A Non-Uniform Data Fragmentation Strategy for Parallel Main-Memory Database Systems

N. Bassiliades* I. Vlahavas
Dept. of Informatics,
Aristotle University of Thessaloniki,
54006 Thessaloniki, Greece.
nbassili@athena.auth.gr, vlahavas@olymp.ccf.auth.gr

Abstract

In multi-processor database systems there are processor initialization and inter-communication overheads that diverge real systems from the ideal linear behaviour as the number of processors increases. Main-memory database systems suffer more since the database processing cost is small compared to disk-based database systems and thus comparable to the processor initialization cost. The usual uniform data fragmentation strategy divides a relation into equal data partitions, leading to idleness of single processors after local query execution termination and before global termination. In this paper, we propose a new, non-uniform data fragmentation strategy that results in concurrent termination of query processing among all the processors. The proposed fragmentation strategy is analytically modeled, simulated and compared to the uniform strategy. It is proven that the non-uniform fragmentation strategy offers inherently better performance for a parallel database system than the uniform strategy. Furthermore, the non-uniform strategy scales-up perfectly till an upper limit, after which a system re-configuration is needed.

1. Introduction

Modern data-intensive database applications strive for enhanced performance. Many parallel database systems [4, 10, 27] have appeared both as research prototypes [5, 9, 16, 22, 23] and as commercial products [24, 25] to capture the demand for speed in database processing. Another direction for the search of performance improvement for time-lined database applications have been main-memory database systems [12, 13]. The integration of the above two research directions have been studied in the context of PRISMA/DB project [1, 28].

During the last decade, Object-Oriented Databases (OODBs) gained considerable attention [17] mainly because they reduce the "semantic gap" between real world concepts and data representation models. 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.

Recently, in order to overcome the above limitations, parallel OODB systems have been proposed [2, 19, 26, 27], following the research for parallel relational databases. Among them, the PRACTIC parallel main-memory OODB system [2] includes an object data model and an abstract hierarchical multi-processor architecture that its performance on query execution has been studied analytically [3]. The hierarchical architecture has been proved inherently better than a flat one [28].

In this paper we try to improve performance of query execution further, by proposing a non-uniform data fragmentation strategy on the nodes of a multi-processor database system, to compensate for the effect of sequential query start-up at each processor from the coordinator. Results show that this new approach a) improves by 44% the peak performance of the parallel database system, and b) scales-up perfectly until an upper bound, after which a system re-configuration is needed. Furthermore, the strategy can be easily implemented on any multi-processor database system, both relational and object-oriented.

The rest of the paper is structured as follows: section 2 refers to related work on the performance analysis of parallel database systems; section 3 presents the non-uniform data fragmentation strategy; section 4 analyses the performance, fragmentation and hashing of the proposed strategy, while section 5 focuses on system scalability. Section 6 presents simulation results, and finally, section 7 concludes this paper with a summary of the main points and a discussion of future work.

* Supported by a scholarship from the Greek Foundation of State Scholarships (F.S.S. - I.K.Y.).

**Proceedings of the 21st VLDB Conference
Zurich, Swizerland, 1995**

2. Related Work

The performance of query execution in parallel database systems using various data fragmentation strategies has been extensively studied [5, 7, 9, 10, 14, 15]. Most of the systems use full and uniform declustering as data fragmentation strategy, i.e. all relations are distributed over all nodes uniformly, to advantage from the parallel execution of an operator among the partitions.

In the Bubba system [5] it is argued that full and uniform declustering is not always the best strategy [7, 14], since the access frequency of each relation partition may vary, therefore the most frequently accessed partition may become a bottleneck to the system performance. The analysis in [7, 14] concerns multi-user database systems, assuming certain workloads that arise from concurrent database accesses and/or small transactions. Therefore, the purpose of their declustering strategies is to achieve higher transaction throughput, i.e. more transactions per second.

In order to achieve this, they base their declustering strategies not only on the size of each partition, but also on the access frequency of the tuples that are stored within its partition. This kind of non-uniform partitioning results in a uniform workload per partition, which is a pre-requisite for the full exploitation of a multi-processor system.

Our work instead concerns the performance improvement of single large, decision-support queries or batch transactions in a parallel database system. Therefore our purpose is to minimize query response time, sacrificing (probably) memory and disk utilization.

The performance of such queries has also been studied in [6, 28]. More specifically, in [28] it is concluded both theoretically and experimentally for the PRISMA/DB parallel main-memory relational database system, that there is always an upper limit in speed-up. This upper limit is lower in main-memory than disk-based database systems, because main-memory processing is faster and the processor initialization costs are comparable to the actual query processing costs. The limited scalability is due to the fact that the query execution coordinator node initializes only one node each time, sequentially.

The main concept behind our fragmentation strategy is to distribute data non-uniformly into partitions in order to achieve non-uniform workloads per processing node. The non-uniform workload distribution will compensate for the effect of sequential initialization of the processing nodes. As a result all nodes will terminate query processing at the same time, achieving minimal global query response time.

Another major difference of our fragmentation strategy for an OODB with previous approaches for relational databases is that they use hash or range partitioning [7, 14], based on the most frequently accessed attribute of the relation. In contrast, our strategy is based on the most frequently evoked method of the class. Furthermore, the single hashing attribute is the object identifier, since in OODBs objects are usually accessed through that. Thus, multi-attribute declustering strategies [11, 15, 20, 21] do not apply to our case, except if they are adapted for multiple methods.

In the following sub-section, the analytical performance of PRISMA/DB query execution is summarized as background knowledge, because it will be used as a measure of comparison with our analysis.

2.1. Background

The abstract architecture of a parallel shared-nothing database system [8, 28] consists of a flat, linear multi-processor network (fig. 1). There is one master processor that coordinates query execution and several slave processors that execute relational operations on the partition of the relation they store. Relations are fragmented into equally sized partitions for the shake of uniformity and simplicity.

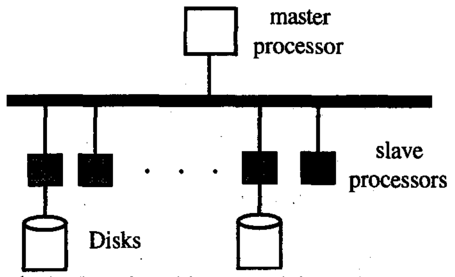

Figure 1. A shared-nothing parallel database architecture

The master processor is assumed to initiate query execution on each slave processor sequentially and then each slave processor proceeds on its own.

Theorem 1. The response time of a uniformly fragmented system is:

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

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. The index u is used to denote that the fragmentation strategy is "uniform".

Theorem 2. The "speed-up" that a uniformly fragmented system offers is:

$$^{u}S = \frac{\alpha + cN}{\alpha n + c\frac{N}{n}}$$

Theorem 3. The optimal configuration for a uniformly fragmented system is:

$$^{u}n_o = \sqrt{\frac{cN}{\alpha}} \tag{2}$$

where the maximum speed-up and the minimum response time are achieved:

$$^{u}S_o = \frac{^{u}n_o}{2} \qquad ^{u}R_o = 2\sqrt{\alpha c N}$$

Proofs. See [28].

