ELSEVIER

Information and Software Technology 44 (2002) 331–341

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Signature caching in parallel object database systems

Kjetil Nørvåg

Department of Computer and Information Science, Norwegian University of Science and Technology, 7491 Trondheim, Norway

Received 1 February 2001; revised 20 December 2001; accepted 21 December 2001

Abstract

In many application areas, the access pattern is navigational and a large fraction of the accesses are perfect match accesses/queries on one or more words in text strings in the objects. One example of such an application area is XML data stored in object database systems. Such systems will frequently store large amounts of data, and in order to provide the necessary computing power and data bandwidth, a parallel system based on a shared-nothing architecture can be necessary. In this paper, we describe how the signature cache approach can significantly reduce the average object access cost in parallel object database systems.

Keywords: Parallel database systems; Object database systems; Signature files; Buffer management; Query processing

1. Introduction

In order to provide the desired computing power and data bandwidth, parallel systems are necessary in many cases. This will be the case in many of the emerging application areas for object database systems, where large amounts of data will be stored. A typical example of such an application area is XML/Web storage.

The shared-everything architecture which symmetric multiprocessors are based on has a very limited scalability, so our primary interest is in object database systems based on shared-nothing multicomputers. With the advent of high-performance computers, and high-speed networks, we expect multicomputers based on commodity workstations/servers and networks to be cost effective.

In many of the emerging application areas for database systems, data is viewed as a collection of objects, and the access pattern is navigational. A typical characteristic of the new applications is that a large fraction of the accesses are perfect match accesses (PMA) on one or more words in text strings in the objects/tuples in these databases (these accesses can also be part of other operations, for example join). For such accesses, *signature files* can be used to reduce the query cost. The main drawback of traditional signature files is that signature file maintenance can be relatively costly. If one of the attributes contributing to the signature in an object (or a tuple) is modified, the signature file has to be updated as well. To be beneficial, a high read to write ratio is necessary. This is also the case for dynamic signature files. In addition, high selectivity is needed at query time to make it beneficial to read the signature file in addition to the objects themselves. A better alternative in the case of dynamic data is to use the signature cache (SigCache) approach [11]. Instead of storing the signatures in separate signature files, the signatures are stored together with the objects, and the most frequently accessed signatures are in addition stored in a SigCache. A signature is in general much smaller than an object, so that the number of signatures we can keep in the SigCache is much higher than the number of objects we can store in the main memory buffer. When an object is updated and the signature is stored on the same page, the extra insert cost is only marginal. The signatures can also be used to reduce the CPU cost when the objects are already in memory.

In this paper, we describe in detail the use and maintenance of the SigCache in a parallel ODB (object database system) and analyze its performance by the use of cost functions. As for all access methods, the gain depends on access patterns. We show that the gain from using the SigCache approach is significant for most access patterns. The discussion here is done in the context of parallel ODBs, but the approach is also applicable for parallel relational and object-relational database systems experiencing a navigational access pattern. The techniques presented here should also be applicable in a distributed ODB, where the potential gain is even higher because of the higher communication cost.

The organization of the rest of the paper is as follows. In Section 2 we give an overview of related work. In Section 3 we describe the parallel ODB used as context for this paper.

E-mail address: kjetil.norvag@idi.ntnu.no (K. Nørvåg).

0950-5849/02/$ - see front matter
PII: S0950-5849(02)00003-4

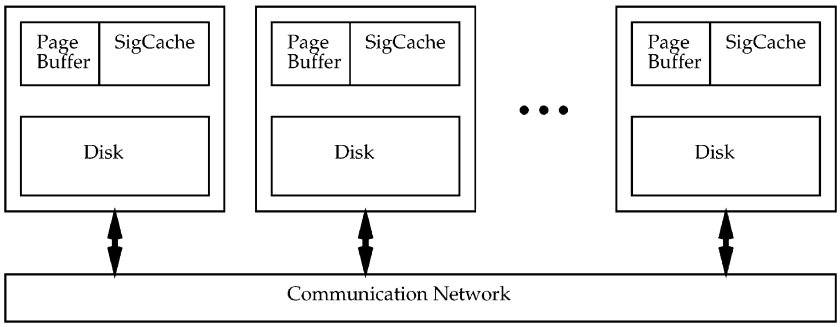

Fig. 1. Architecture of a parallel or distributed database system.

In Section 4 we give a brief introduction to signatures, and describe the SigCache approach. In Section 5 we develop a cost model, which we use in Section 6 to study the performance when a SigCache is used. Finally, in Section 7, we conclude the paper and outline issues for further research.

2. Related work

Several studies have been performed on using signatures as a text access method, e.g. Refs. [1,3,4,9]. Less has been done in using signatures in ordinary query processing, but signature file techniques have been shown to be beneficial in queries on set-valued objects [6].

How to employ signatures in queries in ODBs is described in Refs. [8,10], where the signatures are assumed to be stored in traditional signature files. Another approach, similar to field replication [12], is to store the signature of a referred object inside an object, together with the pointer to the referred object, in order to make it possible to filter objects during forward traversal queries [15].

A more detailed analysis of the performance and gain from using the SigCache approach is given in Ref. [11], where the focus is the disk bottleneck in a non-parallel ODB.

3. System model

In this paper we assume a page server ODB executing on a number of nodes and communicating through a message based communication network as shown in Fig. 1. An instance of the database system runs on each node. On each node, a number of object pages are stored.

In general, a client is connected to one of the nodes. Simple queries can be run on the same node as the client, more complex queries are parallelized and run in parallel on all nodes. When an object is to be accessed, the page where the object is stored has to be retrieved. If the actual object page is not resident in page buffer, the page has to be retrieved from the node where it is stored (possibly the same node as the transaction is running).

When objects are created or updated, they are stored on one of the nodes according to a declustering strategy. A possible strategy is to store related pages (locality sets) on the same node, another is to distribute new created pages round robin on the nodes, in order to reduce the probability of access skew. It has been shown by Venkataraman et al. [14] that the last strategy is most appropriate in a system with multiple large clients. In order to reduce the disk access costs pages can be buffered on servers with idle memory even if they are not stored in the buffer at their home node.

