DATABASE PARTITIONING IN A CLUSTER OF PROCESSORS

EXTENDED ABSTRACT*

Domenico Saccà
CRAI
Rende, Italy

and

Gio Wiederhold
Computer Science Department
Stanford University, Stanford CA

1. INTRODUCTION

In a distributed database system the partitioning and allocation of the database over the processor nodes of the network is a critical aspect of database design effort. A poor distribution can lead to higher loads and hence higher costs in the nodes or in the communication network, so that the system cannot handle the required set of transactions.

We consider here a system where multiple processors are clustered at one location in order to increase the system's processing capability. The local network used in such a system has typically a high communication bandwidth but any single processor will have inadequate processing and input-output capacity to deal with the transaction load. We develop and evaluate algorithms which perform in a computationally feasible manner the design steps of partitioning and allocation of a database to the processors.

More precisely, we investigate the optimal non-redundant partitioning and allocation of a database to a number of processors nodes. Given is a set of source relations of the database and their attributes and a set of transactions which are to be executed during some period of interest. A transaction performs operations on some subset of the database. The initial network node, associated with every transaction, is not prespecified, but to be assigned as part of the design.

This problem differs from the problem in distributed systems where the processors are remote from each other and, presumably, close to their users. In those systems a transaction enters a known, local processor node, and the final response is issued from that node as well. The unclustered model with local transactions has been treated previously in the literature, for instance in [Ap82] and [CNW81].

We model the content of the database as a collection of relations. The given conceptual relations may be too large to be effectively assigned to single processors. We will consider initially how the relations can be fragmented, and will then allocate those fragments to the processor nodes. The possibility that files may be fragmented is not considered in treatments of the file allocation problem, as surveyed for instance in [DoFo82].

Once the database is put into operation each transaction will access some subset of the tuples and the attributes of each original relation. In order to complete transactions which do not find all their data on the same processor, a scheduling algorithm is invoked which optimizes the processing over the network. We do not investigate this scheduling algorithm itself, but simply assume that it exists, and that we can use it in order to obtain the costs for the execution of a given transaction over a proposed database allocation for some specified but fictitious processor network.

In this presentation the following section will formulate our problem precisely. In Sec. 3 we show the intrinsic complexity of the problem, present the heuristic *greedy with first-fit* algorithms we propose, and prove statements about their behavior. The conclusions we draw from this work are presented in Sec. 4. Further background material, proofs, conjectures on the feasibility of the solutions, design algorithms, and evaluations can be found in [SW83].

2. FORMULATION OF THE DATABASE PARTITIONING PROBLEM

2.1 The General Database Partitioning Problem

We investigate the optimal non-redundant partitioning and allocation of database in a cluster of processes. Given is $\mathbf{P} = \{P_1, \ldots, P_p\}$, a small set of processor nodes of a densely and reliably interconnected network. Associated with every processor P_i of $\mathbf{P}$ is the input-output capacity CI_i, expressed in terms of maximum number of blocks that can be processed, and the processor capacity CC_i, expressed in terms of maximum number of cycles of the processor. Furthermore, associated with each pair of processors (P_i, P_j) is the communication capacity CM_{ij}, expressed in terms of maximum volume of messages that can be transmitted between the two processors.

We also have a set of source relations $\mathbf{R} = \{R_1, \ldots, R_r\}$. We denote the set of attributes of each relation R_i of $\mathbf{R}$ by A_i. We can split the relations into fragments for allocation to processors in order to satisfy the capacity constraints of those processors.

*This work was performed and supported at the IBM Research Laboratory, San Jose CA, while the authors were respectively on leave from CRAI and on partial leave from Stanford University.

Fragments Traditional terminology in allocation for distributed systems uses the notions of replication, and vertical and horizontal partitioning. Vertical partitioning allocates columns of a relation over distinct nodes. Horizontal partitioning allocates rows of a relation over distinct nodes. In this analysis we do not consider replication of relations, although tuple-identifiers (TIDs) [Codd80] will be replicated as a byproduct of vertical partitioning.

