Buffer Performance Modeling in the Context of Unclustered Index Accesses with Non-Uniform Access Pattern

Kjetil Nørvåg [1]

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

Abstract

Cost models are powerful tools for analyzing algorithms, and important in cost-based query optimization. With increasing amounts of main memory available, it is important to include buffer performance in the models. In this paper, we describe and validate 1) buffer models for fine-granularity caching in buffers with locking, and 2) a buffer model for nodes in multiway-tree indexes in the context of unclustered accesses and non-uniform access patterns. The validations show a high accuracy of the models. In future self-tuning database systems, the importance of such models will be even higher than today, and the models presented in this paper should be ideally suited for use in these applications.

Key words: index, main memory buffer, analytical cost models

1 Introduction

Cost models are powerful tools for analyzing algorithms, and important in cost-based query optimization. They will also be necessary in self-tuning systems, which will become more important with the increasing complexity of both software and hardware [2]. Prediction of buffer hit rates is important in order to divide main memory resources between different tasks/queries.

Analytical modeling in database research has mostly focused on I/O costs. This is done under the assumption that I/O is the bottleneck. While this is in general true, buffer performance has an increasing impact on performance, as a result of increasing amounts of main memory available for buffering. It is very interesting (and

---

[1] E-mail: Kjetil.Norvag@idi.ntnu.no

alarming!) that many papers using the analytical approach *do not* include buffering aspects, or only do some very simplified assumption, not taking into account access skew (this is particularly evident in index-related papers).

In this paper we present a model for fine-granularity caching, and a model for multiway-tree indexes in the context of unclustered accesses and non-uniform access patterns. Our models are based on the Bhide, Dan and Dias LRU buffer model (BDD) [1], which models buffer performance with non-uniform access distributions. Our models extends the BDD model on the following:

(1) In a mixed workload with both read and write accesses, dirty items in the buffer can not be immediately replaced, they have to be written back first. These items are *locked*. This impact the models, and we study what impact this has on the accuracy of the model.
(2) Hierarchical accesses (for example in the context of multiway-tree indexes) are also considered.

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 outline the motivation for the work presented in this paper. In Section 4 we describe the access model which is assumed in the BDD model, and we give a brief overview of the BDD model itself in Section 5. In Section 6 we study modeling of fine-granularity caching in buffers with locking. In Section 7 we describe our generalized index buffer model. Finally, in Section 8, we conclude the paper.

2   Related work

Estimations of block accesses are closely related to our work. The most well-known model is the Yao model, which assumes uniform distribution [16]. An approximation to Yao's formula was presented by Whang et al. [15]. A study of errors when using Yao's formula with non-uniform distribution and non-uniform block size was performed by Luk [6]. Diehr and Saharia studied upper and lower bounds for the Yao function in [3]. Common for all these models are that they assume unlimited buffer. Mackert and Lohman modeled index scans with limited buffer [7], but did not consider hot spots, only uniformly completely random access to an unclustered index. An extension of this study was performed by Swami and Schiefer [14].

Concurrently with the development of our work, an index buffer model based on the BDD model has been presented by Leutenegger and Lopez in [5]. Their index buffer model was done in the context of buffering of R-tree nodes, but can also be applied to B-trees. The main difference between our model and the one presented by Leutenegger and Lopez, is that their model only consider a uniform access pattern.
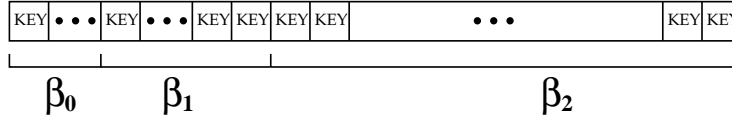
Fig. 1. Logical access partitions.

3 Motivation

Simulations and analytical modeling have been used extensively in database related research in order to compare different algorithmic and architectural options. In evaluations of complex systems, simulations have been most commonly used, mainly because of their proximity to implementation, the sometimes unrealistic assumptions sometimes needed to make analytical models, and the problem of incorporating concurrent and dynamic events into analytical models. However, simulations have some very important shortcomings: 1) they are *time consuming*, and as a result only a small part of the parameter space can be explored, and 2) *it is not always easy to explain the results*! If analytical models are used, we can study the performance with many different combinations of parameters in a short time. It is also easier to determine dependencies and explain results with the help of analytical models. Analytical modeling can also be used together with simulations, to determine which parameters should be used in the simulations, and to help explain the results of the simulations.

Our application of the models presented in this paper has in particular been OID indexes in object database systems, where we have successfully [8–10,12,13] employed the buffer models in our analysis. We have also deeply appreciated analytical modeling as a powerful tool in the design of Vagabond [11]. Several bottlenecks were identified using the analytical models, and many of the solutions were also developed with the help of the models. Another example of an application area where the models in this paper would be very suitable, is in the full-text indexes (inverted lists), which have similar access patterns.

The results in this paper are not limited to analysis of algorithms or systems. Other important application areas include cost-based query optimization and dynamic system tuning. In this context, it is important to note that the evaluation of buffer performance using the models in this paper has a low cost, making them suitable in an on-line/dynamic context.

4 Access model

We assume accesses to items to be random, but skewed (some items are more often accessed than others). We assume it is possible to (logically) partition the range of

Set	β_0	β_1	β_2	α_0	α_1	α_2
3P1	0.01	0.19	0.80	0.64	0.16	0.20
3P2	0.001	0.049	0.95	0.80	0.19	0.01
3P3	0.30	0.60	0.10	0.70	0.10	0.20
3P4	0.70	0.10	0.20	0.80	0.10	0.10

Table 1
Partition sizes and partition access probabilities for the access patterns.

KEYs into partitions, where each partition has a certain size and access probability. This is illustrated in Fig. 1. Note that there is no correlation between KEY and partition, i.e., KEY=1 can be in partition 2, KEY=2 can be in partition 1, etc.

Our access model is the same as used by Bhide, Dan and Dias in their LRU buffer analysis [1], to be described in Section 5. This model, based on the independent reference model, considers a database with a size of N items (for example pages or objects). The database can be partitioned into p partitions. Each partition contains β_i of the items, as a fraction of the total database size or number of items (see Fig. 1), and α_i of the accesses are 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) for a set of items as Π. Π has p partitions, and the following has to be true:

$$\sum_{0}^{(p-1)} \alpha_i = 1.0$$

$$\sum_{0}^{(p-1)} \beta_i = 1.0$$

The studies in this paper use the access patterns summarized in Table 1. Even though the 80/20 model has been widely employed in many analysis and simulations, and has been satisfactory in the analysis of many problems, it has a major shortcoming: when applied to calculate the number of distinct items accessed, it gives a much higher number of distinct accessed items than in a real system. The reason is that for most applications, inside the hot-spot partition (20% in this case), there is an even hotter and smaller partition, with a much higher access probability. This has to be reflected in the model, and is incorporated into the access patterns 3P1 and 3P2, which each consists of three partitions. In both access patterns, the 20% hot-spot partition from the 80/20 model is further partitioned. In 3P1 the 20% hot-spot partition is further divided into a 1% hot-spot partition and a 19% less hot partition. The access pattern 3P2 resembles the access pattern close to what we expect it to be in temporal ODBMSs (which has been the main application area for some of our research), with a large cold partition, consisting of old versions. In

Partition Fraction	Partition Size (Distinct Words)			# of Words in the Book			Access Probability α_i		
	DB	EG	FS	DB	EG	FS	DB	EG	FS
$\beta_0 = 0.01$	241	25	94	432188	16341	87174	0.66	0.42	0.50
$\beta_1 = 0.19$	4583	465	1774	178910	17744	64640	0.27	0.46	0.37
$\beta_2 = 0.80$	19294	1960	7470	44830	4435	23531	0.07	0.12	0.13

Table 2
Partition sizes and partition access probabilities for three analyzed books, which shows that our assumptions regarding access pattern can be justified.

order to study the validity of the models also in the case of other parameter ranges, we have also included the access patterns 3P3 and 3P4.