4. Signatures and the SigCache approach

In order to make this paper self contained, we give in this section an introduction to signatures and the SigCache approach. For a more detailed discussion of these issues we refer to Ref. [11].

4.1. Signatures

In this section we describe how to generate signatures, how to use signatures to reduce the cost of PMA, and signature storage alternatives.

4.1.1. Signature generation

A signature can be generated by applying a hash function on some or all of the attributes of the object. By applying this hash function, we get a signature of F bits. If we denote the attributes of an object O_i as $A_1, A_2, \ldots, A_n$, the signature of the object is $s_i = S_h(A_j, \ldots A_k)$, where S_h is a hash value generating function, and $A_j, \ldots, A_k$ are some or all of the attributes of the object (not necessarily including all of $A_j, \ldots, A_k$).

It is possible to generate the signature from the hash value of the concatenation of one or more of the attributes. However, such signatures can only be used for queries on the same set of attributes that were used to generate the signature. For this purpose, using an index will in many cases have a lower cost. In order to be able to support several query types, that do perfect match on different sets of attributes, a technique called *superimposed coding* can be used. In this case, a separate attribute signature is generated for each attribute. The object signature is generated by

performing a bitwise OR on each attribute signature. For example, for an object with three attributes the signature is $s_i = S_h(A_0)$ OR $S_h(A_1)$ OR $S_h(A_2)$. This results in a signature that is very flexible in use. It can support several types of queries with different attributes.

It is not necessary to use all attributes when creating the superimposed signature. If we know that only D of the attributes will be frequently used in PMAs, we can generate the signature from these attributes only. In the case of string attributes, for example in an XML object, we will generate a separate signature from each distinct word in the string attributes and superimpose these word signatures. D will in this case be the number of distinct words in the object.

4.1.2. Using signatures

A typical example of the use of signatures is a query to find all objects in a collection of objects where the attributes match a certain number of values, i.e. the query predicate is $Q = (A_j = v_j, \ldots, A_k = v_k)$. This can be done by calculating the query signature s_q of the query: $s_q = S_h(A_j = v_j, \ldots, A_k = v_k)$ (or $s_q = S_h(v_j)$ OR $S_h(v_i)$ OR…OR $S_h(v_k)$ if superimposed coding is used). The query signature s_q is then compared to all the signatures s_i in the signature file to find possible matching objects. A possible matching object, a *drop*, is an object that satisfies the condition that all bit positions set to 1 in the query signature also are set to 1 in the object's signature. The drops form a set of candidate objects. An object can have a matching signature even if it does not match the values searched for so all candidate objects have to be retrieved and matched against the value set that is searched for. The candidate objects that do not match are called *false drops*.

4.1.3. Traditional signature files

Traditionally, the signatures have been stored in one or more signature files separate from the objects/tuples. The files contain s_i for all objects O_i in the relevant set. The sizes of these files are in general much smaller than the size of the relation/set of objects that the signatures are generated from and a scan of the signature files is much cheaper than a scan of the whole relation/set of objects. Two well-know storage structures for signatures are sequential signature files (SSF) and bit-sliced signature files (BSSF).

In the simplest signature file organization, SSF, the signatures are stored sequentially in a file. A separate *pointer file* is used to provide the mapping between signatures and objects. In an ODB, the pointer file will typically be a file with OIDs, one for each signature. During each search for perfect match, the whole signature file has to be read. Updates can be performed by updating only one entry in the file.

With BSSF, each bit of the signature is stored in a separate file. With a signature size F, the signatures are distributed over F files instead of one file as in the SSF approach. This is especially useful if we have large signatures. In this case, we only have to search the files corresponding to the bit fields where the query signature has a '1'. This can reduce the search time considerably. However, each update implies updating up to F files, which is very expensive. So, even if retrieval cost has been shown to be much smaller for BSSF the update cost is much higher 100–1000 times higher is not uncommon [6]. Thus, BSSF based approaches are most appropriate for relatively static data.

Several improvements of the BSSF have been proposed, most of them imply some vertical or horizontal decomposition [5,7]. Variants that use signature compression and multilevel signatures also exist.

To better support insertions, deletions, and updates several dynamic signature file methods have been proposed. These are multiway tree variants and hash file variants.

Using traditional signature files in a parallel or distributed object database system is not trivial because signature file pages and object pages will be differently clustered because the number of objects on an object page is different from the number of signatures in a signature file page.

4.2. The SigCache approach

An alternative to store the signatures in separate signature files is to store them together with the objects on the object pages and in addition store the most frequently accessed signatures in a SigCache. This is shown in Fig. 2. The SigCache is a lookup table where replacement of signatures is done according to an LRU policy. In this paper we assume that a clock algorithm with one access bit for each signature is used for this purpose.

A signature is stored on the same page as its object. This implies that if an object is resident in main memory (i.e. the page where the object resides is resident in the page buffer), its signature will also be resident because it is stored in the same page. However, the opposite is not true: a signature can be resident in the SigCache even though its object is not resident in the buffer. When a page is discarded from the page buffer, signatures of some of the objects on the page can be resident in the SigCache.

PMA can use the signatures to reduce the number of objects that have to be retrieved from disk or another node. Only the candidate objects with matching signatures need to be retrieved. In order to identify a signature in the SigCache each signature needs to have a unique identifier. In an ODB this will be the OID, in a relational- or object-relational database system this can be a physical tuple identifier or the concatenation of relation and key.

