Multi-dimensional Declustering Methods for Parallel Database Systems

M. Barrena, J. Hernández, J. M. Martínez, A. Polo
Computer Science Department.
University of Extremadura. Spain.
E-mail: barrena@unex.es

P. de Miguel, M. Nieto
Architecture and Technology of Information Systems Department.
Technical University of Madrid. Spain.

Abstract. This paper describes and evaluates new methods for relation declustering in parallel databases. To process queries in parallel, relations are partitioned across multiple processors, typically by using the value of one single attribute. This kind of declustering has resulted in poor performance in the presence of data skew. Alternatively, the work contained herein proposes several strategies to decluster a relation through the use of multiple attributes. To demonstrate the validity of our approach, a thorough performance evaluation is done. The findings demonstrate the effectiveness of this kind of partitioning methods as opposed to traditional ones. In addition we analyze the performance of the different strategies relative to the speed-up and scale-up metrics.

To sum up, performance results reveal that multi-dimensional declustering methods constitute a very promising alternative to conventional one-dimensional methods to partition relations in parallel database systems.

I Introduction.

Most of the prior work on parallelization of relational queries use single-attribute partitioning mechanisms (usually hash or range partitioning) to distribute data among processors [4]. This approach suffers from two main drawbacks which have a critical incidence on the performance of parallel database systems:

1. A potential load imbalance in presence of data skew.
2. The inability to efficiently manage relational queries whose predicate includes an attribute other than the partitioning attribute.

Recent papers on query processing in parallel relational systems [5,13] have addressed only one of the two aforementioned problems. Some of the work dealing with data skew and load balancing [7] focuses mainly on parallel join algorithms.

Provided that input relations are always declustered, the join operation can be considered as a kind of multiscan operator, where tuples in different fragments may have to be compared. From this perspective, to obtain the load balance being sought, an additional mechanism to redistribute tuples is required. Conversely, in uniscan operators (Select, aggregation operators, etc.), in which the processing of each tuple is independent of the processing of any other tuple, the original declustering of relations plays a crucial role in achieving the desired load balance.

This research presents a summary on innovative partitioning techniques which overcome bothpreviously mentioned shortcomings, by following a multi-dimensional approach to decluster relations. In this case, focus is placed on the uniscan operator Select, which is obiquitous in relational queries. The multi-dimensional partitioning under study in multiscan operators is the goal of our current research [8].

Thus, our multi-dimensional partitioning mechanism offers a general proposal which meets the following requirements:

- The ability to efficiently answer queries with predicates involving any number of attributes.
- A support tool for the three basic kinds of sear types, i.e. range, exact match and partial match.
- Tolerance in the presence of data skew.
- Load balance in the assignment of fragments to processors.

II Multi-dimensional declustering strategies.

The distinguishing feature of our proposals on multi-dimensional declustering lies in the implementation of a novel multi-attribute indexing method referred to as the Q-tree, which establishes a balanced division of the tuple space. The so-called Q-tree is balance-paginated and is based on the use of k-d-trees. Our structure incorporates key ideas for data and index node splitting taken from hB-trees [6].

At the leaf level of the Q-tree, each page embodies a data container where tuples can be found. When a data container overflows, it is split by applying the median value for all the tuples on a selected attribute. The minimum utilization ratio of the two resulting data containers ensuing the split can then be defined by the user. In order to support this feature, in addition to the median value for the chosen attribute, the primary key value may be eventually employed to make a balanced split in the event that a resulting containers does not reach the pre-defined utilization ratio.

Although there is a wide range of policies from which to choose the splitting attribute [1], several tests demonstrate the *best attribute* strategy renders markedly high performance. According to this strategy, all attributes which participate in the index are selected as splitting attributes with a similar frequency. What is more, by following the path from the root to any one leaf, each and every attribute can be found the same number of times. Therefore, in order to locate the splitting attribute, an accurate record must be kept of the number of times each one has been used to split a full container. From those that have been used less frequently, we select the one which makes the split better-balanced.

