TR-0897

Implementation of Parallel Database Management System on KLIC

by
M. Kawamura & T. Kawamura

November, 1994

Implementation of Parallel Database Management System on KLIC

Moto KAWAMURA, and Toru KAWAMURA
Institute for New Generation Computer Technology (ICOT)
21F. Mita-Kokusai Bldg., 1-4-28, Mita, Minato-ku, Tokyo 108, Japan
e-mail: {kawamura, toru}@icot.or.jp

Abstract: A parallel database management system (DBMS) called *Kappa* is developed in order to provide efficient database management facilities for knowledge information processing applications. The data model of Kappa is based on a nested relational model to treat complex structured data efficiently. The system is written in KL1, and works on parallel and/or distributed environments of conventional machines with KLIC. In this paper, we give an overview of Kappa.

Keywords: Parallel DBMS, Concurrent Logic Programming Language

1 Introduction

In the Japanese FGCS (Fifth Generation Computer System) project, many knowledge information processing systems (KIPSs) were developed under the framework of logic and parallelism. In these systems, R&D of databases and knowledge-bases[9] aims at an integrated knowledge-base management system (KBMS) under a framework of deductive object-oriented databases (DOODs). *Kappa* [1] is a database management system (DBMS) located in the lower layer and is also the name of the project. The objective is to provide database management facilities for many KIPSs. In the Kappa project, we developed a sequential DBMS, *Kappa-II*[8] and a parallel DBMS, *Kappa-P*[3]. Both systems adopt a nested relational model, and run on experimental hardware and software environment developed in the FGCS project. We have been developing a parallel DBMS called *Kappa*, which works on parallel and/or distributed environments on conventional machines, to make it easier to adopt our research results.

The following is a short description of Kappa-II, Kappa-P, and Kappa. Kappa-II is written in ESP, which is a kind of prolog with object-oriented features, and runs on the PSI sequential inference machine with the SIMPOS operating system. The system showed us that our approach based on the nested relational model is sufficient for KBMSs and KIPSs, for instance natural language processing systems with electronic dictionaries, proof checking systems with mathematical knowledge, and genetic information processing systems with molecular biological data. Kappa-P is logically based on Kappa-II with the configuration and query processing extended for a parallel environment. Kappa-P is written in KL1, which is based on FGHC, and runs on PIM parallel inference machines with the PIMOS operating system. Kappa is based on Kappa-P, modified to be suitable for parallel and/or distributed environments on conventional machines with KLIC, that is; as a portable implementation of KL1.

We give an outline of Kappa in Section 2. We give an overview of data placement in Section 3, and Section 4 covers implementation issues.

[1]Knowledge Application-Oriented Advanced Database Management System

2 Outline of Kappa

Our environment contains a variety of data and knowledge with complex data structures. For example, molecular biological data treated by genetic information processing systems includes various kinds of information and huge amounts of sequence data. The GenBank/HGIR database[2] has a collection of nucleic acids sequences, physical mapping data, and related bibliographic information, and the amount of data has been increasing exponentially. The size of the sequence data ranges from a few characters to 200,000 characters. The data becomes longer as genome data, and is analyzed gradually: the size of a human genome sequence is about 3,000,000,000. The conventional relational model is not sufficient for efficient data representation and efficient query processing. Moreover, the rapid increase of data will require more processing power and secondary memory to manage it.

Such situations requires a parallel computational environment with database management facilities providing a data model which can treat complex structured data efficiently, and countermeasures for huge amounts of data. We have two solutions to these requirements. One is a parallel environment using PIM machines, the PIMOS operating system, and the Kappa-P DBMS in KL1, which was developed in the FGCS project. Another is a parallel and/or distributed environment for conventional machines with KLIC, and the Kappa DBMS, which we have developed.

Some features of Kappa are listed below.

Nested Relational Model

As there are various data and knowledge with complex data structures in our environment, the conventional relational model is not appropriate for efficient data representation and efficient query processing. In order to treat complex structured data efficiently, we adopt a nested relational model. The nested relational model with a set constructor and hierarchical attributes can represent complex data naturally, and can avoid unnecessary divisions of relations. Semantics of nested relations matches a knowledge representation language, *QUIXOTE*[7] of KBMS. There have been other nested relational models[5, 6, 1] since the proposal in 1978[4]. Specifically, although syntactically the same, their semantics are not necessarily the same. Operations on nested relations are extended relational algebra, which is a simple extension of relational algebra.

The model is the same Kappa-P's data model which shows us to be able to handle complex structured data efficiently, but the implementation of the model is a little different. Kappa-P has two kinds of operations: primitive commands and a query language based on extended relational algebra. To reduce the code size of the system, Kappa provids primitive commands only. Primitive commands are the lowest operations for nested relations based on tuple identifiers. Term is one of the data types in both systems, because term is a primitive data structure in KL1. While the character code in Kappa-P is a 2-byte code because this is usual in the PIM and PIMOS environments, the character code in Kappa is a 1-byte code because this is usual in a KLIC environment.

