Hierarchical Query Execution in a Parallel Object-Oriented Database System

N. Bassiliades and I. Vlahavas
Dept. of Informatics,
Aristotle University of Thessaloniki,
54006 Thessaloniki,
Greece.

Tel: +3031-998145
Fax: +3031-998419
E-mail: nbassili@athena.auth.gr
vlahavas@olymp.ccf.auth.gr

Abstract

This article presents a hierarchical query execution strategy for a parallel object-oriented database (OODB) system. The system, named PRACTIC, is based on a concurrent active class management model and is mapped to an abstract hierarchical multiprocessor architecture. The proposed strategy is studied analytically and by simulation on a transputer-based machine, verifying the theoretical results. Although the analysis suits both main-memory and disk-based database systems, it becomes significant for main-memory systems where the multiprocessor initialization and communication overheads are comparable to the actual workload. The hierarchical query execution strategy is proved much better than the usual flat strategy of parallel database systems, except some clearly identified extreme cases, where flat processing is better. Furthermore, we propose a declustering scheme for space optimization to improve processor utilization and single-class query performance, by having different classes share memory and computation power of neighboring processing elements.

Keywords: parallel main-memory database system; object-oriented databases; multiprocessor architecture; parallel query execution; analytic performance model; simulation

1. INTRODUCTION

The advantages of Object-Oriented databases (OODBs) [11] are well established since the fall of the previous decade. OODBs reduce the "semantic gap" between real world concepts and data representation models and offer great flexibility and extensibility due to their complex structuring capabilities and dynamic binding of messages to methods at run-time. A major drawback of OODBs is the low speed of query execution, due to the sequential processing of independent entities and the slow disk-based access of large objects.

To enhance the performance of database systems, apart from traditional optimization techniques, multiprocessor database systems have been proposed, both for relational [4, 7, 8] and object-oriented [2, 13, 15] databases. Furthermore, main-memory systems have been reported [9, 10] to provide better performance for time-lined database applications. Parallel main-memory relational database systems have been studied in the context of the PRISMA/DB project [1, 17], and also in [5], while the conjunction of sequential object-oriented and main-memory databases received some attention as well [14]. However, there is no account for parallel, main-memory OODB systems so far in the literature to eliminate both the above mentioned drawbacks in the same system.

Our previous work includes the proposal and description of a parallel OODB system named PRACTIC [2]. The system is based on a concurrent object model that supports parallel management of active classes. Furthermore, an abstract machine (APRAM) that consists of a hierarchically structured multiprocessor system has also been proposed to map the above model. The model and its architecture are suitable for both disk-based and main-memory database systems.

In this paper we propose and study analytically the performance of a hierarchical main-memory execution strategy of multiclass queries on APRAM, while simulation results on the transputer-based implementation platform are also presented to verify the theoretical analysis. A first, minimally functional, prototype of PRACTIC has been implemented on a flat transputer network, but due to limited resources it could not be used to test the fully functional hierarchical query execution strategy. However, it gave us some valuable information about the parameters of the simulation.

The results are valid both for main-memory and disk-based database systems. However, they become significant for main-memory databases since the costs for initializing query processing and for interprocessor communication are comparable to the actual workload in such systems. We compare, both by analysis and simulation, our findings with the equivalent performance obtained from a flat parallel processing technique, which is used in most parallel database systems [1, 4, 6, 7, 17]. The comparison shows that hierarchical query processing is better than flat processing on most cases of multiclass queries, except some extreme cases where flat processing is better, which are clearly identified in the paper.

The shortcomings of the hierarchical query processing strategy, namely the low processor utilization and the non-optimal performance of single class queries, have also been overcome in the paper by extending the hierarchical data fragmentation strategy into an overlapping declustering scheme, where classes share storage and computational power of neighboring processing elements of the multiprocessor architecture. This extension offers more processor utilization and increases the performance of single class queries, while it keeps the same or slightly improves the multiclass query execution performance.

The rest of the paper is structured as follows: section 2 refers to related work on the performance analysis of main-memory database systems; section 3 briefly describes the PRACTIC model and its architecture before it presents our proposal for the hierarchical query execution. Section 4 analyses the performance of the model and compares it to the flat processing strategy. Furthermore, in section 4, the cases where hierarchical processing is inferior to flat processing are studied and clearly identified, while performance tuning strategies for the system are presented. In section 5 the problem of single class queries is described and the hierarchical declustering scheme is extended through overlapping of neighboring class resources, enhancing processor utilization and/or query execution performance for a single class. Section 6 discusses implementation issues and presents the simulation results for the proposed query execution strategy on a transputer-based machine. Finally, section 7 concludes this paper with a summary of the main points and a discussion of future work.

2. Related Work

The performance of query execution in parallel database systems using different data fragmentation strategies has been extensively studied by the designers and implementors of such systems [4, 6, 7, 8, 16]. Most of the systems use full declustering as their data fragmentation strategy, i.e. all relations are distributed over all nodes. The implementors of Bubba [4] argue that full declustering is not always the best strategy [6], since the access frequency of each relation partition may vary, therefore the most frequently accessed partition may become a bottleneck to the system performance.

All of the above systems are disk-based DBMSs and their performance and parallel behavior is influenced by the use of disk as main data storage. The performance of query execution in a parallel main-memory relational database system has been studied in the context of the PRISMA/DB project [1, 17], where it is concluded that there is an upper bound in speed-up in any parallel DBMS. This bound is lower in main-memory than disk-based database systems, because main-memory processing is faster, and the query initialization and interprocessor communication costs are comparable to the actual workload. In the following subsection, we summarize the analytical performance of PRISMA/DB query execution, because we will use it as a measure of comparison for our analysis.

Comparing the query processing strategy of PRACTIC to other parallel OODB databases [13, 15] we can identify the following differences:

- The class-hierarchy parallelism of [13] is limited to interclass parallelism, while query processing inside the class is done sequentially. In PRACTIC this type of parallelism is extended to data level parallelism, to provide intraclass parallelism that further speeds-up query processing.
- The vertical partitioning of [15] that provides attribute-level parallelism and decreases the I/O is meaningful only for disk-based databases. PRACTIC is a main-memory database that would not benefit from vertical partitioning, because the intersection of the various partitions would incur more overhead than gain. Furthermore, the system of [15] does not allow for intraclass data-level parallelism, as well.

The work described in this paper concerns the performance improvement of single large, decision-support queries or batch transactions in a parallel OODB system. Therefore our purpose is to minimize query response time, sacrificing (probably) memory and disk utilization. A similar goal was presented in [3], where a non-uniform declustering scheme was proved to achieve a 40% performance and scalability improvement for similar queries on single classes. The non-uniform scheme can be combined with the hierarchical declustering and query processing strategy presented here, to achieve even better results.

2.1. Analysis of Flat Parallel Query Execution

The abstract architecture of PRISMA/DB [1, 17], which is more or less the same for all parallel database systems [8], consists of a flat linear multiprocessor network (fig. 1). There is one master-processor that coordinates query execution and several slave-processors that execute relational operators on the partition of the relation they store. Relations are fragmented into equally sized fragments for the sake of uniformity and simplicity.

The master-processor is assumed to initiate query execution on each slave-processor sequentially and then each slave processor proceeds on its own in parallel (fig. 2).

Theorem 1. The response time of the system is:

$$R = \alpha n + c\frac{N}{n} \tag{1}$$

where α is the initialization time of each slave processor, c is the processing time per tuple in respect to a specific relational operator, N is the total number of tuples of the relation and n is the number of slave-processors that the relation is distributed into.

Theorem 2. Flat parallel query execution (compared to sequential processing) bears the following speed-up:

$$S = \frac{\alpha + cN}{\alpha n + c\frac{N}{n}} \tag{2}$$

Theorem 3. The speed-up of the flat parallel query execution becomes maximum (S_o) for n_o processors:

$$n_o = \sqrt{\frac{cN}{\alpha}} \tag{3}$$

$$S_o = \frac{1}{2} n_o \tag{4}$$

Proofs. See [17].

Thus in order to increase n_o, i.e. increase the scalability of the system (for a fixed number N of relation tuples), either c must be increased (as in disk-based DBMSs) or α must be decreased, which is a more rational solution, because it reduces the query response time as well [12]. In this paper we argue that in the parallel OODB system PRACTIC, it is possible to use a hierarchical, instead of the flat, query execution strategy to increase the performance and scalability of the system, mainly for multiclass queries.

3. The PRACTIC OODB System

This section reviews the PRACTIC parallel OODB system, along with its abstract multiprocessor architecture and describes the hierarchical query execution strategy. Before that, the object-oriented terminology and database parameters, used throughout the paper, are presented.

3.1. Object-Oriented Database Terminology

It is assumed that the database consists of instance objects, which are the actual data, and classes, which aggregate objects with common structure (attributes) and behavior (methods). The set of instances of a class is called the class extension.

Classes can be parallelized with relations, but with a far more richer structure. More specifically, classes can be specialized in class-hierarchies, that refine object descriptions according to their real world semantics. Therefore, the extension of a subclass is a subset of the extension of all its superclasses and the entire class-hierarchy can be viewed as a single relation, the one that corresponds to the extension of the class at the root of the hierarchy. Furthermore, classes have more operators than relations, because methods can be arbitrary programs, instead of the fixed relational operators.

In our discussion we assume that the database schema consists of a single class-hierarchy with k classes. We do not assume any specific type of the class-hierarchy, i.e. balanced tree, acyclic graph, etc. We just examine the query execution of queries sent to the root-class of the hierarchy, which involve all k classes, in order to avoid studying specific types of hierarchies. The results can be easily converted to queries sent to an arbitrary class of the schema, provided that the total number of its subclasses is known.

