecutive archiving of ten Windows 7, followed by six XP PCs. The graph shows the cumulative values for the size of all files submitted for archiving, the size of files that have been archived, and the size of duplicate files.

We gain correct archive size by examining the results in figures 7 to 9. They show the cumulative size of archived and duplicate files. Again the difference in results between Windows 7 (figure 7) and XP (figure 8) are caused by greater number of user specific files on XPs. Although the difference between sizes of old and new files is smaller that it would seem by the number of file hashes, the result is still encouraging. Only by preventing the archiving of duplicate files, almost half of the capacity of the archive can be saved.

In the progressive scenario we re-archive the same personal computers after a certain time period. We expect that not many files would change during that time period, meaning that the archiving process will find most of the metadata hashes unchanged. Consequently the file hashes would not be calculated, thus saving processing time and effort, but for the files with changed metadata and new files the process will of course run in its entirety. Results of re-archiving one of the PCs after two months show that, out of the total number of files on the PC, the archiving system found: 2.21% of new metadata hashes, out of those 1.36% were new files, and 0.85% were old files with changed metadata. Thus, for our archiving application the real benefit lies in the periodical re-archiving of a PC, which is done with far less effort than the first archive episode.

6 Conclusion

In this paper we propose a new general purpose distributed archiving system that can be easily adapted to many different archiving scenarios, tasks and applications. The core module of the system works using the FHFEC algorithm [4], proven to outperform other file existence checking algorithms. Based on the proposal, we have developed and implemented a test application that archives files from personal computers connected to an IP network. The test results show that the application saves almost half of the required archiving storage space. Even better results are achieved by re-archiving where only a small percent of files go through the entire archiving process.

There is a lot of possible future tasks and work to be done on the proposed archiving system. One of them is working with fixed size file chunks instead of working with entire files. It is expected that even more storage space can be saved in this way.

7 References

[1] John Gantz, David Reinsel, *Extracting Value from Chaos*, Research paper sponsored by IDC and EMC corporations, June 2011.

[2] http://en.wikipedia.org/wiki/Archive, referenced on 22.1.2013.

[3] http://www.thefreedictionary.com/archive, referenced on 22.1.2013.

[4] Sašo Tomažič, Vesna Pavlović, Jasna Milovanović, Jaka Sodnik, Anton Kos, Sara Stančin, Veljko Milutinović, *Fast file existence checking in archiving systems*, ACM transactions on storage, vol. 7, no. 1, June 2011

[5] Anton Kos, Sašo Tomažič, *Hash Based Archiving: A Study System*, YUINFO 2009, conference proceedings, Kopaonik, Srbija, 2009