A signature is in general much smaller than an object, so the number of signatures we can keep in the SigCache is much higher than the number of objects we can store in the main memory buffer. The fact that the effective space utilization in an object page buffer is low because of bad clustering on object pages further increase the amount of relevant signatures relative to relevant objects. The optimal size of the SigCache depends on access pattern and total buffer size relative to the total database size. For optimal

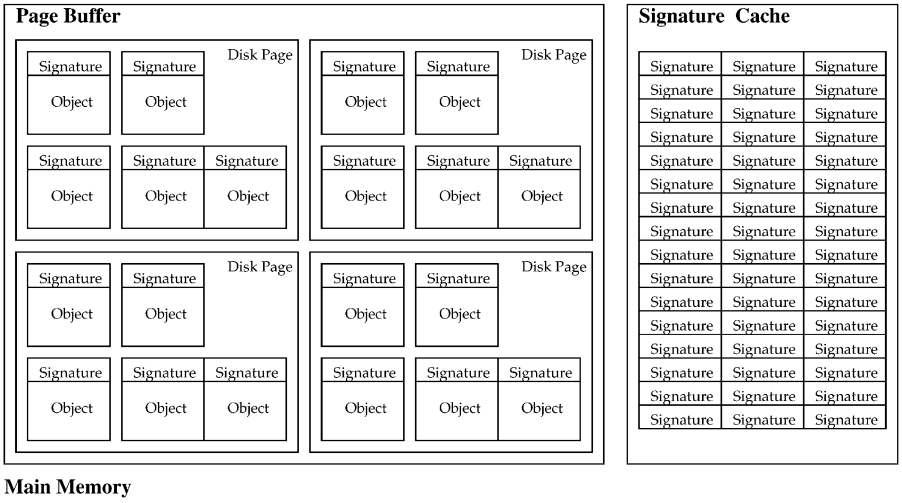

Fig. 2. Storage of object pages and signatures in main memory. Page buffer to the left and signature cache (SigCache) to the right.

performance, the SigCache size can have a size that is adaptively changed by using the cost model that is presented later in this paper. However, even a fixed SigCache size can give a relatively stable performance for a wide range of workload parameters [11].

Signatures are not maintained for all objects in the system, only when it is beneficial. This can for example be decided on the granularity of an object class or object container (also called file). Even when signatures exist, they are only used in a query if it is considered beneficial with respect to query cost. This is similar to traditional secondary indexes where an index is created and maintained only if it is considered beneficial (this is decided by the database administrator) and the index is only used in a query if it is considered beneficial (this is decided during query planning).

4.2.1. The advantages of the SigCache approach

The SigCache approach has many advantages. The most important are:

- Traditionally, signatures have only been beneficial for relatively static data (i.e. a low update rate), for example text documents. Even when dynamic signature files are used, the update rate must be low if the use of signatures should improve performance. Dynamic signature files also have higher space requirements and search cost compared to SSF and BSSF.
  In contrast to using separate signature files the SigCache approach is also useful in the case of high update rates. A signature is in general much smaller than the object it is created from, so that when an object is updated and the signature is stored on the same page the extra insert cost is only marginal. If only a moderate amount of main memory is used for the SigCache the page buffer hit rate is only marginally reduced.
- Read accesses in a relational database system are frequently set accesses, which can benefit from traditional signature files. In contrast, read accesses in ODBs are mostly navigational. Even in the case of collection (for example a set) queries, navigation will often be the result. Unlike a relational database system where the queried set is a relation on storage, in an ODB, a collection can be a collection of references to objects (OIDs) rather than the objects themselves (an object can in this way belong to more than one collection). This navigation can make the average signature retrieval cost high if the signatures are stored in separate files. If the most frequently accessed signatures are stored in the SigCache, this cost can significantly be reduced.
- Previously, signatures have mostly been used to reduce the I/O-costs. However, signatures can also be used to reduce the CPU costs. Even if an object is resident in main memory, a signature comparison can be used before matching the attributes. Especially in the case of many or large attributes, this can reduce the CPU cost for a PMA.

4.2.2. Query processing using the SigCache

When a PMA is performed on one or more attributes of an object, the following algorithm is used to determine if an object O_i is a match, and at the same time maintaining the contents of the SigCache:

1. A lookup is done for the object's signature s_i in the SigCache. If successful, the *access bit* of the signature in the SigCache is set. If this lookup is not successful, the signature has to be retrieved from the page where the object is stored. This page may be resident in the page buffer, but if it is not, the page has to be retrieved from disk or from its home node. When the page is found, the signature s_i which is stored on the page is inserted into the SigCache. The access bit for the signature is set when the signature is inserted.

2. The signature s_i is compared to the query signature s_q. If not all bit positions set to 1 in s_q are set to 1 in s_i, we know for sure that the object does not match the query predicate. If all bit positions set to 1 in s_q are set to 1 in s_i the object is a possible match and we have to retrieve the object in order to compare the value of its attributes with the query predicate. The page where the object is stored on might already be in the page buffer (because it has been accessed recently, or has been brought into the buffer in order to retrieve the signature of one of the objects on the page), if not, the page has to be retrieved from disk or from its home node.

A query on a collection of objects is performed in the same way once for each object. Also note that by employing signatures as outlined in the algorithm above the CPU cost can also be reduced because the signatures are compared before the object is compared with the search predicate.

4.2.3. Object updates and SigCache maintenance

Every time an object is modified, its signature has to be modified as well. A signature is stored together with its object and with respect to concurrency control logging and recovery the signature is treated as a part of the object. If the signature of the object is resident in the SigCache, the signature in the SigCache has to be updated as well. This implies that a SigCache lookup has to be done for each object update. However, compared to the number of CPU instructions necessary to provide persistent storage of an object, this lookup cost is only marginal.

In a parallel ODB, it is possible that the signature of an object is stored on one or several nodes different from the node where its object is to be updated. In order to maintain signature consistency, all the signatures of the objects on a page that are stored on other nodes have to be invalidated before the page can be updated. This is similar to pages cached at remote servers. Those pages also need to be invalidated before a transaction can be granted a write lock on one of the pages.

Only signatures that can contribute in read queries are beneficial to keep in the SigCache so that when a signature is updated in the SigCache the access bit is not set. Read accesses are necessary to make a signature stay in the SigCache.

4.2.4. Signatures and object-orientation

An object is not necessarily just a collection of simple value attributes as a tuple in a relational database system. An object can contain methods and references to other objects and the objects in a queried collection can be objects of different classes due to the concept of inheritance. For a more detailed discussion of these issues we refer to Ref. [11].