3.2. The PRACTIC Object Model and its Abstract Architecture

The above object model is common to many class-based OODB systems. For our parallel main-memory object-oriented database system, we have proposed a novel object data model, called PRACTIC [2], which is based on the concurrency of the active database objects. *Active* objects are the objects that can have direct instances, like the normal classes of an OODB schema. Furthermore, active objects are the metaclasses of an OODB, which have classes as their instances. Metaclasses are very useful for providing structure and behavior for the classes, but since their instances are not data, they should not be allocated considerable processor resources.

Furthermore, the PRACTIC model includes *passive* objects which are the actual data of the OODB, i.e. the instances of the normal, user classes, and the *semiactive* objects, which are all the mixin and abstract classes of the OODB schema, which have no direct instances, but serve as structure and behavior placeholders.

The abstract machine that directly reflects the PRACTIC model (APRAM) consists of a hierarchically structured network of shared-nothing processing nodes (fig. 3). Each node has its own memory and/or disk. It is assumed that the memory and disk sizes are large enough to hold the entire amounts of class partitions. APRAM consists of two kinds of processors:

- The *Active Object Processors* (AOP), where the processes that correspond to the active objects run. They consist of a CPU, memory and links to other AOPs. AOPs are responsible for a) running the permanent procedure of the active objects, b) accepting messages destined at either the active object they host or at a non-active instance of the active object, c) evoking the semiactive processes, and d) coordinating query execution for the passive instances on the AEMs. There are as many AOPs, as the active classes. All the (active) metaclasses of the OODB, share a common AOP, which is also the *master-processor* of the system, since it communicates with the host system and coordinates query execution on the entire database.
- The *Active Extension Managers* (AEM), that store partitions of and execute queries on the active class extensions. AEMs are exclusively devoted to one active class and consequently they are attached to the corresponding AOP. They can directly communicate only to the AOP and to each other, through a local communication network and not to the rest of AOPs and AEMs.

The most important sources of parallelism that the APRAM architecture provides are [2]:

- *Class-hierarchy parallelism*. Queries received by a class are evaluated in parallel by that class and its subclasses, since the extension of a class is a subset of all its superclasses. Class-hierarchy parallelism offers a significant speed-up in query evaluation [13], and is realized by the multiple AEMs.
- *Inter-Object parallelism*. Queries received by a class are evaluated in parallel among the AEMs, since the class extension is partitioned, providing thus a form of interobject parallelism. Queries are distributed to the AEMs, thus methods are executed on objects in parallel.

3.3. The Hierarchical Query Execution Strategy

The main feature of query execution of the PRACTIC model on the abstract machine is class-hierarchy parallelism. A class concurrently evaluates a message with its subclasses, because the instances of a subclass belong to its superclass, as well. Class-hierarchy parallelism speeds-up query execution significantly [13]. However, in APRAM, a class is not stored on a single node, as in [13], but it is further partitioned into an array of AEMs that host instances of one class strictly. Therefore, intraclass parallelism should offer more speed-up. The outline of the query execution strategy is the following:

- The system receives a query:

```
foreach x in X such that P(x) do M => x
```

where `X` is the first class of the class-hierarchy, that has k-1 subclasses, `P(x)` is an object selection predicate and `M` is an arbitrary method defined on class `X` and is inherited (or overridden) at all subclasses of `X`.

- The master-processor (i.e. the AOP of the metaclasses) distributes the query to the AOP of class `X` and each one of `X`'s subclasses. The master-processor cannot initialize all the AOPs at the same time, but each one in turn, sequentially (fig. 4).
- When an AOP receives the query and it is ready to evaluate it, it starts the method execution processes on each of its AEMs. Again, the AOP cannot initialize all AEMs at once, but each one in turn, sequentially (fig. 4).

3.4. The Flat Query Execution Strategy

In order to compare the performance of hierarchical query execution with the parallel query processing on a conventional flat architecture [17], we describe in this section a flat query execution strategy for the OODB query model presented in the previous section.

The class-hierarchy can be viewed as a single relation with N tuples (the total number of instance objects). Furthermore, the specific operations performed on the instance objects have operator execution time per tuple c, which can be calculated as the average method execution time among the classes of the hierarchy (see equation 12). The instance objects are distributed among n slave-processors, which are equivalent to AEMs. Thus, there are no AOPs in the flat architecture (fig. 1). The master-processor initializes the AEMs sequentially and each AEM processes its objects independently, in parallel (fig. 2).

4. PERFORMANCE ANALYSIS

In this section an analytic performance model is developed for the described hierarchical query execution strategy, based on the analysis of an equivalent parallel, main-memory relational database system [17]. First the query response time of the system is calculated and then it is compared with the equivalent flat processing performance. The results of the comparison are used to form guidelines for tuning the database performance. Finally, other system parameters, like speed-up and processor utilization are calculated.

It is assumed that the instance objects of the k active classes are distributed over the AEMs uniformly, such that the workload is the same on each AEM processor, regardless of the class it belongs to. Furthermore, each class extension is equally distributed among its AEMs.

Definition 1. The workload of an AEM node is the number of objects per node, times the average method execution time per object:

$$W_i^j = c_j N_i^j \tag{4}$$

where W_i^j is the workload at the i-th AEM of the j-th AOP, N_i^j is the number of objects at that AEM, and c_j is the average method execution time per object of the j-th class. □

The average method execution time is, generally, different for each class, since (due to overloading) the same method name may correspond to different method body to some or all classes. We expect that some of the classes do not redefine methods of their superclass and the method execution time is the same for those classes. However, in our analytic model we cater for the most general case, where each class has a different method body for the same method name. In contrast, relational operators have the same processing cost per tuple, regardless of the type of the relation.

Definition 2. The workload of an AOP node is the sum of the workloads of all its AEM nodes:

$$W^j = \sum_{i=1}^{n_j} c_j N_i^j \tag{5}$$

where W^j is the workload of the j-th AOP and n_j is its number of AEMs. □

Corollary 1. The number of objects residing at each AEM is:

$$N_i^j = \frac{N^j}{n_j} \tag{6}$$

where N_j is the total number of objects of the j-th class. This is a consequence of the uniform object distribution among the AEMs. □

Lemma 1. The workload of an AOP for the uniform object distribution is:

$$W^j = c_j N^j$$

Proof. This follows easily from definition 2 and corollary 1, since the quantities inside the sum of equation (5) do not depend on i. □

4.1. Query Response Time

In this section the response time of the system is calculated.

Lemma 2. The response time of the j-th class relative to the beginning of query execution on the j-th AOP is:

$$R_j^{rel} = \alpha n_j + c_j \frac{N^j}{n_j} \tag{7}$$

where α is the initialization cost for query processing/method execution on each AOP. Here, we assume that this cost is the same for every AOP, regardless of the class it belongs to, since it mainly involves copying queries/methods and spawning processes on the AOP.

Proof. Each individual class can be seen as a relation, which is uniformly distributed to n_j processors. Therefore, according to theorem 1, its response time, after the AOP receives the query, should be given by the above equation. □

Theorem 4. The absolute response time of the *j*-th AOP is:

$$R_j = \beta j + \alpha n_j + c_j \frac{N^j}{n_j}$$

where β is the initialization time of an AOP which is in general different from the initialization time of an AEM and is independent from the nature of the *j*-th class.

Proof. The master-processor initializes AOPs sequentially. Therefore the *j*-th AOP does not start query execution until the previous *j*-1 AOPs are initialized. Therefore, in order to calculate the absolute response time of the *j*-th AOP, we must add the initial idle time to the relative response time of equation (7):

$$R_j = \beta_j + R_j^{rel} = \beta_j + \alpha n_j + c_j \frac{N^j}{n_j}$$

The initial idle time equals the sum of the initialization times of all previous *j*-1 and *j*-th AOPs:

$$\beta_j = \beta(j-1) + \beta = \beta j$$

□

The response time of the whole system is the response time of the "slowest" among the AOPs. In order to figure out which this AOP is we must have a knowledge of the specific OODB schema, since the distribution of the workload per class is important. However, we can use the index *max* to indicate the class with the maximum workload:

$$W^{max} = max(W^j) = max(c_j N^j)$$

Lemma 3. The response time of the system is bounded:

$$\beta + \alpha n_{max} + c_{max} \frac{N^{max}}{n_{max}} \le R \le \beta k + \alpha n_{max} + c_{max} \frac{N^{max}}{n_{max}} \tag{8}$$

where k is the total number of classes.

Proof. Let us consider the worst case for the system, which is when the last class (k-th) has the largest workload. This means that the "slowest" among the AOPs will start execution last, which makes it also the last one to finish. The response time of the system would then be:

$$R_{up} = \beta k + \alpha n_{\max} + c_{\max} \frac{N^{\max}}{n_{\max}}$$

On the other hand, the maximum workload AOP can be placed first, therefore it will start query execution earlier than the rest of the AOPs. Now we do not know if this first, maximum workload AOP will finish execution last, because there might be some other AOP that finishes later, even if it has less workload. However, the response time for the system can never be less than the response time of the maximum workload class:

$$R_{low} = \beta + \alpha n_{\max} + c_{\max} \frac{N^{\max}}{n_{\max}}$$

□

Definition 3. The workload fraction f_j of an AOP is the fraction of the workload of the j-th AOP, divided by the total workload of the system:

$$f_j = \frac{W^j}{W} \tag{9}$$

□

Notice that f_j is always less than one, since it is a fraction of a positive quantity divided by a sum that includes this quantity.