Configuration

Kappa is constructed of a collection of element DBMSs which manage their own data. Each element DBMS contains full database management facilities, and manages a sub-database. This is called shared nothing architecture. A global map of relations is managed by element DBMSs called server DBMSs to improve availability, and to decentralize access to it. Element DBMSs with the exception of server DBMSs are called local DBMSs.

Interface processes are created to mediate between application programs and Kappa as a collection of element DBMSs, and receive queries as messages in KL1. A query is processed by some element DBMSs, in which relations accessed by the query exist. To allow the element DBMSs to cooperate, Kappa provides distributed transaction mechanisms based on the two phase commitment protocol.

Data Placement

Placement of relations is the responsibility of the database designer, because it should be determined in consideration of relationships among relations and kinds of typical queries to the database. Kappa provide three kinds of data placement: distribution, horizontal partition, and replication.

In order to use parallelism, relations can be located in some element DBMSs. The simple case is distribution of relations like distributed DBMSs. When some queries to a relation need large processing power, the relation can be declustered as a horizontally partitioned relation and located in some element DBMSs. For example, a molecular biological database including sequence data which is increasing rapidly requires homology search by a pattern called motif. If a relation is frequently accessed in any queries, multiple copies of the relation can be made and located in some element DBMSs. The replicated relation is implemented as a global map only.

3 Data Placement

In order to obtain larger processing power, relations should be located in different element DBMSs. Kappa provides three kinds of data placement: distribution, horizontal partition, and replication.

Distribution

Distribution of relations is as for the simple case of distributed DBMSs. When relations are distributed in some element DBMSs, larger processing power is obtained, but communication overheads are generated at the same time. The distribution of relations is the responsibility of the database designer.

A query to access these relations is divided into sub-queries for some element DBMSs by an interface process, and sub-queries are processed as distributed transactions.

Horizontal Partition

A horizontally partitioned relation, which is a kind of declustered relation, is logically one relation, but comprises some sub-relations containing tuples according to some declustering criteria. A horizontally partitioned relation is effective, if some queries to the relation need large processing power, or the relation is too large to treat in an element DBMS. Horizontal partition of relations is also the responsibility of the database designer.

A query to access horizontally partitioned relations is converted to sub-queries to access each sub-relation. Each sub-query is processed in parallel in different element DBMSs in which the sub-relations exist. Especially, when the query is an unary operation or a binary operation suitable for the declustering criteria, each sub-queries can be processed independently and communication overheads among element DBMSs can be disregarded. In other cases, communication overheads among element DBMSs cannot be disregarded, and it is necessary to convert the queries to reduce the overheads.

In primitive commands, a tuple filter is taken as an argument of a read operation. The read operation invokes filters for each sub-relations and a merger for gathering output of the filters (Figure 1).

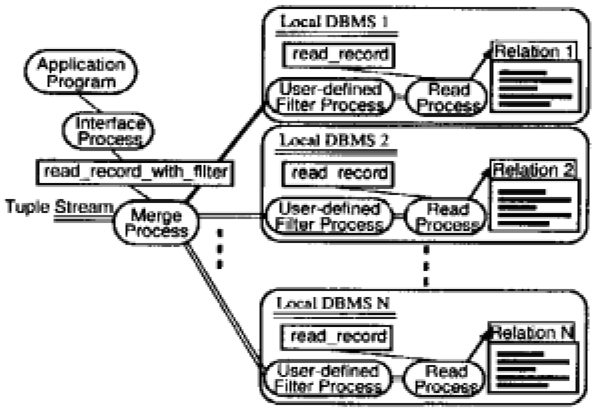

Figure 1: Read operation with a filter

Replication

Replication of a relation enables us to decentralize access to the relation, and to improve availability. A global map managed by server DBMSs is the only replicated relation.

Kappa has two kind of relation identifiers: global relation identifiers and local relation identifiers. Every relation has it own local identifier managed by an element DBMS in which the relation exists. Some server DBMSs manage global relation identifiers as a global map centrally, freeing the identifiers from element DBMS information in which the relations exist. The centralization does not affect performance, because the information is only referred to find target element DBMSs from relation identifiers at the beginning of query processing, and is modified when global relations are created or deleted.

We decided to implement the replicated global map based on the weighted voting protocol, because the protocol satisfies the above requirements, is not complicated to implement, and does not work incorrectly in any kinds of failures. A read operation for the global map is translated to read operations for randomly selected server DBMSs, the number of which is one or two over the read vote. The operation is completed when the number of read vote results are received. A write operation for the global map is translated to read operations for randomly selected server DBMSs and write operations for server DBMSs replying to the read operations. As the latter write operations are performed by using distributed transaction mechanisms based on the two phase commitment protocol to make implementation simple, failures in the write operations are not treated by the weighted voting protocol, but by the two phase commitment protocol. This is not a big problem because the reason for implementing the replicated global map is to improve availability and to decentralize access to it.