4.2.5. Optimal signature size

The signature size is a tradeoff. Using large signatures reduces the false drop probability but large signatures also reduces the number of signatures that can be kept in the SigCache. Too large signatures will also break the assumption that signatures are much smaller than the objects and in that way increase the signature maintenance cost. The signature size should be chosen so that it minimizes the average object retrieval cost for a large range of parameter values.

When deciding the signature size it is also important to keep in mind that the signature size can not easily be changed afterwards if it is too small. If this should be done, reorganizing the database is necessary because there is no reserved space for larger signatures on the object pages.

5. Analytical model

The goal of our analysis is to study the gain of using the SigCache approach in a system with N_N nodes and for this purpose we develop a cost model that models the average object access cost. The bottleneck in a parallel system is normally communication and the transported data volume of the objects can be used as a measure. We denote the cost of accessing an object in terms of the average number of pages required for each object access. We do not consider the additional update cost caused by storing the signatures because the size of a signature compared to an object is small (between 3 and 6% in the study of this paper). The purpose of this paper is to analyze the effect of using signatures in an ODB so we focus on navigational accesses and restrict this analysis to PMAs and signatures generated by the use of the superimposed coding technique.

The database system modeled in this paper is a parallel page server ODB. Each node in the ODB has a total of M bytes available for buffering. Thus, when we talk about the memory size M, we only consider the part of main memory used for buffering. The most recently used object pages are kept in a page buffer of size M_{pbuf} and the most recently used signatures are kept in a SigCache of size M_{scache}. The main memory size M is the sum of the size of the page buffer and the SigCache, $M = M_{pbuf} + M_{scache}$.

Note that the goal of the model is to estimate the impact of using the SigCache in a parallel system and to calculate the gain. For this purpose, transferred data volume is an appropriate measure. If the goal was to find the actual cost the transferred data volume can be weighted with the communication cost for the actual system.

5.1. Buffer hit rates

It is important to have an accurate buffer hit model. For this purpose, we use the Bhide, Dan and Dias LRU buffer model (BDD) [2]. An important feature of the BDD model, which makes it more powerful than some other models is that it also can be used with *non-uniform access distributions*.

The analysis in the paper assumes 'warm' buffers. This includes the page buffer as well as the SigCache. In most

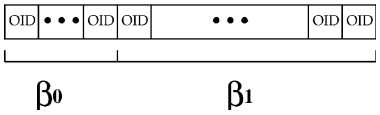

Fig. 3. Partitioned data area. Each partition contains a fraction β_i of the data granules and α_i of the accesses are done to each partition.

systems, this will be the case during normal operation. We consider this the most important case. In systems where this assumption is not valid, object pages will be in the buffer and performance will be bad anyway.

A database in the BDD model has a size of N data granules (e.g. pages) partitioned into p partitions. As shown in Fig. 3, each partition contains a fraction β_i of the data granules, and α_i of the accesses are done to each partition. The distributions *within* each of the partitions are assumed to be uniform and all accesses are assumed to be independent. We denote an access pattern (partitioning set) $\Pi = (\alpha_0, \ldots, \alpha_{p-1}, \beta_0, \ldots, \beta_{p-1})$. For example, for the 80/20 model, $\Pi = (0.8, 0.2, 0.2, 0.8)$. In this paper, we will study performance with the 80/20 access pattern, which we denote 2P8020 and a 90/10 access pattern, which we denote 2P9010.

5.1.1. The BDD buffer model

We will briefly explain the main equations in the BDD model, the derivation and details behind the equations can be found in Ref. [2].

After n accesses to the database, the number of distinct data granules (pages, objects, or index entries) from partition i that have been accessed is:

$$N^i_{\text{distinct}}(n, N, \Pi) = \beta_i N\left(1 - \left(1 - \frac{1}{\beta_i N}\right)^{\alpha_i n}\right)$$

The total number of distinct data granules accessed is:

$$N_{\text{distinct}}(n, N, \Pi) = \sum_{i=1}^{p} N^i_{\text{distinct}}(n, N, \Pi)$$

When the number of accesses n is such that the number of distinct data granules accessed is less than the buffer size B (the number of data granules that fits in the buffer), $\sum_{i=1}^{p} B_i(n) \leq B$, the buffer hit probability for partition i is:

$$P_i(n, \Pi) = 1 - \left(1 - \frac{1}{\beta_i N}\right)^{\alpha_i n}$$

The overall buffer hit probability is:

$$P(n, \Pi) = \sum_{i=1}^{p} \alpha_i P_i(n, \Pi)$$

The steady state average buffer hit probability can be approximated to the buffer hit ratio when the buffer becomes full, i.e. n is chosen as the largest n that satisfies:

$$\sum_{i=1}^{p} N^i_{\text{distinct}}(n, N, \Pi) = B$$

We denote the average buffer hit probability as:

$$P_{\text{buf}}(B, N, \Pi) = P(n, \Pi)$$

where n is chosen as described earlier.

5.1.2. Page buffer hit rate

The number of disk pages necessary to store a database with N_{obj} objects assuming page size S_{P} and an average object size of S_{obj} bytes:

$$N_{\text{objpages}} = \left\lceil \frac{N_{\text{obj}} S_{\text{obj}}}{S_{\text{P}}} \right\rceil$$

If we store F bits signatures of all objects on the object pages as well the number of disk pages is:

$$N_{\text{objpages}} = \left\lceil \frac{N_{\text{obj}}(S_{\text{obj}} + \lceil F/8 \rceil)}{S_{\text{P}}} \right\rceil$$

With a page buffer size of M_{pbuf} bytes and an overhead for each item in the buffer of S_{oh} bytes we can keep $N_{\text{pbuf}} = \lfloor M_{\text{pbuf}}/S_{\text{P}} + S_{\text{oh}} \rfloor$ of these pages in main memory. The buffer hit rate in this case is:

$$P_{\text{buf\_page}} = P_{\text{buf}}(N_{\text{pbuf}}, N_{\text{objpages}}, \Pi)$$

