DEXA-GLOBE 2006 Keynote

Disk Backup Through Algebraic Signatures in Scalable Distributed Data Structures

Witold LITWIN[1], Riad MOKADEM[1], Thomas SCHWARZ, S.J.[2]

Abstract :

A Scalable Distributed Data Structure (SDDS) allows to store a large scalable file over a distributed RAM. The file scales up transparently for the application over the nodes of a multicomputer, e.g., a local network of PCs. The prototype system termed SDDS 2000, designed by CERIA experiments with this technology for Wintel multicomputers. The application may manipulate data much faster than on local disks.

We present the functions we have added to SDDS-2000 to backup the RAM on each storage node onto the local disk. Our goal is to write only the data that has changed since the last backup. We experiment for this purpose with the algebraic signatures. We present our architecture and design choices. Performance measures validate our implementation. It is now available for the non-commercial use as part of new version of our prototype termed SDDS-2002.

1. *Introduction*

A Scalable Distributed Data Structure (SDDS) allows to store a large scalable file over a distributed RAM. The file scales up transparently for the application over the storage nodes of a multicomputer, e.g., a local network of PCs and WSs. The storage nodes are called the (SDDS) *server* nodes. The application manipulates through the SDDS interface at its node, called the *(SDDS) client* node. The client node performs the (scalable) address calculus and handles the application requests accordingly. Various SDDS schemes have been investigated, [C]. Probably the most studied were the linear hash partitioning variants of the LH* scheme, and the range partitioning using the RP* schemes.

The prototype system termed SDDS-2000, designed by CERIA put the SDDS technology into practice for Wintel multicomputers. The application manipulates data much faster than on local disks. Experiments with the SDDS files over 1.8 GHz Dell nodes linked by the 1 Gbs Fast Ethernet at CERIA show the successful key search time for the 100-byte record under 30 µs. That is thus about 300 times faster than the typical disk search time of 10 ms.

In SDDS-2000, each server node keeps the SDDS records in the RAM storage area termed *bucket*. It appeared useful to have buckets backed up to the disk. For the efficiency of this process, it was clearly best to copy to the disk only the data that has changed in the bucket since its last backup. As usual, it appeared potentially best to divide then the bucket into smaller units we called *pages* and write

[1] University Paris 9 Dauphine, 75016 Paris, France. mailto:Witold.Litwin@dauphine.fr
[2] Santa Clara University, Santa Clara, CA. mailto:tjschwarz@scu.edu

only the modified pages. The usual way to detect the page modification is to create the "dirty" bit table. The bit corresponding to a page is set up when a modification occurs. One backs up then only the "dirty" pages, cleaning the dirty bits back to the clean status afterwards.

To add the backup service in this way to SDDS-2000 turned out to be impossible in practice. The service was to be added to the existing complex code, built over years by different designers. It appeared a daunting task to properly identify all the pieces of the code that write at some point to the bucket. These should be updated to write the dirty bits as well. An approach not modifying the existing code, but only adding a new one was necessary.

We have chosen to provide each page in RAM and on the disk with a signature. We recall that a signature calculus scheme guarantees that a rapidly computed and a few-byte long *signature* computed for a modified data unit, e.g., our page, differ from the original signature of the unit for sure or at least almost surely. We may compute then the RAM page signature at the backup only and compare it to that of its disk image. We then write the page only if the signatures differ. We can perform the whole backup service by an additional software module. Without any change to those already working[V1].

Many signature schemes are known. The SHA-1 standard is perhaps the most used [11]. For our purpose, the *algebraic* signatures appeared nevertheless more practical [LS2], [LS3], [X&al3]. To implement these in practice required various design choices, e.g., over the page size and signature length. In what follows, we present our implementation and report on our final choices. We justify them through the experimental performance analysis. We consider the reader familiar with the theory of the algebraic signatures [LS2].

Section 2 overview our scheme. Section 3 discusses the experimental performance analysis validating our design. Section 4 presents the conclusions and directions for further work.

2. The SDDS-2002 Backup Scheme

2.1. System Architecture

In SDDS-2002, we offer to the application two new commands. These are the **store** command to backup or evict the file and the **load** command to restore it. The **store** writes to the disk only the data selected as we will describe. The **load** reads to the RAM from the disk the whole saved bucket. See [M2] for the command syntax.

