In: Proc. 5th Int. Conf. on Database Systems for Advanced Applications (DASFAA'97), Melbourne, 1997, pp. 1-10.

A Cost Model for Parallel Navigational Access in Complex-Object DBMSs

Michael Gesmann

University of Kaiserslautern
67653 Kaiserslautern, Germany

email: gesmann@informatik.uni-kl.de

Abstract

In contrast to relational query processing, one of the most important extensions of query processing in object-oriented DBMSs (OODBMSs) is navigational access to objects. So far, optimization of this kind of access has been primarily supported by special access path structures or object clustering strategies. Parallelism, although an important topic in large relational systems, has not been explored comprehensively in OODBMSs. In order to evaluate parallel access strategies when collecting a set of objects, we present a cost model to estimate the necessary amount of page I/O. In contrast to others, our cost model covers parallel execution of navigational retrieval operations under a restricted number of available buffer frames. Finally, some measurements validate the cost model.

Keywords: parallelism, query processing, OODBMS

1 Introduction

One of the most important reasons why parallel processing has not been considered for OODBMSs is the lack of set-oriented descriptive query languages. However, such query languages are prerequisites for exploitation of parallelism which has been proven a highly effective tool for providing high performance in RDBMSs. Nowadays, standardization of OQL in ODMG [2, 3] and object-oriented extensions in SQL3 [14] enable compilers and optimizers of OODBMSs implementing these standards to choose among various, possibly parallel, query execution strategies. Therefore, integrating parallel strategies into query processing engines seems to be a promising idea for achieving short query response times in OODBMSs, too. Unfortunately, due to different query processing characteristics [6] and access operations [8, 10], well known parallel query processing techniques of relational systems often cannot be applied in OODBMSs. Especially, navigational access from objects to referenced objects which is a key property in OODBMSs and complex-object DBMSs is not supported in relational systems. In principle, implementation of a 'reference' operation using well-known join algorithms from relational database system (RDBMS) [7] is possible, but explicit usage of references instead can improve performance significantly [16, 11]. However, investigations in this field have mostly considered sequential processing. In this paper, we focus on parallel processing of such operations. In contrast to [12] we do not rely on physical references which directly point to pages containing the required objects. Since pointer-based sort-merge join algorithms turned out to be most efficient in sequential processing [16] our algorithms rely on this kind of algorithms.

Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia, April 1-4, 1997.

In the past, a lot of work has been done for the support of navigational access by appropriate storage structures like path indices [1, 10], storage structures [18], clustering strategies [8], and mapping of object identifiers to physical addresses [5]. However, these approaches aimed only at the optimization of sequential processing. [4] recently described how to parallelize OO7 benchmark's complex traversal operations. Their measurements revealed that parallel traversals can reduce response times substantially. In contrast to our approach, their implementation encapsulates parallel strategies in methods belonging to objects. They did not try to integrate exploitation of parallelism in a more general query processing framework.

In this paper, we investigate parallel navigational access to objects via object identifiers within a general query processing framework. Based on some well-known cost formulas we derive a detailed cost model which can explain and predict performance for complex-object traversals and complex-object construction from elementary objects. The initial problem, how to determine the number of I/Os when accessing multiple sets of referenced objects with buffer size limitations, is very similar to [13] who discuss retrieval of tuples in an RDBMS with a finite buffer using an unclustered index. In contrast to [13], this paper additionally considers simultaneous execution of multiple requests for referenced objects of the same type. On one hand, such parallel fetch operations can lead to increased locality, i.e. repeated references to a page already contained in the buffer. On the other hand at a high system load, processing of concurrent object requests may degenerate to their sequential processing and lead to repeated I/O of the same page. This means, performance of concurrent object requests strongly depends on actual system load and therefore cannot be preplanned at query compilation and optimization time. Hence, decisions about parallelism have to be made at execution time of a query. The presented cost model is simple enough to consider load dependent parameters at run time in order to determine an optimal degree of parallelism.

Most other cost models derived for complex-object construction procedures, eg. [15, 18], neither consider restricted buffer size nor parallel object requests. Only [12] examines parallel strategies. However, their model relies on physical references and on processing in a shared-everything environment with distributed objects. Therefore, we cannot translate results gained there to our environment presented in the next section.

For reasons of simplicity, this paper concentrates on navigational access without exploiting any path indices and clustering strategies. This is motivated by the fact that clustering of complex objects (CO) would imply a huge degree of replication and is therefore not possible for all potential CO structures required by some applications. Furthermore, due to this immanent overhead path indices cannot be supported for every attribute and path in CO structures. Therefore, navigational access often has to be processed without support of these structures.

In the next section, we outline some characteristics of CO processing as well as the terminology used in this paper. In Section 3 and 4, we derive the cost model for parallel access to referenced objects and describe some measurements in order to validate our cost model. The paper concludes with the most important results and an outlook to future work. An appendix summarizes the notations of our cost model.

2 Complex-Object Processing

Processing of dynamically defined application queries in CO-DBMS and OODBMS shows the following entirely different characteristics to query processing in relational DBMS:

- In OODBMS, direct representation of relationships between objects via references allows for (parallel) traversal of complex structures. The corresponding traversal algorithms are sometimes much more efficient than relational join operations embodying the only method to materialize relationships between tuples in relations [11, 16].
- Objects can reference various objects of different types and can themselves be referenced from multiple objects. As a consequence, COs often incorporate highly meshed network structures; they may share components among separate COs [9]. Moreover, these structures may be dynamically definable in query languages.
- In relational DBMS, data distribution across multiple sites (declustering) is one of the most important prerequisites to achieve data parallelism in query processing. Apparently, it is much more difficult in CO-DBMS because every distribution strategy can only support some dedicated CO structures. Even worse, dynamically defined structures may not be preplanned at all. Therefore, in this paper we assume storage schemes which only cluster objects of a single type in a single storage structure. Clustering of heterogenous objects, i.e. they belong to different object types, in a single CO will not be considered here. Moreover, for the same reason we only consider query processing in a shared-everything system architecture.
- Finally, due to the enhanced semantics in OO data models and query languages, OO query processing requires comparatively more CPU resources than relational query processing [9, 17]. These resources are mainly required for retrieval or manipulation of composite data types; obviously, they are also needed by more sophisticated query operators like recursion. Furthermore, due to structural overlap, when COs are assembled from elementary objects or from other COs, these objects may appear in multiple roles in a single CO or in different COs. Therefore, query processing facilities have to separate objects' data and COs' structure information (composition relationships). This means, attribute values of a single object, appearing in various roles in (a single or in multiple) resulting COs should be represented only once without any replication. At the same time, since (data) objects in different roles can comprise multiple references, this structure information has to be represented separately. Due to the increased demands for CPU resources, pipelining in the CO-construction procedure becomes more interesting than in RDBMS.