Some people might question the validity of the assumptions above, and wonder whether such a large amount of requests going to a small area is a realistic assumption. To show that this is quite reasonable, even on non-temporal data, we will give an example. Imagine a spell checker, which spell checks a document by comparing each word in the document against the words in a dictionary which is accessed by an index. Each word in the dictionary can be though of as an item, while each word in the document that is spell checked can be considered an item access (we have to look up this word in the dictionary). The access probability for a book with W words in total, and N distinct words, can be found as follows:

(1) Define partition sizes β_i, for example 0.01, 0.19 and 0.80 as in the 3P1 access pattern.
(2) Count the number of occurrences for each distinct word. [2]
(3) Sort the word/occurrence counts tuples on occurrence counts. The $\beta_0 N$ most frequent words are in partition 0, the next $\beta_1 N$ words in partition 1, and the $\beta_2 N$ less frequently used words in partition 2.
(4) The access probability for partition i is the sum of the occurrence count for each word in the actual partition, divided on W.

As an example, we have used three books: a Danish bible (DB), an English version of the Genesis (EG), and a Norwegian book on access methods and query processing (FS). If we consider the number of distinct words in each of the books as dictionaries (24118, 2450 and 9338 unique words, respectively), and do a spell check of the books (655928, 38520, and 175345 word accesses), we obtain the access probabilities for the 1%/19%/80% partitions as summarized in Table 2. They access probabilities show that our assumptions regarding access probabilities can be justified, and is also consistent with a comparison with the Zipf distribution.

---

[2] Note that each conjugation of a word will be present in the spell checker dictionary. The actual number of words as would appear in a traditional dictionary would be smaller.

5 The BDD LRU buffer model

In our analysis, we need to estimate the buffer hit probability in an LRU-managed buffer given a certain access pattern. This is done based on the LRU buffer model developed by Bhide, Dan and Dias (the BDD model). In this section, we present the main results and equations from the model. The derivation and details behind the equations can be found in [1].

Distinct items. After n accesses with access pattern Π to a database containing N items, the number of *distinct* items that has been accessed is (p is the number of partitions as defined in Section 4):

$$N_{distinct}(n, N, \Pi) = \sum_{i=0}^{p-1} N_{distinct}^i(n, N, \Pi)$$

where $N_{distinct}^i(n, N, \Pi)$ is the number of distinct items from partition i that have been accessed:

$$N_{distinct}^i(n, N, \Pi) = \beta_i N(1 - (1 - \frac{1}{\beta_i N})^{\alpha_i n}) \tag{1}$$

Buffer hit probability. When the number of accesses n is such that the number of distinct data items accessed is less than the buffer size B (the number of items that fits in the buffer), $\sum_{i=0}^{p-1} N_{distinct}^i \leq B$, the buffer hit probability for partition i is:

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

and the overall buffer hit probability is:

$$P(n, \Pi) = \sum_{i=0}^{p-1} \alpha_i P_i(n, \Pi)$$

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 : [3]

$$\sum_{i=0}^{p-1} N_{distinct}^i(n, N, \Pi) = B$$

---

[3] Binary search can be used to find n, because $N_{distinct}^i$ are monotonically increasing functions of n.

```
┌─────────────────────────┐
│     Trace Generator     │
└─────────────────────────┘
             │
┌─────────────────────────┐
│       Item Cache        │
└─────────────────────────┘
             │
┌─────────────────────────┐
│          Index          │
└─────────────────────────┘
```
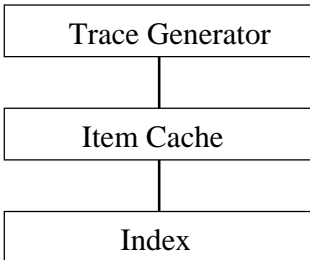
Fig. 2. TSIM architecture.

We denote the average buffer hit probability as:

$$P_{buf}(B, N, \Pi) = P(n, \Pi) \tag{2}$$

where n is chosen as described above.

6 Buffer hit probability in a buffer with locking

The BDD model assumes that all items in the buffer are eligible for replacement. However, this is not the case in a buffer with dirty items, for example in a page buffer, object buffer, or index item cache. To avoid a very costly synchronous writeback of items in the buffer, items are written back to the database/index asynchronously in the background, using some strategy to reduce disk arm movement. This means that some of the items in the buffer are locked and cannot be discarded until they have been written back to disk. In this section, we study the inaccuracies of the original model in the context of a buffer with locking, and describe two possible approaches to reduce the inaccuracies: the *DCOMP* and the *insert-compensating BDD (IC-BDD)* models.

6.1 The simulation approach

In the validation of the model we used the TSIM simulator, a toolbox originally developed for studies of different temporal object identifier indexes. Fig. 2 gives an overview of the parts of the architecture of the TSIM that is relevant for this paper:

- **Trace generator:** The trace generator generates access requests (i.e., keys), which are handled by simulator. Requests from the trace generator can be generated on the fly, based on defined access patterns, or they can come from traces from real programs running against a temporal ODBMS. The results reported in this paper are from an artificially generated workload, as will be described shortly.
- **Item cache:** This is a cache, with cache replacement according to the clock algorithm.

- **Index:** This module contains an index, which during the simulations reported in this paper is simulated by a main-memory structure.

The workload consists of read, write, and create requests. There are no separate delete requests, but delete can in this context be considered a write operation, because the key will be removed. The following parameters are used to specify the workload:

- Π: The access pattern (see Section 4). Both read and write accesses are performed according to this access pattern.
- P_{write}: The probability that an operation is a write or create operation.
- P_{new}: The probability that a write operation creates a new item (object, index entry, etc.).
- F_{dirty}: The fraction of the items in the item cache that are allowed to be dirty, i.e., not yet inserted into the index.
- N: The total number of items in the index.

We have performed the simulations with the access patterns in Table 1. Default values for the other parameters are $P_{write} = 0.2$, $P_{new} = 0.2$, and $F_{dirty} = 0.2$. The total number of items has been set to $N = 50000$. This might seem too small, but the difference between using $N = 50000$ and a larger value of N is only marginal.

The simulations have been run in two modes, *READ_DB* and *FILL_DB*:

- In the *READ_DB* mode we are only interested in the hit rate for a read-only workload when the index is in a stable condition. When using this mode, we first insert all N items into the index. After the items have been inserted, a large number of read accesses to the index are performed in order to "warm up" the item cache. After the warm-up phase, we assume the item cache is in a stable condition, and we measure the hit rate from a number of read accesses.
- In the *FILL_DB* mode, we want to measure the hit rate under a more typical workload, with both read and write accesses applied to the system. One problem when doing this, is that the analytical models are based on an index in a stable condition.