5.1.3. SigCache hit rate

For each signature in the SigCache, we need to store object identifying information in order to know which object it belongs to. It is not necessary to store the whole OID for each signature, variants of prefix compression or multilevel access tables can be employed. Assuming S_{ID} bytes in average are needed to know which object a signature belongs to, the number of signatures that fits in the SigCache is:

$$N_{\text{scache}} = \left\lfloor \frac{M_{\text{scache}}}{\lceil F/8 \rceil + S_{\text{oh}} + S_{\text{ID}}} \right\rfloor$$

The SigCache hit rate is:

$$P_{\text{scache}} = P_{\text{buf}}(N_{\text{scache}}, N_{\text{obj}}, \Pi)$$

5.2. Object access cost

One or more objects are stored on each page. When an object page is to be retrieved, the probability that this page resides on the requesting node is $1/N_{\text{N}}$. We denote the cost of retrieving a page from another node as S_{P}. Thus, the average cost of reading one page from the database is:

$$C_{\text{readpage}} = \left(1 - \frac{1}{N_{\text{N}}}\right)(1 - P_{\text{buf\_page}})S_{\text{P}}$$

In order to reduce the object access cost, objects are usually placed on pages in a way that makes it likely that more than one of the objects on a page that is read will be needed in the near future. This is called clustering. In our model, we define the clustering factor C as the fraction of an object page that is relevant, i.e. if there are $N_{\text{o\_page}} = N_{\text{obj}}/N_{\text{objpages}}$

Table 1
Default system model parameters

Parameter	Value
S_P	8 KB
N_N	4 and 32 nodes
S_{oh}	8 bytes
S_{ID}	4 bytes

objects on each page and n of the objects on the page will be used before the page is discarded from the buffer $C = n/N_{o\_page}$. If $N_{o\_page} < 1.0$, i.e. the average object size is larger than one page, we define $C = 1.0$. The average object access cost is:

$$C_{readobj}^{nosig} = \frac{1}{CN_{o\_page}} C_{readpage}$$

We model the database read accesses as (1) *ordinary object accesses* and (2) *perfect match accesses*, which can benefit from signatures. We assume the PMA to be a fraction P_{PMA} of the read accesses and that P_A is the fraction of matched objects that are actual drops. The false drop probability when a signature with F bits is generated from D attributes is denoted $F_d = (1/2)^m$, where $m = F \ln 2/D$.

The average object access cost employing signatures is:

$$\begin{aligned} T_{readoj}^{sig} &= \text{Cost of non-PMA access} \\ &\quad + \text{Cost of PMA access where} \\ &\quad \text{signature not in SigCache} \\ &\quad + \text{Cost of PMA access where} \\ &\quad \text{signature in SigCache} \\ &= (1 - P_{PMA}) T_{readobj}^{nosig} \\ &\quad + P_{PMA}(1 - P_{scache}) T_{readoj}^{nosig} \\ &\quad + P_{PMA} P_{scache} \Big(P_A T_{readobj}^{nosig} \\ &\qquad + (1 - P_A) F_d T_{readoj}^{nosig} \Big) \end{aligned}$$

This means that of the P_{PMA} accesses that are for perfect match, we only need to read the object page in the case of actual or false drops.

6. Performance

In this section, we study how different workloads and system parameters affect the gain from using signatures. Based on the cost functions from Section 5, we calculate the gain from using signatures as:

$$\text{Gain} = 100 \left(\frac{T_{readobj}^{nosig} - T_{readobj}^{sig}}{T_{readobj}^{sig}} \right)$$

When studying parallel database systems, we can choose to study *speedup* or *scaleup*. If the choice is speedup, we keep the database size constant but increase the processing power and disk bandwidth by adding more nodes. If scaleup is the choice, the database size is increased linearly with the number of nodes. The most important reason for adding more nodes is to be able to handle larger databases, so in this paper we concentrate on scaleup. The number of objects N_{obj} is scaled with the number of nodes so that when the number of nodes is increased by a factor S the number of objects is increased with the same amount.

Table 2
Default workload parameters

Parameter	Workload I	Workload II
S_{obj}	128 bytes	2048 bytes
C	0.2	0.8
N_{obj}	$N_N \times 32$ mill.	$N_N \times 2$ mill.
D	4 attributes	128 attributes
P_A	0.001	0.001
P_{PMA}	0.4	0.8
Π	2P9010	2P9010
F	32 bits	1024 bits

The system model parameters are summarized in Table 1. For the workload, we consider two main cases:

Workload I: This is the workload of a traditional application. The object size is relatively small and the signatures are generated from a small number of attributes. In such applications, there will be a mix of access types and only some of them can benefit from using signatures.

Workload II: This is a possible workload in one of the emerging application areas, for example XML storage. As described by DeWitt et al. [13], the space overhead would be very high if each XML element was stored as a separate object. Instead, one object is used to store one XML document and the XML elements stored as 'light-weight objects' inside one storage object. The result is larger objects than in workload I, and each object contains a higher number of attributes. The attributes are frequently text strings and a large fraction of the queries in such systems will be for perfect match of one or more words. In order to be able to use signatures for such queries, the object signatures are generated by superimposing the individual text word signatures. As a result, the value of D will be large.

The default parameter values for the two workloads are given in Table 2. Note that with the default parameters, we keep the total database size constant. The signature sizes used are $F = 32$ for workload I, and $F = 1024$ for workload II. The signature sizes are chosen so that they minimize $T_{readobj}^{sig}$ for a large range of parameter values.