Navigational access along references from already present objects to referenced objects consists of reading OIDs of references and demanding referenced objects. In the following, execution of such object requests which is one of the most important operations in CO construction is called **object scan**.

In order to allow for object replication in CO-clusters, to ensure OID-uniqueness beyond deletion of an object, and to simplify database reorganisation we only consider logical OIDs (surrogates). Therefore, we require a two phase execution of object scans. In a first phase (called **address scan**) OIDs have to be mapped to physical addresses. Thereafter, in a second step (called **base scan**), the requested objects have to be read from the addressed pages. Both phases are assumed to be executable in parallel for various object scans. Current thread technology from operating systems should allow for such fine grained parallel execution because thread creation and administration there is extremely cheap. The DBMS component which processes address and base scans is called **Access System**.

From a query processing engines point of view, concentrating on referencing objects, we distinguish various object scan granules:

- objectwise invocation, i.e., immediately after having received a requested object (or a set of objects) for the CO construction procedure we immediately invoke object scans for referenced objects. For reasons explained later on, every object scan requests its entire set of referenced objects. This means, even in cases of very extensive references we do not cut such a set of OIDs into multiple independent requests.
- COwise invocation, i.e., we do not invoke any object scan before having received all referencing objects within a single or a set of COs, and finally,
- querywise invocation, i.e., we invoke object scans for referenced objects after having received all referencing objects of all COs.

Objectwise invocations induce a lot of object scans each usually demanding a comparatively small number of objects. This obviously leads to some administration overhead but at the same time, it allows for parallel construction of a single as well as parallel processing of multiple COs. On the other hand, querywise requests provoke only a small number of requests each asking for a lot of objects. In contrast to the previous procedure, this one does not exploit that much parallelism. Finally, COwise invocations implement a middle course. In the following section we will develop a cost model which can determine the most efficient type of invocations.

3 Cost Model

In order to determine an optimal degree of parallelism for a CO construction procedure as well as a scan execution, we need a cost model which can efficiently predict execution costs. This section presents such a cost model which can assess different granules for object scans, data characteristics, and system parameters like buffer size or system load. Our cost model only considers page I/O because CPU resources required for scan operations are comparatively small. Even in cases with many object scans, dividing the set of objects to be read into multiple subsets is expected to induce a comparatively small amount of CPU overhead. But, unfortunately, it can cause a substantial number of additional page requests for the following reason:

Given k objects (or address information for OIDs) evenly distributed among m pages. Then, we can estimate the number of pages storing n randomly selected objects ($n \le k$) by Yao's formula [19] as [1]

$$yao(k, m, n) = m \times \left(1 - \prod_{i=1}^{n} \frac{k \times d - i + 1}{k - i + 1}\right)$$

where d=(1-1/m). Fig. 1 shows the fraction of pages which have to be read, if 100*n/k percent of existing objects were selected with k/m = 1, 10, 100 or 1,000 objects per page. In this example, we assume to have a constant number of m=10,000 pages; for m>100, the illustrated curves differ only marginally. If all objects are read within a single scan, no page has to be processed more than once. In this case, yao() corresponds to the number of necessary page requests to fetch the required n objects. However, when invoking multiple object scans to retrieve n objects this procedure usually requires additional page requests because yao() is not a linear function in general. In Fig. 1 we see, with k/m=1, i.e. a single object per page, this function behaves in a linear way. Then, retrieving 50% of the objects in a single scan requires the same amount of page requests as in the case with 5 scans each fetching 10% of the objects. But, with k/m=10, the function clearly shows a non-linear behavior. In this situation, when accessing 50% of the objects the Access System has to read almost all pages. When dividing this scan into 5 independent scans each fetching 10% of the objects the Access System has to request 5 times 65% = 325% of pages.

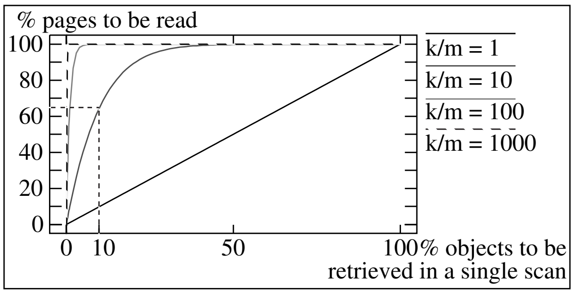

Figure 1: Yao-Function

Of course, in the latter case due to page buffering in the DBMS buffer, not every page request (in the following we call it logical I/O) necessarily causes a physical I/O from a disk to the buffer. Therefore, a parallel scan's performance depends on the number of available buffer frames. For example, if the buffer can keep all pages, even in the parallel case only a minimum number of physical I/O operations is needed to get all relevant pages. On the other hand, if there is only a single page available in the buffer almost every logical I/O induces a physical I/O (except in cases where subsequent requests refer to the same page).

Consequently, a cost model has to estimate the number of physical I/Os instead of the number of logical I/O operations for two reasons. First, page requests which do not require physical I/O are almost for free. Second, parallel navigational access can cause a lot of repeated logical I/O to the same page which do not require additional physical I/O due to reference locality. This means, when decomposing a scan into parallel requests a cost model has to comprise actual workload parameters, e.g. the number of

1. An approximation of this formula by some linear equations allows for an efficient evaluation [18].

available buffer frames.