6.2 Results from using the unmodified model

Fig. 3 shows the simulated item cache hit rate for read-only accesses (obtained using the *READ_DB* mode), compared with the calculated hit rates based on the BDD buffer model. For all access patterns, the deviations between the model and the simulations are in general less than 1%. The accuracy of the model in the case of a read-only workload is as expected, because the accesses are done following the assumptions behind the BDD model. The reason for the few inaccuracies that can be observed, is that the item cache uses the clock algorithm, which is only
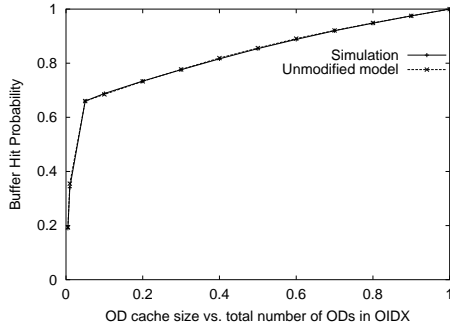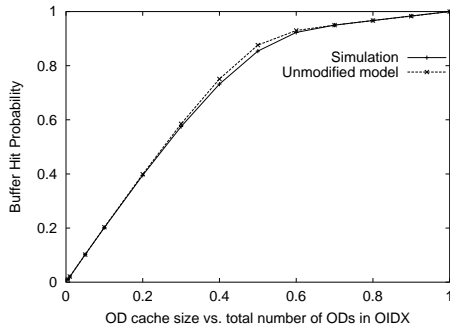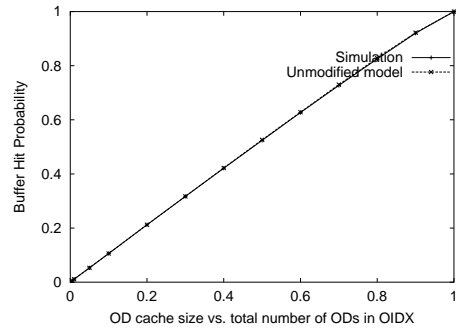
(a) 3P1 access pattern.
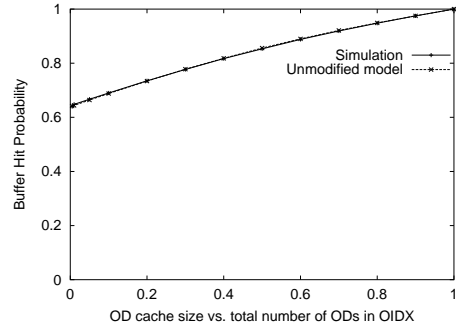
(b) 3P2 access pattern.
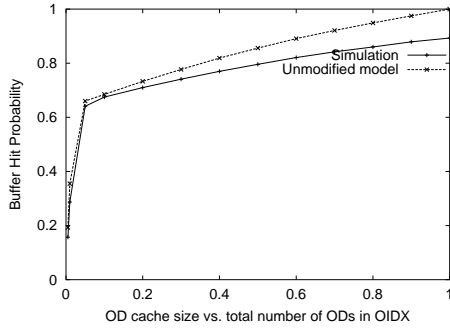
(c) 3P3 access pattern.

(d) 3P4 access pattern.

Fig. 3. Item cache hit rate for read-only accesses.

an approximation to the LRU algorithm (for sufficiently large buffers the clock algorithm is a good approximation of the LRU strategy [4]).

Fig. 4 shows the simulated item cache hit rate in the case of a workload of both read and write accesses (obtained from using the *FILL_DB* mode). In this case, the deviation between *unmodified* model and the simulation is much higher. The reason for this, is that dirty items are locked in the item cache until they have been written to the index. Many of these dirty items are not hot-spot items, and would otherwise have been replaced in the item cache.

6.3 The DCOMP model

The fraction of the items in the buffer that are locked depends on the writeback rate. A high writeback rate results in less locked items and increases the buffer hit rate, but in general it also increases the average writeback cost. We denote the fraction of the items in the buffer that are allowed to be dirty as F_{dirty} (which means that the writeback rate should be high enough to keep the number of dirty items less than

(a) 3P1 access pattern.
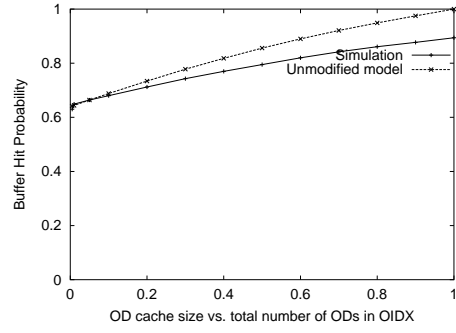
(b) 3P2 access pattern.
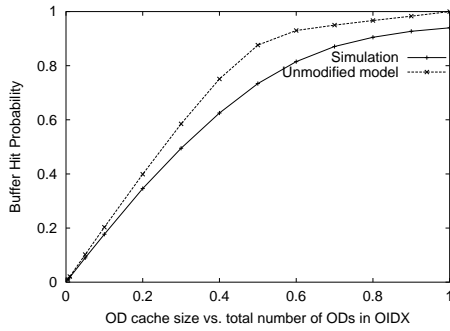
(c) 3P3 access pattern.
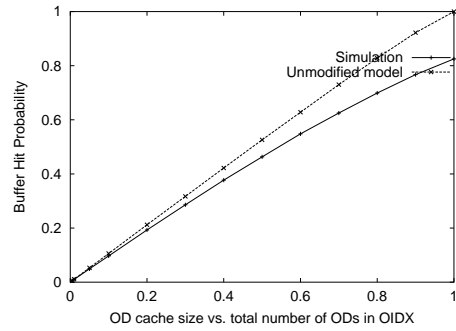
(d) 3P4 access pattern.

Fig. 4. Item cache hit rate with mixed workload.

$F_{dirty}B$). To model the behavior of the dirty items in the buffer we use a simple extension of the BDD mode, which we call the *DCOMP model*. In the DCOMP model we use Equation 2, but exclude the slots in the buffer used by the dirty items when we calculate the buffer hit ratio:

$$P_{bufD}(B, N, \Pi, F_{dirty}) = P_{buf}((1 - F_{dirty})B, N, \Pi) \qquad (3)$$

This equation is valid with relatively low values of F_{dirty}. In Section 6.3.3 we will study how larger values of F_{dirty} affect the accuracy of the model.

6.3.1 Model validation

To model the behavior of the dirty items, we exclude the slots in the item cache used by the dirty items when we calculate the item cache hit ratio, i.e., we calculate the hit ratio for a buffer less than the one we actually have, by using the DCOMP model as described in Section 6.3. Fig. 5 illustrates the much higher accuracy we obtain when using the DCOMP model.
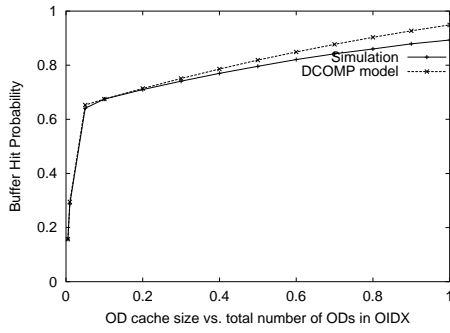
(a) 3P1 access pattern.
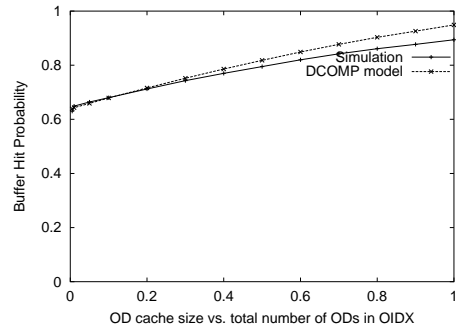
(b) 3P2 access pattern.
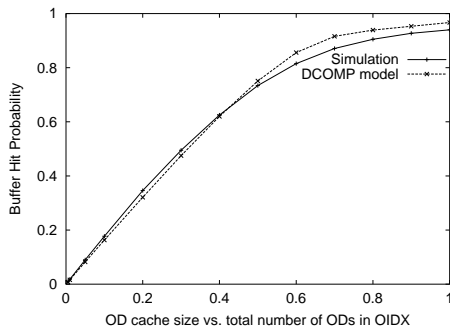
(c) 3P3 access pattern.
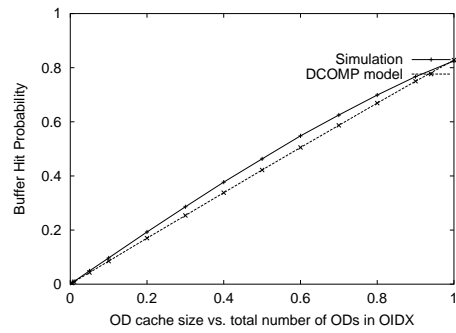
(d) 3P4 access pattern.