In [28] it is argued that in order to increase n_o, i.e. increase the scalability of the system, either c must be increased (as in disk-based DBMSs) or α must be decreased, which is a more rational solution, since it decreases the query response time as well.

In this paper we argue that if the data fragmentation strategy is changed into a non-uniform one, it is possible to increase the scalability (and performance) of a parallel database system, another 44% more than the maximum of theorem 3. In addition to the use of a hierarchical abstract architecture and query execution strategy proposed in [3], our data fragmentation strategy can provide for a very increased performance in a parallel OODB system.

3. Non-Uniform Fragmentation Strategy

In this section we first review the query execution model of the commonly used in parallel database systems uniform strategy and then we discuss the proposed non-uniform data fragmentation strategy, which is the main contribution of this paper.

The theoretical characteristics of the parallel execution of an operation on a uniformly fragmented relation [28], are depicted in fig. 2, where each line represents one operation process. It is inevitable, that the last node starts execution some time after the first one, since the master processor cannot initialize all slaves at once. It is obvious that when a node finishes with local operation execution, it stays idle till the very last node finishes as well; this is a consequence of the uniform distribution of data among nodes, i.e. each node has an equal workload, therefore if it starts late, it will end late.

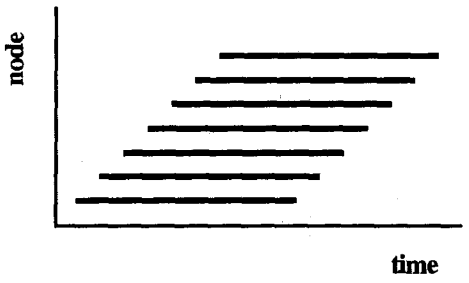

Figure 2. The behaviour of uniform fragmentation

We argue that the last effect can be avoided if the data distribution is not uniform. In fig. 3 there is a schematic approach to our proposal. The main point is that nodes should terminate local query processing at the same (or at least around the same) time, in order not to waste computational power. Since, query execution cannot start at the same time on each node, the amount of data on each processor should be different.

If the processing time per object/tuple is not the same for all objects, the above model can still be used by considering workload distribution instead of data distribution. Therefore, in the following analysis the term "data fragmentation" can be substituted by the term "workload distribution" without any problem.

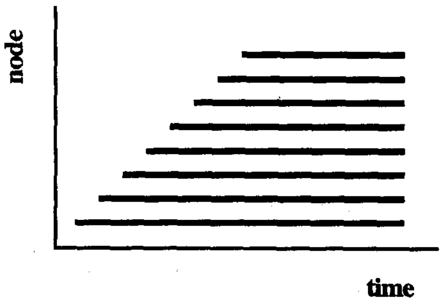

Figure 3. The behaviour of non-uniform fragmentation

4. Analytic Performance Model

In this section an analytic performance model for the proposed query execution model and data fragmentation strategy is developed. The study is based on the analysis of [28], using a slightly different terminology:

- *Class* is used instead of *relation*. A single class isolated from its sub- or super- classes of the class-hierarchy, will only be considered.
- *Method* is used instead of *operation*.
- *Object* is used instead of *tuple*.

Throughout this and later sections the results obtained for the non-uniform fragmentation strategy are compared to the equivalent results for the uniform strategy.

4.1. System Response Time

In this section the global query response time is studied and compared to the uniform strategy.

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

$$W_i = cN_i$$

where N_i is the number of objects on the i-th node. The method execution time per object, is assumed to be common for all objects, or at least very near around an average time.

Definition 2. The initialization time $INIT_i$ for each processor is defined as the sum of the actual initialization time that is spent for each node from the query coordinator processor, plus the waiting time for each node, which equals the sum of the initialization times of all previous (i-1) nodes:

$$INIT_i = \alpha + \alpha(i-1) = \alpha i$$

where α is the initialization cost, common for each node.

Definition 3. The response time R_i of the i-th node is the sum of the node workload and the initialization time:

$$R_i = INIT_i + W_i = \alpha i + cN_i$$

Lemma 1. In order to achieve simultaneous termination according to the non-uniform fragmentation strategy, the number of objects N_i per node should be:

$$N_i = \frac{N}{n} + \frac{n+1-2i}{2}\frac{\alpha}{c}$$

where N is the total number of class instances and n the total number of nodes.

Proof. The proposed data fragmentation assumes that the response time R_i of each node is the same (fig. 3):

$$R_i = R_{i+1}, \forall i \in [1, n) \tag{3}$$

If definition 3 is substituted:

$$\alpha(i+1) + cN_{i+1} = \alpha i + cN_i \Rightarrow N_{i+1} = N_i - \frac{\alpha}{c} \tag{4}$$

Thus, the number of objects at each node is an arithmetic progression, as equation (4) implies, with a common (negative) difference of $-\alpha/c$. The i-th node has N_i objects:

$$N_i = N_1 - (i-1)\frac{\alpha}{c} \tag{5}$$

To calculate N_1, the objects in all nodes are summed and set equal to the total number of objects N:

$$\sum_{i=1}^{n} N_i = N \Rightarrow \frac{n}{2}\left[2N_1 - (n-1)\frac{\alpha}{c}\right] = N \tag{6}$$

Thus N_1 is calculated as:

$$N_1 = \frac{N}{n} + \frac{n-1}{2}\frac{\alpha}{c} \tag{7}$$

Finally, the substitution of N_1 in (5) yields N_i. □

The inverse operation, i.e. to find in which node a certain object resides, is performed by the hashing function, which is analyzed in a later section.

Corollary 1. The response time ^{nu}R of the system (*nu* stands for "non-uniform") is:

$$^{nu}R = R_1 = \alpha + cN_1 = \alpha\frac{n+1}{2} + c\frac{N}{n} \tag{8}$$

Note that it does not matter which R_i we choose, since they are all the same, as equation (3) suggests.

Theorem 4. The non-uniform fragmentation strategy provides faster response time than the uniform strategy, i.e. it is $^{nu}R < {}^{u}R, \forall n > 1$.

Proof. Equations (8) and (1) differ only in the first part, which is $\alpha(n+1)/2$ and αn, respectively. However, for each $n>1$, it is:

$$n > 1 \Rightarrow n + n > 1 + n \Rightarrow n > \frac{n+1}{2}$$

therefore, it is also $^{nu}R < {}^{u}R$. □

The above result is depicted in fig. 4. We note here that certain assumptions were made for the system parameters used for all the plots. These are shown in table 1, along with the deduced, optimal number of processors for both strategies (explained later). The order of magnitude for the system parameters is based on actual measurements on experimental systems [28].