Theorem 5. The response time of the system is:

$$\beta + \alpha f_{max} n + c\frac{N}{n} \leq R \leq \beta k + \alpha f_{max} n + c\frac{N}{n} \tag{10}$$

where n is the total number of AEMs in the system, N is the total number of instance objects, and c is the average method execution time.

Proof. The workload on each AEM of any AOP is the same, given by equation (4). If equation (6) is substituted, we derive:

$$c_1 \frac{N^1}{n_1} = c_2 \frac{N^2}{n_2} = \ldots = c_k \frac{N^k}{n_k} = \frac{\sum_{j=1}^{k} c_j N^j}{\sum_{j=1}^{k} n_j} = c\frac{N}{n} \tag{11}$$

where c is the average method execution time:

$$c = \frac{\sum_{j=1}^{k} c_j N^j}{N} \tag{12}$$

From equations (11) and (9) we can derive that for each AOP it holds:

$$n_j = \frac{c_j N^j}{cN} n = \frac{W^j}{W} n = f_j n \tag{13}$$

Therefore, if equations (13), (11) are substituted in (8) with index *max*, we get equation (10). □

In the following we use the upper bound of theorem 5 as the response time of hierarchical query processing, bearing in mind that better performance can be obtained by assigning the "slowest" class to the first AOP. Whenever the lower bound is used, it is clearly articulated.

4.2. Hierarchical vs. Flat Query Processing

If the response time of the hierarchical query processing (10) is compared to the response time of flat processing (1), their difference is:

$$\Delta R = R_h - R_f = \beta k - \alpha n(1 - f_{max}) \tag{14}$$

We observe that there is no single, definite answer to the question if the hierarchical processing is faster than the flat processing. We must examine the behavior of ΔR according to the quantity f_{max}, which is a factor that depends on the schema and the volume of data of the OODB and not on the specific hardware resources.

The difference is monotonically increasing with f_{max}:

$$\frac{\partial}{\partial f_{max}}(\Delta R) = \alpha n > 0$$

therefore we must study the special cases for f_{max}.

4.2.1. Worst Case

The worst case is when f_{max} is close to its maximum (1), because the difference (14) is also maximum. This happens when the workload of the *max* class is "almost" equal to the total workload:

$$f_{max} \cong 1 \Rightarrow W^{max} \cong W$$

If the above equation is substituted in (14) we derive that:

$$\Delta R \cong \beta k > 0 \Rightarrow R_h > R_f \tag{11}$$

which means that hierarchical processing is worse than flat in this case and therefore it should not be used.

This can also be rationally explained by observing that a single class is distinctively "slower" than the rest and dominates the processing time, therefore class-hierarchy is degenerated into a single class. Instead of processing though this single class through flat query execution, we use instead hierarchical processing that introduces the extra overheads caused by the AOP initialization times (βk).

The same conclusion is drawn if the lower bound of hierarchical query response is considered (theorem 5). If the maximum workload AOP is placed first, then the difference becomes just β which means that the two processing strategies have almost the same performance. Indeed in this case we do know that the maximum workload AOP will finish last, since it is much "slower" than any other AOP.

4.2.2. Best Case

The best case is when f_{max} is minimum. In order to find out this minimum we can observe that f_{max} is always bigger than or equal to the rest of f_j's, because it depends directly on the maximum workload, therefore the minimum f_{max} can only be achieved when all f_j's are equal to each other and to f_{max} . Of course this means that all classes have the same workload:

$$f_{max} = \frac{W^{max}}{W} = \frac{W^{max}}{\sum W^j} = \frac{W^{max}}{kW^{max}} = \frac{1}{k}$$

If the above quantity is substituted in equation (14) we get:

$$\Delta R = \beta k - \alpha n(1 - \frac{1}{k}) = \frac{\beta k^2 - \alpha nk + \alpha n}{k} \tag{15}$$

The sign of the nominator determines the sign of the difference as well. The quadratic function of the nominator can be studied with the help of the "discriminant":

$$\Delta = (-\alpha n)^2 - 4\beta\alpha n = \alpha n(\alpha n - 4\beta)$$

<u>Case 1 (Δ<0):</u> In order for Δ to be negative the following must hold:

$$\frac{\beta}{\alpha} > \frac{n}{4}$$

When the above condition holds the sign of the nominator of (15) is positive, therefore hierarchical processing is worse than flat processing. The above condition actually means that the AOP initialization time is very high compared to the AEM initialization time, therefore it is useless to introduce new processors with such a high non-useful initialization time.

Case 2 (Δ=0): In order for Δ to be zero the following must hold:

$$\frac{\beta}{\alpha} = \frac{n}{4}$$

When the above condition is satisfied then the difference becomes:

$$\Delta R = \frac{\alpha n (k-2)^2}{4k}$$

The difference is now positive for every k, except for k=2, which means that only for 2 classes the two strategies have the same performance, while for any other case, flat processing is faster. There is no point using the more complex hierarchical processing if it brings the same performance with hierarchical. The explanation given for case 1 applies here as well.

Case 3 (Δ>0): In order for Δ to be positive the following must hold:

$$\frac{\beta}{\alpha} < \frac{n}{4} \tag{16}$$

When condition (16) is satisfied then the difference becomes:

$$\Delta R = \frac{\beta}{k}\left(k - \frac{\alpha n + \sqrt{\alpha n(\alpha n - 4\beta)}}{2\beta}\right)\left(k - \frac{\alpha n - \sqrt{\alpha n(\alpha n - 4\beta)}}{2\beta}\right)$$

The difference is now positive for every k that lies between the two solutions of the above equation and negative (or zero) for any other value. Therefore hierarchical query processing should be used only when the number of classes k satisfies the following condition (when also the condition (16) holds):

$$\frac{\alpha n - \sqrt{\alpha n(\alpha n - 4\beta)}}{2\beta} < k < \frac{\alpha n + \sqrt{\alpha n(\alpha n - 4\beta)}}{2\beta} \tag{12l}$$

The rationale for using efficiently hierarchical processing on condition (16) is that the AOP initialization is not so high, allowing more useful work to be done concurrently.

In this case there is no need to examine the lower bound of theorem 5, since we know that the last AOP is the slowest, because it starts last and has the same workload with every other AOP.

4.3.3. General Case

If f_{max} is neither minimum nor maximum we can still find some condition that will decide whether hierarchical query processing is worthwhile for a certain system. Difference of equation (14) must be negative, thus:

$$\beta k - \alpha n(1 - f_{max}) < 0 \Rightarrow \frac{1}{n}\frac{\beta}{\alpha} < \frac{1 - f_{max}}{k} \tag{17}$$

The left-hand side of the equation includes only system-specific parameters, like processor initialization times and number of available AEMs, while the right-hand side contains only OODB specific parameters, like number of classes and maximum workload fractions, which can be determined from the actual OODB to be used on the system. If condition (17) is satisfied, then the hierarchical scheme will increase the performance of parallel query processing, compared to a flat processing scheme.

If condition (17) is not satisfied for a specific system and/or OODB, the system administrator may tune the system's parameters, based on (17) to make hierarchical processing worthwhile. We will investigate in the next section how to tune the performance.

4.3. Performance Tuning

The performance of the hierarchically structured system is based both on system parameters, such as available processors and processor initialization times, as well as database parameters, such as the number of classes and the distribution of workload. Based on condition (17) we will study in this section ways to increase the performance of a hierarchical query processing system, in order to become better than flat processing.

4.3.1. System Tuning

The left-hand side of condition (17) should become less than the right-hand side in the following ways:

- Increasing the number of AEMs. If the number n of AEMs is increased in the system, assuming that the rest of the parameters remain constant, then the left-hand side of condition (17) increases, because n is at the denominator. This means that adding new AEMs to the system will make hierarchical processing perform better, until a crosspoint n_c, where it will become better than flat processing:

$$n_c = \frac{\beta}{\alpha} \frac{k}{1 - f_{max}}$$

Both flat and hierarchical processing response times (equations 1 and 10, respectively) depend similarly on n, however hierarchical processing benefits more from the increase of the number of AEMs, since not all AEMs go to the same AOP, but they are distributed to the AOPs according to their workload fraction (equation 13). This means that the initialization idleness that goes with the new AEMs is not added linearly, since many of the new AEMs will be initialized concurrently. This is not the case however for flat processing, where each new AEM adds to the initialization time linearly.

We can examine the best and worst cases for the value of n_c. For the worst case it is:

$$f_{max} \to 1 \Rightarrow n_c \to \infty$$

which actually means that there is no value for *n*, such that hierarchical processing is better than flat processing, a conclusion already drawn. For the best case it is:

$$f_{max} = \frac{1}{k} \Rightarrow n_c = \frac{\beta}{\alpha} \frac{k^2}{k-1}$$

- Decreasing the AOP initialization time. Since the AOP initialization time β is at the nominator of the left-hand side of condition (17), the latter will decrease if β decreases, as well. Decreasing β is a matter of system implementation optimization or hardware improvement though, and it does not have anything to do with the database system administrator. Therefore, for a given working system nothing can be done for decreasing β.
- Increasing the AEM initialization time. Since the AEM initialization time α is at the denominator of the left-hand side of condition (17), the latter will decrease if α increases. However, this again has nothing to do with the database system administrator, for a given working system. Furthermore it is not a rational solution since increasing α increases the overall response time of the system, for both query processing strategies (equations 1, 10).

