One Approach to Efficient Management of Zillion Signatures

Rudan, Sasa; Kovacevic, Aleksandra; Babovic, Zoran; Jovic, Darko; Milutinovic, Veljko; and Milligan, Charles

Abstract—*In the introductory part, this paper defines the general environment for this research and defines the terms of interest. Then we define the research problem: How to manage efficiently billion file signatures from a specially introduced new file signature management layer; this problem is important because it enables the files signatures to be handled in a fast way. Existing solutions to the problem are briefly surveyed in the third part, and their drawbacks are underlined; each surveyed piece of research is analysed according to the same template. Then, the essence of the proposed solution is presented: Efficient storage of 1billion 20-byte digital signatures, their fast lookup, insert and delete, fast rebuild of the storage digital signature index; it also includes primitives that can be directly ported into hash functions and other appropriate mechanisms used for management of file signatures; this idea has several versions. The fifth part defines conditions and assumptions of the analysis to follow. Analytical analysis of all above is presented next. Elements of a simulation study, to compare performance, are presented in part seven. The conclusion is presented in the part eight.*

1. *Introduction*

NEW storage systems deal with billion of files. These files are usually stored on classical hard disk based storages. This environment is characterized with inefficiency of hierarchical file systems.

New concepts of raw file systems use digital signatures as Uniform Resource Identifier's (URI) for stored files. This implies a need for extremely fast digital signature management, i.e. searching, storage, and deleting of digital signatures in the domain of $2^{20*8}=2^{160\tilde{}}\,10^{48}$ values.

One of the basic feature of digital signatures is sharp miscorrelation between two digital signatures that come (are calculated) from two extremely correlated documents (even if documents differ only in one bit). It implies an approximately uniform division of digital signature values created from a finite number of documents.

Manuscript received January 30, 2006. This work was supported in part by the StorageTech.

Sasa Rudan, Aleksandra Kovacevic, Zoran Babovic, Darko Jovic, and Veljko Milutinovic are with Faculty of Electrical Engineering, University of Belgrade, Serbia.
Charles Milligan is with StorageTech.

2. *Problem Statement*

The Digital Signature Index Mechanism represents a service used by the archive platform. This service provides fast and reliable lookup/insert functionality. The archive system is designed to handle up to 1 billion unique data files. Each file is identified by a unique digital signature, calculated using at least two different algorithms.

The archive system has to be able to quickly determine if a data file is either a unique data file or a duplicate of an existing file. This is accomplished by generating the digest on the file, then querying the existing digital signatures looking for a match.

The Digital Signature Index Mechanism has to be durable to system failures, redundantly stored, and secured from unauthorized manipulation.

The primary challenges are:

1. Efficient storage of 1 Billion 20-Byte digital signatures.
2. Fast lookup, insert and delete of a digital signatures.
3. Fast rebuild of the storage digital signature index.

The digital signature storage mechanism should be used as a lookup system only. It does not act as a system of records for digital signatures. The index lookup returns either a duplicate status, or a unique status. If the status is unique, the signature is stored in the index.

The archive system executes lookups/inserts in two distinct ways: real time and batch mode. Batch mode provides log files with the results of each lookup/insert request.

The signature index should allow multiple readers, and a single writer.

Upon system restart, the signature index needs to recover to a state that is consistent with the reset of the archive system. The signature index should be able to be rebuilt quickly and efficiently.

3. Existing Solutions and Their Criticism

This chapter will present two solutions which are currently in use for management of the storage systems.

The first of the proposed solutions is a log-structured file system, which is presented at the Berkeley, University of California by Mendel Rosenblum and John K. Ousterhout [1]. The essence of this solution is based on the assumption that files are cached in main memory and that increasing of memory size will make a cache more and more effective. A log structured file system writes all modifications to disk sequentially in a log-like structure, thereby speeding up both file writing and crash recovery. The log is the only structure on disk; it contains indexing information so that files can be read back from the log efficiently. In order to maintain large free areas on disk for fast writing, the log is divided into segments and uses a segment cleaner to compress the live information from heavily fragmented segments.

Another solution of the storage management is based on the use of specialized storage devices known as "Appliances." This solution is developed at the University of Wisconsin-Madison by John Bent and his associates [2]. The appliances are designed solely to serve files to clients. These devices usually have the ability to adapt to the characteristics of the underlying hardware and operating system. The appliance developed by this team is called NeST, the open-source, user-level, software-only storage appliance.

4. Proposed Solution

Our proposed solution has several versions; all of them are developed in order to improve all of requests and drawbacks of other solutions. The main advantage of our model is its simplicity. It does not require additional memory and hardware.