Lemma 2. The response time is minimal when the number of processors is $^{nu}n_o$:

$$^{nu}n_o = \sqrt{2\frac{cN}{\alpha}} \tag{9}$$

Proof. The response time in equation (8) is a function of the number of processors n and the number of objects N. In this section it is assumed that N is constant. The derivative of ^{nu}R is calculated and set to zero:

$$\left.\frac{d\,^{nu}R}{dn}\right|_{n={}^{nu}n_o} = 0 \Leftrightarrow \frac{\alpha}{2} - c\frac{N}{^{nu}n_o^2} = 0 \Leftrightarrow {}^{nu}n_o = \sqrt{2\frac{cN}{\alpha}}$$

For $n < {}^{nu}n_o$, it is:

$$n < {}^{nu}n_o = \sqrt{2\frac{cN}{\alpha}} \Rightarrow \frac{\alpha}{2} - \frac{cN}{n^2} < 0$$

therefore the derivative is negative, and for $n > {}^{nu}n_o$ it is equivalently positive. Thus at $^{nu}n_o$ the response time is minimal. □

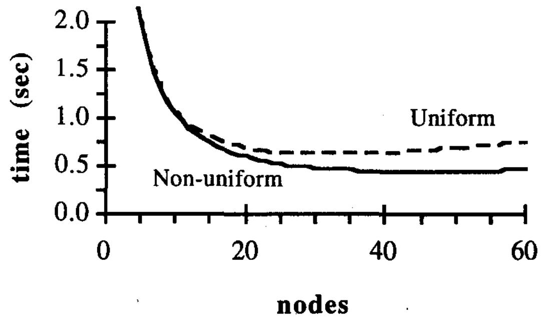

Figure 4. Global query response time

$a = 10^{-2}$ sec
$c = 10^{-5}$ sec
$N = 10^6$ objects
$^{u}n_o = 32$ $\quad$ $^{nu}n_o = 45$

Table 1. Assumptions for system parameters

Theorem 5. The non-uniform fragmentation strategy offers 44% better scalability than the uniform strategy.

Proof. Comparing the result of lemma 2 with the optimal number of processors for the uniform strategy (equation 2):

$$^{nu}n_o = \sqrt{2}\,^{u}n_o$$

it is derived that the non-uniform fragmentation strategy is inherently more scalable than the uniform strategy, by about 44% ($\sqrt{2} \approx 1.44$). □

Corollary 2. If $^{nu}n_o$ is substituted in equation (8), the minimum response time of the non-uniform strategy is found to be:

$$^{nu}R_o = \sqrt{2}\sqrt{\alpha c N} + \frac{\alpha}{2}$$

or in other words, 44% less[1] than the uniform fragmentation strategy (theorem 3).

[1] It is assumed that the constant term $\alpha/2$ becomes insignificant for large databases ($N \gg \alpha$).

4.2. Scalability Limitation

The number of objects on each node, as it is calculated using lemma 1, decreases linearly with i. Since N_i is physically constrained to be greater than zero, i cannot grow indefinitely.

Theorem 6. The number of objects per node N_i is positive for every $i \leq n$.

Proof. Since N_i is a decreasing function of i, it suffices to show that $N_n > 0$, for every plausible value of n, i.e. for each $n \leq {}^{nu}n_o$.

$$N_n > 0 \Leftrightarrow \frac{N}{n} + \frac{n+1-2n}{2}\frac{\alpha}{c} > 0 \Leftrightarrow n(n-1) < 2\frac{cN}{\alpha}$$

If ${}^{nu}n_o$ is substituted (equation 9), the above becomes:

$$n^2 - n < {}^{nu}n_o^2$$

The in-equality is true, since:

$$n \leq {}^{nu}n_o \Leftrightarrow n^2 \leq {}^{nu}n_o^2 \Leftrightarrow n^2 - n < n^2 \leq {}^{nu}n_o^2$$

Thus $N_i > 0$, for every $i \leq n \leq {}^{nu}n_o$. □

Theorem 7. The minimum number of objects that can reside at the last (n-th) node, which is also the very lowest number of objects that can reside on any node, is $\alpha/2c$.

Proof. Following lemma 1, the last node has N_n objects:

$$N_n = \frac{N}{n} - \frac{n-1}{2}\frac{\alpha}{c}$$

The derivative of the above quantity:

$$\frac{dN_n}{dn} = -\frac{N}{n^2} - \frac{1}{2}\frac{\alpha}{c} < 0$$

suggests that N_n is a decreasing function of n, therefore N_n is minimum when n is maximum:

$$N_{n_o} = \frac{N}{\sqrt{2\frac{cN}{\alpha}}} - \frac{\sqrt{2\frac{cN}{\alpha}} - 1}{2}\frac{\alpha}{c} \Rightarrow N_{n_o} = \frac{\alpha}{2c} \quad □$$

The above result suggests that it is not possible to add more nodes to the system than the optimal number of nodes ${}^{nu}n_o$, because equation (4) would yield a negative number of objects for the new node, subtracting the quantity α/c.

4.3. Interpretation of the Optimal Case

An interesting conclusion to be drawn from the interpretation of the optimal case is the following theorem.

Theorem 8. When the global query response time is minimal, the average initialization time per node equals the average workload per node.

Proof. Transforming equation (9) it is derived that:

$${}^{nu}n_o = \sqrt{2\frac{cN}{\alpha}} \Rightarrow \alpha\frac{{}^{nu}n_o}{2} = \frac{cN}{{}^{nu}n_o} \qquad (10)$$

From definition 2 it is derived that the average initialization time is:

$$\langle INIT_i \rangle = \frac{1}{n}\sum_{i=1}^{n} \alpha i = \alpha\frac{n+1}{2}$$

The average initialization time calculated above differs from the quantity in the left-hand side of (10) by a constant factor of $\alpha/2$ which can be ignored.

The average workload per node can be calculated using definition 1:

$$\langle W_i \rangle = \frac{c}{n}\sum_{i=1}^{n} N_i = \frac{c}{n}\left(\frac{N}{n}n + \frac{n(n+1)}{2}\frac{\alpha}{c} - \frac{n(n+1)}{2}\frac{\alpha}{c}\right) \Rightarrow$$

$$\langle W_i \rangle = c\frac{N}{n} \quad □$$

The above theorem implies that if more nodes are added to the system, then the initialization time will overpower the method processing time and the performance will degrade.

Following the same line of reasoning for the uniform fragmentation strategy [28]:

$$\alpha\,{}^{u}n_o = \frac{cN}{{}^{u}n_o} \qquad (11)$$

it is derived that at the optimal case, the maximum initialization time of a node (the last node) equals the common (and also average) workload per node.

The qualitative difference between the uniform and non-uniform fragmentation cases is that in the uniform case (equation 11) the scalability of the system is limited to nodes that have more (or the same) workload than initialization time, while in the non-uniform case (equation 10) the scalability of the system is extended beyond those nodes that spend the same time with the useful working time, waiting. The "average" initialization time implies that there are nodes with initialization time more than the average working time (fig. 11).