Fig. 5. Item cache hit rate with mixed workload, compared with DCOMP model.
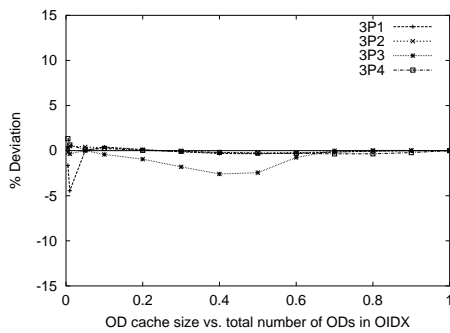
(a) Read-only (*READ_DB*).
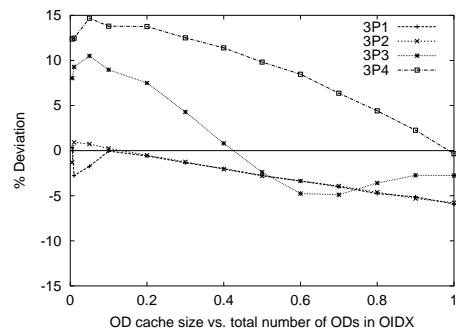
(b) Mixed workload (*FILL_DB*).

Fig. 6. Deviation between simulation and the DCOMP model, using the default parameters.

When studying in detail the accuracy of the models, we will use the deviation between model and simulation as a measure. The deviation is calculated according
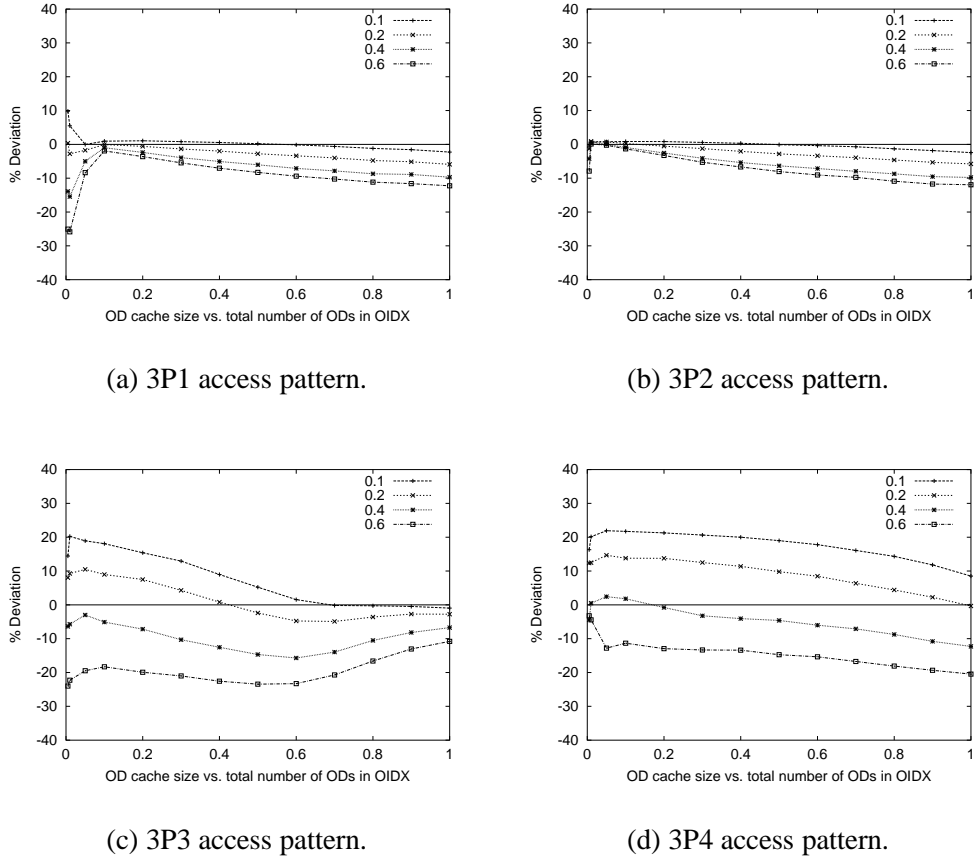
(a) 3P1 access pattern.

(b) 3P2 access pattern.

(c) 3P3 access pattern.

(d) 3P4 access pattern.

Fig. 7. Deviation between simulations and the DCOMP model with different write rates.

to:

$$D = \frac{P_{buf}^{Simulated} - P_{buf}^{Model}}{P_{buf}^{Model}} 100\%$$

Fig. 6 shows the deviation using the default parameters when simulating in the *READ_DB* and *FILL_DB* modes.

6.3.2 The effect of different write and create rates

Fig. 7 illustrates the effect different write rates have on the deviation. For high write rates the deviation is higher than expected. However, this is a result of the way the creation rate is linked to the write rate in our simulation model: the create rate is given as a percentage of the write rate. This becomes very evident when we study the deviation with different create rates, illustrated in Fig. 8.

As mentioned, the buffer models assume a stable condition. A high insert rate (high P_{new}) breaks this assumption, as new items are inserted into the index so fast that the
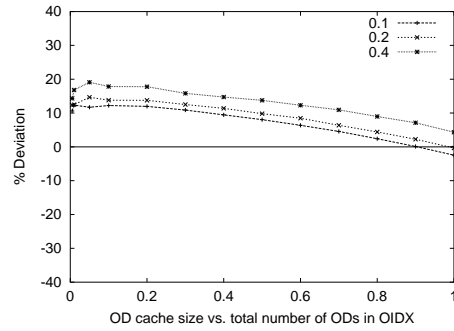
12
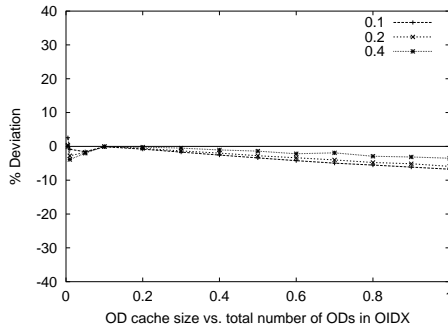
(a) 3P1 access pattern.
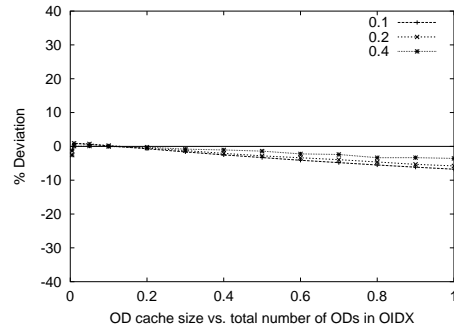
(b) 3P2 access pattern.
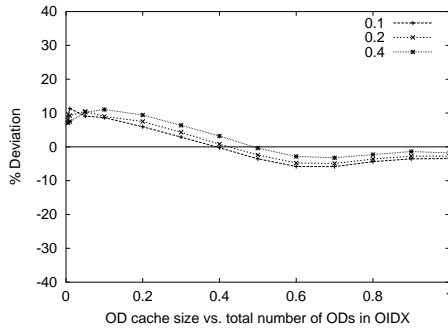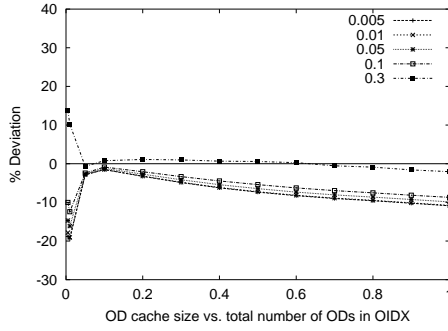
(c) 3P3 access pattern.

(d) 3P4 access pattern.

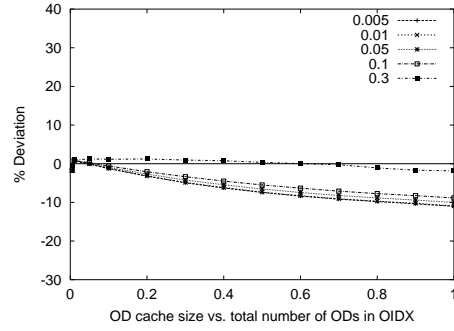Fig. 8. Deviation between simulations and the DCOMP model with different create rates.

item cache does not necessarily stabilize. However, as long as the number of dirty items in the item cache is low (set by the parameter F_{dirty}) and the DCOMP model is used, the high insert rate in itself is not the main contributor to the deviation between the model and the simulations. The deviations are a side effect of the way the simulations are done. The proportional size of the access pattern partitions are constant, which means that the number of items in the hot-spot partition is constantly increasing. In a real system, we can expect the number of items in this partition to be more constant.