In the following, we present some basic cost formulas which predict the number of physical I/Os induced by a couple of scans each requesting multiple objects (addresses) of a single object type. These cost formulas consider restricted buffer size similar to [13]. Thereafter, we extend this model in order to cover parallel scan processing of a single object scan as well as parallel processing of multiple scans on a single object type. Finally, we shortly summarize and discuss some important results.

3.1 Estimation of necessary I/Os

When retrieving n randomly selected objects from *no_of_objects* objects which are evenly distributed across *no_of_pages* pages Yao's formula yields

$$N'_{PR}(n) = yao(no\_of\_objects, no\_of\_pages, n)$$

pages containing the requested objects. Fetching n objects in t scans each covering n_i objects leads to

$$N''_{PR}(n) = \sum_{i=1}^{t} N'_{PR}(n_i) \qquad \text{with } n = \sum_{i=1}^{t} n_i$$

logical I/Os. When assuming even distribution of objects to t scans, this equation simplifies to [2]

$$N_{PR}(n, t) = t \cdot N'_{PR}(n/t)$$

Unfortunately, as already mentioned, $N_{PR}(\ )$ does not calculate the number of physical I/Os because it comprises multiple requests to the same page. To overcome this deficiency, we examine this situation in more detail. Therefore, in the following we assume to have a constant number of available buffer frames (*buffersize*) for the entire scan operation.

Obviously, if $buffersize \geq N'_{PR}(n)$ every page remains in the buffer until the end of the scans. Therefore, the number of physical I/Os $N_{IO}(\ )$ is equal to $N'_{PR}(\ )$. On the other hand, if $buffersize < N'_{PR}(n)$ at request time, pages may or may not be contained in the buffer. In this situation, we have at least $N'_{PR}(\ )$ physical I/Os to read all pages containing requested objects. Once our buffer is filled with *buffersize* pages, the probability that a requested page is not in the buffer is $(1 - buffersize/N'_{PR}(\ ))$. According to [13] the number of physical I/Os for t sequential requests approximately is

$$N_{IO}(n, t, buffersize) =$$
$$N'_{PR}(n) + (N_{PR}(n, t) - N'_{PR}(n)) \cdot \left(1 - \frac{buffersize}{N'_{PR}(n)}\right)$$

Fig. 2 illustrates the effect of $N_{IO}(\ )$ with respect to our example from the beginning of this section. For two different numbers of objects per page (k/m=10, 100), it shows the amount of physical I/O when fetching 1% and 10% of the objects by $t \in \{1, 5, 10\}$ scans.

2. Note, even distribution is only assumed to simplify calculation of $N_{PR}()$. In principle, more sophisticated distributions could also be exploited but they complicate estimation of $N_{PR}()$.

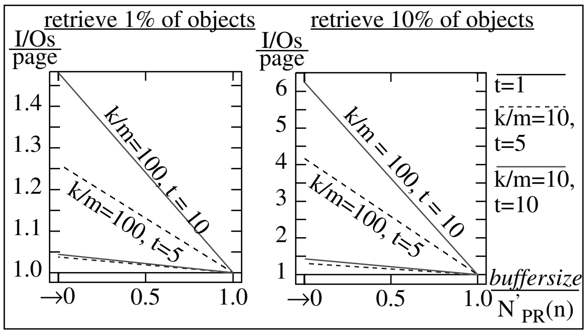

Fig. 2: physical page I/Os depending on buffersize, numbers of objects per page, and number of scans

The x-axis presents the ratio between *buffersize* and number of relevant pages ($N'_{PR}(\ )$). Of course, we assume to have at least one page in the buffer. The y-axis illustrates the ratio between physical page I/O ($N_{IO}(\ )$) and number of different pages ($N'_{PR}(\ )$). This ratio corresponds to the average number of physical I/Os per page. Fig. 2 demonstrates an escalating number of physical I/Os with an increasing number of scans as well as number of objects per page. Moreover, it indicates that overhead increases the more objects have to be read from the database.

Unfortunately, as illustrated by some examples in Fig. 3, $N_{IO}(\ )$ is only an approximation for sequential

Assume we have to access 10 pages (numbered from 1 to 10) in 1 scan reading all pages, 2 scans each reading 8 pages, and 5 scans each reading 6 pages. Page numbers to be read in every scan are randomly distributed accross the scans, and they are sorted in each scan according to their page numbers. Finally, all pages are accessed with almost the same frequency.

Let us compare our estimated number of physical I/Os with the effective number of physical I/Os for the following three scheduling strategies:

a) sequential processing of all scans
b) 1st request of every scan then 2nd request of every scan and so on, starting with the first scan
c) as in b) but starting with the last scan

With buffer size of 2, 4, 6, 8, 10 pages, respectively, $N_{IO}()$ yields the following number of physical I/Os:

No.	scans with page requests:	strategy:	buffer sizes: 2	4	6	8	10
1	1 2 3 4 5 6 7 8 9 10	a, b, c) *NIO()*	10 *10*	10 *10*	10 *10*	10 *10*	10 *10*
2	1 2 4 5 6 8 9 10 2 3 4 6 7 8 9 10	a) b) c) *NIO()*	16 10 12 *15.8*	16 10 10 *13.6*	16 10 10 *12.4*	11 10 10 *11.2*	10 10 10 *10*
3	1 2 3 5 7 9 2 4 7 8 9 10 1 3 4 5 6 8 3 5 6 7 9 10 1 2 4 6 8 10	a) b) c) *NIO()*	30 19 20 *26*	27 16 14 *22*	23 10 10 *18*	20 10 10 *14*	10 10 10 *10*
4	1 2 3 5 7 9 2 4 7 8 9 10 1 3 4 5 6 8 3 5 6 7 9 10 1 2 4 6 8 10	a) b) c) *NIO()*	30 25 21 *26*	27 21 21 *22*	23 15 14 *18*	20 13 10 *14*	10 10 10 *10*

Figure 3: Example for inaccuracy of $N_{IO}()$

scan processing because it assumes that $N_{PR}(\ )$ accesses are evenly distributed to $N'_{PR}(\ )$ pages. In reality, however, this is not the case because scans access pages only once and within every scan logical page requests are sorted according to the page numbers of affected pages. Therefore, repeated access to the same page is less probable than expected in the formula. Consequently, when processing t scans in a sequence the number of physical I/Os exceeds the estimated value by $N_{IO}(\ )$. On the other hand, when processing parallel requests, because of sorted access to pages repeated requests to a single page within a short period of time become more probable. Therefore, the number of physical I/Os falls under the estimated number of physical I/Os.