4.4. 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 stay idle. This always equals the total number of processors, times the response time of query execution at the "slowest" node, which is also the system's response time:

$$T_{tot} = Rn$$

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 3. The processor utilization is calculated as follows, if definitions 4 to 6 are combined:

$$u_{avg} = \frac{cN}{Rn}$$

Corollary 4. The processor utilization for the two fragmentation strategies is:

$$^{u}u_{avg} = \frac{cN}{\alpha n^2 + cN}$$

$$^{nu}u_{avg} = \frac{cN}{\alpha \frac{n(n+1)}{2} + cN}$$

if equations (1) and (8) are substituted into u_{avg}.

Theorem 9. Processors are used more productively in the non-uniform fragmentation strategy, than the uniform one (fig. 5).

Proof. The denominator of $^{nu}u_{avg}$ is less than the denominator of $^{u}u_{avg}$ for each $n>1$, (see proof of theorem 4), thus it is $^{nu}u_{avg} > ^{u}u_{avg}$, for each $n>1$. □

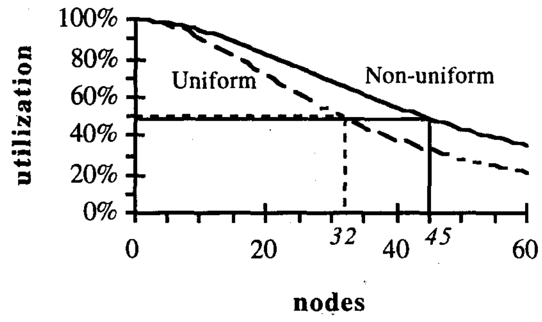

Figure 5. Processor utilization

Theorem 10. At the minimum response time case, the non-uniform distribution has worse utilization than the corresponding case of the uniform distribution.

Proof. If the optimal number of processors, for both strategies (equations 2, 9) is substituted into the average processor utilization, for both strategies, it becomes:

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

$$^{nu}u_{avg_o} = \frac{cN}{\alpha\frac{^{nu}n_o(^{nu}n_o+1)}{2} + cN} = \frac{cN}{cN + \sqrt{\alpha\frac{cN}{2}} + cN} \Rightarrow$$

$$^{nu}u_{avg_o} = \frac{\sqrt{cN}}{\sqrt{\frac{\alpha}{2}} + 2\sqrt{cN}} < \frac{1}{2}$$

Thus, it is proved that $^{nu}u_{avg_o} < ^{u}u_{avg_o}$. □

However, the difference between the optimal cases is insignificant, since usually it is $cN \gg \alpha$, in the denominator of $^{nu}u_{avg_o}$, and utilization becomes $\approx 1/2$. Note also that utilization at the optimal cases ($^{u}u_{avg_o}$, $^{nu}u_{avg_o}$) is computed for different number of processors, while general utilization equations ($^{u}u_{avg}$, $^{nu}u_{avg}$) are computed for the same number of processors. In fig. 5 this small difference is demonstrated.

This similarity at the low-end of processor utilization is based on the logical observation that it is no use to operate a multi-processor system when less than half processor time is used productively. The worst case (for processor utilization) is when half the processor time is spent on "useful" work and the other half is either idle time or is spent on "auxiliary" work, like communication, initialization, etc.

Here we note that at the non-uniform strategy, the last nodes are utilized very little, when working close to the optimal case (fig. 11). This is the trade-off between global query response time minimization and local processor utilization. Some of the individual processors are used too little in order to achieve 44% faster query response.

The analysis of this paper has only considered a single class. If more classes of the OODB schema are considered for query processing (inter-class parallelism [2, 3, 19]), then multiple classes can be interlaced within the same processors. This will achieve better processor utilization both at the local and the global levels, because the idle time of a processor during query processing for a certain class will be used for its neighbouring class. However, this requires new scheduling algorithms and fragmentation strategies to be developed. Furthermore, the analysis of a such a system is more complex, exceeding the scope of this paper.

4.5. The Hashing Function

This section studies how a hashing function can be defined for the proposed strategy, in order to locate where objects reside. Since objects are usually referred to by their object identifier (OID), it is also used here for the definition of the hashing function [2]. But first we define an auxiliary function.

Definition 7. The "accumulating" function $\sigma(i)$ is defined as the total number of objects of all the nodes from the first up to *i*-th (*i*-th included):

$$\sigma(i) = \sum_{j=1}^{i} N_j, i \in [1, n]$$

where N_j is the number of objects in the *j*-th node. Note that the "accumulating" function is defined for any fragmentation strategy.

Corollary 5. From the definition of σ it is understood that $\sigma(n)=N$.

The following definition makes understood why such a function is defined.

Definition 8. The object *z%class* resides on node *i*, if the following condition holds:

$$\sigma(i-1) < z \leq \sigma(i)$$

where *z* is a positive integer and $i \leq n$.

The following theorem establishes the relationship between the "accumulating" function and the hashing function.

Theorem 11. The hashing function $H(z)$ is related to the inverse "accumulating" function as follows:

$$H(z)=\lceil \sigma^{-1}(z)\rceil$$

Proof. We must first prove that the inverse function σ^{-1} of the "accumulating" function σ always exists. From the definition of σ:

$$\sigma(i+1)-\sigma(i)=\sum_{j=1}^{i+1}N_j-\sum_{j=1}^{i}N_j=N_{i+1}>0$$

since the number of objects per node is always a positive integer. Thus σ is an increasing function of i, therefore its inverse function is always defined and is also an increasing function of i.

The hashing function returns the i-th node that a given object *z%class* resides. Definition 8 combined with the above result about σ^{-1} gives the condition that holds:

$$z\le\sigma(i)\Leftrightarrow\sigma^{-1}(z)\le\sigma^{-1}(\sigma(i))\Leftrightarrow i\ge\sigma^{-1}(z)\Leftrightarrow$$

$$i=\lceil\sigma^{-1}(z)\rceil=H(z) \qquad \square$$

It is clear now that in order to calculate the hashing functions of both strategies, we must first calculate the corresponding "accumulating" functions and their inverses.