4.3.2. Database Tuning

The right-hand side of condition (17) should be greater than the left-hand side, in any of the following ways:

- Decreasing the number of AOPs. Since the number of AOPs k is at the denominator of the right-hand side (17), decreasing k, increases the right-hand side of condition (17). Literally, decreasing k does not mean decreasing the number of classes of the system, because these are fixed for a given

OODB schema, but decreasing the number of AOPs that 'host' them. This can be achieved when some of the classes share the AEMs of an AOP. The sharing is done by re-distributing the objects of both classes to the sum of the AEMs. The two classes are managed by the same AOP concurrently.

This solution decreases the number of AOPs and therefore the initialization time associated with them, without increasing the workload per AEM, because the sum of the workloads of the two AOPs are distributed to the sum of the number of their AEMs, therefore their quotient remains fixed. However, the response time of the new AOP is increased because the sum of the AEMs are now initialized by the single AOP sequentially, therefore the relative response time of the AOP is:

$$R_j^{rel} = \alpha(n_j + n_{j+1}) + c\frac{N}{n}$$

according to lemma 2. Thus, in the position of two AOPs we have one larger, slower AOP. However, if the above AOP is still faster than the AOP with the maximum workload, this means that even if one AOP got slower, we do have one less AOP in the system and the maximum workload class begins execution earlier, thus the whole system's response time is decreasing.

On the other hand, if the response time of the new AOP is more than the response time of the maximum workload AOP we risk that the new system is slower than the previous one.

Now we consider the case when the maximum workload AOP is placed first (lower bound of theorem 5). Condition (17) becomes:

$$\frac{1}{n}\frac{\beta}{\alpha} < 1 - f_{max}$$

and it does not depend on *k*. In this case the merging of two classes is immaterial because the slowest AOP is the first one and its response time is not affected by the number of AOPs. Furthermore, merging would increase the risk of the new AOP becoming the slowest one.

- Decreasing the maximum workload fraction. Since the maximum workload fraction f_{max} is at the nominator of the right-hand side of condition (17) with a negative sign, decreasing f_{max} will increase the right-hand side expression. However, the maximum workload fraction is a parameter of the database and cannot be changed by the database administrator, because the workload of each class and the total workload depend on the application and the users of the database. However, the above statement can be interpreted as: "*The less the maximum workload is, compared to the total workload, the better hierarchical processing performs*".

4.4. Other System Parameters

This section examines some more useful system parameters, like speed-up and processor utilization [8, 16].

4.4.1. Speed-up

The speed-up of parallel query processing measures how better the system's performance gets when more resources are added. Speed-up is usually the quotient of sequential to parallel processing times.

Corollary 2. The speed-up of hierarchical query processing is:

$$\frac{\alpha + cN}{\beta k + \alpha f_{max} n + c\dfrac{N}{n}} \le S \le \frac{\alpha + cN}{\beta + \alpha f_{max} n + c\dfrac{N}{n}} \tag{18}$$

in analogy to theorem 2 and by substituting the bounds of response time from theorem 5. ˆ

We note here that the worst case for the speed-up is the lower bound of equation (18) because the higher the speed-up, the better the parallel system is [8]. The ideal speed-up is linear, which virtually means that the system's performance is improved with the same rate of resource addition. However, in real systems speed-up is usually sublinear. In order to study the behavior of speed-up we must examine the behavior of the first derivative of equation (18) assuming that only the number n of AEMs is variable.

Theorem 6. Hierarchical processing has a maximum speed-up S_o', which is achieved for n_o' processors:

$$n_o' = \frac{1}{\sqrt{f_{max}}}\sqrt{\frac{cN}{\alpha}} \tag{19}$$

Proof. By differentiating equation (18) and finding the *n* for which the derivative equals to zero. □

Note that the maximum speed-up case is achieved for n_o' processors calculated by theorem 6, regardless of which bound (the lower or the upper) of equation (18) we consider. Therefore, we can consider n_o' as an important parameter of the hierarchical processing strategy.

Theorem 7. Hierarchical processing is more scaleable than flat processing, by a factor that depends on the maximum workload fraction:

$$\frac{n_o'}{n_o} = \frac{1}{\sqrt{f_{max}}} \tag{20}$$

Proof. Comparing equations (2) and (19) we derive the above equation. However, we must still prove that the quantity on the right-hand side of (20) is larger than 1.

Considering definition 4 of workload fraction and equation (11), we derive that:

$$\frac{1}{\sqrt{f_{max}}} = \sqrt{\frac{cN}{c_{max}N_{max}}} = \sqrt{1+\frac{\sum_{j=1,\, j\neq max}^{k} c_j N_j}{c_{max}N_{max}}} > 1$$

□

We note here, that at the worst case, f_{max} is close to 1 and the scalability of the two systems becomes asymptotically the same. Moreover, at the best case, it is:

$$\frac{n_o^{'}}{n_o} = \sqrt{k}$$

Theorem 8. The maximum speed-up of the hierarchical processing strategy is bounded:

$$\frac{n_o^{'}}{1+\frac{1}{f_{max}}} < S_o^{'} \leq \frac{n_o^{'}}{2}$$

Proof. If $n_o^{'}$ (equation 19) is substituted in the lower bound of equation (18), we get:

$$S_o^{'} = \frac{\alpha + cN}{\beta k + \alpha f_{max}\frac{1}{\sqrt{f_{max}}}\sqrt{\frac{cN}{\alpha}} + c\frac{N}{\frac{1}{\sqrt{f_{max}}}\sqrt{\frac{cN}{\alpha}}}} = \frac{\alpha + cN}{\beta k + 2\sqrt{f_{max}}\sqrt{\alpha c N}} \tag{21}$$

The denominator of the above equation can be transformed using condition (17):

$$\beta k - \alpha\frac{1}{\sqrt{f_{max}}}\sqrt{\frac{cN}{\alpha}}(1-f_{max}) < 0 \Rightarrow \beta k < \frac{1-f_{max}}{\sqrt{f_{max}}}\sqrt{\alpha c N}$$

Substituting the above in equation (21) we get:

$$S_o^{'} > \frac{\alpha + cN}{\frac{1-f_{max}}{\sqrt{f_{max}}}\sqrt{\alpha c N} + 2\sqrt{f_{max}}\sqrt{\alpha c N}} \cong \frac{cN}{\left(\frac{1-f_{max}}{\sqrt{f_{max}}} + 2\sqrt{f_{max}}\right)\sqrt{\alpha c N}} \Rightarrow$$

$$S_o^{'} > \frac{n_o^{'}}{\frac{1}{\sqrt{f_{max}}}\left(\frac{1-f_{max}}{\sqrt{f_{max}}} + 2\sqrt{f_{max}}\right)} = \frac{n_o^{'}}{\frac{1-f_{max}}{f_{max}} + 2} = \frac{n_o^{'}}{1+\frac{1}{f_{max}}} \tag{22}$$

Now we substitute $n_o^{'}$ (equation 19) in the upper bound of equation (18):

$$S_o^{'} \le \frac{\alpha + cN}{\beta + \alpha f_{max} \frac{1}{\sqrt{f_{max}}}\sqrt{\frac{cN}{\alpha}} + c\frac{N}{\frac{1}{\sqrt{f_{max}}}\sqrt{\frac{cN}{\alpha}}}} = \frac{\alpha + cN}{\beta + 2\sqrt{f_{max}}\sqrt{\alpha cN}} \Rightarrow$$

$$S_o^{'} \le \frac{cN}{2\sqrt{f_{max}}\sqrt{\alpha cN}} = \frac{n_o^{'}}{2} \tag{23}$$

In the last equation we assumed that α, β are negligible compared to the total workload.

□

Theorem 9. The maximum speed-up of the hierarchical processing strategy, when the maximum workload class is assigned at the first AOP, is better than the maximum speed-up of the flat processing strategy, by the same factor as that of theorem 6:

$$\frac{S_o^{'}}{S_o} = \frac{1}{\sqrt{f_{max}}}$$

Proof. By comparing equations (3) and (23) and substituting equation (20). □

Note that a similar theorem cannot be proved for the lower bound of the maximum speed-up, because if equations (3) and (22) are compared:

$$\frac{S_o^{'}}{S_o} > \frac{n_o^{'}}{n_o}\frac{2}{1+\frac{1}{f_{max}}} = \frac{2}{\sqrt{f_{max}} + \frac{1}{\sqrt{f_{max}}}} \tag{24}$$

we come up with a condition that implies nothing about the relationship between the maximum speed-up of the two processing strategies. To understand why is this so, we examine the right hand side of (24):

$$\frac{2}{\sqrt{f_{max}} + \frac{1}{\sqrt{f_{max}}}} = \frac{2\sqrt{f_{max}}}{f_{max}+1} = \frac{2\sqrt{f_{max}}}{f_{max}+1-2\sqrt{f_{max}}+2\sqrt{f_{max}}} =$$

$$= \frac{2\sqrt{f_{max}}}{\left(1-\sqrt{f_{max}}\right)^2 + 2\sqrt{f_{max}}} < 1 \tag{25}$$

In-equations (24) and (25) are not consistent, therefore they do not resolve whether $S_o^{'} > S_o$. What we can derive, though, from condition (24) is what happens at the extreme cases.

At the worst case, where $f_{max}=1$, it is $S_o^{'} > S_o$. At the best case, where $f_{max}=1/k$, it is:

$$\frac{S_o'}{S_o} > \frac{2}{\sqrt{\frac{1}{k}} + \sqrt{k}}$$

which again, does not imply anything about the relationship between S_o' and S_o, since the right-hand side quantity can be proved to be less than 1, using a similar technique with equation (25).

Note that S_o' and S_o are achieved for different number of processors (n_o' and n_o, respectively). Therefore, the techniques are just compared regarding their absolute best performance, whereas a relative comparison would include speed-up obtained from the same number of processors. This comparison is not necessary, because speed-up is the inverse of response time, therefore whenever hierarchical processing's response time is less than flat processing (see section 4.2), the speed-up is greater as well, for the same number of processors.