As expected, a comparison of the estimated values for physical page I/Os with observed values first of all shows, that $N_{IO}(\ )$ underestimates the number of physical I/Os in the case of sequential scan processing. On the other hand, $N_{IO}(\ )$ overestimates the number of physical I/Os in almost all cases with parallel scan execution; in the worst case (schedule no. 3, strategies b) and c)) the deviation is 80%. Therefore, in the following, we will define a more accurate cost function which at the same time is easy to calculate. Of course, the derived cost model can again be only an approximation because the amount of physical I/O strongly depends on time dependent scheduling decisions. For example, strategies b) and c) for schedules number 2 and 3 show different numbers of physical page I/Os although both behave equivalently from a logical point of view.

The main reason for inaccurate results of $N_{IO}(\ )$ originates from the assumption of independent and random access to pages. These premises are reflected in the summand $N_{PR}(n, t)$ which inherently expresses that all page requests are independent from each other. This assumption, however, does not hold any more for concurrent, set-oriented, and ordered page requests. As schedule 2 in Fig. 3 exemplifies, the more scans access the same or similar sets of pages the more locality, i.e. higher probability for repeated references to the same pages, occurs. Therefore, the number of physical I/Os decreases. In order to model this observation we consider sets of parallel scans as a single unit of processing. In the following, such a set of parallel scans is called **SPS**. We estimate the number of relevant page accesses within each SPS and add these values for all sets in order to get the total number of relevant page requests. This means, from $N_{PR}(n, t)$ we eliminate those requests in each set which probably do not require physical page I/Os.

Now, let us assume an SPS consists of p parallel scans (p≤t) each requesting $N'_{PR}(n/t)$ pages. We dnote a set of concurrent page requests, i.e. all page requests on the same position within all of the p scans, as a **run**. For example, in Schedule 3 of Fig. 3 (1, 2, 1, 3, 1) builds the first run, (9, 10, 8, 10, 10) forms the sixth run. Since all page requests in a single scan are executed in a sequel, parallel processing of the p scans is equivalent to sequential processing of $N'_{PR}(n/t)$ runs with parallel execution of every individual run. Because of sorted access to pages in every scan this behavior leads to an increasing locality if various scans access the same pages (see Schedules 2 and 3 in Fig. 3). The degree of locality primarily depends on the number of page requests in all scans in relation to the number of different pages referenced by all scans. This can be expressed as

$$avg\_paf(n, p, t) = \frac{p \cdot N'_{PR}(n/t)}{N'_{PR}(p \cdot n/t)} \in [1, p]$$

Normalizing this formula to values between 0 and 1 yields

$$avg\_paf_n(n, p, t) = \begin{cases} \dfrac{avg\_paf(n, p, t) - 1}{p - 1} & \text{for } p>1 \\ 0 & \text{for } p=1 \end{cases}$$

So far, all parallel scans are assumed to be invoked and executed concurrently. This, of course, usually does not hold; e.g. when requesting objects for various COs in independent scans. Then, scan executions will overlap but they will not run in parallel from scan invocation to final result transmission. As Schedule 4 in Fig. 3 shows, this asynchronous executionalso influences the number of physical I/Os in an SPS. Therefore, we compute the effective degree of parallelism as

$$eff\_par(n, p, t) = \frac{p \cdot N'_{PR}(n/t)}{number\_of\_runs} \in [p, 1]$$

Again, normalizing this to values between 0 and 1 leads to

$$eff\_par_n(n, p, t) = \begin{cases} \dfrac{eff\_par(n, p, t) - 1}{p - 1} & \text{for } p>1 \\ 0 & \text{for } p=1 \end{cases}$$

In our simplified cost model, we now define the degree of locality as

$$locality(n, p, t) = avg\_paf_n(n, p, t) \cdot eff\_par_n(n, p, t)$$

This product expresses the extent of overlapping references to pages from various concurrent scans each requesting $N'_{PR}(n/t)$ pages. If both values for avg_paf_n and eff_par_n are equal to 1 we achieve highest locality, i.e. in a single run all scans always access the same page. If avg_paf_n is equal to 0 then all pages are requested by a single scan only. Finally, a value of 0 for eff_par_n corresponds to sequential processing of the p scans. Summarizing, *locality(n, p, t)* calculates a normalized factor which covers overlapping page requests in an SPS of p scans each requesting n/t objects. Using this factor we can calculate the number of different page requests in a single run as

$$N^{run}_{PR}(n, p, t) = \underbrace{p}_{\text{page requests in a single run}} - \underbrace{(locality(n, p, t) \cdot (p - 1))}_{\text{reductions caused by overlapping requests}}$$

According to the computation of $N_{IO}(\ )$, the number of relevant page requests in a single run which can cause physical page I/O is

$$N^{run}_{PRrel}(n, p, t, bf) =$$

$$\begin{cases} N^{run}_{PR}(n, p, t) + (p - N^{run}_{PR}(n, p, t))\left(1 - \dfrac{buffersize}{N^{run}_{PR}(n, p, t)}\right) & \\ \qquad \text{for } (buffersize \le N^{run}_{PR}(n, p, t)) & \\ N^{run}_{PR}(n, p, t) \qquad \text{for } (buffersize > N^{run}_{PR}(n, p, t)) & \end{cases}$$

and the number of relevant page references of t scans executed in t/p SPSs consisting of p scans is

$$\frac{t}{p} \cdot (N'_{PR}(n/t) \cdot N^{run}_{PRrel}(n, p, t, buffersize))$$

Replacing $N_{PR}(n, t)$ in $N_{IO}(\ )$ by this product yields the following number of expected physical I/Os when processing t scans in SPS of p scans

$$N^{par}_{IO}(n, p, t, buffersize) =$$

$$N'_{PR}(n) + \left(1 - \frac{buffersize}{N'_{PR}(n)}\right) \cdot$$

$$\left(\frac{t}{p} \cdot (N'_{PR}(n/t) \cdot N^{run}_{PRrel}(n, p, t, buffersize)) - N'_{PR}(n)\right)$$

In the case of p=1, $N^{par}_{IO}(\ )$ yields the same results as $N_{IO}(\ )$ because locality is equal to 0 and therefore, $N^{run}_{IO}(\ ) = N^{run}_{PRrel}(\ ) = 1$ and $t \cdot N'_{PR}(n/t) = N(n, t)$. However, comparing $N^{par}_{IO}(\ )$ to our previous example yields the results shown in Fig. 4 for the parallel schedules. These results correlate much more with the observed values than those of $N_{IO}()$.