6.1. Gain

Fig. 4a shows the gain from using signatures under workload I with different access patterns and number of nodes. We see that using signatures is especially beneficial for access patterns with a narrow hot spot area. The gain is

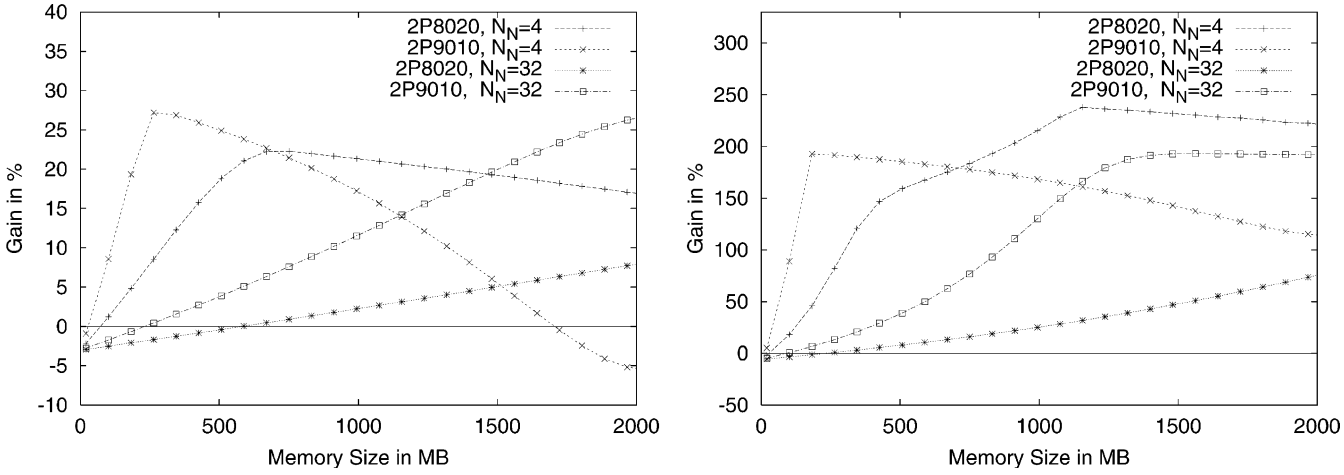

Fig. 4. Gain from using signatures with different access patterns and number of nodes, workload I to the left and workload II to the right. The memory size is the amount of main memory available for buffering on each node.

highest when all signatures of hot spot objects fit in main memory, but the main memory size is too small for the objects themselves to fit. With a larger number of nodes ($N_N = 32$ in the figure), this point is not reached on the memory range used in Fig. 4a.

Fig. 4b shows the gain from using signatures under workload II. In this case, the gain is high in the order of 200%.

6.2. The effect of P_{PMA}

The value of P_{PMA} is the fraction of the read accesses that can benefit from signatures.

Fig. 5 shows the gain with different values of P_{PMA}. As can be expected, a small value of P_{PMA} results in small or negative gain. As we increase the value of P_{PMA}, the gain increases. With large values of P_{PMA}, as we have assumed for case II we can achieve a very high gain.

6.3. The effect of P_A

The value of P_A is the fraction of PMA that are actual drops (selectivity). The purpose of signatures is to reduce the number of objects that actually have to be retrieved and we can only benefit from signatures if this number is sufficiently low.

Fig. 6 shows the gain with different values of P_A for workload I and II and it is interesting to note that signatures will be beneficial even with a relatively large value of P_A.

6.4. Optimal SigCache size

The fraction of memory that should be used for caching signatures depends on the total memory size, database size and workload. The SigCache size can either be static (but tunable) or it can be adaptive (using cost functions to determine the size). Fig. 7 shows the optimal SigCache size as the fraction of the total main memory size.

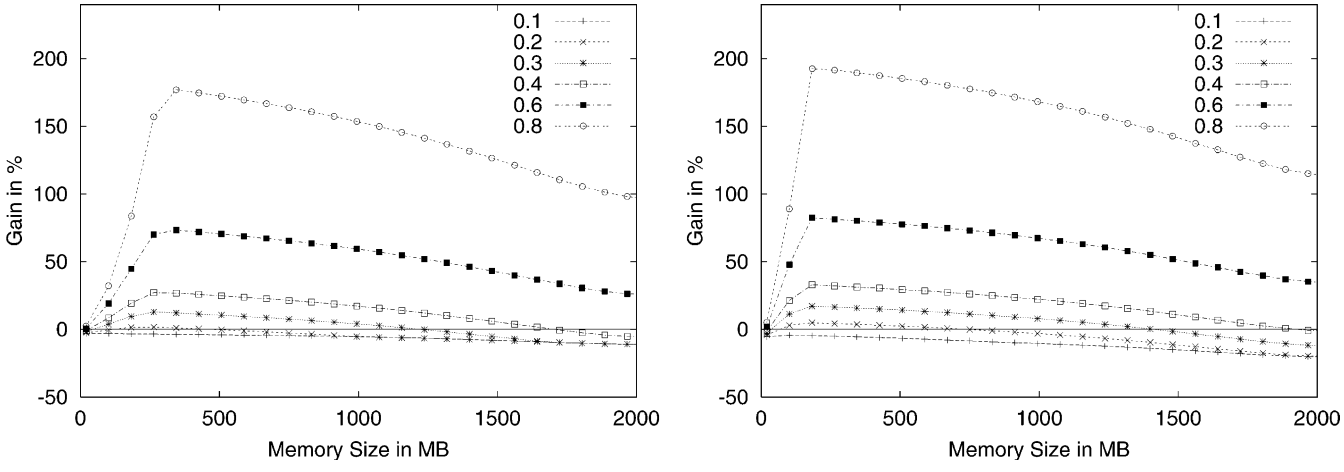

Fig. 5. Gain from using signatures with different values of P_{PMA} with $N_N = 4$. Workload I to the left and workload II to the right.