By combining vertical and horizontal partitioning, we can associate to any relation R_i of **R** a set of fragments $F(R_i) = \{F_{i_1}, \ldots, F_{i_h}\}$. Given a set of fragments $F(R_i)$ for all R_i in **R**, we denote by $\mathbf{F} = \{F_1, \ldots, F_f\}$ a set of fragments such that

$$\mathbf{F} = \bigcup_{i=1}^{r} (F(R_i))$$

. Since we may define several different fragmentations of any relation R_i in **R** (for instance, by choosing different subsets of A_i for the vertical partitioning, or by providing different predicate expressions on some attribute of A_i for horizontal partitioning) there could be several sets of fragments **F** associated with **R**.

Transactions A set of transactions $\mathbf{T} = \{T_1, \ldots, T_t\}$ perform some operations on **R** (or on **F** if we refer to a set of fragments of **R**). Each transaction T_i of **T** is executed during some period of interest with frequency q_i and performs operations op_i on some subset $\mathbf{X}_i$ of **R** (or **F**). In general, a transaction T_i in **T** will access only some rows and some columns of each relation (or fragment) X_{i_j} in $\mathbf{X}_i$, thus it performs operation only on a fragment of X_{i_j}. Associated with each transaction T_i in **T** is an initial network node P_{T_i}, which is not prespecified, but to be assigned as part of the design.

Allocation Given the set of transactions **T** and a some set of fragments **F**, let $\mathbf{O} = \mathbf{T} \bigcup \mathbf{F}$ be the set of objects to be allocated on the network. Furthermore, given a set of fictitious processor nodes $\mathbf{N} = \{N_1, \ldots, N_n\}$, where **N** not necessarily equals **P**, an allocation of **O** to **N**, denoted by $\mathbf{L(O, N)}$, is a mapping of **O** into **N**. In the case that $\mathbf{N} = \mathbf{P}$, the allocation $\mathbf{L(O, N)}$ represents a possible real allocation of the objects (fragments and transactions) to the processors **P** of the network.

Objective The objective of this analysis is to find an allocation design **L** for the set of fragments **F** and the set of transaction **T**, so that an aggregate cost function is minimized, while the capacity constraints $\{CI_i, CC_i, CM_i, i = 1, \ldots, p\}$ are observed. The elements of the cost are the load parameters produced at execution time by a transaction optimizer as part of its planning. The analysis model is limited by the capability of the optimizer, that is no design should be produced which implies a transaction processing strategy which will not be generated by the optimizer.

Cost evaluation In order to evaluate the cost function of some processor configuration we invoke at design time the program which is eventually to be used for the optimization of transaction execution. Such a program determines the execution load components once for each transaction T_i in **T**.

The cost evaluation function *Cef* is a mapping defined as follows: Given a set **F** of fragments of **R**, a set **N** of nodes (not necessarily equal to **P**), the set of objects $\mathbf{O} = \mathbf{T} \bigcup \mathbf{F}$, and an allocation $\mathbf{L(O, N)}$, $Cef(\mathbf{L(O, N)})$ is the triple $< \mathbf{vi}, \mathbf{vc}, \mathbf{vm} >$, where

i) $\mathbf{vi} = \{vi_1, \ldots, vi_n\}$ and vi_i is the load in terms of the number of block accesses required in the node N_i.

ii) $\mathbf{vc} = \{vc_1, \ldots, vc_n\}$ and $\mathbf{vc}_i$ is the load in terms of the number of processor cycles consumed by the node N_i.

iii) $\mathbf{vm} = \{vm_{ij} | 1 \leq i, j \leq n\}$ and $vm_{i,j}$ is the load due to the message traffic generated between the nodes N_i and N_j or, more generally, the communication cost between the two nodes.

Notice that $\mathbf{vi}, \mathbf{vc}, \mathbf{vm}$ represent the costs necessary to execute the transactions in **T** on the fictitious network of nodes **N**. Such costs can be determined analytically from the generated allocation and knowledge of the available access strategies and the transactions [W83]. Estimates of such costs are produced for one transaction at a time by a transaction execution optimizer (like the optimizer in System R* [WSA81]) as part of its planning.