No.	scans with page requests:	strategy:	buffer sizes: 2	4	6	8	10
2	1 2 4 5 6 8 9 10 2 3 4 6 7 8 9 10	b)	10	10	10	10	10
		c)	12	10	10	10	10
		$N_{IO}()$	*15.8*	*13.6*	*12.4*	*11.2*	*10*
		$N^{par}_{IO}()$	***10.96***	***10.72***	***10.48***	***10.24***	***10***
3	1 2 3 5 7 9 2 4 7 8 9 10 1 3 4 5 6 8 3 5 6 7 9 10 1 2 4 6 8 10	b)	19	16	10	10	10
		c)	20	14	10	10	10
		$N_{IO}()$	*26*	*22*	*18*	*14*	*10*
		$N^{par}_{IO}()$	***19.6***	***14.8***	***13.2***	***11.6***	***10***
4	1 2 3 5 7 9 2 4 7 8 9 10 1 3 4 5 6 8 3 5 6 7 9 10 1 2 4 6 8 10	b)	25	21	15	13	10
		c)	21	21	14	10	10
		$N_{IO}()$	*26*	*22*	*18*	*14*	*10*
		$N^{par}_{IO}()$	***23.6***	***18.4***	***15.6***	***12.8***	***10***

Figure 4: Comparison of $N_{IO}()$ with $N^{par}_{IO}()$ using our example in Figure 3

Finally, let us compare behavior of $N^{par}_{IO}(\ )$ to $N_{IO}(\ )$. Fig. 5 shows I/O behavior for the same environment as described for Fig. 2 (10, 100 objects/page, retrieval of 1%, 10% of objects). Now, for all experiments buffersize is set to $0{,}5 \cdot N'(n)$. The x-axis shows the number of scans which are executed to get the required objects. The y-axis shows the ratio between necessary physical I/Os per page with multiple scans and the necessary amount of I/O per page with a single scan. Furthermore, Fig. 5 distinguishes two degrees of effective parallelism p (1, t). With p=1 all scans are executed sequentially, with p=t they are all processed in parallel. As expected, the figure shows that an increasing number of scans always induces additional physical page I/Os. A comparison of the two presented diagrams shows that the more objects are to be retrieved, the more this extra overhead increases. Furthermore, in both diagrams comparing the curves for 10 objects per pages with the curves for 100 objects per page, we can observe that an increasing number of objects per page always leads to an increasing number of I/Os. Finally, especially for small numbers of scans $N^{par}_{IO}(\ )$ yields considerably smaller numbers of I/Os than $N_{IO}(\ )$ which corresponds to $N^{par}_{IO}(n, 1, t, buffersize)$. This means, if there are enough resources available to process parallel scans, locality increases drastically due to parallel and sorted pages accesses.

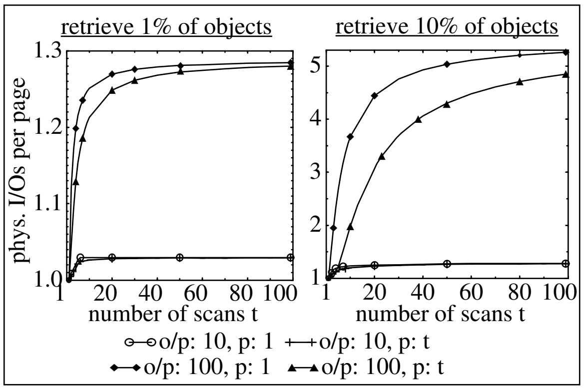

Fig. 5: Page I/Os with variable degrees of parallelism

3.2 A scan's response time

The cost model developed so far estimates the number of physical I/Os. It turned out that extra I/Os inherently restrict the achievable speedup. Consequently, response times can increase if the I/O tasks cannot be performed in parallel. On the other hand, the more CPU and I/O resources are available, the more work can be done in parallel. Hence, response times for scan operations will decrease. This means, the primary optimization criterion response time is not proportional to the number of physical I/Os. Therefore, the cost model explicitly has to estimate response times which in a simplified way can be done by the following formula (bf is an abbreviation for buffersize):

$$rt(n, p, t, bf) = avg\_seqIO\_SPS(n, p, t, bf) \cdot t/p$$

with

$$avg\_seqIO\_SPS(n, p, t, bf) = \frac{avg\_IO\_SPS(n, p, t, bf)}{p + 1 - avg\_paf(n, p, t)}$$

$$avg\_IO\_SPS(n, p, t, bf) = avg\_IO\_task(n, p, t, bf) \cdot p$$

$$avg\_IO\_task(n, p, t, bf) = \frac{N^{par}_{IO}(n, p, t, bf)}{t}$$

When again applying the parameters of our initial example, Fig. 6 shows response time behavior for parallel scan operations obtained from the simplified estimation. With p=1, response time of course is proportional to the number of physical I/O. Furthermore, we see how parallel scan execution can decrease response times. Summarizing, the response time behavior of a scan operation strongly depends on its decomposition, the number of objects per page, the number of requested objects, the number of available buffer frames, as well as on the available CPU and I/O resources which determine the effective degree of parallelism. However, the more objects have to be retrieved and the more objects are contained in a page, the more resources are indispensable to benefit from parallel processing. Otherwise, less available resources and more objects within a page lead to substantially increased response times because of the explained degradation to sequential processing and the induced I/O overhead.