Lemma 3. The "accumulating" function σ for the uniform fragmentation strategy is:

$${}^{u}\sigma(i)=\frac{N}{n}i$$

Proof. Each node has the same number of objects N/n. We substitute this into the sum in the definition of σ:

$${}^{u}\sigma(i)=\sum_{j=1}^{i}\frac{N}{n}=\frac{N}{n}i \qquad \square$$

Theorem 12. The hashing function for the uniform fragmentation strategy is:

$${}^{u}H(z)=\left\lceil\frac{z}{N/n}\right\rceil$$

Proof. The inverse "accumulating" function is calculated from lemma 3:

$$z={}^{u}\sigma(i_z)=\frac{N}{n}i_z\Leftrightarrow{}^{u}\sigma^{-1}(z)=i_z=\frac{z}{N/n}$$

This, combined with theorem 11, proves the theorem. $\square$

Lemma 4. The σ function for the non-uniform fragmentation strategy is:

$${}^{nu}\sigma(i)=\frac{\alpha}{2c}\left(\frac{n_o^2}{n}+n-i\right)i$$

Proof. We substitute N_j from lemma 1 into the sum in the definition of σ:

$${}^{nu}\sigma(i)=\sum_{j=1}^{i}\left(\frac{N}{n}+\frac{n+1-2j}{2}\frac{\alpha}{c}\right)=\frac{Ni}{n}+\frac{i(n-i)}{2}\frac{\alpha}{c}\Rightarrow$$

$${}^{nu}\sigma(i)=\frac{\alpha}{2c}\left(2\frac{cN}{\alpha n}+n-i\right)i$$

and then we replace the maximum number of processors ${}^{nu}n_o$ (equation 9) and the theorem is proved. $\square$

Theorem 13. The hashing function of the non-uniform fragmentation strategy is:

$${}^{nu}H(z)=\left\lceil\frac{1}{2}\left(\frac{n_o^2}{n}+n-\sqrt{(\frac{n_o^2}{n}+n)^2-\frac{8cz}{\alpha}}\right)\right\rceil$$

Proof. In order to calculate σ^{-1}, the following equation must be solved:

$$z={}^{nu}\sigma(i_z)\Leftrightarrow z=\frac{\alpha}{2c}\left(\frac{n_o^2}{n}+n-i_z\right)i_z\Leftrightarrow$$

$$i_z^2-\left(\frac{n_o^2}{n}+n\right)i_z+\frac{2c}{\alpha}z=0 \qquad (12)$$

The above equation has two real numbers as solutions:

$$i_z=\frac{1}{2}\left(\frac{n_o^2}{n}+n\pm\sqrt{(\frac{n_o^2}{n}+n)^2-\frac{8cz}{\alpha}}\right) \qquad (13)$$

because its "discriminant" $\varDelta$ is positive[2]:

$$\varDelta=(\frac{n_o^2}{n}+n)^2-\frac{8cz}{\alpha}\ge(\frac{n_o^2}{n}+n)^2-\frac{8cN}{\alpha}=$$

$$=(\frac{n_o^2}{n}+n)^2-4n_o^2=(\frac{n_o^2}{n}+n+2n_o)(\frac{n_o^2}{n}+n-2n_o)=$$

$$=\left[\frac{1}{n}(n_o+n)(n_o-n)\right]^2\ge0$$

The equality holds only when $z=N$ and $n=n_o$, where there is only one solution ($\varDelta$=0).

The solution with the positive sign (equation 13) is rejected, since it leads to a false conclusion:

$$n\ge i_z=\frac{1}{2}\left(\frac{n_o^2}{n}+n+\sqrt{(\frac{n_o^2}{n}+n)^2-\frac{8cz}{\alpha}}\right)>\frac{1}{2}\left(\frac{n_o^2}{n}+n\right)\Rightarrow$$

$$n^2>n_o^2\Rightarrow n>n_o$$

Therefore, the inverse "accumulating" function ${}^{nu}\sigma^{-1}$, is the solution of equation (12) with the negative sign (equation 13). $\square$

Comparing theorems 12 and 13, it is clear that the hashing function of the non-uniform fragmentation strategy is more complex than the uniform one, but the overhead is insignificant compared to the performance improvement.

5. Analytic Model of System Scalability

In the previous section the performance of the system has been studied according to the response time. This section focuses more on the extensibility of the system, i.e. its ability to evolve both in terms of processor numbers and database size, using two metrics: "speed-up" and "scale-up" [10, 27].

[2]We use the inequality $z\le N\Rightarrow -z\ge -N$ and equation (9).

5.1. Processor-only Scale-up

The performance of the system depends heavily on the number of processors, as equation (8) demonstrates. In order to improve performance new nodes are added to the system, while the database size remains fixed. The appropriate quantity that is used to measure how well the system "scales-up" in this case is called "speed-up".

Definition 9. "Speed-up" is the fraction of the query response time of sequential query processing divided by the equivalent time on a parallel system:

$$S(n) = \frac{R_{seq}(N)}{R_{par}(n, N)}$$

where $R_{seq}(N)$ is the response time of the sequential execution of a query on N objects, while $R_{par}(n,N)$ is the response time of the parallel execution of the same query, on the same number of objects, distributed over n processors.

Definition 10. A system speeds-up "ideally" if S is a linear function of n [10].

Definition 11. The response time of the sequential query execution is always the same and equals the total workload, plus the initialization time of a single processor:

$$R_{seq}(N) = \alpha + cN \tag{14}$$

Corollary 6. The speed-up of the non-uniform strategy is (from definitions 9-10 and corollary 1):

$$^{nu}S = \frac{\alpha + cN}{\alpha\frac{n+1}{2} + c\frac{N}{n}}$$

Theorem 14. The non-uniform fragmentation strategy offers better speed-up than the uniform strategy (fig. 6).

Proof. From theorem 4 and definitions 9 and 11:

$$^{nu}R < {}^{u}R \Rightarrow \frac{R_{seq}}{^{nu}R} > \frac{R_{seq}}{^{u}R} \Rightarrow {}^{nu}S > {}^{u}S \qquad \square$$

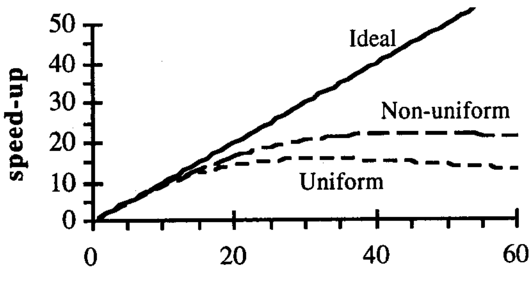

Figure 6. Speed-up

Theorem 15. The maximum speed-up that the non-uniform strategy offers is 44% more than the uniform strategy.

Proof. Since the minimum response time is achieved for n_o processors, speed-up is maximum for the same number of processors, for both strategies. Maximum speed-up for the non-uniform strategy is calculated by substituting $^{nu}n_o$ (equation 9) in ^{nu}S:

$$^{nu}S_o = \frac{^{nu}n_o}{2} = \frac{\sqrt{2}}{2}{}^{u}n_o = \sqrt{2}\,{}^{u}S_o \tag{5.1.4}$$

or in other words, 44% more speed-up than the uniform fragmentation strategy (theorem 3). $\square$

Notice that even if the two fragmentation strategies offer different performances, both at the maximum speed-up case achieve half of the linear speed-up [28].

5.2. Data-only Scale-up

The response time of both fragmentation strategies depends also on the number of the objects N (equations 1, 8). In the previous sections it was assumed that N was constant. In this section we consider the case where more objects are added to the system, but these are stored on existing nodes, i.e. no new nodes are added to the system. Speed-up is no longer a valid metric, since the system remains constant, so the response time is studied instead.

Theorem 16. The response time grows linearly with the number of added objects, with the same rate for both strategies.