The problem of database allocation to the nodes in a cluster of processors is the following:

The DBNCP-Problem over Relations:

Let **R** *be a set of relations,* **T** *be a set of transactions,* **P** *be a set of* p *processors, and Cef be a cost evaluation function. Find a set* **F** *of fragments of* **R** *and an allocation* $\mathbf{L(O, P)}$, *where* $\mathbf{O} = \mathbf{T} \bigcup \mathbf{F}$, *such that the cost*

$$\sum_{i=1}^{p} (vi_i + vc_i) + \sum_{i=2}^{p} \sum_{j=1}^{i-1} vm_{ij} \qquad \textit{is minimum}$$

subject to

i) $vi_i \leq CI_i$ *for all* P_i *in* P

ii) $vc_i \leq CC_i$ *for all* P_i *in* P

iii) $vm_{ij} \leq CM_{ij}$ *for all* P_i, P_j *in* P.

where $< \mathbf{vi}, \mathbf{vc}, \mathbf{vm} > = Cef(\mathbf{L(O, P)})$. ∎

We note that this problem is different from the database partitioning problems treated in literature (see, for instance [Ap82,CNW81]). In particular, in this case the initial network node of each transaction is not predefined. This means that we cannot neglect the capacity construct as it is assumed in previous works, because otherwise we would find a trivial solution, that is allocating all transactions and relations in only one processor.

2.2 The database partitioning problem with a defined set of fragments

We note that the DBNCP-problem considers any possible fragmentation of the relations in **R**. If we assume that a set **F** of fragments is given, a variant of the DBNCP-problem is the following:

The DBNCP-Problem over fragments

Let **F** *be a set of fragments of a given set of relations* **R**. *Let* **T** *be a set of transactions,* **P** *be a set of* p *processors, and Cef be an cost evaluation function. Find an allocation* L(O,P), *where* $O = T \bigcup F$, *such that*

$$\sum_{i=1}^{p}(vi_i + vc_i) + \sum_{i=2}^{p}\sum_{j-1}^{i-1} vm_{ij} \qquad \textit{is minimum}$$

subject to

constraints i), ii), iii) of the DBNCP-problem over relations. ∎

We note that, in general, the solution of this second problem is not a solution of the general problem. However, if the initial set of fragments contains units of allocation, i.e., elementary objects of allocation that cannot be further fragmented, then the solution of the DBNCP-problem over a set of predefined fragments is also the solution of the DBNCP-problem over relations.

This means that, since the transactions cannot be fragmented, we have to find an initial set **F** of fragments such that each fragment in **F** will not be further partitioned but only, eventually, combined. In [SW83] a method, following [Ap82], is proposed to obtain the initial units of allocation.

From now on, we will consider as input of the DBNCP-problem over fragments either a set of functional elementary objects or whatever set of fragments has been predefined by the database designer. The size of the set of fragments is polynomially bound by the size of the input of the DBNCP-problem over relations. Our conjecture is that such an approach gives a good suboptimal solution of the DBNCP-problem over relations. Furthermore, since the optimal solution of both problems cannot be found in a reasonable time, as we shall prove in the next section, the above assumption is the only one suitable to provide polynominal-time heuristic algorithms.

Before concluding with the formulation of the database partitioning problems we point out that, after the solution of the problem is found, a postanalysis will recombine any fragments of the same source relations which are allocated into the same processor.

3. HEURISTIC ALGORITHMS FOR THE PARTITIONING PROBLEM

3.1 Complexity of the Problem

The solution of the two database partitioning problems introduced in the previous section is strongly dependent on the cost evaluation function *Cef*. It is obvious that, if the computation of such a function requires exponential time in the size of the relations in **R** and the transactions in **T**, we do not have any hope to find a solution, even suboptimal, in a reasonable amount of time.