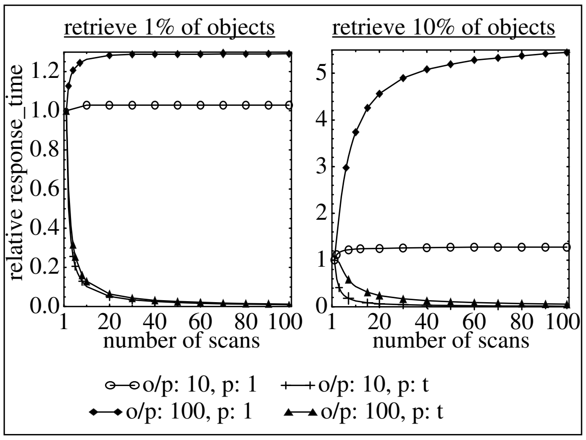

Figure 6: Scan response times with variable degrees of parallelism

3.3 Some observations

Now, let us point out some important results from our theoretical analysis considering parallel navigational access to objects:

- Despite the observation that navigational access may induce repeated access to a page (cf. Fig. 5) parallel processing of scan operations may reduce response times (cf. Fig. 6) because the amount of extra physical I/Os does not increase to the same extent as the number of scans grows. Response times of course only decrease if enough I/O resources are available to process page requests in parallel.
- Especially, in the case where only a few large COs composed of a lot of objects are retrieved from the database, navigational algorithms request comparatively large sets of objects in relatively few scans. Then, parallel processing of these requests is very efficient because they do not cause much overhead. The reason for this behavior originates from yao() which shows a linear behavior when retrieving a small set of objects. In this situation, almost every object request induces a physical I/O. Hence, processing such requests in individual scans does not provoke additional I/Os.
- Benefit from parallel processing strongly depends on local scan parameters like number of requested objects and number of objects per page as well as run-time parameters like currently usable CPU and I/O resources. As a consequence, parallelism in CO construction cannot be preplanned in advance. This first of all means, definition of scan descriptions for a query execution plan (QEP) determining whether to execute individual phases of a set-oriented scan operation in a pipeline or in a sequence is not possible before query execution time. Moreover, decisions about the CO construction procedure cannot be made in advance, too, that is at query compilation time. Apparently, objectwise requests in CO construction cause very much overhead. On the other hand, decisions about COwise or querywise object requests which cause different access patterns for object requests are not possible a priori at query compilation time. Therefore, the obtained results have to be considered for dynamic query optimization within the CO construction procedure.
- Relying on our execution model and on our cost model, it obviously does not make sense to divide a given set of OIDs or addresses into multiple sets in order to process them independently in parallel. If address and base scans process multiple pages these scans can be decomposed into parallel subscans each processing dedicated sets of pages. However, the presented cost model does not capture this feature because it does not consider increased system load by additional scans and it does not cover that these subscans never operate on the same pages. In a future step the cost model has to be extended into this direction.
- Finally, our results show that pipelining in the CO-construction procedure can be a good choice if there are enough resources (available buffer frames). This observation can be generalized for cases where access path structures are additionally processed at the beginning of a scan operation. Then, scan processing in multiple requests does not induce much more physical I/O. Furthermore, it allows for an optimal resource utilization because multiple Access System components may run in parallel. Without pipelining, resource utilization may decrease, because parallel processing of access path structures and address information does not allow for substantial parallelism because these operations affect only a comparatively small number of pages. On the other hand, with less available resources response times deteriorate with an increasing number of parallel scans due to additional physical I/Os. In this situation, of course, pipelining should be avoided.

4 Measurements

After having derived an I/O based cost model which can estimate the amount of necessary I/O for parallel navigational access to referenced objects in a restricted buffer, we now describe some measurements with our CO-DBMS PRIMA [9, 16] in order to validate the model.

4.1 Environment

The measurements use a SEQUENT Symmetry (S27) shared-memory multi-processor system with 8 processors (6MHz Intel 80386) running DYNIX V3.0.18. Every processor has a cache of 64 kbytes. Processors and shared memory are connected by a 64-bit system bus with a channel bandwidth of 80 megabytes per second. Data has been allocated on a single disk, accessible by each of the 8 processors. Since the measurements aim at a verification of our cost model, i.e. the number of physical I/Os rather than response times, the number of disks has no effects on the results. To guarantee parallel processing of requested scans all measurements are executed in exclusive mode in a conventional environment, i.e. neither the hardware nor the operating system have been modified. The system has been set up with special measurement tools to observe response times as well as I/O behavior. The PRIMA system is running in 7 processes, which all can execute every request from all applications. The 8th processor is reserved for operating system processes and a client process which creates the workload for the measurements.

4.2 Workload Characteristics

Since the derived cost model considers parallel access to objects of a single object type within a more sophisticated CO construction procedure, we also consider only a single object type in our measurements. This object type has 235,000 instances in the database. A client generates the workload for our measurement when requesting sets of objects. It requests 2% and 5% of the objects randomly selected by their OIDs. We guarantee the usage of address scans instead of complete object type scans. Varying degrees of scan sizes (i.e. the number of objects retrieved within a single object scan) and scan parallelism (i.e. the number of simultaneously invoked object scans) allow for the investigation of system behavior under different workload decompositions and degrees of parallelism. To simulate the behavior of smaller databases, we repeated our measurements on a subset of our database consisting of 23,500 instances (small DB). Moreover, to explore different workload situations, we exploit two different buffer sizes (500KB, 2.5MB) which correspond to about 1% and 5% of the object type's storage utilization. In order to compare system behavior for various numbers of entries per page, we implement two databases with different page sizes. The first database consists of 1KB-pages (1K-DB), the second database of 8KB-pages (8K-DB). The following table summarizes the essential characteristics of our databases:

	1K- DB		8K- DB	
	objects	addresses	objects	addresses
no_of_pages	42392	7847	4912	837
entries/page	6	30	48	282

We will compare the measured amount of physical I/Os with estimated values derived from our cost model in Section 3. The actual formulas are a little bit more sophisticated than the already presented ones because the derived cost model considers only sequences of parallel scans on objects of a single type and address information, respectively. But, when processing a scan operation in our environment, we process address and object information in an interleaved way. Therefore, the LRU replacement policy of the buffer manager has to be considered in more detail than presented here. Due to space limitations, we cannot further explain the refined formulas. Nevertheless, the basic cost estimation for parallel address and base scans is obtained by our cost model.