Example. To get a handy number, we assume a system configuration with the parameters of table 1. The order of magnitude for the system parameters is based on actual measurements on experimental systems [17]. We derive the maximum speed-up and the maximum number of processors for each strategy in table 1. It is clear that hierarchical processing achieves better maximum speed-up because it is far more scaleable. Notice that this conclusion could not be reached using a formal proof (equations 24, 25). □

4.4.2. Processor Utilization

Definition 4. The total "useful" execution time is defined as the total workload:

$$T_{use} = cN$$

□

Definition 5. The total "occupation" time is defined as the total time that all processors are occupied by global query execution, including the time they remain idle. This always equals to the total number of processors, times the response time of query execution at the "slowest" node, which is also the system's response time:

$$T_{tot} = Rn_{tot}$$

□

Corollary 3. The total number of AEMs is n, for flat processing, whereas for hierarchical processing the k AOPs must be added, too. Therefore, the following equations hold:

$$T_{tot}^{f} = R_f n \qquad\qquad T_{tot}^{h} = R_h(n+k)$$

for the two processing strategies, respectively. □

Definition 6. The average processor utilization is defined as the fraction of the "useful" time, divided by the "occupation" time:

$$u_{avg} = \frac{T_{use}}{T_{tot}}$$

□

Corollary 5. The processor utilization for the two parallel query processing strategies is:

$$u_{avg}^{f} = \frac{cN}{\alpha n^2 + cN} \qquad\qquad u_{avg}^{h} = \frac{cN}{\beta kn + \alpha f_{max} n^2 + cN}$$

if equations (1), (10), and corollary 3 are substituted into u_{avg}.□

Theorem 10. The hierarchical processing strategy utilizes processors more than flat processing, when the following condition holds:

$$k < \frac{-\left(\beta n + \alpha f_{\max} n + c\frac{N}{n}\right) + \sqrt{\left(\beta n + \alpha f_{\max} n + c\frac{N}{n}\right)^2 + 4\beta\alpha(1 - f_{\max})n^2}}{2\beta}$$

Proof. According to definition 6, utilization is inversely proportional to the total "occupation" time. Therefore, from corollary 3 we have:

$$u_{avg}^{h} > u_{avg}^{f} \Rightarrow T_{tot}^{h} < T_{tot}^{f} \Rightarrow R_h(n+k) < R_f n \Rightarrow \beta k^2 + (\beta n + \alpha f_{\max} n + c\frac{N}{n})k - \alpha(1 - f_{\max})n^2 < 0$$

The above inequation is satisfied only when the determinant of the quadratic equation is positive and the number of classes k lies between the two solutions of the quadratic equation. The determinant is always positive:

$$\left(\beta n + \alpha f_{\max} n + c\frac{N}{n}\right)^2 + 4\beta\alpha(1 - f_{\max})n^2 > 0$$

and therefore k must satisfy the following:

$$\frac{-B - \sqrt{B^2 + 4\beta\alpha(1 - f_{\max})n^2}}{2\beta} < k < \frac{-B + \sqrt{B^2 + 4\beta\alpha(1 - f_{\max})n^2}}{2\beta}$$

where:

$$B = \beta n + \alpha f_{\max} n + c\frac{N}{n}$$

Actually, the lower bound of k is not the negative solution of the quadratic equation, but 1, since at least 1 class must exist. □

Corollary 5. The hierarchical query processing strategy can only become better at processor utilization than flat processing when the following condition holds:

$$1 < \frac{-\left(\beta n + \alpha f_{\max} n + c\frac{N}{n}\right) + \sqrt{\left(\beta n + \alpha f_{\max} n + c\frac{N}{n}\right)^2 + 4\beta\alpha(1 - f_{\max})n^2}}{2\beta}$$

Corollary 5 is important because it gives a rule that should be obeyed in order to construct a system that can utilize processors more than flat processing, using hierarchical processing.

□

Theorem 11. At maximum speed-up, hierarchical processing has worse utilization than the corresponding case for flat processing.

Proof. If the optimal number of processors, for both processing strategies (equations 2 and 19) is substituted into the average processor utilization, for both strategies, it becomes:

$$u^f_{avg_o} = \frac{cN}{n_o^2\alpha + cN} = \frac{cN}{\frac{cN}{\alpha}\alpha + cN} = \frac{1}{2}$$

$$u^h_{avg_o} = \frac{cN}{\beta k n_o^{'} + \alpha f_{max} n_o^{'2} + cN} = \frac{cN}{\beta k \frac{1}{\sqrt{f_{max}}}\sqrt{\frac{cN}{\alpha}} + 2cN} = \frac{1}{\frac{\beta k}{\sqrt{\alpha f_{max} cN}} + 2} < \frac{1}{2}$$

Thus, it is proved that $u^h{}_{avg_O} < u^f{}_{avg_O}$. □

However, the difference between the optimal cases is not very significant, since usually it is $\alpha f_{max} cN \gg \beta k$, in the denominator of $u^h{}_{avg_O}$, and utilization becomes ≈1/2. This little difference however pays off, since the extra processors speed-up query execution significantly. Note also that utilization at optimal cases (theorem 11) is compared for different number of processors, while general utilization (theorem 10) is compared for the same number of processors.

5. SINGLE CLASS QUERY EXECUTION

So far in the analysis, it was assumed that a query targets the class at the top of the hierarchy. The maximum performance is thus achieved for such multiclass queries. In this section queries targeted at a single-class are also considered. This class is at the bottom of the class-hierarchy. It is usually not possible (or desirable) to isolate individual classes at the middle of the class-hierarchy in OODBs. Furthermore, we remind here that hierarchical query processing is particularly useful for multiclass queries.

5.1. Problem Analysis

Lemma 4. When the system operates with the global optimum number of processors, each AOP s operates with ${}^{s}n_o^{'}$ processors:

$$ {}^{s}n_o^{'} = f_s \frac{1}{\sqrt{f_{max}}} \sqrt{\frac{cN}{\alpha}} \qquad (26) $$

Proof. An AOP s, operates with n_s AEMs, given by equation (13). At maximum speed-up it is $n{=}n_o^{'}$ and therefore equation (26) is proved. □

Corollary 6. A single class s can be viewed as a relation, therefore its response time to a query is:

$$ R_s = \beta + \alpha n_s + c_s \frac{N_s}{n_s} $$

Furthermore, the speed-up for class s would become maximum for ${}^{s}n_o$ AEMs:

$$ {}^{s}n_o = \sqrt{\frac{c_s N_s}{\alpha}} \qquad (27) $$

□

From lemma 4 and corollary 6 we understand that AOP s cannot be assigned more than ${}^{s}n_o^{'}$ AEMs, while it could operate optimally at ${}^{s}n_o$ AEMs. The following theorem states which is better.

Theorem 12. When the hierarchical processing strategy operates optimally, every class except the maximum workload class, operates (individually) with a suboptimal performance.

Proof. Combining equations (27) and (26) we get:

$$ \frac{{}^{s}n_o^{'}}{{}^{s}n_o} = \frac{f_s\sqrt{cN}}{\sqrt{f_{max}}\sqrt{c_s N_s}} = \frac{\frac{c_s N_s}{cN}\sqrt{cN}}{\sqrt{\frac{c_{max}N_{max}}{cN}}\sqrt{c_s N_s}} = \sqrt{\frac{W_s}{W_{max}}} \leq 1 \Rightarrow {}^{s}n_o^{'} \leq {}^{s}n_o $$

If AOP *s* operates with less processors than the optimum, it also means that the performance is less than the maximum. □

This is a consequence of the requirement that the class hierarchy as a whole should operate with the maximum possible speed-up. However, if the distribution of the class workloads is not very sharp near the maximum workload class, then AOPs operate close to the optimum number of processors and the maximum performance, as the above equation implies.

5.2. Overlapping of Neighboring Class Resources

In this section we propose a new declustering scheme that will increase the utilization of processors in the hierarchical query processing scheme. Furthermore it will increase the performance of individual classes, since the number of AEMs per class is increased.

The main idea behind the proposed declustering scheme is that some AEMs of neighboring AOPs could be shared by the AOPs, at different time instances. This means that certain AEMs will host objects from both AOP classes. However, this sharing must not disturb: a) the uniform declustering of objects among the AEMs of one AOP, and b) the hierarchical query processing strategy.

The former means that every AEM should contain the same number of instances of a certain class, no matter if it is an exclusive or a shared AEM. The latter means that the scheduling of hierarchical query processing should not be disturbed, thus a shared AEM should be exclusively engaged by a single AOP, at any given time instance.

A simple schematic description of the overlapping declustering scheme is shown in fig. 5. It is obvious that in order not to disturb hierarchical processing an AEM must process objects from a second class, when it is still waiting to be initialized from the first class. This can be achieved if AOP_2 initializes the shared AEMs before AOP_1 of the first class gets there, because the latter is still initializing previous AEMs. Thus we utilize the idle initial time of AEMs, by fitting useful work in there. Furthermore the performance of individual classes is increased, since they now use more processors for query processing.

Some initial remarks that can be made before the study of the behavior of the new declustering scheme, are:

- The sum of the AEMs for each AOP does not result in the total number of AEMs, since some of the AEMs are shared and therefore counted twice:

$$n \neq \sum_{j=1}^{k} n_j$$

- The number of AEMs of the j-th AOP is the sum of the exclusive and shared AEMs, both with the previous and the next AOPs:

$$n_j = n_{j-1,j} + n_{jj} + n_{j,j+1} \tag{28}$$

where n_{jk} is the number of shared AEMs between the j-th and k-th AOP and n_{jj} are the exclusive AEMs of the j-th AOP.