In this paper, we suppose that the evaluator computes the costs in polynomial time. This is not a strong assumption since the optimizer must compute, in order to be practical, the cost of transactions in a reasonable amount of time. However, we have to point out that such a computation cannot be considered as an elementary computer operation. This means that in our complexity analysis we will evaluate how many times the optimizer cost function is invoked separately from the evaluation of the number of simple operations. This analysis is found in Sec. 3.3.

Despite the above simplifying assumption, the complexity of the two partitioning problems is still hard. In fact, we have proven not only that there is no polynomial-time algorithm to solve the problem, unless $P = NP$ [GJ79], but also that no polynomial-time algorithm is able to find a feasible solution of the problems.

THEOREM 1. *Finding a feasible or optimal solution of the DBNCP-problem over relations or over fragments is NP-hard.*

The proof of Theorem 1 can be found in [SW83].

From Theorem 1 we decide that we cannot find a solution of the database partitioning problem, even only a feasible and not optimal solution, in a reasonable amount of time, without using heuristic procedures.

In the next sections we will give heuristic algorithms for the solution of the DBNCP-problem, based on combining the greedy method [HS76] and the First-Fit algorithm [JDUGG74]. Since a feasible solution of the DBNCP-problem over fragments is also a feasible solution of the DBNCP-problem over relations, and since starting from predefined set of elementary fragments is more suitable to provide heuristic algorithms, we only consider the DBNCP-problem over fragments.

3.2 The greedy and the first-fit algorithms.

Our task is to select fragments from a large set of fragments and allocate them to the processors. At the same time transactions have to be allocated. Fragments should be combined with other fragments and transactions should be allocated with fragments if placing them together leads to a great benefit in terms of reduction of communication, CPU, or IO load. Their allocation to processor nodes is subject to capacity constraints.

Greedy selection of fragments The aspect of selecting fragments suggests a greedy method, i.e., an algorithm which works in stages, considering one input at a time At each stage, an optimal "local" solution is found for a particular input. Such solutions may or may not lead to the optimal solution of the problem. However, most of the time such a method will result in algorithms that generate suboptimal solutions. In our case, the greedy method reduces to the following algorithm.

We start from the set **O** of **o** objects (transactions and relation fragments) and we consider a set of **n** nodes **N** such that $\mathbf{n} = \mathbf{o}-1$. For all pairs of objects O_i, O_j, we consider an allocation on $\mathbf{L(O,N)}$ such that one node N_h of **N** contains the combination of two objects O_i, O_j and that each of the remaining nodes contains only one object of $\mathbf{O} - \{O_i, O_j\}$. Let $\mathbf{tc}_{ij}$ be the total cost of this allocation,

$$\mathbf{tc}_{ij} = \sum_{i=1}^{n}(vi_i + vc_i) + \sum_{i=2}^{n}\sum_{j=1}^{i-1} vm_{ij}$$

where $< \mathbf{vi}, \mathbf{vc}, \mathbf{vm} >= \mathit{Cef}(\mathbf{L(O,N)})$.

We select the pair of nodes O_i, O_i for which $\mathbf{tc}_{ij}$ is minimum and the capacity constraints of the problem are satisfied. We then modify the set of objects **O** by replacing the object O_i, O_j with the compound object. In further stage, we repeat the above step by considering the modified set of objects **O** and by reducing the number of nodes by one. Eventually, the algorithm stops when either $\mathbf{n} = \mathbf{p}$ or no two nodes can be further combined. If the algorithm reaches the stage where $\mathbf{n} = \mathbf{p}$ the compound objects in **O** define the final allocation of the objects in the processors. The greedy method fails if it reaches the point where no nodes can be combined while $\mathbf{n} > \mathbf{p}$, although a feasible solution of the problem may exist.

We note that the same algorithm was proposed in [Ap82] for the solution of a database partitioning problem without capacity constraints but with predefined starting point of each transaction. In that paper it was shown that such an application gives a good suboptimal solution of the predefined transaction entry problem.