### 6.3.3 The effect of different amounts of dirty items in the item cache

We assumed previously that the dirty items in the item cache did not contribute to the hit rate, because many of these items would be "cold" items that would have been discarded if they were not dirty. If we allow a large percentage of the items in the item cache to be dirty, this will result in an underestimation of the hit rate, because a larger amount of these dirty items will actually be read and contribute to the simulated hit rate. This is illustrated on Fig. 9. A fraction of dirty items larger than approximately $F_{dirty} = 0.3$ results in an unacceptable high deviation
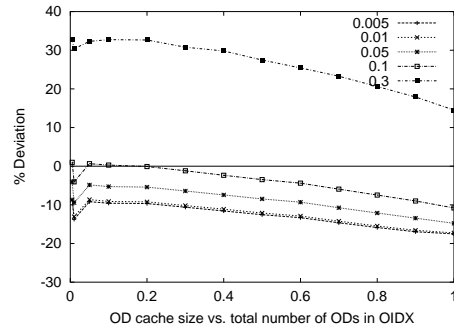
13

(a) 3P1 access pattern.

(b) 3P2 access pattern.
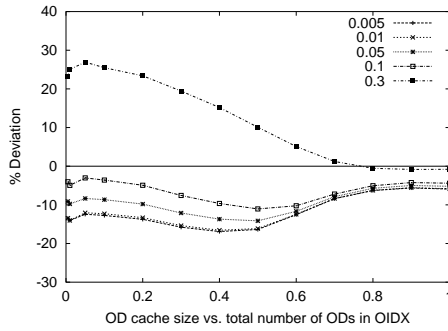
(c) 3P3 access pattern.

(d) 3P4 access pattern.

Fig. 9. Deviation between simulations and the DCOMP model with different amounts of dirty data in the item cache.

between the simulations and the model. However, in practice F_{dirty} will be less than 0.3 because a large number of dirty items (or objects) means a large checkpoint interval (to keep recovery time short, the checkpoint interval should not be too long). In practice, F_{dirty} will also be much less than the default value used in this study, resulting in less deviation.

6.4 The IC-BDD model

We have just seen that the accuracy of the DCOMP model is reduced in the case of a combined high write and create rate. The reason for this, is that the creation of a new items essentially can be viewed as an access to a "non-existing item"; at creation time it does no have an access behavior related to any of the partitions in the access pattern. Thus, another approach is to model the creation of an item as an access to a very large partition, where the probability of accessing an item twice is (close to) zero. In order to model this within our framework, we introduce a new large partition with access probability equal to the creation rate $P_C = P_{write}P_{new}$.
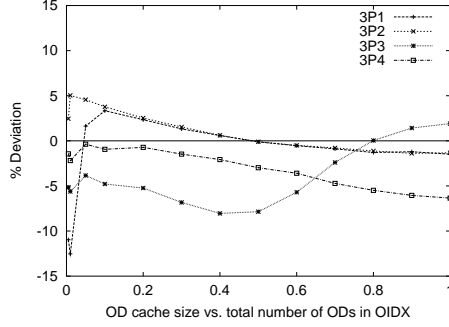
14

Fig. 10. Deviation between simulation and the IC-BDD model, using the default parameters and mixed workload (*FILL_DB*).

Based on the original access pattern, we calculate the hit rate using the BDD model with the following modified access pattern Π_{IC}:

- The new partition has $\alpha = P_{write}P_{new}$ and $\beta = 1 - \frac{1}{S}$, where S is a large scaling factor.
- The original partitions are redefined to be $\alpha_i = \alpha_i(1 - P_{write}P_{new})$ and $\beta_i = \beta_i/S$.

and hit rate calculated by:

$$P_{bufIC}(B, N, \Pi) = P_{buf}(B, NS, \Pi_{IC}) \tag{4}$$

We will now study the accuracy of this model using the same parameters as in the previous section.

Fig. 10 shows the deviation using the default parameters when simulating in the *READ_DB* and *FILL_DB* modes. It shows reduced deviation (i.e., increased accuracy) compared with the DCOMP model.

Fig. 11, 12 and 13 illustrate the deviation between simulations and the IC-BDD model with different parameters. The result is a significantly reduced deviation in the case of high write rates and large amounts of locked data, but slightly increased deviation in the case of high create rates.

7 Multiway-tree index model

Multiway-tree indexes are heavily used, and models of buffer performance for accesses to these are important. However, the assumptions behind the BDD model include independent accesses, while in accesses to hierarchical indexes, accesses are only *partially* independent. In this section, we describe an extension to the BDD model making it suitable for modeling such indexes as well, and provide results from our validations with have been performed using different index sizes, buffer
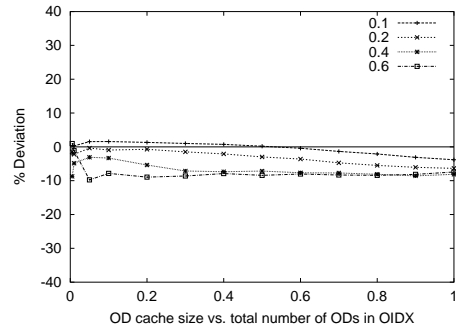
15
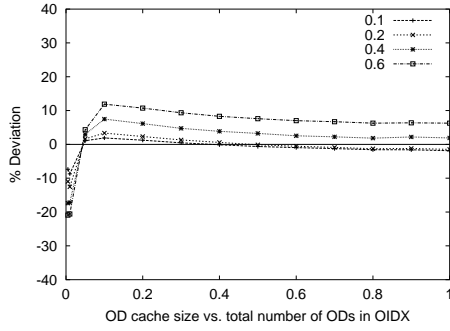
(a) 3P1 access pattern.
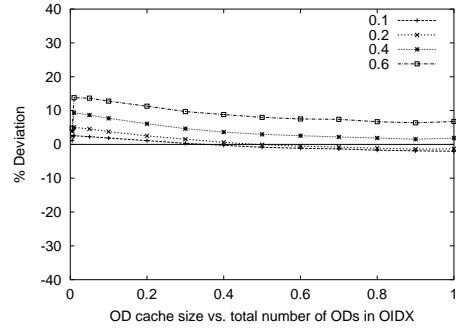
(b) 3P2 access pattern.
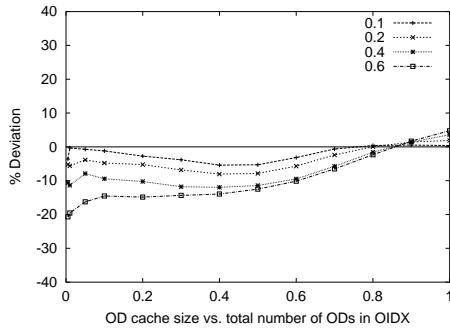
(c) 3P3 access pattern.

(d) 3P4 access pattern.

Fig. 11. Deviation between simulations and the IC-BDD model with different write rates. sizes, index page fanouts, and access patterns.

7.1 Index access model

In our model, we assume low locality in index pages. We also assume accesses to index entries to be random, but skewed (some items are more often accessed than others). We assume it is possible to (logically) partition the range of index entries into partitions, where each partition has a certain size and access probability. This is illustrated in Fig. 14 (note that this is not how it is stored on disk, this is just a model of accesses). The upper part illustrates the physical realization of an index tree. The lower part of the figure shows the index from a logical view. We have indicated with arrays how the entries are distributed over the leaf nodes.

We denote the number of leaf nodes as N_{tree}^0. A leaf node containing index entries contains unrelated entries from different partitions. This means that *the access pattern for the leaf nodes is different from the access pattern to the index entries from a logical view.* We use the initial index entry partitioning (the index entry access

16

(a) 3P1 access pattern.

(b) 3P2 access pattern.