Proof. We differentiate the response times for both strategies using equations (1), (8):

$$\frac{d^{u}R}{dN} = \frac{d^{nu}R}{dN} = \frac{c}{n} \qquad \square$$

New objects must be added uniformly to existing nodes, in a round-robin manner, as figs. 7 and 8 show.

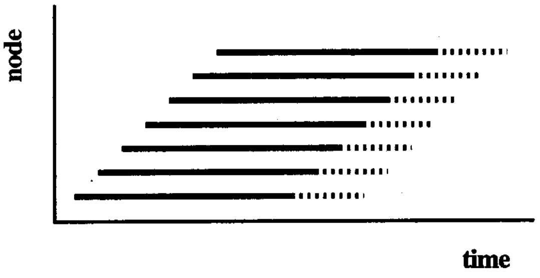

Figure 7. Data-only scale-up (uniform)

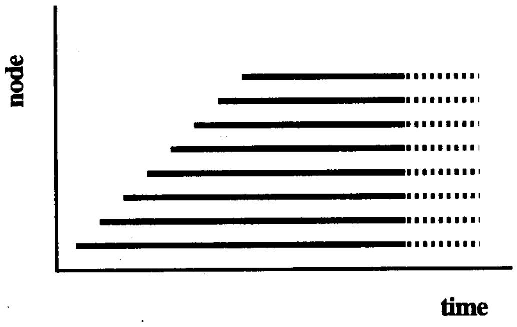

Figure 8. Data-only scale-up (non-uniform)

5.3. Processor and Data Scale-up

This section considers the case where new data are stored on new nodes that are added to the system. This is called "system scale-up" [10] and its purpose is to achieve the same performance despite data increase. The speed-up is no longer a valid measure, since the sequential and the parallel performances are not achieved with the same data size. Instead, "scale-up" L is used to measure the parallel system's "goodness".

Definition 12. "Scale-up" is the fraction of the query response time of sequential query processing divided by the equivalent time of a larger query on a parallel system:

$$L(n) = \frac{R_{seq}(N)}{R_{par}(n, N')}$$

where $R_{seq}(N)$ is the response time of the sequential system on N objects and $R_{par}(n,N')$ is the response time of the parallel system with n nodes and N' objects ($N'>N$), where N' also depends on n and the rest of the known system parameters.

Definition 13. A system scales-up "ideally" if L equals 1 (or close) for all n's [10].

Lemma 5. The "scale-up" of the uniform strategy is:

$$^{u}L = \frac{\alpha + cN}{\alpha n + cN}$$

Proof. According to the uniform fragmentation strategy, when there are N objects stored on a single node, then there should be $N'=nN$ objects stored on n nodes, since the system configuration insists that an equal amount of objects resides at each node. Therefore, scale-up is, according to definition 12 and equations (14, 1):

$$^{u}L = \frac{\alpha + cN}{\alpha n + c\frac{N'}{n}} = \frac{\alpha + cN}{\alpha n + c\frac{nN}{n}} = \frac{\alpha + cN}{\alpha n + cN} \qquad \square$$

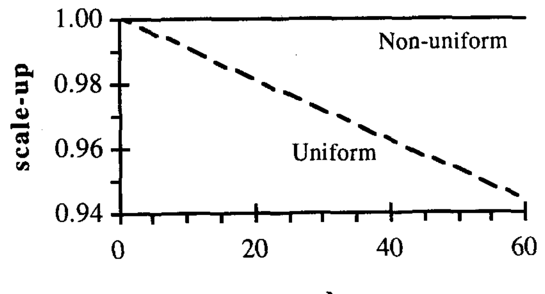

Figure 9. Scale-up

Theorem 17. The uniform fragmentation strategy does not "scale-up" ideally and its "scalability" worsens with the number of nodes (fig. 9).

Proof. From lemma 5 it is clear that $^{u}L<1$, $\forall n>1$. Taking also the derivative of ^{u}L:

$$\frac{d^{u}L}{dn} = -\frac{\alpha(\alpha + cN)}{(\alpha n + cN)^2} < 0$$

we derive that the more nodes (and objects) are added to the system, the worse it scales up. $\square$

The explanation for this behaviour is depicted in fig. 10, where it is shown how new nodes with new data behave in time. Since new nodes are still initialized sequentially by the same master processor, processing begins later at each new node.

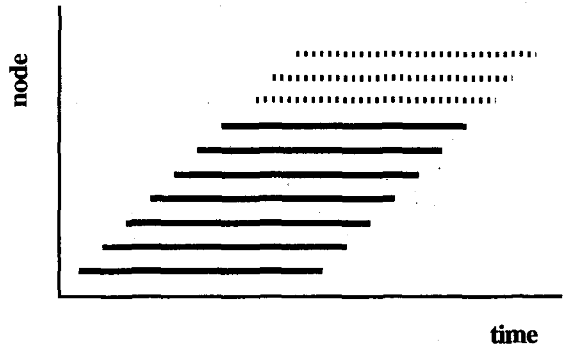

Figure 10. Processor and data scale-up (uniform)

Theorem 18. The response time of a uniformly fragmented system is monotonically increased when new nodes and objects are added to the system:

$$\frac{d^{u}R}{dn} = \alpha > 0$$

Proof. We must calculate the derivative of ^{u}R, bearing in mind that both N and n are variables:

$$\frac{d^{u}R}{dn} = \frac{\partial^{u}R}{\partial n} + \frac{\partial^{u}R}{\partial N}\frac{dN}{dn}$$

The partial derivatives are calculated directly from equation (1). In order to calculate the derivative of N as a function of n, we note that the number of objects per node is constant and equals to N/n, thus:

$$N_i = Q = \frac{N}{n} \Rightarrow N = Qn \Rightarrow \frac{dN}{dn} = Q = \frac{N}{n}$$

We substitute all the derivatives:

$$\frac{d^{u}R}{dn} = (\alpha - c\frac{N}{n^2}) + \frac{c}{n}\frac{N}{n} = \alpha \qquad \square$$

Theorem 19. The non-uniform fragmentation strategy scales-up "ideally" (fig. 9).

Proof. According to definition 12 and equations (14, 8) the "scale-up" of the non-uniform strategy is:

$$^{nu}L = \frac{\alpha + cN_1}{\alpha\frac{n+1}{2} + c\frac{N}{n}}$$

where N_1 is the number of objects that reside in the first node of a non-uniformly fragmented system, if we consider that an one-node system is equivalent to a sequential system.

If we substitute N_1 using equation (7):

$$^{nu}L = \frac{\alpha + c\left(\frac{N}{n} + \frac{n-1}{2}\frac{\alpha}{c}\right)}{\alpha\frac{n+1}{2} + c\frac{N}{n}} = \frac{\alpha + c\frac{N}{n} + \alpha\frac{n}{2} - \frac{\alpha}{2}}{\frac{\alpha}{2} + \alpha\frac{n}{2} + c\frac{N}{n}} = 1$$

According to the above equation and definition 13, a non-uniformly fragmented system scales-up ideally. □

When new nodes with new objects are added to the system (fig. 11), they do not affect the response time of the system, since the fragmentation strategy insists that all nodes terminate at the same time.