The SDDS client sends the **store** or **load** command for a given file *F* using a UDP multicast request to all the servers, Figure 1. Those that carry a bucket of *F* process the commands and acknowledges the execution to the client. The acknowledgements include the RP* range of each bucket (we recall that an RP* scheme range partitions the file). The client unions all the ranges to find out whether all the servers of *F* that should reply did it (we recall that an SDDS client may not be aware of all the existing servers). This is the case only if the union reaches the whole key space of *F*. In this case, the client informs the application of the successful termination. Otherwise, messages are resent to the missing servers after a timeout and if some still do not reply, a server recovery action may start.

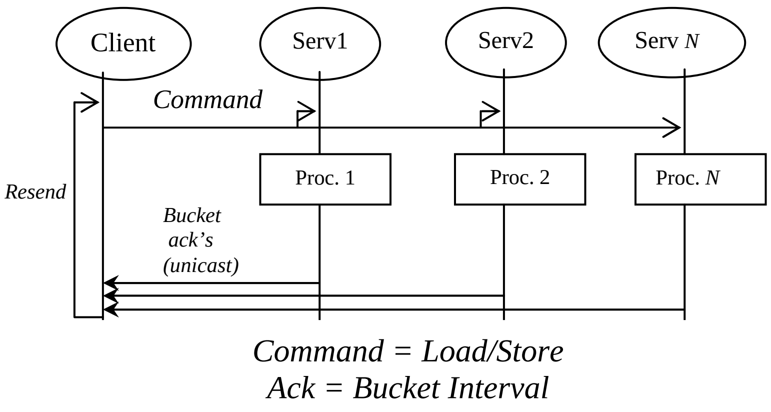

Figure 1 Gross Architecture for Bucket Disk Backup in SDDS-2002

Figure 2 shows the internal bucket structure for the RP* file under the SDDS-2002. The bucket is organized as a kind of RAM B+-tree. The index area is usually a few KByte long, a fraction of the data area. The index is modified whenever a tree leaf in the data area splits. Hence, the index gets modified more often than any part of its size in the data area[DL1].

To implement our backup schema, we have divided the data and index area into pages. The index is entirely one page. Each page has an algebraic signature. We compute all the signatures when the whole bucket is saved for the 1st time. The collection termed the (*bucket signature*) *map* is saved itself and kept in RAM. Any subsequent **store** leads to the computation of the actual signatures of all the RAM pages. We compare each to its copy in the map. We write the pages whose signatures differ to the disk.

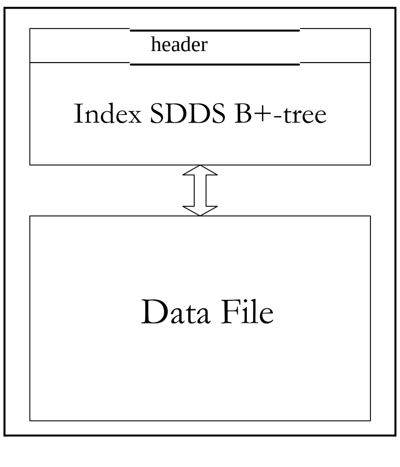

Figure 2 Internal Organization of Bucket in SDDS

The size of a disk write is a page at least. The granularity of data pages should thus be chosen carefully. Smaller pages possibly decrease the disk write size so the time to backup if a few updates only occur. In turn they potentially increase the number of writes when many updates occur. Our present data page size is 16 KB and index page size is 256 B. Reasons for that will be discussed below.

We have organized the disk storage at each server as follows. For each bucket to save, there are two files. One contains the backed up data pages. The other has the index pages. At each server, there is also an additional file that stores various parameters of each file saved. The **load** command needs these parameters to restore the buckets. These are the bucket file name, page size, number of pages….

The **store** command may keep the saved bucket in the RAM. It remains then available for access. Alternatively, the backup may free the RAM space, i.e., may *evict* the bucket from the RAM. The storage becomes available for another bucket. To manipulate records in the saved bucket, one must then execute the **load** command.