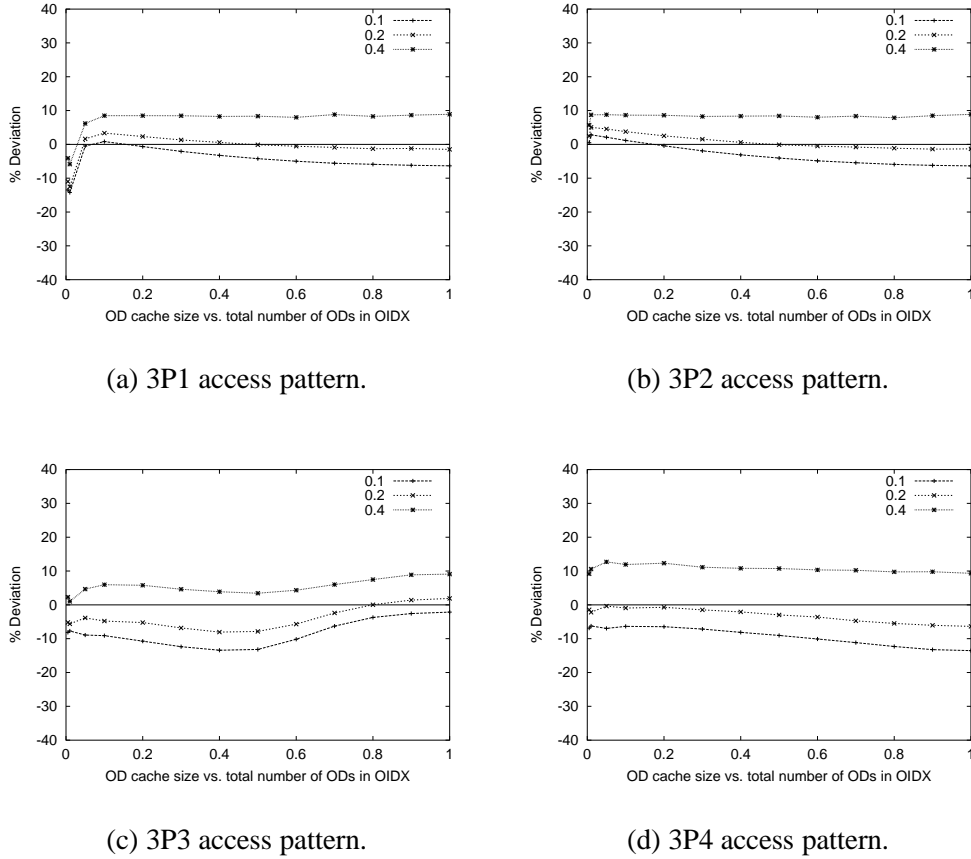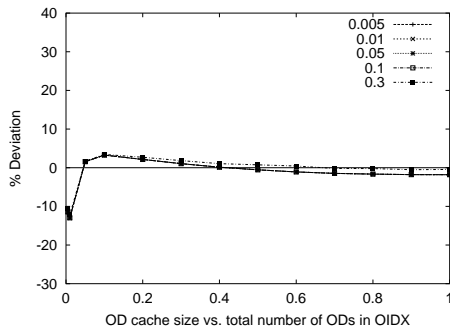(c) 3P3 access pattern.

(d) 3P4 access pattern.

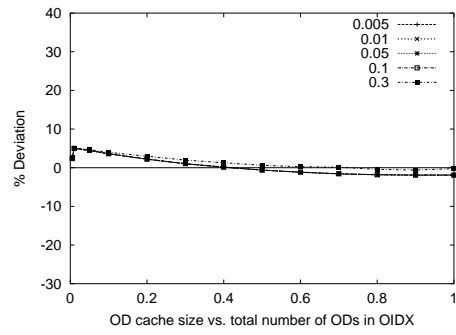Fig. 12. Deviation between simulations and the IC-BDD model with different create rates.

pattern) as basis for deriving the index page partitioning (the index page access pattern) (see Fig. 14). With a totally unclustered index, the index entries from the different partitions are distributed over the leaf nodes with binomial distribution. To simplify this analysis, we make some approximations and assumptions: The second most hot area has a number of entries sufficiently larger than the number of leaf nodes, i.e., $\beta_1^0 > \frac{1}{F}$. This is a reasonable assumption, with a page size of 8 KB, this gives $\beta_1^0 > 0.002$. We assume that the accesses to entries not belonging to the hottest hot spot can be modeled as random and uniform, and it is the hottest hot spot area alone that decides the index page access pattern. Further, we approximate the binomial distribution by assuming the index entries are distributed over the leaf nodes with uniform distribution. Simulation results show that the error is acceptable.

We consider two cases: 1) the number of hot spot entries is smaller than the number of leaf nodes, $\beta_0^0 N < N_{\text{tree}}^0$, and 2) the number of hot spot entries is larger than the number of leaf nodes, $\beta_0^0 N > N_{\text{tree}}^0$.
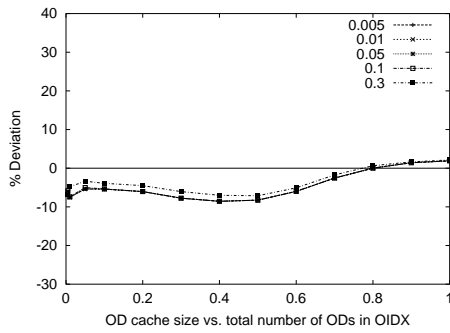
The number of hot spot entries is smaller than the number of leaf nodes: When the number of items in the hot spot area is less than the number of leaf nodes, pages
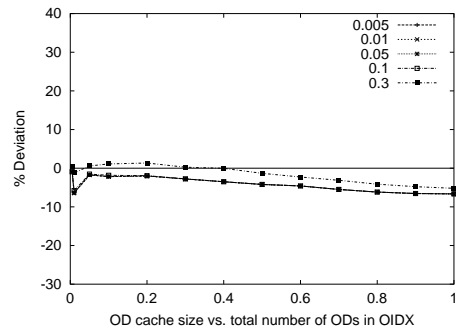
17

(a) 3P1 access pattern.

(b) 3P2 access pattern.

(c) 3P3 access pattern.

(d) 3P4 access pattern.

Fig. 13. Deviation between simulations and the IC-BDD model with different amounts of dirty data in the item cache.
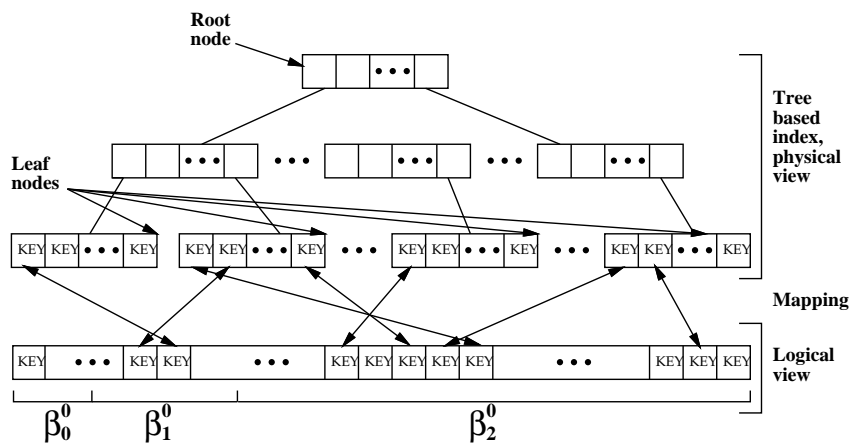

Fig. 14. Multiway-tree index.

belong to one of two partitions $L0$ and $L1$: those that have a hot spot entry, and

those that have not:

$$\beta_{L0} = \beta_0^0 N / N_{\text{tree}}^0 \qquad \alpha_{L0} = \alpha_0^0 + \beta_{L0} \sum_{i=0}^{p-1} \alpha_i^0$$

$$\beta_{L1} = 1 - \beta_{L0} \qquad \alpha_{L1} = 1 - \alpha_{L0}$$

The number of hot spot entries is larger than the number of leaf nodes: In the case where there is one or more hot entry in each page, we will with a uniform distribution over the pages have some of the pages with one hot index entry more than the others. Especially in the case where there are few hot index entries in each page, it is important to capture this fact in the model. We now have two partitions, one with the pages containing that extra index entry, and the other with those pages that do not:

$$\beta_{L0} = (\beta_0^0 N - \lfloor \beta_0^0 N / N_{\text{tree}}^0 \rfloor N_{\text{tree}}^0)/N_{\text{tree}}^0 \qquad \alpha_{L0} = \beta_{L0} N_{\text{tree}}^0 \frac{\lceil \beta_0^0 N / N_{\text{tree}}^0 \rceil}{\beta_0^0 N}$$

$$\beta_{L1} = 1 - \beta_{L0} \qquad \alpha_{L1} = 1 - \alpha_{L0}$$

With an increasing number of hot spot index entries in each page, the access pattern will get more and more uniform.