At the next upper level in the Q-tree, every page contains a local k-d-tree which records the successive splits of the data containers. Upon division of a full container, a new node is inserted in the k-d-tree as usual. When an index node overflows, a sub-tree is transferred to a new index page, and the complete path going from the root to the extracted sub-tree is posted to its parent in the Q-tree. A detailed description of the Q-tree can be examined in [1].

To the best of the authors' knowledge, all prior proposals on data partitioning have compulsory used previous information about the workload. However there are scenarios in which this information is difficult to obtain, as well as being a considerably short-term representative. In such cases, the application of a general partitioning method which does not draw any information from the workload (a *blind* method) would be decidedly useful.

Thereby, two fundamental partitioning strategies based on Q-trees will be introduced and analyzed: (i) multi-dimensional partitioning based on an existing index, and (ii) workload-based multi-dimensional partitioning.

The former, *multi-dimensional partitioning based on an existing index* (EI) qualifies as a blind strategy. In this approach, the partitioning process departs from an existing Q-tree and distributes data containers among processors giving three possible policies:

- The *fine grain*, which simply uses the leaf level of the Q-tree by distributing data containers among a number of processors.
- The *medium grain*, in which the tuple space is perceived as a set of sub-spaces, each containing a number of data containers. In this strategy, the algorithm takes each sub-space separately (an index page on parent level of the data) and spreads it among processors in a balanced way.
- The *coarse grain* distribution, which considers the tuple space as merely composed of sub-spaces, regardless of its content. In this case, the algorithm traverses the parent level of the data by assigning complete sub-spaces to the processors. If the assignment lacks evenness, some sub-spaces can eventually be broken up into smaller ones in order to achieve load balance.

The latter, *workload-based multi-dimensional partitioning* (WB) uses a prior knowledge of both the workload and the processing features of the system to finally build a Q-tree, which is used as a partitioning directory. The data containers are assigned to the processors during the final stage of the partitioning process.

With a view to briefly describe the building process of the Q-tree for the WB mechanism, let us assume a workload consisting of a number of selection queries Q_i. Each query Q_i retrieves and processes $\{Q_i\}$ tuples from the database. So as to establish the resources needed for a typical query, we compute the average number of tuples satisfying the query, whose value is termed as $\{Q_{avg}\}$. Once the number P of processors which should participate in a typical query has been identified, the ideal objective is to make all P processors share in the processing of $\{Q_{avg}\}$ tuples.

With this goal in mid, the size of an average data container may be defined as $\{Q_{avg}\}/P$ tuples. At this point, data containers are distributed in a balanced fashion across the P processors, firstly by defining the size of the sub-spaces as $\{Number\_of\_containers\}/P$, and secondly, by using a medium grain distribution approach to allocate data to the processors.

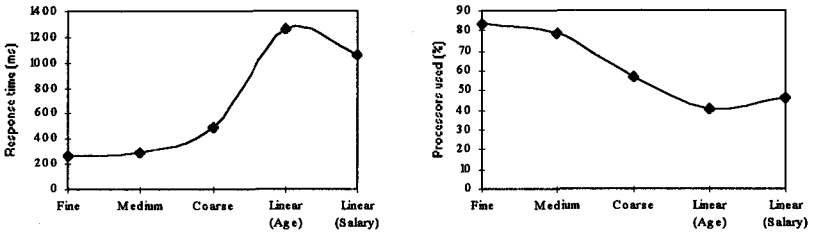
III. Performance Results

In this section, a sampling of relevant performance figures of our proposal will be presented. An in-depth evaluation of the multi-dimensional partitioning strategies based on Q-trees can be consulted in [2].