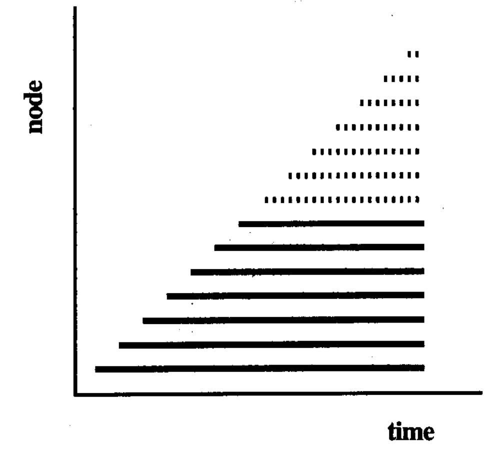

Figure 11. Processor and data scale-up (non-uniform)

Theorem 20. The response time of a non-uniformly fragmented system remains constant when new nodes and objects are added to the system:

$$\frac{d^{nu}R}{dn} = 0$$

Proof. Using the same line of reasoning with the proof of theorem 18, we calculate the derivative of the response time, using equation (8):

$$\frac{d^{nu}R}{dn} = \frac{\partial^{nu}R}{\partial n} + \frac{\partial^{nu}R}{\partial N}\frac{dN}{dn} = \left(\frac{\alpha}{2} - c\frac{N}{n^2}\right) + \frac{c}{n}\frac{dN}{dn}$$

The derivative of N is calculated using equation (6):

$$\frac{dN}{dn} = \frac{d}{dn}\left(nN_1 - \frac{n^2-n}{2}\frac{\alpha}{c}\right) = N_1 - \frac{2n-1}{2}\frac{\alpha}{c}$$

Now N_1 (which is considered as a constant above) is substituted using equation (7):

$$\frac{dN}{dn} = \frac{N}{n} + \frac{n-1}{2}\frac{\alpha}{c} - \frac{2n-1}{2}\frac{\alpha}{c} = \frac{N}{n} - \frac{n}{2}\frac{\alpha}{c}$$

Finally, the above derivative is substituted in the total derivative:

$$\frac{d^{nu}R}{dn} = \frac{\alpha}{2} - c\frac{N}{n^2} + \frac{c}{n}\left(\frac{N}{n} - \frac{n}{2}\frac{\alpha}{c}\right) = 0 \qquad □$$

The scalability of a non-uniformly fragmented system is ideal, but limited to a certain number of objects and nodes. When the system is scaled-up the number of objects at the last node is constantly decreased, until the optimal configuration, with n_o nodes, is reached. Then no further scale-up can take place, because the last node has the lowest allowable number of objects (see section 4.2), that cannot be further reduced (fig. 11). If more objects (and nodes) are to be added to the system there must be a system reconfiguration in order to fit into the nodes.

6. Simulation Measurements

In this section we present the simulation we performed to justify the theoretical analysis of the non-uniform fragmentation strategy. The simulation was performed on a multi-processor network of 5 transputers, using CS-Prolog [18] as the simulation language. The same hardware/software platform is used to build a prototype of the PRACTIC system [2].

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 processing at the slave nodes. When slaves finish processing they inform the master by sending a message. When the master processor gathers all the messages, it terminates the global process.

$a = 1$ sec
$c = 4 \cdot 10^{-3}$ sec
$N = (1..5) \cdot 10^3$ objects

Table 2. Simulation parameters

The global query response time we measure starts before master begins initialization and ends after master receives all the terminating messages from the slave nodes. The results are shown in table 3 for both fragmentation strategies.

	N	1	2	3	4
Non-uniform	1,000	9.47	6.61	**6.57**	-
	2,000	17.90	10.84	9.43	**9.39**
	5,000	43.02	23.55	18.04	**15.88**
Uniform	1,000	10.20	**7.41**	7.70	9.48
	2,000	19.46	12.94	**12.27**	12.70
	5,000	44.60	25.09	20.05	**18.19**

Table 3. Simulation response time measurements (sec)

From table 3, it is clear that the non-uniform strategy outperforms the uniform strategy at all cases. Figs. 12-14 compare graphically the two fragmentation strategies and the simulation time to the theoretical time calculations, for different numbers of objects. Simulation time is depicted with squares, while calculated time with rhombuses. Furthermore, uniform strategy is depicted with solid marks, while non-uniform with outlined symbols.

Although the measured time is different from the theoretical, the behaviour of the curves is identical. The difference between theoretical and simulated results is around 100% and is mainly due to extra overheads caused from program execution, inter-processor communication, etc.

However, these overheads add similarly to both strategies, therefore the comparison between strategies is not affected. Furthermore, the simulated times show minima at the same as the calculated number of processors (table 4).

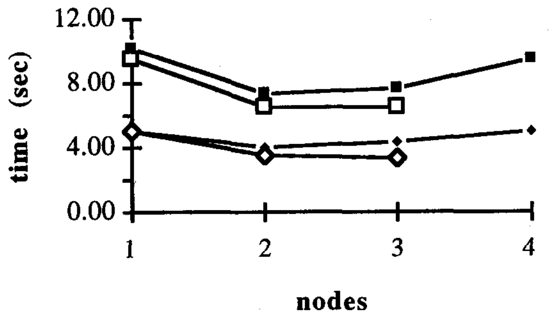

Figure 12. Simulation vs. calculation (1000 objects)

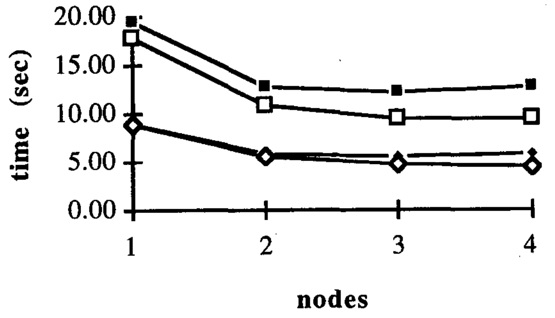

Figure 13. Simulation vs. calculation (2000 objects)

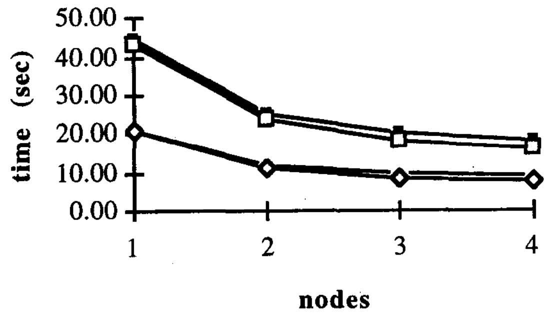

Figure 14. Simulation vs. calculation (5000 objects)

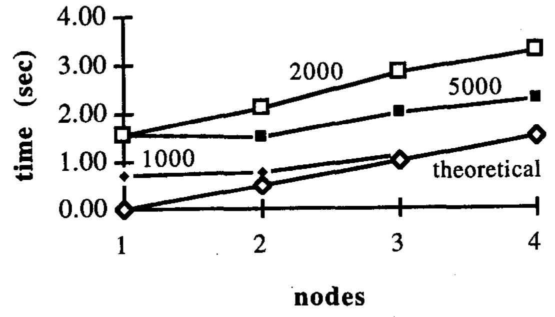

Figure 15. Response time difference between strategies

The difference in time between the two strategies is plotted in fig. 15. The theoretical difference is linear and depends only on the number of processors[3], while the simulated time difference is not perfectly linear but follows a linear trend. Furthermore, fig. 15 shows that the time difference between the two strategies depends on the number of objects, since the overheads affect both the master and the slaves.