- The main feature of the *simple* hierarchical approach described in section 4, that each AEM from any AOP has the same workload cannot hold any more. This is obvious since the shared AEMs host more objects than the exclusive ones of both neighboring AOPs.

For the sake of simplicity we will here study a more simple case with only two overlapping AOPs. Therefore equation (28) becomes for two neighboring AOPs:

$$n_j = n_{jj} + n_{j,j+1} \tag{29}$$

In the following the conditions that must be met by the parameters of the two AOPs, in order for the overlapping to take place, are presented. The main condition, as can be understood from fig. 5, is that the processing time of the AEMs of the j+1-th AOP, must be equal to or less than the initial idle time of at least one AEM of the j-th AOP, plus/minus some constants:

$$R_{j+1}^{1} = I_j^d - \alpha \tag{30}$$

where R_j^i is the response time of the i-th AEM of the j-th AOP and I_j^i is the initial idle time of the i-th AEM of the j-th AOP. We need to subtract the initialization time α from the initial idle (fig. 5) because the AEM must be free to communicate with the AOP during that period. In equation (30) it is assumed that the d-th AEM of the j-th AOP is the first shared AEM with the j+1-th AOP.

Lemma 5. The number of AEMs that the j+1-th AOP must have, depends not only on its workload, but also on the number of AEMs of the j-th AOP and the number of shared AEMs:

$$n_{j+1} = \frac{c_{j+1}N^{j+1}}{\alpha(n_j - n_{j,j+1} - 1) - \beta}$$

Proof. Equation (30) expands to:

$$\beta(j+1) + \alpha + \frac{c_{j+1}N^{j+1}}{n_{j+1}} = \beta j + \alpha d - \alpha \Rightarrow \frac{c_{j+1}N^{j+1}}{n_{j+1}} = \alpha(d-2) - \beta \Rightarrow n_{j+1} = \frac{c_{j+1}N^{j+1}}{\alpha(d-2) - \beta}$$

The d-th AEM of the j-th AOP is the first shared AEM, or else the first AEM after the exclusive ones of the j-th AOP. If we also consider equation (29) we get:

$$d = n_{jj} + 1 = n_j - n_{j,j+1} + 1$$

If the above equations are combined, the lemma is proved. □

The next theorem states how large can the overlapping between two neighboring classes be.

Theorem 13. The maximum number of shared nodes between two neighboring AOPs is:

$$n_{j,j+1} \le n_j - n_{j+1_o} - \frac{\beta}{\alpha} - 1$$

Proof. The number of nodes of the j+1-th AOP is restrained to be less than its optimum:

$$n_{j+1} \le n_{j+1_o} \Rightarrow \frac{c_{j+1}N^{j+1}}{\alpha(n_j - n_{j,j+1} - 1) - \beta} \le \sqrt{\frac{c_{j+1}N^{j+1}}{\alpha}} \Rightarrow \frac{\sqrt{\alpha c_{j+1}N^{j+1}}}{\alpha(n_j - n_{j,j+1} - 1) - \beta} \le 1 \Rightarrow$$

$$\frac{\beta}{\alpha} + \sqrt{\frac{c_{j+1}N^{j+1}}{\alpha}} \le n_j - n_{j,j+1} - 1 \Rightarrow n_{j,j+1} \le n_j - n_{j+1_o} - \frac{\beta}{\alpha} - 1$$

□

Theorem 13 cannot always hold, because certain requirements on the relative workload sizes of the neighboring AOPs must be satisfied first. In fig. 5 it is obvious that the first AOP must have quite a few more AEMs and greater workload than the second, in order to have a large enough initial idle time for the second AEM's workload to fit in.

Theorem 14. The minimum number of AEMs that an AOP must have in order to be overlapped with a second AOP is larger than the optimum number of AEMs of the second AOP:

$$n_j \ge n_{j+1_o} + \frac{\beta}{\alpha} + 2$$

Proof. Theorem 13 gives the maximum number of shared nodes. However, in order to have overlapping this number must be at least 1:

$$1 \le n_{j,j+1} \le n_j - n_{j+1_o} - \frac{\beta}{\alpha} - 1 \Rightarrow n_j \ge n_{j+1_o} + \frac{\beta}{\alpha} + 2$$

□

The following corollary of theorem 14 establishes the condition on the workloads of the two classes that must be satisfied to accomplish the overlapping.

Corollary 7. In order to have overlapping between two classes, the following condition on their workloads must be met:

$$\sqrt{\frac{c_j N^j}{\alpha}} \geq \sqrt{\frac{c_{j+1} N^{j+1}}{\alpha}} + \frac{\beta}{\alpha} + 2$$

Proof. Directly from theorem 14 and the restriction that: $n_j \leq n_{jo}$. □

Therefore, corollary 7 is a key to choosing the correct neighborhood for the classes, through a process that will search for valid pairs that satisfy it. However, there might be more than one combinations of valid pairs. A plausible heuristic for the allocation of neighbors is to favor the combination that produces the highest total overlapping between the classes, i.e. the sum of the number of overlapping nodes, given by theorem 13, to be maximum.

5.3. Performance and Utilization Improvement

When two AOPs operate with n_j and n_{j+1} AEMs before the overlapping takes place, they have the following options afterwards:

- Operate with the same total number of AEMs (n_j+n_{j+1}), but with better performance (fig. 6a). Should this happen, it means that both AOPs have extended their number of AEMs using the shared nodes and more AEMs means better performance. This requires that the AOPs were not operating with the optimal number of processors, prior to overlapping.

The global performance depends on the performance of the maximum workload class, therefore if the latter operated with its optimum number of AEMs before the overlapping, it cannot operate better afterwards. Thus, global performance improvement is not achieved.

- Operate with fewer total number of AEMs, keeping the same performance (fig. 6b). Should this happen, it means that both AOPs keep their number of AEMs (n_j and n_{j+1}, respectively), some of which are shared. However, keeping the same number of AEMs does not improve performance, but improves utilization instead.

When the above overlapping scheme occurs for every pair of overlapping AOPs, then the total number of AEMs in the system will be decreased. This will cause the total "occupation" time (see definition 5) to be decreased as well. Processor utilization is inversely proportional to the total "occupation" time (definition 6), therefore the overall effect of AEM degradation will be the increase of processor utilization.

- A hybrid approach, where one of the AOPs operates with the same number of processors it had before the overlapping, while the other extends its number of AOPs, sharing some of the nodes of the first AOP (fig. 6c). This will decrease the response time of the second AOP and it will increase the utilization of the resources of both AOPs.

5.4. Non-Uniform Declustering

Another technique to increase the performance and scalability of single-class queries is the non-uniform declustering presented in [3]. According to this scheme, the instances of a class are not uniformly distributed to all the AEMs of the AOP, because this causes a cascaded termination effect (fig. 2). More specifically, the relative response time of each AEM is the same and each AEM starts execution slightly later than the previous one, therefore it will also finish a little later. The non-uniform declustering strategy achieves the concurrent termination of all the AEMs, by having different workload (and therefore different number of objects) on each AEM (fig. 7).

More details on the non-uniform declustering are out of the scope of this paper and can be found in [3], where it is proved by analysis and simulation that this scheme increases by 40% the performance of a single-class. If this intraclass declustering scheme is combined with the hierarchical query processing described in this paper, even better results are going to be achieved.

6. Simulation Results

In this section we present the simulation of the hierarchical query processing on a multiprocessor network of 5 transputers, using CS-Prolog [12] as the simulation language. The same hardware/software platform has been used to build a minimally functional prototype of the PRACTIC system [2]. Simulation was necessary, though, to test every aspect of the hierarchical query processing strategy presented in this paper, because due to limited resources of the implementation platform a large enough configuration for the prototype could not be built. However, most parameters used in the simulation are based on actual values measured on the prototype.

We note here that the target machine is not hierarchical, because what matters is the hierarchical nature of query processing and not the architecture itself. The hierarchical abstract machine is very easily emulated (with a satisfactory performance) on any commercial (flat) shared-nothing MIMD machine, with a re-configurable topology of processors, like the transputer network.

The simulation treats processor initialization and method execution times as "do-nothing" periods of time. Simulation parameters are shown in table 2. The master-processor coordinates the execution by sequentially starting-up query processing at the AOPs. Each AOP in turn coordinates the local execution by sequentially starting-up query processing at each AEM it owns. When AEMs finish processing they inform their AOP by sending a message. When an AOP gathers all the messages from its own AEMs, it informs the master-processor with a message. Finally, when the master-processor receives all the messages from the AOPs, it terminates the global process.

Due to limited resources, we could not perform extensive simulations for a variety of system parameters. Furthermore, we had to compromise our full hierarchical abstract architecture to a more mundane scheme, in order to be able to map a minimal OODB configuration onto our transputer machine. More specifically, the minimal test model we could build was a class-hierarchy with two classes (two AOPs), having the same number of instances, with at least two AEMs per AOP. This makes a total of 7 processors, if we add the master processor (fig 8a). Our machine had only 5 transputers therefore it was not adequate to build even the minimal model.

To resolve this, we decided to sacrifice the separate AOP processors without a great performance loss, by putting the functionality of each AOP to the last AEM of the AOP cluster (fig. 8b). According to this scheme, the last AEM initializes sequentially the rest of the AEMs of the same cluster and initializes itself last. This saves 1 processor per class, therefore we could map our minimal model onto our 5 transputer network. This trick could not be duplicated for the PRACTIC prototype, because the model demands a dedicated processor for the permanent foreground procedure of the active class management [2].