The performance results shown here have been obtained by using a workload consisting of a 50,000-tuple relation *EMP* with six attributes, two of which, termed *Age* and *Salary*, being search attributes. Queries in the workload have been marked to capture a high degree of data skew. On the other hand, the query execution model and simulation parameters themselves are those conventionally found in shared-nothing parallel dbms. They can also be reviewed in [2].

For the purpose of comparison between multi-dimensional declustering as opposed to one-dimensional or *linear* approach, a mechanism to horizontally

decluster the relation EMP based on Bubba's Extended-Range Declustering Strategy [3], has been implemented. Furthermore, two possible approaches to linearly decluster our EMP relation have been employed. The first one uses *Age* as the primary attribute and *Salary* as the secondary one. By contrast, the second approach uses the same primary and secondary attributes, but in the inverse order. Figure 1a depicts the response time for a typical query in the workload obtained for both of the two strategies considered. As can be noted, the multi-dimensional partitioning significantly outperforms the linear partitioning.



(a) (b)

Figure 1. (a) Response times and (b) percentage of processors involved in the execution of a typical query for multi-dimensional and linear partitioning methods.

The performance differences between linear and multi-dimensional partitioning strategies are better realized by examining the percentage of processors involved in the execution of queries. As figure 1b demonstrates, the average percentage of processors participating in the execution of a query is much greater when multi-dimensional partitioning is used. Nonetheless, the use of a high percentage of resources in a parallel machine does not necessarily mean that the method is efficient. Rather, the careful examination of the way in which such resources are used is required. In this sense, figure 2 draws out the fact that compared to the linear partitioning method, the multi-dimensional method presents higher degrees of balance in the use of resources. It should be noted that a a multi-computer system with eight processors has been employed to conduct this experiment.

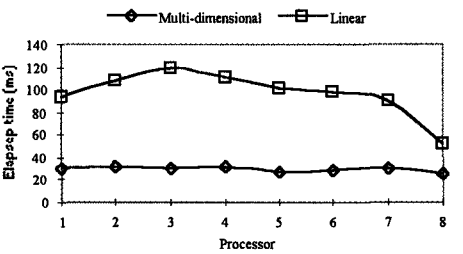
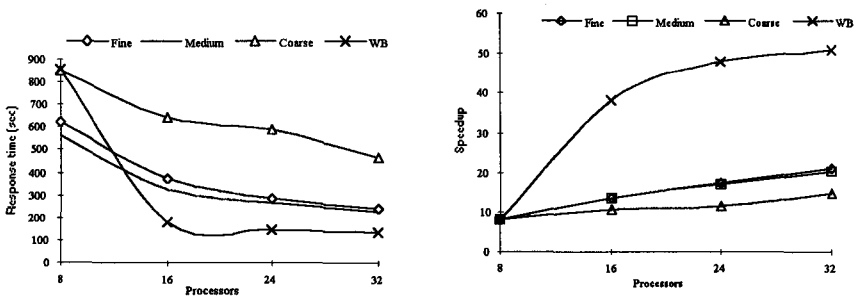


Figure 2. Elapsed time of each processor in multi-dimensional partitioning as compared to linear partitioning.

As can be visualized in figure 2, not only does linear partitioning devote a longer time at each processor in order to execute the query, but it also shows significant

fluctuations from processor to processor in the inverted time. Conversely, multi-dimensional partitioning displays the time taken to be virtually constant among all processors. Therefore, the observable evidence reveals an important feature of multi-dimensional partitioning in parallel systems, that being, load balance even in the presence of data skew.

To measure just how our partitioning models affect the scale-up property of the parallel system, we varied the number of processors from eight to thirty-two, and at the same time increased the source relation from 50K to 200K tuples respectively. Additionally, with regard to the speed-up experiments, we submitted queries in the workload on a 100K-tuple relation by varying the number of processors from eight to thirty-two.



(a) (b)

Figure 3. Speed-up features for EI and WB methods. (a) Response times. (b) Corresponding speed-up curves.