4.1 Linear Area of Region to Bit Mapping

This represents a simple, but powerful solution for our problem. It is presented by a simple bit array structure. For each equivalent width region from the whole digital-signature-value domain there is one corresponding bit. In this way, we have some kind of or-hash function inside the specified regions. Figure 1 specifies the concept.

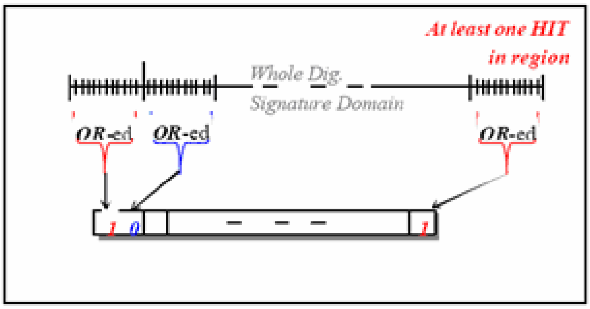

Figure 1: Bit Array Structure

4.1.1 Client-System Communication Protocol

Client-System communication is quiet simple. Client sends the request about specific digital signatures. Our system component checks in local structures and responds with an appropriate answer. Answer is one of the following:

NO – stands for absolutely no existence of the specified Digital Signature (DS) in the whole system.

YES – stands for an absolutely positive statement that the specified DS exists somewhere in the system.

Also in the subcomponents of the system (i.e., volatile storage memory subcomponent) we can get the following answers:

POSSIBLE_NO – This means that system subcomponents "believes" that DS does not exist in the system.

POSSIBLE_YES – This means that system subcomponents "believes" that DS exists in the system.

Now we can explain more precisely the client-system communication. When our DSM (DS Management) system component receives request from the client, it proceeds DS to the "volatile" system subcomponent which retrieves answer (NO or POSSIBLE_YES) from "region to bit" array. If the answer is NO, it can be sent back to client. If the answer is POSSIBLE_YES, we have to find absolute answers. Therefore, we access to the *Disk Regions* structure which resides on disk and is explained later in this chapter. There will be only one disk access since DSs are arranged on appropriate way.

4.1.2 Required Disk Organization

Our proposal is to use a dedicated hard disk which will only serve for managing digital signatures. This will result in Disk Regions structure that holds digital signatures organized by regions in which they falls. The main idea is to organize storing of DSs to provide direct access to disk sector that contains requested DS. It has to involve gaps that can be populated in later system growth. Since each region is covered with 0.125 digital signatures it is quite reasonable (because of uniform division) to allocate 0.25 DSs per region. We have control over the hard disk so we can choose that each sector (common size of 512 bytes per sector is assumed in this calculation) holds RegionsPerSector = 512 / 20 / 0.25 = 102 regions that are "mixed" inside the sector but ordered between sectors.

This organization also gives us space of 12 bytes for (global) specific fields and for a pointer (for the case of overloading). This will results in about 40 GB of disk consumption.

4.1.3 Process of Structure Creation

Initial structure creation is quite simple. We have to go through each digital signature stored in our system and to determine to which region it belongs to.

This determining process could be done easily if each region is mapped into a specific bit of digital signatures. In that case we just have to use these bits to access the appropriate bit in the

Bit Area Structure. When we find a specific bit we just set its content to 1.

4.1.4 Process of Adding Signatures

Adding of new digital signature to system is just iteration in the process of creation of the array structure. It implies modification (setting to 1) of a single bit, responsible for the region in which value of digital signature falls.

As one can see, this is not a time consuming process. However, we have to store this change to non-volatile memory before system is powered off. This storing can be done in the process of shutting down the system. Nevertheless, we have to be prepared for the system failures, also. To create a failure-resistant system, we have to store all the changes of our structure to the non-volatile memory (i.e., hard disk) before giving announcement to client about completion of storing operation. This would unconditionally imply at least one access to the hard disk and this is not allowed in our system (this presumption is one of the most important requirements in the software). All what we can do is to allocate a small amount of memory *NewlyAddedDS* for storing newly added digital signatures (Figure 2):

Sporadically, we withdraw collected digital signatures to disk. With this action we enter to the new consistent state.