4.3 Results

Initially, (not illustrated here) we validated the basic estimation of $N_{IO}(\ )$ by randomly and sequentially selecting objects from our databases using a finite buffer. For various numbers of retrieved objects, for various buffer sizes and for all types of databases (1K-DB, 8K-DB) the measured numbers of physical I/Os always differed at most by about 3% from estimated values. This observation justifies application of $N_{IO}(\ )$ in our cost model and allows for further investigations which additionally consider parallel scan execution.

With two different buffer sizes (500KB (1%), 2.5MB (5%)), we have requested 4700 (2% of objects in the database) and 11750 (5%) objects, respectively. In order to investigate system behavior with different scan sizes we have varied the number of object requests in a single scan (1, 100, 1000). Furthermore, variable degrees of parallelism, i.e. concurrently running requests, allow for exploration of parallel scan processing. Fig. 7 shows some results in terms of number of physical I/Os.

In all diagrams, the bottom line shows the estimated number of different pages affected by the presented measurements, i.e. the minimum number of physical page I/Os. When correlating the curves for the estimated values with the curves for measured amount of physical I/O we see that both curves show almost the same behavior in all cases. Consequently, our measurements verify the observations described in Section 3.3, which until now had only been obtained from estimations. Moreover, the results illustrate that the derived cost model yields more realistic results than the much simpler $N_{IO}(\ )$ formula derived in [13]. For comparison, the values obtained from $N_{IO}(\ )$ are shown as the first values in all curves with estimated values on the x-axis.

5 Conclusions

In this paper, we have investigated parallel access to referenced objects in a complex-object DBMS. The developed cost model can predict the number of necessary I/Os when executing parallel set-oriented requests for identified objects. Due to the strong performance impact of run-time parameters, the degree of parallelism for scan operations cannot be determined before run time. The results of this paper show that parallel navigational access has to be considered very carefully because repeated access to pages can induce substantial overhead. On the other hand, we observed that complex-object construction can benefit from parallel navigational access, especially when there are a lot of free resources or when clients request only small numbers of objects or if pages store only a small number of objects. In these situations, it is highly probable that every object request refers to another page. Finally, the derived cost model has been verified in the framework of the PRIMA system.

The results obtained in this paper do not depend on any specific algorithm (e.g. depth first, breadth first, or more sophisticated ones) for navigation through highly meshed complex-object structures. But, on the other way around, our results are helpful to determine optimal run-time parameters for an efficient parallel execution of these algorithms in complex-object construction.

An extension of this model, e.g. exploitation of access path structures, is straight forward in only adding another phase. Then, the input for an address scan is determined by the result of this access path scan and not any more by the Access System's client. This means, after having received logical addresses (OIDs) from an access path server the Access System additionally has to determine appropriate scan invocations for address scans. Fortunately, we can use the same