As expected, we observe that the response time decreases as the number of nodes increase (Figure 3a). When WB strategy is used, there is a significant reduction upon passing from eight to sixteen nodes. This situation is mainly caused by the fact that eight no longer remains an optimal number of processors, since the response size of a typical query has actually doubled. A view of figure 3b shows a superlinear speed-up for the WB strategy. In order to realize the differences in this figure, it is neccesary to say that data containers in the WB method are five times larger than data pages in EI strategies.In this way, the greater number of nodes utilized causes more spreaded data containers are. Hence, when the WB method is applied, the seek time overhead is reduced at each node.

Finally, figure 4 displays how the different partitioning strategies are affected by scalability criterion. Here, the ordinate references the time needed to execute a typical query in the workload. The first conclusion we draw from it is directly related to the reliability of the WB strategy. Since the WB Q-tree was precisely designed by taking into account both the resource requirements of a typical query and the number of processors, it is expected that by doubling the resource requirements and the number of nodes simultaneously, the computation will not be affected. However, this reasoning only holds true as long as the load distribution is maintained in a balanced manner. Figure 4 demonstrates hence, the accuracy the WB partitioning method.

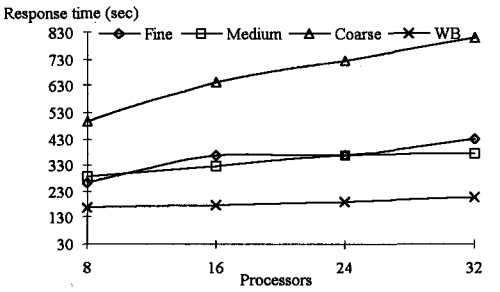


Figure 4. Scale-up for EI and WB partitioning methods.

With regard to the EI strategies, observation clarifies that fine and medium grain distributions behave similarly concerning scalability. It is worth mentioning that even without any prior information on the workload, both EI strategies present a notably operative adaptation to system scalability.

IV Conclusions

In this paper, innovative methods for declustering a relation in parallel database systems have been introduced. Our proposals are based on the resulting division prompted by a multi-attribute search structure (the Q-tree) in the tuple space. This summarized version, which is part of a much larger study, includes some interesting performance figures of the methods employed. In conclusion, the full analysis of the methods used by the research team confirm the successful application of multi-dimensional partitioning based on the use of the Q-tree structure according to the block of observable knowledge presented.

References

[1] M. Barrena, "Técnicas de Particionamiento Multidimensional Basadas en Índices Multiatributos en Bases de Datos Paralelas". *Doctoral Dissertation. Madrid.* Dic, 1995.

[2] M. Barrena et al., "Multi-dimensional Declustering Methods for Parallel Database Systems", *Tech. Report. University of Extremadura.* 1996.

[3] H. Boral et al, "Prototyping Bubba, A Highly Parallel Database System", *IEEE Trans. on Knowledge and Data Engineering*, pp: 4-24, vol 2(1), March 1990.

[4] D. DeWitt, and J. Gray "Parallel Database Systems: The Future of High Performance Database Systems", *CACM*, Vol 35, No. 6, June 1992.

[5] S. Ghandeharizadeh et al. "MAGIC: A Multiattribute Declustering Mechanism for Multiprocessor Database Machines", *IEEE Tr. P&D. Syst.*, Vol. 5, No. 2, May 1994.

[6] Lomet D. B. et al. "The hB-Tree: A Multi-attribute Indexing Method with Good Guaranteed Performance". *ACM Trans. on Database Systems*, Vol. 15, N° 4, Dec. 1990

[7] H. Lu et al. "Query Processing in Parallel Relational Database Systems". *IEEE Computer Society Press*, Los Alamitos, Calif. 1994.

[8] A. Polo et al., "Multi-dimensional Partitioning for Masivelly Parallel Database Systems".*Proc. 3th Euromicro Workshop on Parallel and Distributed Processing. IEEE CS Press.* Jan. 1995.