N	*Non-uniform*		*Uniform*	
	theore-tical	**simu-lation**	**theore-tical**	**simu-lation**
1,000	2.8	3	2.0	2
2,000	4.0	4	2.8	3
5,000	6.3	>4	4.5	~4

Table 4. Simulation vs. calculation (optimal processors)

As a general remark, fig. 12 shows that the uniform strategy can work non-optimally after 2 processors, while non-uniform strategy cannot operate at all after 3 processors. As the number of objects increases the difference between the two strategies decreases, because the optimal number of processors is more than 4. Unfortunately our hardware resources were limited to 4 processors.

7. Conclusions and Future Work

In this paper, we have studied the analytic performance of query execution in a parallel object-oriented database, using a non-uniform object fragmentation policy in a flat multi-processor architecture. The proposed fragmentation strategy is based on the simultaneous termination of query processing among the processors into which a class is partitioned, despite the sequential initialization of the processes from the master processor. The process initialization time can be significant compared to actual query processing time, in main-memory database systems.

Results show that the new approach improves by 44% the fragmentation ability of the database, and thus provides better query performance than the uniform fragmentation strategy that has been widely used in parallel database systems [28]. Furthermore, the non-uniform fragmentation strategy offers perfect scale-up for both the system and data, until a certain upper limit is reached, after which system re-configuration must occur.

In practical terms the non-uniform fragmentation method can be used in a parallel database system, either relational or object-oriented, based on the most frequently accessed attribute or executed method, respectively. However, it must be noted that if the system is to be scaled-up frequently, then the initial configuration must not be too close to the optimal configuration, because then scalability is strictly limited to the optimal number of processors, in contrast to the uniform fragmentation, where scalability beyond the optimal configuration keeps the system working, non-optimally though.

[3]If equation (8) is subtracted from (1): $\Delta R=\alpha(n\text{-}1)/2$.

Current work includes the study of improved object partitioning strategies that will further increase the upper limit in the scalability of the system. These strategies will combine the non-uniform fragmentation of objects with a hierarchically structured multi-processor architecture [3], to take advantage of both. Furthermore, data of neighbouring classes will be interlaced into common processors, to provide better processor utilization. Finally, all the above fragmentation strategies will be incorporated and tested into the parallel OODB system PRACTIC [2], which is under development.

Acknowledgments

We wish to thank the anonymous referees for their valuable comments on the paper.

References

[1] P. Apers, C. Berg, J. Flokstra, P. Grefen, M. Kersten, A. Wilschut, PRISMA/DB: A parallel, main memory relational DBMS, in [12] 541-554.

[2] N. Bassiliades, I. Vlahavas, PRACTIC: A concurrent object data model for a parallel object-oriented database system, to be published, *Information Sciences*, 1995.

[3] N. Bassiliades, I. Vlahavas, A hierarchical abstract architecture for parallel query execution in an object-oriented database system, submitted for publication, 1994.

[4] B. Bergsten, M. Couprie, P. Valduriez, Overview of parallel architectures for databases, *The Computer Journal*, 36(8) (1993) 734-740.

[5] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, P. Valduriez, Prototyping Bubba, a highly parallel database system, in [22], 4-24.

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

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

[8] Z. Cvetanovic, The effects of problem partitioning, allocation and granularity on the performance of multi-processor systems, *IEEE Trans. Computers*, 36(4) (1987).

[9] D. DeWitt, S. Ghandeharizadeh, D.A. Schneider, A. Bricker, H. Hsiao, R. Rasmussen, The GAMMA database machine project, in [22], 44-62.

[10] D. DeWitt, J. Gray, Parallel database systems: The future of high performance database systems, *Comm. ACM*, 35(6) (June 1992) 85-98.

[11] H. Du, J. Sobolewski, Disk allocation for cartesian product files on multiple disk systems, *ACM Trans. Database Systems*, 7(1) (1982) 82-101.

[12] M. Eich (ed.), Special section on main-memory databases, *IEEE Trans. Knowledge & Data Eng.*, 4(6) (Dec. 1992).

[13] H. Garcia-Mollina, K. Salem, Main memory database systems: An overview, in [12], 509-516.

[14] S. Ghandeharizadeh, D. DeWitt, A multiuser performance analysis of alternative declustering strategies, *Proc. 6th Int. IEEE Conf. Data Eng.* (1990) 466-475.

[15] S. Ghandeharizadeh, D. DeWitt, W. Qureshi, A performance analysis of alternative multi-attribute declustering strategies, *Proc. ACM-SIGMOD Int. Conf. Management of Data* (1992) 29-38.

[16] G. Graefe, Volcano, An extensible and parallel dataflow query processing system, *IEEE Trans. Knowledge & Data Eng.*, 6(1) (1994) 120-135.

[17] P. Gray, K. Kulkarni, N. Paton, *Object-Oriented Databases, A Semantic Data Model Approach* (Prentice Hall, 1992).

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

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

[20] J. Neievergelt, H. Hinterberger, K. Sevcik, The grid file: An adaptable, symmetric multikey file structure, *ACM Trans. Database Systems*, 9(1) (1984) 38-71.

[21] J. Srivastava, T. Niccum, B. Himatsingka, Data declustering in PADMA: A parallel database manager, *Data Eng.*, 17(3) (1994) 3-13.

[22] M. Stonebraker (ed.), Special issue on database prototype systems, *IEEE Trans. Knowledge & Data Eng.*, 2(1) (Mar. 1990).

[23] M. Stonebraker, R. Katz, D. Patterson, J. Ousterhout, The design of XPRS, *14th Int. Conf. Very Large Data Bases*, Los Angeles, CA (Sept. 1988) 318-330.

[24] Tandem Performance Group, A benchmark of non-stop SQL on the debit-credit transaction, *Proc. ACM SIGMOD Conf. Management of Data*, Chicago, IL (June 1988) 337-341.

[25] Teradata Corporation, *DBC/1012 Data Base Computer System Manual*, Doc. No. C10-0001-02, Release 2.0 (Nov. 1985).

[26] A. Thakore, S. Su, Performance analysis of parallel object-oriented query processing algorithms, *Distributed & Parallel Databases*, 2(1) (Jan. 1994) 59-100.

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

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