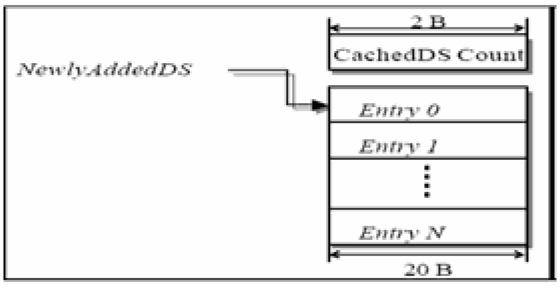

Figure 2: Storing of New Added Digital Signature

4.1.5 Process of Removing Signatures

Process of removing Digital Signatures is a bit tricky. The main problem lies in the nature of hash function; it is not a one-to-one function. It means that we can not decide whether we have to reset the state of bit that presents region to which the specified digital signature belongs, or we should leave it on 1 (meaning that the removed digital signature is not the only one that exists in the specified region). However, process of removing digital signatures is a rear process in archiving large systems, which is the case in our problem.

The simplest way of handling the removal of digital signatures would be ignoring this activity. What we would lose with this solution will be the possibility of one region that “arrogantly” announces Hits.

Another solution is to allocate another small amount of memory *RemovedDS* for storing digital signatures removed from the system. Sporadically, we remove collected digital signatures from disk. With this action we enter to the new consistent state. Also we have to recalculate values of conditioned bits in the array structure. This is not an exhaustive process since we have all digital signatures organized by regions, which they belong.

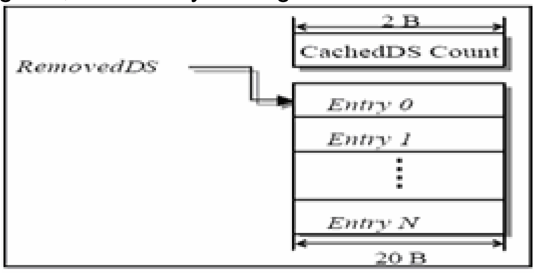

Figure 3: Removing of Digital Signature

4.1.6 Process of Recovering Structure

Recovering of the bit array structure is not an extensive process, since the bit array structure is a secondary data structure and it can be easily recovered from the primary data structure; in this case from *DiskRegions* structure (disk sectors that hold digital signatures from specific regions).

4.2 Client-System Communication Protocol

This solution covers another part of our problem; case for YES answers. It is presented with an array structure for each part of the digital signature. Figure 4 specifies this concept:

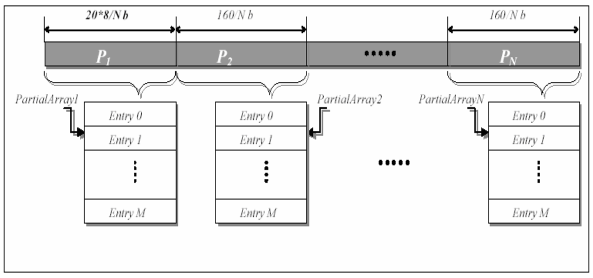

Figure 4: Case for YES Answers

4.2.1 Required Disk Organization

Required disk usage is N times larger than in the previous algorithm. Our decision is not to use additional disk structures which are specific for this model.

4.2.2 Process of Structure Creation

Initial structure creation is quite simple. We have to go through each digital signature stored in our system and to determine the value of each digital signature partition. Each value is used as entry point to an adequate Partial Array. Each entry is a bit that should be set. This determining process requires simple shifting, concatenating, and masking operations.

4.2.3 Process of Adding Signatures

Adding of new digital signature to system is just iteration in the process of creation of array structure. It implies modification (setting to 1) of N bits responsible for value of each digital signature partition. As one can see, it is not time consuming process. However, we have to store this change to non-volatile memory before system is powered off. This storing can be done in the process of shutting down system. Nevertheless, we have also to be prepared for system failures. As in the previous algorithm we have no possibility for creation of a failure-resistant system. All what we can do is to allocate small amount of memory *NewlyAddedDS* for storing newly added digital signatures. Sporadically, we withdraw collected digital signatures to disk. With this action we enter to the new consistent state.

4.2.4 Process of Removing Signatures

Removing Digital Signatures is a bit tricky. The main problem lies in the nature of hash function; it is not a one-to-one function. It means that we can not decide whether we have to reset the state of bit that presents the region to which the specified digital signature belongs, or we should leave it on 1 (meaning that removed digital signature is not the only one that exists in the specified region). However, process of removing digital signatures is a rear process.

The simplest way of handling the removing of digital signatures would be ignoring this activity. What we would lose with this solution will be the possibility of one region that "arrogantly" announces Hits. Another solution is to allocate another small amount of memory *RemovedDS* for storing digital signatures removed from the system. This is a pretty exhaustive process since we can not arrange digital signatures by partitions in which they belongs (we will need N times more disk space than for the previous algorithm). Sporadically, we remove collected digital signatures from disk. With this action we enter to the new consistent state.