The global query response time we measure starts before master begins initialization and ends after master receives the last terminating message from the AOPs. The results are shown in table 3 and fig. 9 for both query processing strategies. From fig. 9 it is obvious that both processing schemes perform almost similarly, therefore our simulation cannot be used to justify the superiority of the hierarchical processing strategy. Unfortunately, due to our limited resources we could not set up an adequately large experiment.

What can our simulation prove though is that the analysis of section 4.3.3 for condition (17) is correct, because it implies that if:

$$\frac{1}{n}\frac{\beta}{\alpha} = \frac{1-f_{max}}{k}$$

then the two strategies should have the same performance. If the simulation parameters are substituted from table 2 into the above equation, it is verified. This gives us a clue about the correctness of the analysis of this paper.

The correctness of the analysis for flat query processing has been established by measurements on the PRISMA/DB system [17] and by simulation on our transputer-based system [3]. In table 3 and fig. 9 we compare the simulation measurements with the corresponding theoretical calculations. Although the measured time is very different from the theoretical, the behavior of the curves is similar. The difference between theoretical and simulated results is mainly due to extra overheads caused by program execution, interprocessor communication, etc. However, these overheads add similarly to both strategies, therefore the comparison between the two strategies is not affected.

7. CONCLUSIONS

In this paper, we have studied and simulated the analytic performance of query execution in a hierarchically structured multiprocessor architecture, for the PRACTIC parallel OODB system [2]. The proposed query execution strategy is based on the hierarchical fragmentation of a class-hierarchy into multiple clusters of processors and the uniform distribution of the class instances among the processors of each cluster.

During the hierarchical processing of a query, the master processor sequentially initializes the master processor of each class cluster (AOP), which in turn sequentially initialize all the processors of the cluster (AEMs). However, query execution (after the initialization) proceeds in parallel among the AOPs and the AEMs.

We study analytically such a system and we derive conclusions about its performance, scalability and utilization. The analytic model can be adapted both to main-memory and disk-based database systems. However, the results become significant for main-memory database systems, since the multiprocessor initialization and communication costs are comparable to the actual workload, in such systems.

The performance and scalability of the system is shown to be bounded, but the bound is inherently larger (by a parameterized fraction) than a flat architecture approach that has been adopted by most of the parallel database systems [4, 7, 8, 17], except some extreme cases which have been clearly identified. The results obtained from the analysis are compared to an equivalent analysis for a parallel main-memory relational database system [17] and verified by simulation on a transputer network.

Finally, we have extended the hierarchical fragmentation strategy into an overlapping declustering scheme, where neighboring AOPs share the storage and computational power of neighboring AEMs. This declustering scheme overcomes certain drawbacks of the hierarchical query processing strategy, providing greater processor utilization and/or increasing the performance of query execution for a single class.

Our current work on the theoretical aspect includes the generalization of the overlapping declustering scheme to an arbitrary number of classes with two neighbors. This also includes the development of algorithms for the placement of classes on the abstract PRACTIC machine, according to the heuristics of section 5, in order to achieve the optimal class neighborhood. Finally, the ultimate goal of this research is to combine the hierarchical and overlapping declustering schemes with the non-uniform declustering of objects of a single class [3], to achieve the maximum performance and utilization improvement by getting the best from each technique.

On the practical side we currently extend the first PRACTIC prototype to capture the full functionality model and to measure its performance on a real-world application.

Acknowledgments

We would like to thank the anonymous referees for their valuable comments. The first author is supported by a scholarship from the Greek Foundation of State Scholarships (F.S.S. - I.K.Y.).

REFERENCES

[1] P.M.G. Apers, C.A. van den Berg, J. Flokstra, P.W.P.J. Grefen, M.L. Kersten and A.N. Wilschut, PRISMA/DB: A parallel, main memory relational DBMS, in Ref. [9], 541-554.

[2] N. Bassiliades and I. Vlahavas, "PRACTIC: A Concurrent Object Data Model for a Parallel Object-Oriented Database System", in *Information Sciences* **86**(1-3) (1995) 149-178.

[3] N. Bassiliades and I. Vlahavas, "A Non-Uniform Data Fragmentation Strategy for Parallel Main-Memory Database Systems", in *Proc. 21st International Conference on Very Large Data Bases* , Zürich, Switzerland (1995) 370-381.

[4] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith and P. Valduriez, Prototyping Bubba, a highly parallel database system, in *IEEE Trans. on Knowledge and Data Engin.* **2**(1) (March 1990) 4-24.

[5] L. Böszörményi, J. Eder and C. Weich, PPOST: A parallel database in main memory, *Proc. Database & Expert System Applications*, Athens, Greece (1994) 754-758.

[6] G. Copeland, W. Alexander, E. Boughter and T. Keller, Data placement in Bubba, in *Proc. ACM-SIGMOD Internat. Conf. on Management of Data*, Chicago, USA (1988) 99-108.

[7] D.J. DeWitt, S. Ghandeharizadeh, D.A. Schneider, A. Bricker, H. Hsiao and R. Rasmussen, The GAMMA database machine project, in *IEEE Trans. on Knowledge and Data Engin.* **2**(1) (March 1990) 44-62.

[8] D. DeWitt and J. Gray, Parallel database systems: The future of high performance database systems, *CACM* **35** (1992) 85-98.

[9] M.H. Eich, ed., Special section on main-memory databases, *IEEE Trans. on Knowledge and Data Engin.* **4**(6) (December 1992).

[10] H. Garcia-Mollina and K. Salem, Main memory database systems: An overview, in Ref. [9], 509-516.

[11] P. M. D. Gray, K. G. Kulkarni and N. W. Paton, *Object-Oriented Databases, A Semantic Data Model Approach*, Prentice Hall, 1992.

[12] P. Kacsuk, I. Futo, Multi-transputer implementation of CS-Prolog, *Proc. AI & Comm. Process Architecture*, (Wiley & Sons, 1989) 131-148.

[13] K.C. Kim, Parallelism in object-oriented query processing, in *Proc. 6th Internat. IEEE Conf. on Data Engineering* (1990) 209-217.

[14] W. Litwin and T. Risch, Main memory oriented optimization of OO queries using typed datalog with foreign predicates, in Ref. [9], 517-528.

[15] A.K. Thakore and S.Y.W. Su, Performance analysis of parallel object-oriented query processing algorithms, *Distributed and Parallel Databases* **2**(1) (1994) 59-100.

[16] P. Valduriez, Parallel database systems: Open problems and new issues, *Distributed & Parallel Databases*, 1(2) (Apr. 1993) 137-165.

[17] A.N. Wilschut, J. Flokstra and P.M.G. Apers, Parallelism in a main-memory DBMS: The performance of PRISMA/DB, in *Proc. 18th Internat. Conf. on Very Large Data Bases*, Vancouver, Canada (1992) 521-532.

Figure & Table Captions

Table 1. Comparative example of maximum speed-up and maximum number of processors.

Table 2. Simulation parameters.

Table 3. Simulation results.

Figure 1. A flat abstract architecture for a parallel database system.

Figure 2. The behavior of parallel query execution for a flat query processing strategy.

Figure 3. The hierarchical architecture of the abstract PRACTIC machine (APRAM).

Figure 4. The behavior of hierarchical query execution.

Figure 5. The behavior of hierarchical query execution with overlapping declustering.

Figure 6. Overlapping of neighboring nodes with total AEMs: a) remain constant, b) increasing.

Figure 7. The behavior of parallel query execution for non-uniform data declustering.

Figure 8. The minimal simulation model architecture: a) abstract, b) concrete.

Figure 9. Simulation vs. theoretical results.

	Flat	**Hierarchical**	**Difference**
k	-	3	-
N	$3 \cdot 10^6$	10^6/class	-
c/α	10^{-3}	10^{-3}	-
n_o	55	95	72.72%
S_o	27.39	43.34	58.23%

Table 1

N. Bassiliades and I. Vlahavas

$\alpha = 1$ sec	$k = 2$ classes
$\beta = 1$ sec	$f_{max} = 0.5$
$c = 10^{-3}$ sec	$N = (4 \text{ to } 20) \cdot 10^3$ objects
$n = 4$ processors	

Table 2

N. Bassiliades and I. Vlahavas

N	4,000	8,000	20,000
Flat	13.21	21.76	47.44
Hierarchical	13.04	21.84	48.07
Theoretical	5.00	6.00	9.00