Let us now analyze the goodness of this greedy algorithm for the solution of the entire problem. Unfortunately, our conjecture is that this algorithm is very unlikely to give a feasible solution of the allocation phase of the problem. Consider, for simplicity, that the CPU load associated with the fragments is independent from their allocation. In this case, finding a feasible solution corresponds to finding a solution of the Bin-Packing problem, where the fragments are the items to be inserted in the bins, represented by the processors. It is known in the literature (for instance, see [GJ79, JDUGG74]) that any algorithm that start a new bin before all the non-empty bins are full, gives a poor solution to the bin-packing problem. In other words, any "good" approximate allocation algorithm must be at least a "First-fit" algorithm.

First-fit allocation Reconsider the basic greedy algorithm. Any time it combines two objects, which were never combined before, it starts what amounts to a new bin although there are bins not yet full. Hence, we have to modify the allocation phase of the greedy algorithm in order to meet the above-mentioned property of First-fit algorithms.

We could use some other better algorithm for the bin-packing problem, like the "First-fit Decreasing" algorithm in [J74]. However, in these approaches the selection phase is also driven by the allocation algorithm. Since allocation is based on the capacity constraints of the processors we loose the optimality objective used for selection.

However, in the case of the simple First-fit algorithm we have the freedom to select fragment combinations on the basis of the greedy method since we the only constraint is to never introduce too many bins. In particular, there should never be more bins than the number of available processors. Since we believe that in reality the capacity constraints cannnot be too restrictive, a first-fit allocation approach should permit the design to respect the constraints.

3.3 Analysis of the greedy first-fit algorithm for various cases of capability.

The greedy first-fit algorithm described above allows for the solutions of the partitioning problems over fragments to the capacity limits specified for the processor network. In particular, as shown in [SW83], the cost evaluation function which computes *Cef* is in all cases computed $O(o)$ times, where o is the initial number of objects to be allocated. However, the complexity of the algorithm in terms of elementary operations (i.e., all operations but calls to cost evaluation function) is different for the various cases of capacity constraints. We considered all such cases. The algorithm is modified only in the part which checks the capacity constraints. We present the results found in [SW83] in a table.

The Table shows that the proposed First-fit Greedy algorithm runs in all but the most general case in $O(\mathbf{o}^3\mathbf{p}^{5/2})$ time. The general case where the algorithm does not appear to run in polynomial time occurs when the network is not homogenous. Such networks, having links of unequal capacity or, more likely, absent links between processor pairs are common in long-haul networks. These are not the type of networks which are addressed in our analysis, since for these networks the initial assumption that the entry point for the transaction can be assigned arbitrarily is highly unlikely.

Case	CPU Capacity	IO Capacity	Communication Capacity	Algorithm Complexity
1	Equal for all processors	Unlimited	Unlimited	$O(o^3p)$
2	Differing	Unlimited	Unlimited	$O(o^3p \log p)$
3	Differing	Equal for all processors	Unlimited	$O(o^3p \log p)$
4	Differing	Differing	Unlimited	$O(o^3p^{5/2})$
5	Differing	Differing	Equal for all pairs of processor nodes	$O(o^3p^{5/2})$
6	Differing	Unlimited	Unlimited	exponential

Recall that **o** is the number of objects (fragments and transactions) and **p** is the number of processor nodes.

4. CONCLUSION

We have addressed the problem of distributing a database over a fixed number of processors. The processors and the network connecting them have limited processing and transmission capacities. The relations of the database are fragmented to provide suitable units of allocation. The entry point of the query transactions is not constrained. The complexity of the general problem is shown to be NP-hard so that an heuristic algorithm is called for. This algorithm should not only provide a feasible solution but a solution that is near to optimal as well, these two criteria are related since without finding a low cost solution no solution which satisfy the processor constraints may be found.

The database is fragmented prior to the analysis so that an appropriate granularity for distribution is obtained. The heuristic algorithm assigns the fragments to processors. It combines a greedy algorithm for the selection of candidate fragments to be assigned with a first-fit bin-packing algorithm for the allocation of the selected fragments to the processor nodes.