4.2.5 Process of Recovering Structure

Recovering of the bit array structure is extensive process, since the bit array structure is a primary data structure and it can be recovered only sequentially from the whole collection of digital signatures.

5. Conditions and Assumptions of the Research

The archive system can handle several streams of data simultaneously. The number of streams, speed of the streams and size of the files will determine the average number of lookup/inserts per second. Formula for the number of lookup/inserts per second is:

(AvgSize / Throughput per second)*NumStreams = Ops per Second

Assumptions:

- Average File Size = 10 MB,
- Throughput = 300 MB/sec,
- Number of Streams = 8,
- Operations per second (for entire system) = 2400,
- For a fully configured system, there are 8 computing nodes,
- Each node handles 300 lookups/inserts per second.

The goal of 300 lookups/inserts per second per node has been set.

6. Analytical Analysis

Lets have a digital signature DS1 that falls in a region R1. Additionally, let's assume that request for digital signature DS2 arises, and DS2 also falls in the region R1. In this case we will respond to the client with POSSIBLE_YES answer. However, adequate answer should be NO. This phenomenon is called groundless Hit.

Main advance of Linear Area of Region to Bit Mapping concept is in its simplicity. Also it gives 100% precisely answers on each request that SHOULD result in NO. On the other hand, this model suffers from groundless Hits.

Digital Signatures Partially Caching concept still suffers from groundless Hits, but in less measure than Linear Area of Region to Bit Mapping; however, the amount of disk utilities is N times larger then in the first algorithm. It could be improved to almost completely avoid groundless Hits, but, as one can see, there is no algorithm and data organization that will give just valid Hits answers. The reason is that the amount of RAM storage needed for storing whole data (20 * 1 Billion = 20 Billion = 20 GB) and because of uniform division of digital signature values.

In the case of uniform division impact, our software allows maximum each third check that results in disk access. If we include human and real-system-utilization impact we will get slightly different results. For example: a user "A" is storing collection of Mozart's compositions that user "B" have just stored on the system. In this specific case, request for each document's digital signature will result in justified Hit. The result will be 100% disk access!

7. Performance Analysis

In our specific case we have:

- DS_COUNT - a 1 billion of digital signatures,
- MEM_SIZE - a 1 GB of volatile memory (RAM),
- DISK_SIZE - almost unlimited disk space (we will limit it to 256 GB).

Let us first consider Linear Area of Region to Bit Mapping concept. With 1 GB RAM we can create 8Gb large Bit Array Structure (BAS). Thereby, we can divide DS value domain on 8G regions and finally we can calculate BAScoverage, the percent of coverage (coverage with existing digital signatures) for each region:

BAScoverage = DS_COUNT / BAS_SIZE= DS_COUNT /8*MEM_SIZE = 1/8 ˜ 12.5%

This result is telling us that in case of the first model, each eight check (client request to our system component) will result in groundless Hits (and need for disk access).

In the case of Digital Signatures Partially Caching concept we have to find on how many parts (N) digital signatures should be split to fit into the specified amount of RAM memory (1 GB). We can find N from the following equitation:

MEM_SIZE=MEM_USAGE=N*2(20*8/N)/8

At first we have to see if it is satisfied with N=1. In this case we have MEM_SIZE = 5KB => equitation is satisfied. This equitation is a transcendental equitation and it can only be calculated by probing (we have to found the smallest N that satisfies the equitation). One can see that N = 6. Since 20*8/6 is not an integral number, we will get the following partitions:

26 + 26 + 26 + 26 + 28 + 28 = 160b

In this case:

MEM_USAGE = (4*2(26) 2*2(28))/8 B = 96 MB

8. Conclusion

In conclusion, the proposed approach proves to more efficient compared the existing solutions, in conditions of interest for this research. The proposed solution is of interest to those who a faced with the management of extremely large volumes of files.

References

[1] Rosenblum, M., Ousterhout, J., “The Design and Implementation of a Log-Structured File System,” *ACM Transactions on Computer Systems*, 1992.

[2] Bent, J., Venkataramani, V., LeRoy, N., Roy, A., Stanley, J., Arpaci-Dusseau, A., Arpaci-Dusseau, R., Livny, M., “Flexibility, Manageability, and Performance in a Grid Storage Appliance,” *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing* (HPDC-11), July 24-26, 2002.