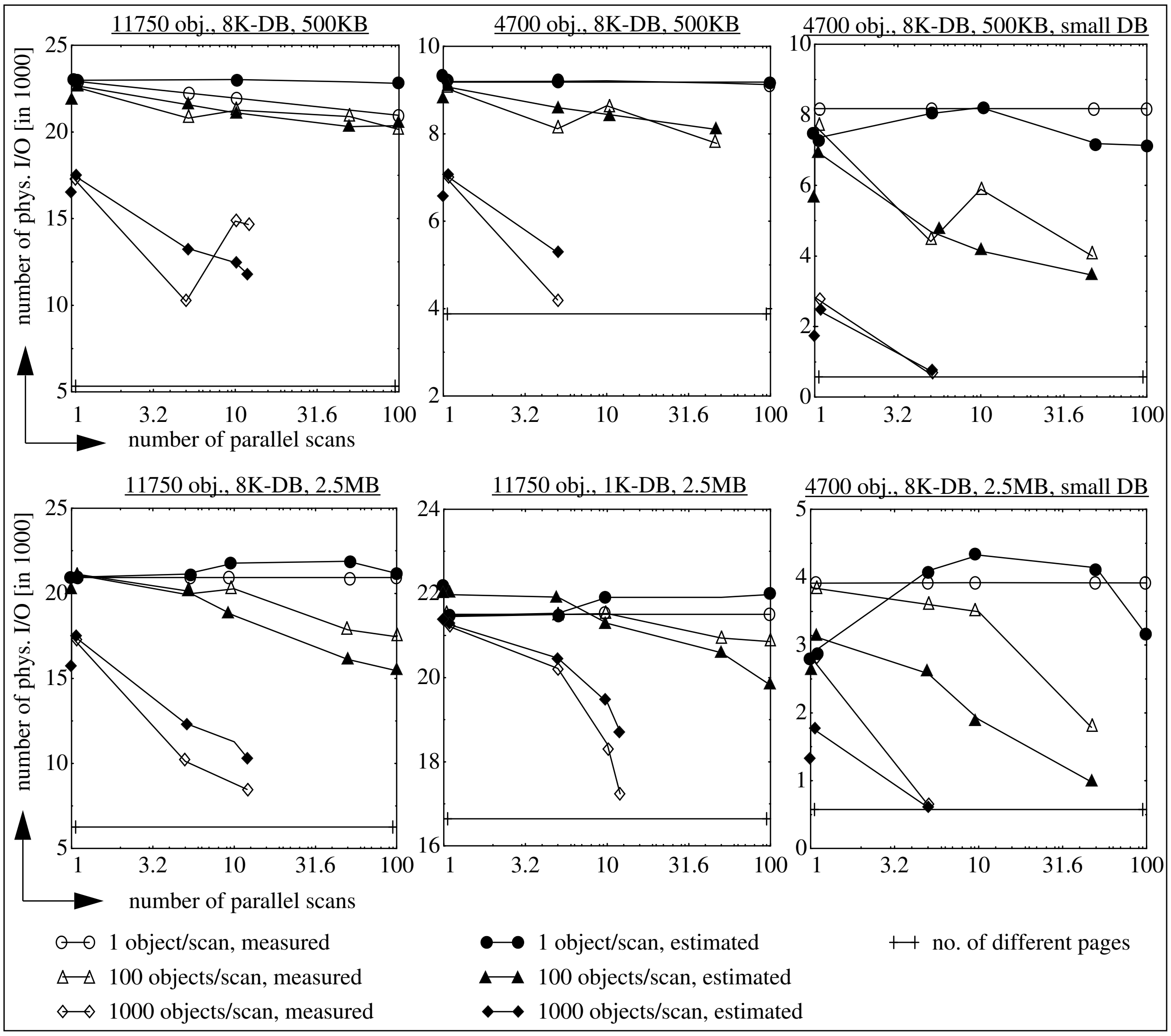

Figure 7: Measurement results compared to cost model estimation

cost model for this decision.

In our future work, we want to investigate how clustering of data affects performance of parallel data access. However, note that clustering can support only special complex-object structures which are frequently used. At the same time, clustering is counterproductive to dynamic definition of COs in query languages especially in cases of bidirectional links. The presented cost model assumes constant response times for every physical page I/O because we only considered random access to pages containing randomly selected address information or objects, respectively. Of course, this does not hold in the case of single large objects or of clustered complex objects which cover multiple pages, organized in page sequences. Therefore, unequal I/O times for random and sequential I/O have to be considered in a more realistic model. Finally, parallel processing of distributed pages has to be explored in more detail.

References

[1] Bertino, E.: A Survey of Indexing Techniques for Object-Oriented Databases, in: Freytag, J.C., Maier, D., Vossen, G.: Query Processing For Advanced Database Systems, Morgan Kaufmann Publishers, Inc., pp. 382-418, 1993

[2] Bancilhon, F., Ferran, G.: The ODMG Standard for Object Databases, Proc. DASFAA '95, Singapore, pp. 273-283, 1995

[3] Chamberlain, D., Mattos, N., Cheng, J., DeMichiel, L.: Extending relational database technology for new applications, IBM Systems Journal, Vol. 33, No. 2, pp. 264-279, 1994

[4] DeWitt, D.J., Naughton, J.F., Shafer, J.C., Venkataraman, S.: Parallelizing OODBMS traversals: a performance evaluation, The VLDB Journal, Vol. 5, No. 1, pp. 3-18, 1996

[5] Eickler, A., Gerlhof, C., Kossmann, D.: A Performance Evaluation of OID Mapping Techniques, Proc. 21st VLDB Conf., pp. 18-29, 1995

[6] Gesmann, M.: Mapping a Parallel Complex-Object DBMS to Operating System Processes, Proc. EURO-PAR 96, pp. 852-861, 1996

[7] Graefe, G.: Query Evaluation Techniques for Large Databases, ACM Computing Surveys, Vol. 25, No. 2, pp. 73-170, 1993

[8] Gardarin, G., Gruser, J., Tang, Z.: A Cost Model for Clustered Object-Oriented Databases, Proc. 21st VLDB Conf., pp. 323-334, 1995

[9] Härder, T. et al.: PRIMA - A DBMS Prototype Supporting Engineering Applications, Proc. 13th VLDB Conf., pp. 433-442, 1987

[10] Kemper, A., Moerkotte, G.: Access Support in Object Bases, Proc. ACM/SIGMOD Conf., pp. 364-374, 1990

[11] Lieuwen, D.F., DeWitt, D.J., Mehta, M.: Pointer-based Join Techniques for Object-Oriented Datases, Technical Report tr1099, University of Wisconsin, 1992

[12] Lieuwen, D.F., DeWitt, D.J., Mehta, M.: Parallel Pointer-based Join Techniques for Object-Oriented Datases, 2nd Int. Conf. on Parallel and Distributed Information Systems, 1993

[13] Mackert, L.F., Lohman, G.M.: Index Scans Using a Finite LRU Buffer: A Validated I/O Model, ACM Trans.on Database Systems, Vol. 14, No. 3, pp. 401-424, 1989

[14] Pistor, P.: Objektorientierung in SQL3: Stand und Entwicklungstendenzen, Informatik Spektrum, Springer-Verlag, Vol. 16, No. 2, pp. 89-94, 1993

[15] Schöning, H.: Anfrageverarbeitung in Komplexobjekt-Datenbanksystemen, Deutscher Universitäts-Verlag, 1993

[16] Shekita, E.J., Carey, M.J.: A Performance Evaluation of Pointer-Based Joins, Proc. ACM/SIGMOD Conf., pp. 300-311, 1990

[17] Schek, H.J., et al.: The DASDBS Project: Objectives, Experiences, and Future Prospects, IEEE Trans. on Knowledge and Data Engineering, Vol. 2, No. 1, pp. 25-43, 1990

[18] Teeuw, W., Rich, C., Scholl, M., Blanken, H.: An Evaluation of Physical Disk I/Os for Complex Object Processing, Research Report, ETH Zürich, Departement Informatik, 1992

[19] Yao, S.: Approzimating the number of accesses in database organizations, Communications of the ACM, Vol. 20, No. 4, pp. 260-261, 1977

Appendix

Summary of the most important notations in the cost model presented in this paper:

buffersize	number of available buffer frames
n	number of requested objects
t	number of scans to request the objects
p	number of parallel scans
SPS	group of parallel scans
run	group of simultaneous page requests in an SPS
$N_{PR}()$	number of logical I/Os
$N'_{PR}()$	number of pages containing randomly selected objects
$N_{IO}()$	number of physical I/Os with t sequential scans
locality()	probability that logical I/Os in a run request the same page
$N^{run}_{PR}()$	number of different page request in a run
$N^{run}_{PRrel}()$	number of page requests in a run which can cause physical I/O
$N^{par}_{IO}()$	number of physical I/Os for t scans in SPS of p scans