2.2 *Algebraic Signature Calculus*

The algebraic signature is a specific power series in a Galois Field (GF), [LS3]. We will now show our calculus of such signatures. We recall that a GF (N) is an algebraic structure with N elements, including elements 0 and 1 with usual properties with respect to the addition, subtraction, multiplication and division in the GF. GF (2^f) with $f = 8, 16…$ are especially useful. In this case the addition $a + b$ is usually computed as XOR of the bytes or words (symbols) representing the elements. The multiplication $a*b$, or ab as usual in short, is often implemented as the calculus:

$$ab = \text{antilog}\ ((\log_\alpha a + \log_\alpha b) \bmod N)$$

where α is a primitive element and $a,b \neq 0$ and $a,b \neq 1$. We recall that in a GF, every $a \neq 0$ is α^i form some $i = 0,1…f\text{-}1$. The log and antilog values are then tabulated in tables. The size of each table can be of $N - 1$ symbols. To avoid the mod N calculus, one can also double the log table to the size of 2N in practice. These calculus methods for the addition and multiplication were or final choice for our implementation. We have also chosen to use GF(2^{16}).

We use the algebraic signatures for our purpose as follows. Let P be a page. We consider P as a vector (1-d array) of symbols p_1 , p_2, …p_n of the GF used, GF(2^{16}) in our case. Technically, each p is one byte or a two-byte word (our case) of P. Let p' denote the element of GF used such that $p = \log_\alpha p'$. Let $\boldsymbol{\alpha}=(\alpha_1, \alpha_2,…\alpha_n)$ be the vector of different non-zero elements of the GF. We consider here specifically that α_1 is primitive and for $i = 2..n$, we have :

$$\alpha_i = \alpha_1^{\,i}.$$

Let it be for each $\alpha \in \boldsymbol{\alpha}$:

$$\text{Sign}_\alpha(\text{P}) = \sum p'_i\, \alpha^i \ ; i = 1,2..n$$

Then, the (n-symbol) *signature* of P is the vector denoted $\text{Sign}_{\boldsymbol{\alpha}}$:

$\text{Sign}_{\alpha}(P) = (\text{Sign}_{\alpha 1}(P), \text{Sign}_{\alpha 2}(P) \ldots \text{Sign}_{\alpha_n}(P))$.

We calculate $\text{Sign}_{\alpha}(P)$ using the log/antilog multiplication calculus. The use of p' in the formulae is purely formal. In practice, we consider P symbols p directly as logarithms. In other words, we do not calculate p', but, for each α_j , $j = 1..n$, we directly add p_i to $\log_{\alpha} \alpha^{i}$. This speeds up the calculus, with respect to the direct application of the signature definition in [LS3].

Likewise, a natural approach to $\text{Sign}_{\alpha}(P)$ calculus is to compute $\text{Sign}_{\alpha 1}(P)$, then $\text{Sign}_{\alpha 2}(P)$ etc. In our case, it appeared notably faster to calculate the contribution of p_1 to $\alpha_1 .. \alpha_n$, followed by this of p_2 to $\alpha_1^2 .. \alpha_n^2$ etc. This approach was our final choice. The reason is likely the influence of L1 and L2 caches.

The crucial propriety of the algebraic signature for our application is that the probability that two objects differ by only few symbols have the same signature, i.e. they *collide*, can be made negligible or even zero, [LS3]. Larger n is, smaller is the collision probability for the same P_1, P_2. For a given n, as long as P_1, P_2 do not differ by more than n symbols, then collision probability is zero, provided the page size under $2^f - 1$ symbols for GF (2^f) used. This property is at present unique to algebraic signatures.

For our purpose, the use of 2-symbol (4-byte) signatures sufficed. All things considered with respect to the page granularity, we have also set up our system for data area pages of 16 Kbytes. For the index, pages of 256 Bytes sufficed. As long as the RAM page P_1 and its previous disk image P_2 differ thus by at most 2 symbols, the collision probability is zero. If the application may make them differ arbitrarily, this probability is 2^{-32}. In general, that probability is 2^{-nf}.