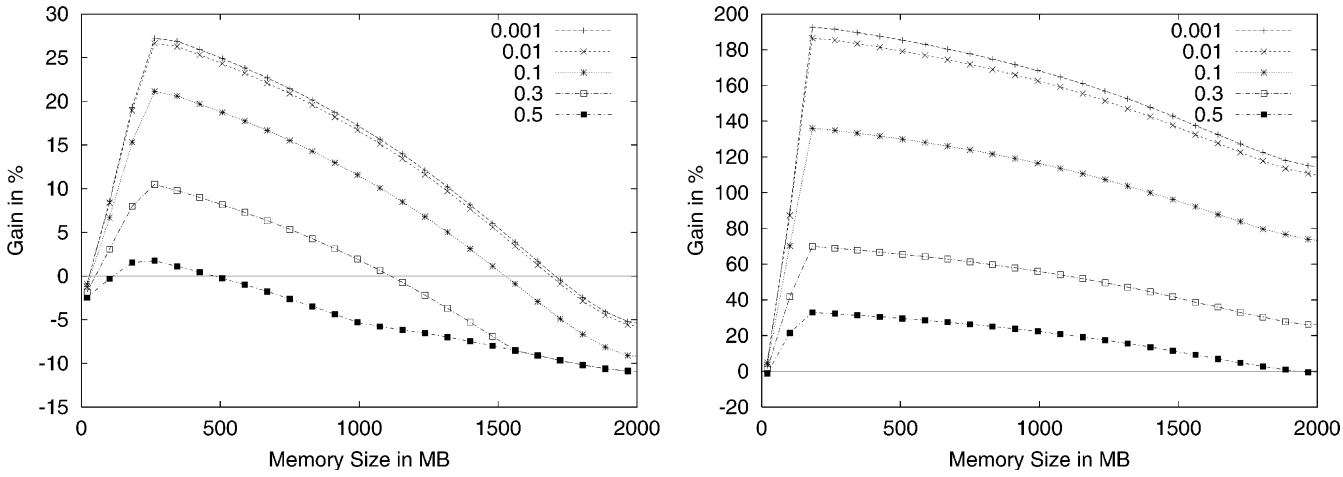

Fig. 6. Gain from using signatures with different values of P_A with $N_N = 4$. Workload I to the left and workload II to the right.

For workload I, it is beneficial to use most of the memory to cache signatures when the main memory size is small. It is most important to keep the most frequently accessed signatures in the SigCache so that when the SigCache is large enough to keep them increasing the SigCache size further has less impact on performance. In that case, the optimal SigCache size decreases.

For workload II, the optimal fraction of memory used for caching signatures is larger than for workload I. The main reason for this is the larger signature size used for workload II. Given a certain SigCache size, the actual number of signatures that fits in the SigCache is smaller.

6.5. Gain with large memory sizes

The analysis have until this point been done under the assumption that only a relatively small amount of data fits in main memory. However, as the price of main memory decreases, it will be common with servers that can keep most of the database in main memory. Fig. 8 shows the gain with larger main memory sizes up to the point where the whole database fits on each node (i.e. the database is replicated on each node).

Fig. 8 shows the gain from using signatures with large main memory sizes. We see that using signatures is especially beneficial for access patterns with a narrow hot spot area. However, there are two cases when gain is negative or only marginal:

1. When the memory size is large enough to keep most of the hot spot object pages. In this case, it is more beneficial to use all the memory for page buffering so that the hot spot objects can be kept in main memory. It should be noted that when the memory size is smaller so that only some of these pages fits in the page buffer using signatures is beneficial.
2. The number of object pages necessary to store a database will be larger if signatures are stored as well. If the main memory is large enough to keep most of the object pages in main memory (large values of M)

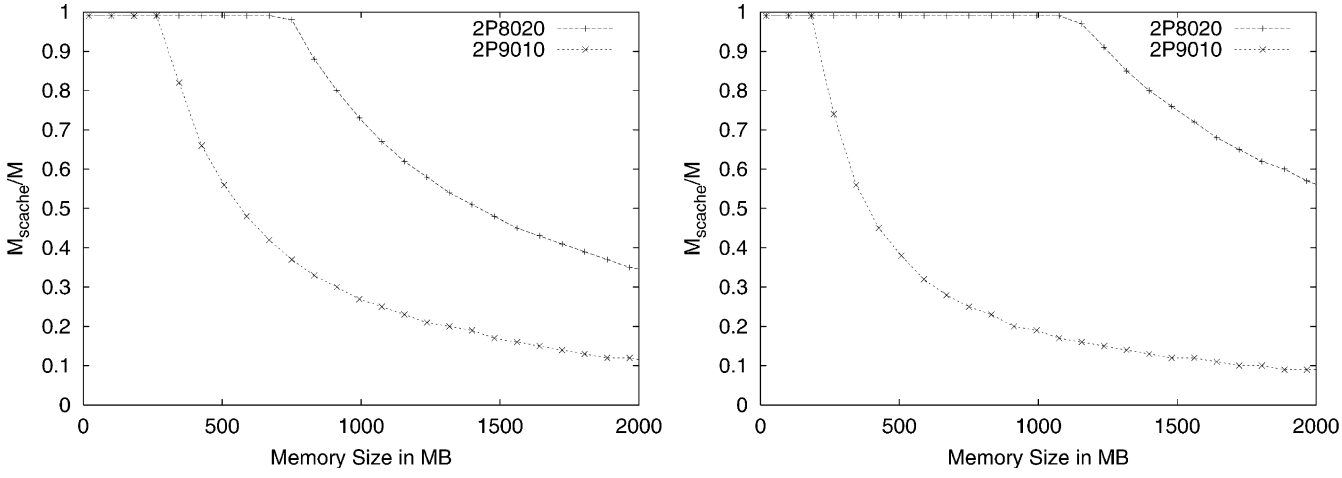

Fig. 7. Optimal SigCache size as the fraction of the total main memory size with $N_N = 4$. Workload I to the left and workload II to the right.