7.2   General index buffer model

We now present a general index buffer model, where the granularity for access is a page. Even though searches to the leaf page can be considered to be random and independent, nodes accessed during traversal of the tree are *not* independent.

In the generic tree used in this model, the size of one index page is S_P, and the size of one index entry S_{ie}. This give a fanout $F = \lfloor S_P/S_{ie} \rfloor$ and with a database consisting of N objects to be indexed, the number of leaf nodes is $N_{\text{tree}}^0 \approx \frac{N}{U \lfloor S_P/S_{ie} \rfloor}$. This assumes 100% space utilization. In a dynamic tree (for example a typical B-tree) the space utilization U will be lower, and the number of leaf nodes is $N_{\text{tree}}^0 \approx \frac{N}{\lfloor S_P/S_{ie} \rfloor}$. The space utilization does not qualitatively affect the results, so for simplicity we will assume $U = 1.0$ for both internal and leaf pages in the rest of this paper.

Our approach to approximate the buffer hit probability, is based on the obvious observation that *each level* in the tree is accessed with the *same probability* (assuming traversal from root to leaf on every search). Thus, with a tree where the index nodes has a fanout F, the number of levels in the tree is $H = 1 + \lceil \log_F N_{\text{tree}}^0 \rceil$. We initially treat each level in the tree as one partition, thus, initially we have H partitions. Each of these partitions is of size N_{tree}^i, where N_{tree}^i is the number of index pages on level i in the tree. The access probability is $\frac{1}{H}$ for each partition.

To account for hot spots, we further divide the leaf page partition into p' partitions, each with a fraction of β_{Li} of the leaf nodes, and access probability α_{Li} relative to the other leaf page partitions. Thus, in a "global" view, each of these partitions have size $\beta_{Li} N_{\text{tree}}^0$ and access probability $\frac{\alpha_{Li}}{H}$. In total, we have $p = p' + (H - 1)$ partitions. The hot spots at the leaf page level make access to nodes on upper levels non-uniform, but as long as the fanout is sufficiently large, and the hot spot areas are not too narrow, we can treat accesses to nodes on upper levels as uniformly distributed within each level. With the modifications described, a tree of height H, and p' leaf page partitions, the equations for α_i and β_i (access probability and fractional size of each partition) becomes:

$$\alpha_i = \begin{cases} \frac{\alpha_{Li}}{H} & \text{if } i < p' \\ \frac{1}{H} & \text{if } p' \leq i < p \end{cases} \qquad \beta_i = \begin{cases} \frac{\beta_{Li} N_{\text{tree}}^0}{N_{\text{tree}}} & \text{if } i < p' \\ \frac{N_{\text{tree}}^i}{N_{\text{tree}}} & \text{if } p' \leq i < p \end{cases}$$

The total number of nodes in the tree, N_{tree}, can then be calculated as:

$$N_{\text{tree}} = \sum_{i=0}^{H-1} N_{\text{tree}}^i \quad \text{where} \quad N_{tree}^i = \left\lceil \frac{N_{\text{tree}}^{i-1}}{F} \right\rceil$$

We denote the overall buffer probability, by using the BDD buffer hit probability equation with α_i and β_i as defined above as $P_{\text{buf\_ipage}}(B, N_{\text{tree}})$ where $B = \lfloor \frac{M_{\text{ipages}}}{S_P} \rfloor$ is the buffer size, and N_{tree} is the total number of index pages in the tree.

7.3 Validation of the index buffer model

In order to validate our hierarchical index buffer model, we have compared simulation results with the analytical model. The simulations have been performed with different index sizes, buffer sizes, index page fanouts, and access patterns.

The index buffer simulations have been performed using the access patterns summarized in Table 1. Access pattern 3P1 is used as default.

7.3.1 The index buffer simulator

The simulator itself is quite simple. It maintains an LRU chain of index pages currently resident in the buffer. The simulator can operate with either a *traverse* or a *no traverse* strategy.

Using a traverse strategy means that a complete traversal is needed from root to leaf for every leaf node access. In some index implementations, however, the leaf page would be self describing. When doing a search in the tree, we would first check if
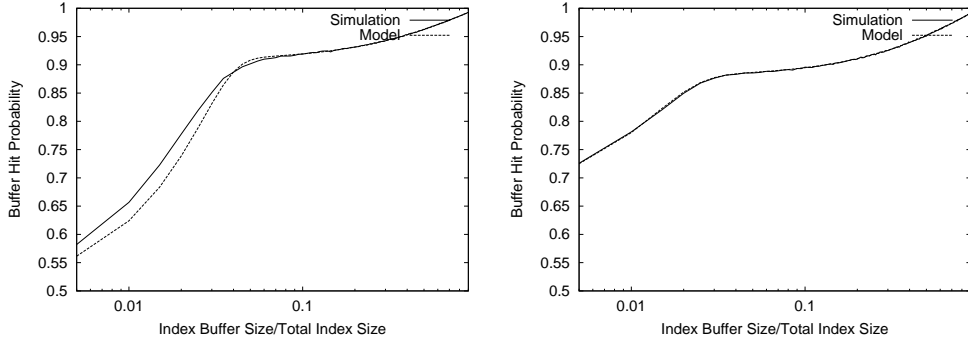
Fig. 15. Overall buffer hit probability for access pattern 3P1 with different buffer sizes, fanout F=64 to the left, and fanout F=2048 to the right.

the relevant leaf page is already in the buffer. Only if the leaf page is not resident, we would need to traverse the tree. We call this a *no traverse strategy*.

The equations in the BDD model assumed a hot buffer. Therefore, we warm up the buffer by doing a large number of requests, before we start measuring the buffer hit probability.

All simulation results in this section are results from simulations with an index tree with 200000 leaf pages. With a leaf page size of 1024 entries, this is sufficient to index approximately 200 million items. We have also performed simulations using larger index trees, but with the same qualitative result.

7.3.2 Results

Fig. 15 shows the overall buffer hit probability for indexes with fanout F=64 and F=2048 (note that the number of leaf pages is the same for both indexes). We see clearly how the buffer hit performance increases close to the point where a new complete index tree level has space in the buffer. After that point, it rises more slowly, until it reaches the point when most of the next level fits completely in the buffer. In the figure, we also see how close the model is to the simulation results (we have used a logarithmic scale for the buffer size axis to emphasize deviations, without logarithmic scale, the curves would be overlapping).

Fig. 16 shows the relative deviation between estimated and simulated buffer hit rate. The deviation is very small for all access patterns.