3. Experimental Performance Analysis

3.1. Overall description

The signature schema in our case makes sense only if it saves time with respect to the straight disk write of the entire bucket. The algebraic signatures themselves are new & the scheme makes sense only if it is more efficient in our case than a known one. Regardless of the signature scheme used, in our case our total time results from that (i) to calculate and compare all the signatures and from that (ii) to write all the modified pages. First, this time is interesting by itself. Especially, as the function of the number of servers for the file. Logically it should decrease when there are more server, the interesting question being how.

Next, our scheme makes sense if we typically need less time for the file write than for the straight bucket(s) write. Our fastest case is when we only calculate the signatures, i.e., it appears that no page was changed. Our worst case is when we perform this calculus and write everything. The intermediate case can be, e.g., assuming that a small part of the bucket has changed, like 5 % of the file. The analytical calculus of such times for various bucket sizes, given CPU and disk speeds, the number of servers for the file etc. appears unfeasible in practice. We have therefore performed the experimental analysis we present now.

Our servers were four 1.8 GHz P4 PCs under Windows 2000 Server. The client was on a P3 800MHz machine. There was also a P3 500 MHz PC for the name sever used by SDDS-2002. A 1Gbs Ethernet linked all the machines.

To evaluate the comparative interest for us of the algebraic signatures with respect to another signature scheme, we have also experimented with the probably best known one which is the SHA-1 scheme, [S5]. We recall that this is the cryptographically secure signature schema implemented, in particular in the MS-Studio.net architecture. The SHA-1 calculus produces a 20-byte long signature. That one is called *message digest*, Figure 3. The reasons for this terminology is that the message digest can be input to the digital signature algorithm (DSA), which generates or verifies the signature for the message. Any change to a message in transit will, with very high probability, result in a different message digest. We have implemented the SHA-1 calculus for our pages and comparatively analysed the message digest as an alternative signature for our backup scheme.

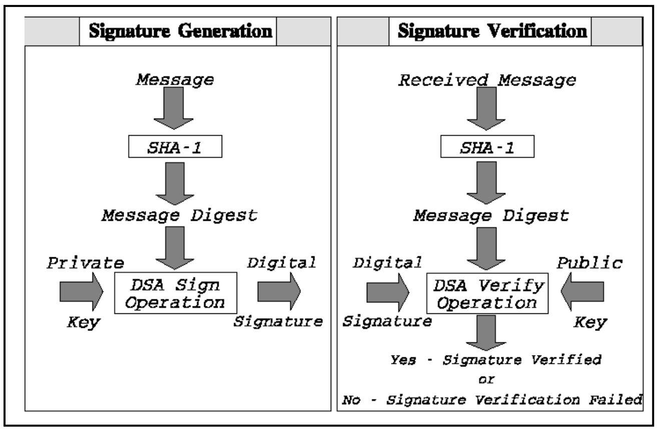

Figure 3 The SHA-1 scheme.

Data Storage

Figure 4 shows the time to store a file with our scheme. The file has up to 25000 records and is generated either entirely at one server, or, using smaller bucket sizes, splits on two or three servers. The bucket sizes and the times for one server are from Table 1. Logically, the figure confirms that using more servers speeds up the process. The speed-up reaches 30 %.

Table 2 sums up experiments for the scheme for various file sizes on a single server. First column shows bucket sizes used. The 3rd column shows the cumulated times to compute the algebraic signatures for all the pages of the bucket. As one could expect, the values appear linear with the bucket size. They are also about constant per Mbyte, around 25 ms. Finally, and fortunately, they are only a small, about 10 %, fraction of the time to store the file, shown in next column. Thus our calculus is globally efficient for our purpose.

Notice that these results are for the 16 KB pages. Better results were obtained for larger pages of 64 KB, leading to the calculus time of 20 Ms per 1 MB of RAM. It is due to less calculus of signatures and probably to the better use of cache.

However, such pages appear globally less efficient for our application at present than our choice. This, because of 4-time higher minimal write time.

The next column shows the time to store when the signatures did not detect any change to the file (0 % change to the file). Nothing is written to the disk. These times are nevertheless understandably slightly higher than for the signature calculus alone. Nonetheless, the total time to perform the backup according to our scheme, saves about 90 % of the write time.