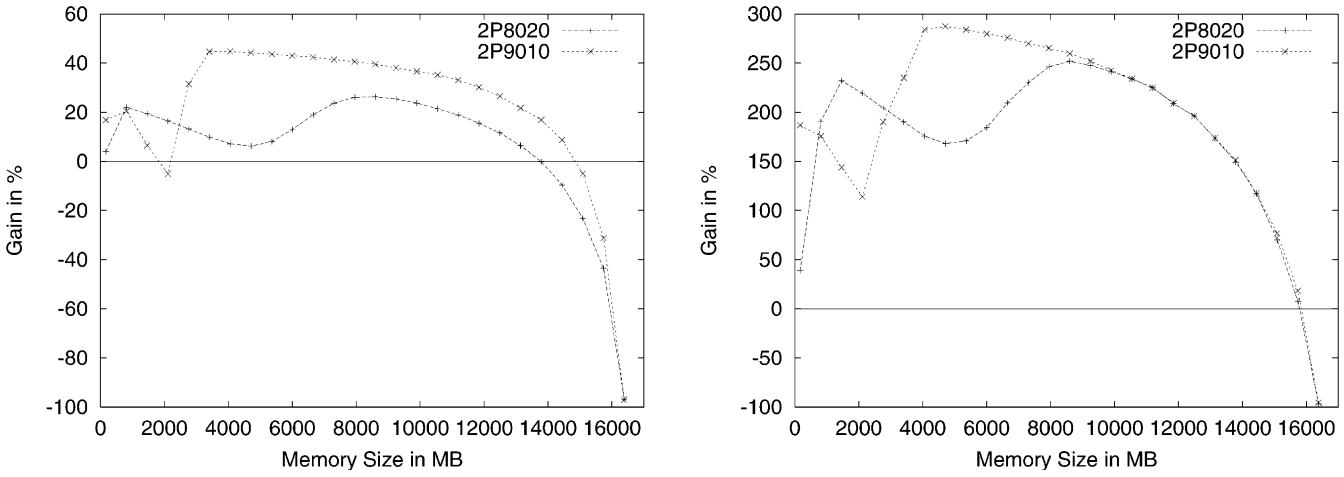

Fig. 8. Gain from using signatures with large memory sizes with $N_N = 4$. Workload I to the left and workload II to the right.

when signatures are not used, using signatures will result in decreased or negative gain because of a lower page buffer hit rate.

6.6. Scalability and speedup

One of the problems with a parallel page server ODB is limited scalability. One of the main reasons for this is navigational accesses to data belonging to locality sets of computations on several nodes in combination with a size of main memory that is not sufficient to cache the actual pages. The SigCache approach does not entirely solve this problem but reduces the impact of the problem because caching signatures can also be regarded as 'cheap replication'.

Speedup in a parallel ODB is a result of increased number of nodes (until a certain point). Using the SigCache approach is orthogonal to this but for a certain number of nodes using the SigCache approach will in most cases be beneficial. This can also be seen on Fig. 4, where the gain when using 32 nodes can be compared with the gain when using four nodes. In general, using a very high number of nodes in an ODB can be counter-productive, as the communication cost will increase with the number of nodes as a result of navigational accesses. The exception is in a context of relatively few updates and sufficient main memory at each node to keep the working set.

7. Conclusions

In this paper, we have described how object signatures can be cached in main memory in a signature cache and how the use of the signatures in perfect match object accesses can be used to reduce the average object access cost in a parallel ODB. We developed a cost model that we used to analyze the performance of the proposed approaches and this analysis showed that substantial gain can be achieved.

When storing signatures together with their objects instead of in separate signature files the signature maintenance cost in terms of disk space as well as I/O is only marginal. This makes the SigCache technique an interesting supplement to traditional signature files as well as traditional indexes, which have a higher maintenance cost and in the case of indexes a higher storage cost as well.

References

[1] E. Bertino, F. Marinaro, An evaluation of text access methods, Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, 1989, vol. II, Software Track, 1989.
[2] A.K. Bhide, A. Dan, D.M. Dias, A simple analysis of the LRU buffer policy and its relationship to buffer warm-up transient, Proceedings of the Ninth International Conference on Data Engineering, 1993.
[3] C. Faloutsos, Access methods for text, ACM Computer Surveys 17 (1) (1985).
[4] C. Faloutsos, R. Chan, Fast text access methods for optical and large magnetic disks: designs and performance comparison, Proceedings of the Fourteenth VLDB Conference, 1988.
[5] N.W. Hiroyuki Kitagawa, Y. Ishikawa, Design and evaluation of signature file organization incorporating vertical and horizontal decomposition schemes, Proceedings of the Seventh International Conference on Database and Expert Systems Applications, DEXA'96, 1996.
[6] Y. Ishikawa, H. Kitagawa, N. Ohbo, Evaluation of signature files as set access facilities in OODBs, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993.
[7] H. Kitagawa, K. Fukushima, Composite bit-sliced signature file: an efficient access method for set-valued object retrieval, Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications (CODAS), 1996.
[8] D. Lee, W.-C. Lee, Signature path dictionary for nested object query processing, Proceedings of the IEEE International Phoenix Conference on Computers and Communications (IPCCC'96), 1996.
[9] D.L. Lee, Y.M. Kim, G. Patel, Efficient signature file methods for text retrieval, IEEE Transactions on Knowledge and Data Engineering 7 (3) (1995).
[10] W.-C. Lee, D. Lee, Signature file methods for indexing object-oriented database systems, Proceedings of the Second International Computer Science Conference, 1992.
[11] K. Nørvåg, Fine-granularity signature caching in object database systems, Journal of Data and Knowledge Engineering 38 (2) (2001).

[12] E.J. Shekita, M.J. Carey, Performance enhancement through replication in an object-oriented DBMS. Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, 1989.
[13] F. Tian, D.J. DeWitt, J. Chen, C. Zhang, The design and performance evaluation of alternative XML storage strategies, SIGMOD Record 31 (1) (2002).

Kjetil Nørvåg *is an Associate Professor in the Department of Computer and Information Science at the Norwegian University of Science and Technology. He received a Dr Ing. degree in computer science from the Norwegian University of Science and Technology in 2000 and was a postdoctoral researcher in the VERSO group at INRIA (France) in 2001. His major research interests include query and storage of XML documents, object database systems, temporal database systems, query processing, and access methods.*

[14] S. Venkataraman, M. Livny, J. Naughton, Impact of data placement on memory management for multi-server OODBMS, International Conference on Data Engineering, 1995.
[15] H.-S. Yong, S. Lee, H.-J. Kim, Applying signatures for forward traversal query processing in object-oriented databases, Proceedings of the Tenth International Conference on Data Engineering, 1994.