We point out that the greedy first-fit algorithm requires $O(o^3)$ calls to the cost evaluation functions (**o** is the number of initial objects to be allocated), in order to obtain the information needed to make design decisions. In most cases the cost evaluation function is the query optimization program of the database management system over hypothetical defined network. One invocation of the function has to consider all specified transactions. This means that in typical cases the database designer cannot afford to call it a large number of times. In order to reduce the number of such computations, three other techniques are presented in [SW83] to drive the greedy selection of the algorithm. These techniques allow for a reduced number of calls to the optimizer (namely, $O(1)$, $O(o)$ and $O(o^2)$ times).

The faster techniques appear to be applicable in earlier stages of the selection algorithm. Furthermore, they are especially attractive in the early stages when the value of $o >> p$. Experimental results have been obtained for some of the techniques and appear in [SW83] in order to verify the analytical results summarized in this paper.

ACKNOWLEDGEMENT

This work was performed in the context of a larger research project on A Highly Available Transaction Support System at the IBM San Jose Research Laboratory under the leadership of Mario Schkolnick. We profited greatly from discussions with members of the laboratory. Jayne Pickering helped with entering the text into the TEXsystem [Kn79], which was used to prepare this report. Gio Wiederhold is receiving partial support from ARPA, contract N39-82-C-250.

REFERENCES

[Ap82] Apers,P.M.G.: *Query Processing and Data Allocation in Distributed Database Systems*; Thesis 1982, Mathematical Centrum, Vrije Univ., Amsterdam.

[Casey72] Casey,R.G.: "Allocation of Copies of a File in an Information Network"; *Proc. 1972 SJCC*, AFIPS Vol.40, pp.617–625.

[Chu68] Chu,Wesley W.: "Optimal File Allocation in a Multicomputer Information System"; *Information Processing-68*, Proc. IFIP Congress, North-Holland 1968, pp.F80–F85.

[Co71] Cook,S.A.: "The complexity of theorem-proving procedure"; *Proc. 3rd ACM Symposium on Theory of Computing*, ACM, New York, pp.151–158.

[CNW81] Ceri,S., Navathe,S., and Wiederhold,G.: *Optimal Design of Distributed Databases*; Stanford Univ., CA, CS Report No.STAN-CS-81-884, Dec.1981.

[CODD80] Codd,E.F.: "Extending the Database Relational Model to Capture More Meaning"; *ACM TODS*, Vol. 4 No.4, Dec.1979, pp.397–434.

[DoFo82] Dowdy,L.W.and Foster,D.V.: "Comparative Models of the File Assignment Problem"; *ACM Computing Surveys*, Vol.14 No.2, June 1982, pp.287–313.

[JDUGG74] Johnson,D.S., Demers,A., Ullman,J.D., Garey, M.R., and Graham,R.L.: "Worst–case performance bounds for simple one-dimensional packing algorithms"; *SIAM Journal on Computing*, Vol.3, pp.299–325.

[GJ79] Garey,M.R. and Johnson,D.S.: *Computers and Intractibility – A guide to the Theory of NP-Completeness*; W.H.Freeman and Co., 1979.

[GJS76] Garey,M.R., Johnson,D.S., and Stockmeyer,L. : "Some simplified NP-complete graph problems"; *Theoretical Computer Science*, Vol.1, pp.237–267.

[J74] Johnson,D.S.: "Approximation Algorithms for Combinatorial Problems"; *JCSS*, Vol.9, 1974, pp.256–278.

[Kn79] Knuth,D.E.: TEX *and METAFONT, New Directions in Typesetting*; Digital Press, 1979.

[MoLe77] Morgan,H.L. and Levin,K.D.: "Optimal Program and Data Locations in Computer Networks"; *CACM*, Vol.20 No.5, May.1977, pp.315–322.

[SW83] Saccá,D. and Wiederhold,G.: *Database Partitioning in a Cluster of Processors*; Stanford University, Computer Science Department, KBMS project, Mar. 1983.

[W83] Wiederhold,G.: *Database Design, second edition*; McGraw-Hill, 1983.

[WSA81] Williams,R., Selinger,P., et al: *R-*: an Overview of the Architecture*; IBM Res. Rep. RJ3325 (40082), Dec.81.