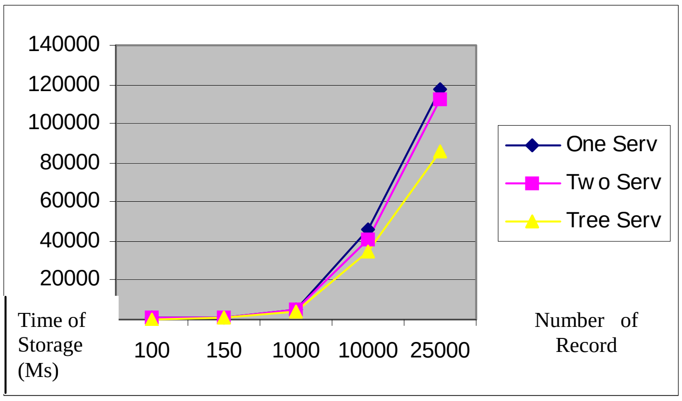

Figure 4 Time to store a file

Likewise, the next column shows the results for the 5 % change to the file, through inserts. The gains are smaller, but still about 90 % in practice. The hidden reason for such a good performance is that SDDS-2002 locates the new records at the end of the bucket. Notice, however that if inserts lead to a bucket split, then the entire splitting bucket is to write, as well as the new one at the server appended to the file.

Bucket size (MB)	Number of record	Signature calculus (ms)	Signature Calculus per/MB (ms)	Total store time (ms)	Store time for 0 % change (ms)	Gain (%)	Store time for 5 % change (ms)	Gain (%)
1.88	100	46	24.46	562	50	91.1	65	88.43
2.7	150	78	28.8	781	82	89.51	95	87.83
17.6	1000	438	24.88	5078	438	91.38	453	91.07
158	10000	4068	25.74	46406	4071	91.23	4085	91.19
393	25000	11003	27.9	117859	11003	91.33	11018	90.65

Table 1 File storage performance analysis

3.2. SHA-1 Signatures

We repeated our experiments using the SHA-1 signature calculus. Results are in Table 2 and in Figure 5.

Bucket size (Mb)	Number of record	Algebraic signature calculus (ms)	SHA-1 calculus (ms)	Storage time using SHA-1 (ms)	Storage time using alg. sign. (ms)	SHA-1 Store time for 5 % change (ms)	Alg. sign Store time for 5 % change (ms)	Gain (%)
1.88	100	46	70	602	562	85	65	30
2.7	150	78	103	799	781	119	95	25
17.6	1000	438	680	5278	5078	697	453	53
158	10000	4068	6088	47906	46406	6102	4085	49
393	25000	11003	15403	119342	117859	15418	11018	40

Table 2 File storage using SHA-1 signature performance analysis

The results confirm that our scheme is substantially more efficient with the algebraic signatures. The signature calculus is itself substantially faster, the gain being about 30 %. Also, the gain for 5 % change is between 30 % for a small file, and 40 ÷ 50 % for larger ones. Besides, we recall that our algebraic signature is 4-byte long, while SHA-1 uses 20 bytes.