The results presented until this point have been achieved using a traverse strategy. When simulating a no-traverse strategy, we are satisfied if the leaf page requested is in the buffer. With the traverse strategy, we assume that the leaf pages in the buffer do not contain enough information to be accessed directly. In this case, we traverse the tree from the root node to leaf page, and do a buffer allocation and replacement each time we ask for a node that is not resident in the buffer. An access to a node
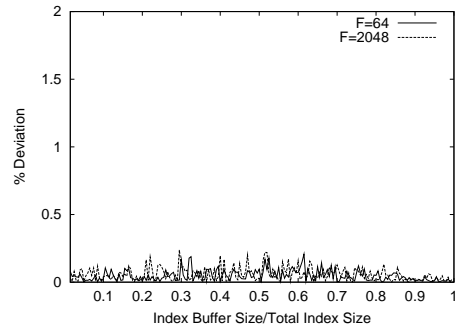
21
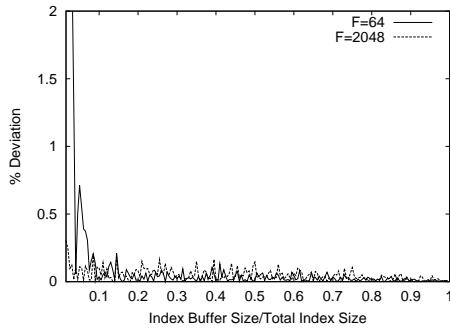
(a) 3P1 access pattern.
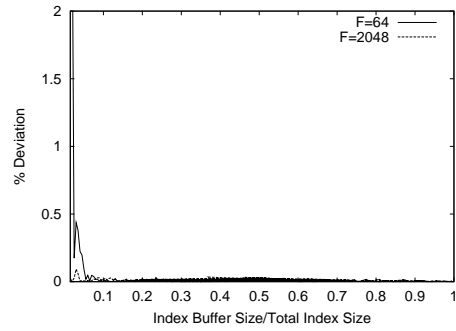
(b) 3P2 access pattern.
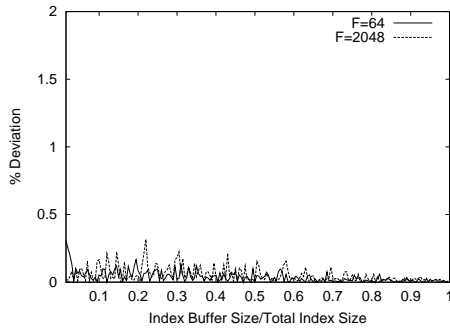
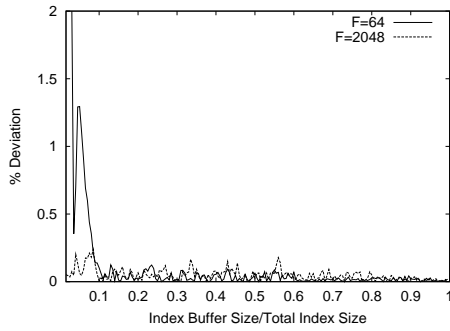(c) 3P3 access pattern.

(d) 3P4 access pattern.

Fig. 16. Relative deviation between buffer hit rate in analytical model and simulations.

resident in buffer, makes the node move to the front of the buffer chain.

To study the impact of a no-traverse strategy on the buffer hit probability, we have compared the simulations done with the no-traverse strategy, and compared the results with the results of the analytical model, which assumed complete traversal. In Fig. 17, we have plotted the relative deviation between estimated and simulated buffer hit rate in the case of a no-traverse strategy. This shows that the deviation is very small in this case as well.

8 Conclusions

Cost models are powerful tools for analyzing algorithms, and important in cost-based query optimization. With increasing amounts of main memory available, it is important to include buffer performance in the models. We have in this paper described a buffer model for fine-granularity caching in buffers with locking, and a buffer model for multiway-tree indexes in the context of unclustered accesses and non-uniform access patterns.
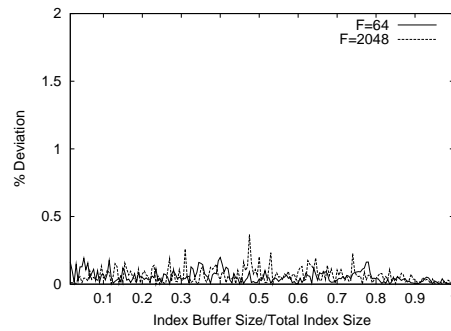
(a) 3P1 access pattern.
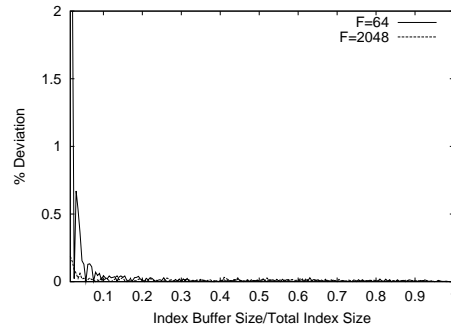
(b) 3P2 access pattern.
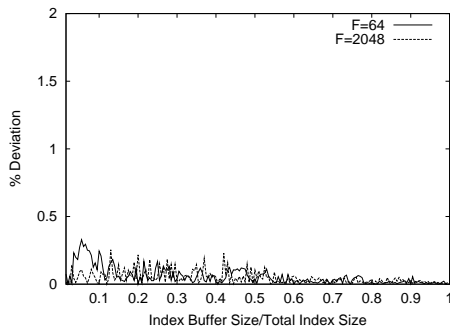
(c) 3P3 access pattern.

(d) 3P4 access pattern.

Fig. 17. Relative deviation between buffer hit rate in analytical model and simulations when the no traverse strategy is used.

In order to validate the analytical models, we have compared them with simulation results. The simulations have been performed with different access patterns and parameter combinations. By comparing simulations with the models, we found that typical deviations between models and simulations are acceptable.

We have already used the models presented in this paper in a number of projects, in order to find bottlenecks and find suitable parameters for resource usage, with improved performance as the result. We will continue using the models for this purpose, and also work on adapting them for the use in other contexts.

References

[1] A. K. Bhide, A. Dan, and D. M. Dias. A simple analysis of the LRU buffer policy and its relationship to buffer warm-up transient. In *Proceedings of the Ninth International Conference on Data Engineering*, 1993.

[2] S. Chaudhuri and G. Weikum. Rethinking database system architecture: Towards a

self-tuning RISC-style database system. In *Proceedings of the 26th VLDB Conference*, 2000.

[3] G. Diehr and A. N. Saharia. Estimating block accesses in database organizations. *IEEE Transactions on Knowledge and Data Engineering*, 6(3), 1994.

[4] W. Effelsberg and T. Haerder. Principles of database buffer management. *ACM Transactions on Database Systems*, 9(4), 1984.

[5] S. Leutenegger and M. Lopez. The effect of buffering on the performance of R-trees. In *Proceedings of the 1998 International Conference on Data Engineering*, 1998.

[6] W. S. Luk. On estimating block accesses in database organizations. *Communications of the ACM*, 26(11), 1983.

[7] L. F. Mackert and G. M. Lohman. Index scans using a finite LRU buffer: A validated I/O model. *ACM Transactions on Database Systems*, 14(3), 1989.

[8] K. Nørvåg. Efficient use of signatures in object-oriented database systems. In *Proceedings of ADBIS'99*, 1999.

[9] K. Nørvåg. The Persistent Cache: Improving OID indexing in temporal object-oriented database systems. In *Proceedings of the 25th VLDB Conference*, 1999.

[10] K. Nørvåg. A comparative study of log-only and in-place update based temporal object database systems. In *Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM'2000)*, 2000.

[11] K. Nørvåg. *Vagabond: The Design and Analysis of a Temporal Object Database Management System*. PhD thesis, Norwegian University of Science and Technology, 2000.

[12] K. Nørvåg and K. Bratbergsengen. An analytical study of object identifier indexing. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications, DEXA'98*, 1998.

[13] K. Nørvåg and K. Bratbergsengen. Optimizing OID indexing cost in temporal object-oriented database systems. In *Proceedings of the 5th International Conference on Foundations of Data Organization, FODO'98*, 1998.

[14] A. N. Swami and K. B. Schiefer. Estimating page fetches for index scans with finite LRU buffers. *VLDB Journal*, 4(4), 1995.

[15] K.-Y. Whang, G. Wiederhold, and D. Sagalowicz. Estimating block accesses in database organizations: A closed noniterative formula. *Communications of the ACM*, 26(11), 1983.

[16] S. B. Yao. Approximating the number of accesses in database organizations. *Communications of the ACM*, 20(4), 1977.