Table 3

N. Bassiliades and I. Vlahavas

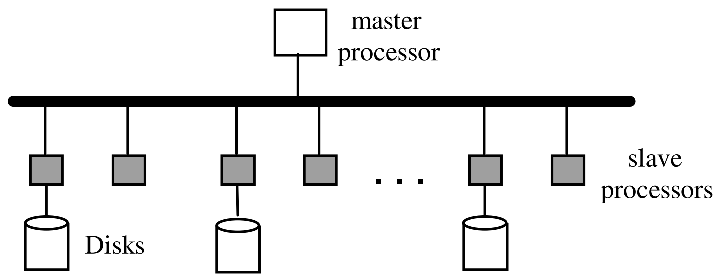

Figure 1

N. Bassiliades and I. Vlahavas

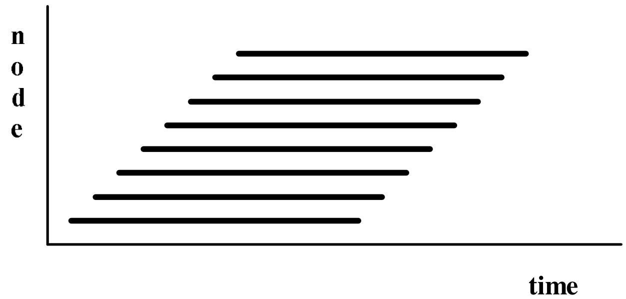

Figure 2

N. Bassiliades and I. Vlahavas

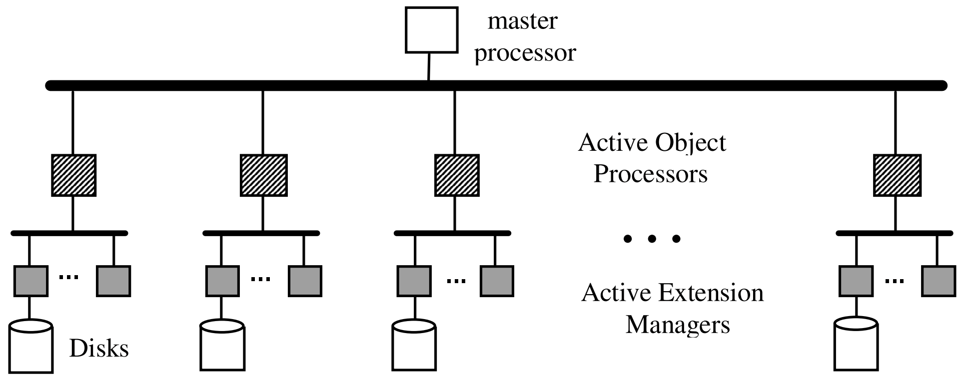

Figure 3

N. Bassiliades and I. Vlahavas

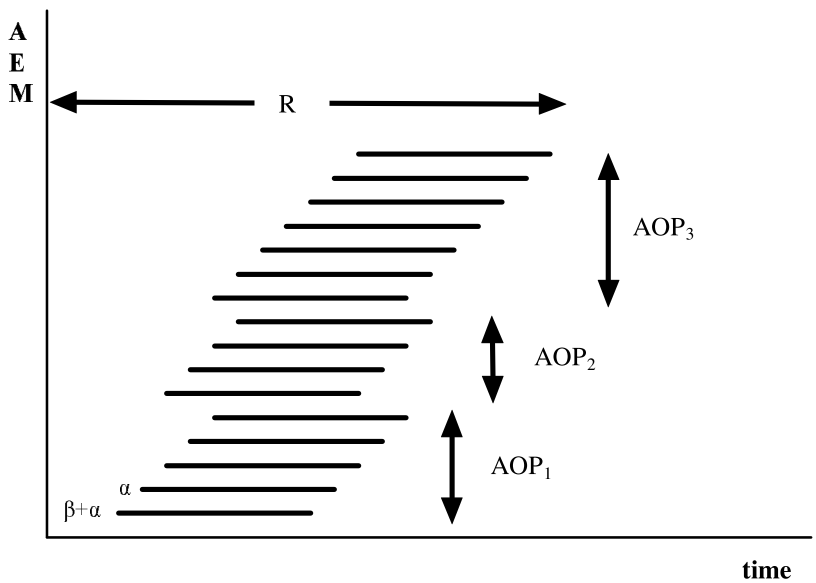

Figure 4

N. Bassiliades and I. Vlahavas

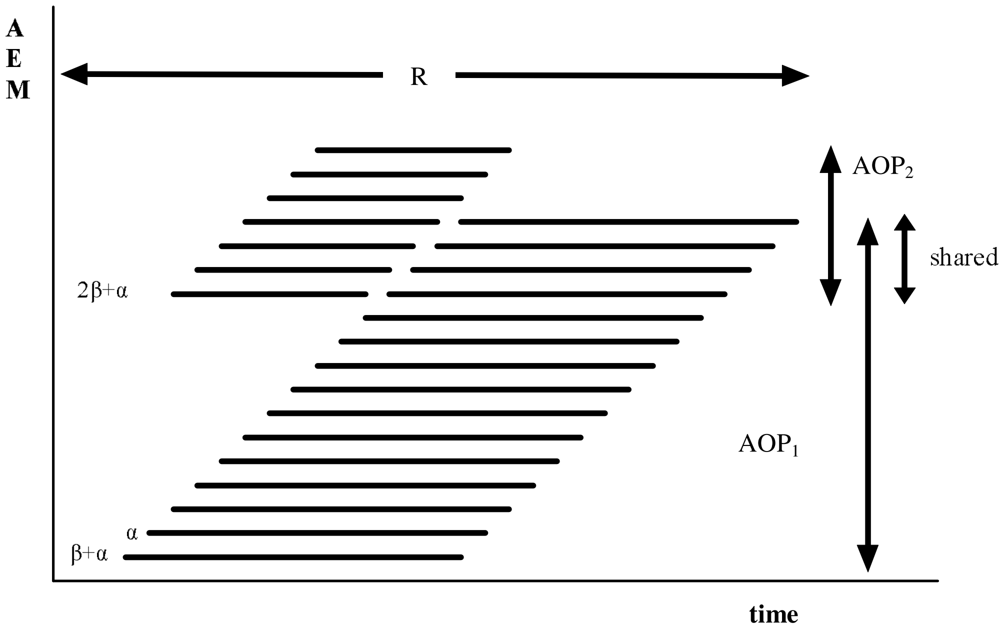

Figure 5

N. Bassiliades and I. Vlahavas

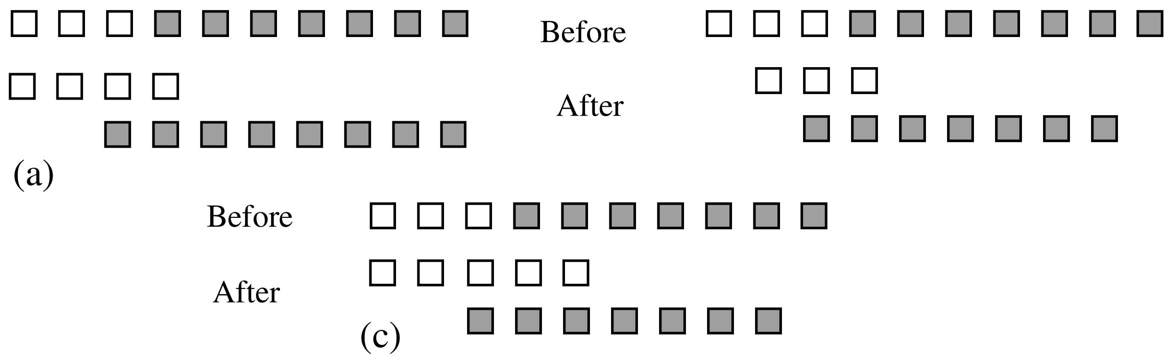

Figure 6

N. Bassiliades and I. Vlahavas

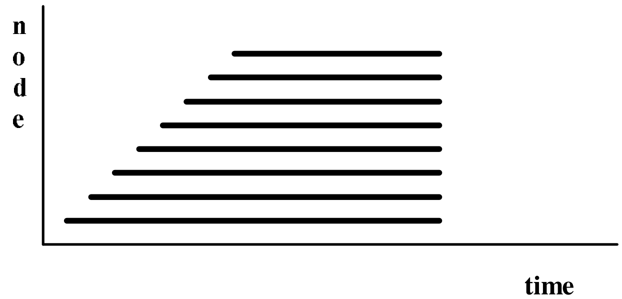

Figure 7

N. Bassiliades and I. Vlahavas

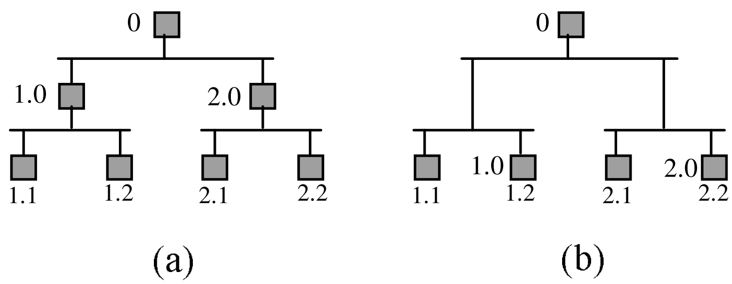

Figure 8

N. Bassiliades and I. Vlahavas

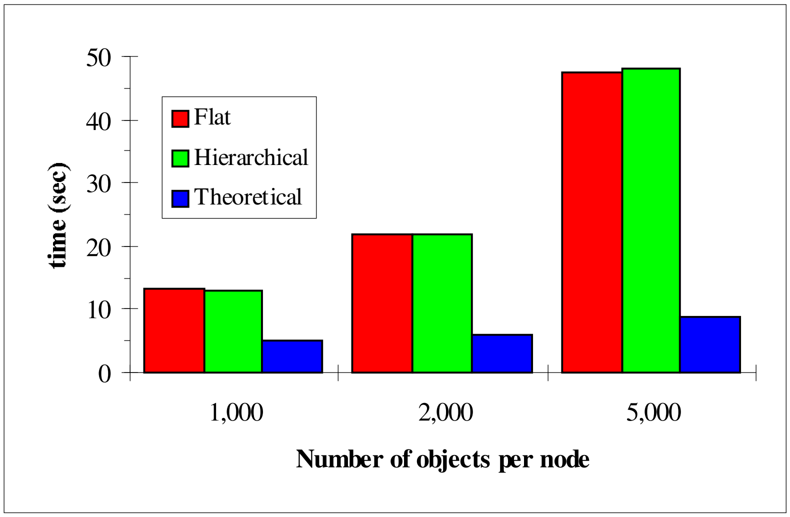

Figure 9

N. Bassiliades and I. Vlahavas