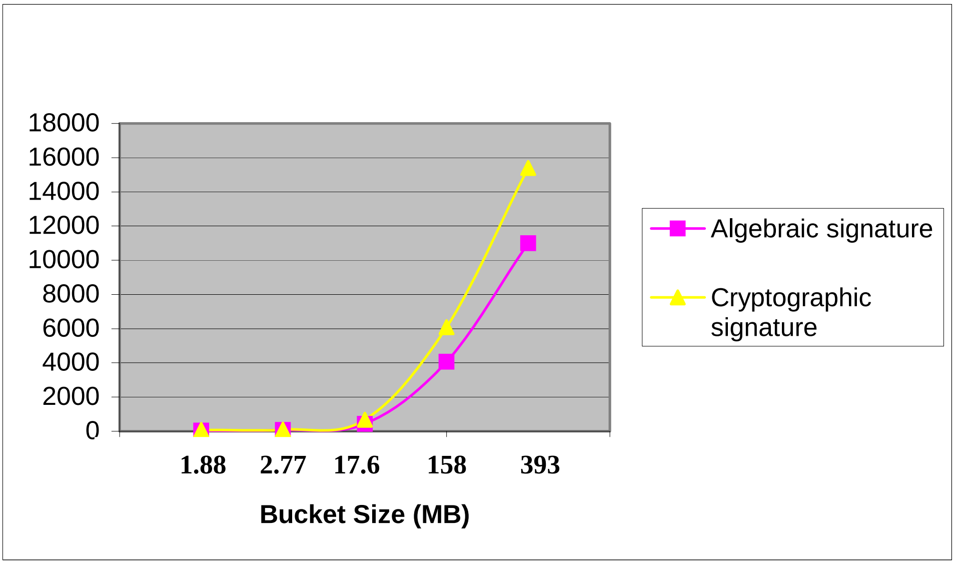

Figure 5 Algebraic / SHA-1 Signature Calculus Time (ms)

4. Conclusion

We have added the SDDS file disk backup storage capability to our prototype SDDS-2000 system. Our implementation is now integrated in the SDDS-2002 prototype. The use of signatures allowed us to add this function without the modification of the existing code. That one would be infeasible in practice. We only needed to add our new code. It turned out to be a three man-month task.

The signature calculus that our approach required could be made fast enough to remain under 10 % of the disk write time. This is a negligible performance penalty in practice. Also the storage for the page signature is a negligible fraction of the page size. Besides, the traditional approach of the "dirty" bit table also carries an overhead. Even more importantly, it introduces the run-time overhead for the update operations, unlike the ours. All together, our approach should be of interest also to other applications with similar needs.

Our future work addresses the automation of the bucket eviction to the disk and of its load, [LSS2]. We also work on other uses of the algebraic signatures for the SDDS management. In [LS3], we report on the record update management, also now integrated in SDDS-2002, [H3]. We also work on the use of signatures for the distributed non-key record search, including the partial (string) search. The algebraic properties of the signatures play there to some extent similarly to properties of hash schemes in [KR87].

Acknowledgements

We thank Jim Gray and Peter Scheuermann for fruitful discussions. This work was partly supported by the research grants from Microsoft Research, and from the European Commission project ICONS project no. IST-2001-32429, as well by the SCU IBM grant 41102-COEN-RSCH-IG-IG09.

References

[C] http://ceria.dauphine.fr/

[DL1] Diène AW, Litwin W Performance measurements of RP*: A Scalable Distributed Data Structure for Range Partitioning. 2000 Intl Conf on Information Society in the 21st Century: Emerging Tech and New Challenger. Aizu City, Japan,2000.

[H3] Hamadi, B. Suppressions et Mises à Jour dans le Système SDDS-2000. CERIA Res. Report, 2003-04-4.

[KR87] Karp,R.M.,Rabin,M.O.1987.Efficientrandomizedpattern-matchingalgorithms.IBMJ.Res.Dev. 31(2):249–260.

[LS2] Litwin, W, Schwartz, Th. Signatures for Scalable Distributed Data Structures. CERIA Technical Report, 2002-30-8.

[LSS02] Litwin, W., Scheuermann, P., Schwarz Th. Evicting SDDS-2000 Buckets in RAM to the Disk. CERIA Res. Rep. 2002-07-24, U. Paris 9, 2002.

[LS3] Litwin, W, Schwartz, Th. Algebraic Signatures for Scalable Distributed Data Structures. WDAS 2003, (June 2003), Thessaloniki.

[M2] Mokadem, R. Storage of data in Scalable and Distributed Data structures (SDDS). CERIA Res. Report, 2002-19-2 [S6] [Sinha A.K] Network Programming in Windows NT. Addison- Wersley Publishing Company 1996.

[M2a] Mokadem R Stockage de données en utilisant les signatures algébriques dans les SDDS. Mémoire DEA. Université Paris Dauphine Septembre 2002.

[V&al1] Vingralek & al Scalable Storage on Network of workstation with balanced load. July 2001. http ://www.bell-labs.com/user/rvingral/publication.html

[S5] Secure Hash Standard Processing Standards publication FIPS PUB 180-1, (April, 1995).

[X&al3] Xin, Q., Miller, E., Schwarz, Th., Long, D., Brandt, S., Litwin, W. Reliability Mechanisms for Very Large Storage Systems. 20th IEEE-NASA Symp. Mass Storage Systs. & Techn. MSST03, "Global Access to Distributed Storage", San Diego USA, 2003.