4 Implementation Issues

4.1 Element DBMS

An element DBMS contains full database management facilities, and accepts primitive commands. An element DBMS is regarded as parallel processing based on a shared memory.

Parallel Processing of Primitive Commands

Primitive commands for nested relations are processed by various parallel processes: for instance, operations on sets and tuple streams driven on demand with double buffering.

A set is a collection of tuple identifiers, and is obtained by restriction operations on a relation. So a set corresponds to sub-relation of the relation, operations on sets: union, intersection, and difference can be defined. A tuple identifier consists of a primary tuple identifier, which specifies a whole tuple, and sub-tuple identifiers, which specify some occurrences of set values in the tuple. The operations on sets are processed on tuple identifiers without tuples. In order to parallelize set operations, a set is partitioned with the range of tuple identifiers.

The way tuples are treated among internal modules of a DBMS is important. For example, a restriction operation without using an index on a relation is performed by two processes, a process to get the tuples of the relation, and a process to test them to satisfy a restriction formula, connected by a tuple stream. This is a typical KL1 program with a generator and a consumer. In a simple implementation of the program, the generator process generates one data contained by a cons cell, and the consumer process receives the data, triggered by the cons cell. If these two processes are not scheduled properly, the generator process will exhaust main memory. There are some ways to prevent this, for instance, to execute the generator process at a lower priority than the consumer process, or to change the direction of triggers. That is, a consumer process creates a cons cell to request a generator process to generate one data, and the generator process waits for the cons cell and then sets one data into the car of the cons cell.

Tuple streams in Kappa are based on the latter. In order to reduce suspensions of goals, a generator process passes the user-defined number of tuples as a list of tuples at a time. Of course, the head of the list is passed when the last tuple is generated, to reduce suspensions in the consumer process. A tuple stream is expressed as follows: $[[Tuple_{11}, Tuple_{12}, \cdots, Tuple_{1N}], \cdots, [Tuple_{M1}, Tuple_{M2}, \cdots, Tuple_{MN}], end, \cdots, end]$. Cons cells of the outer list are created by the consumer process, and the inner lists, each of which calls a buffer, is created by the generator process.

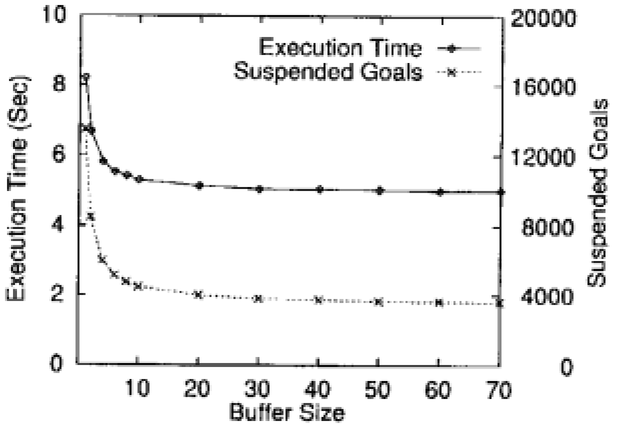

Figure 2: Restriction operation with various sizes of buffers

Figure 2 shows the relationship between the execution time for a restriction operation without using an index on a relation, and the number of suspended goals for various sizes of buffers, on a sequential version of KLIC. The relation contains 10,000 tuples. About 15 tuples of the relation are transferred by one read operation from secondary storages. This figure shows that tuple streams work efficiently.

From a parallel processing point of view, the generator process and the consumer process can run in parallel with double buffering techniques. However a buffer containing tuples is processed sequentially. In order to obtain further parallelism, the consumer process requests multiple buffers at a time, and the generator process performs the requests in parallel.

Modules written in C language

We can use C modules as generic objects in KLIC. But functions of generic objects are executed sequentially. Almost all modules in an element DBMS are suitable to be written in KL1, because of the data structure of tuples in nested relations, and because sets containing tuple identifiers are complex.

The lowest data structure in nested relations is bit-image, stored as files. Kappa accesses the data as byte strings in KL1. Operations on strings can be treated more efficiency in C than in KL1. Important operations on strings, which influence the system performance, are getting a tuple from strings, getting B-tree components from strings, and similar operations.

We decided to implement the following operations in C: getting a tuple from strings, setting a tuple into strings, getting B-tree components from strings, and setting B-tree components into strings. These operations are implemented sequentially in Kappa-P written in KL1 also, because too much parallel coding decreases performance. Parallelism is controlled by the number of tuples to be processed in parallel.

5 Conclusions

In this paper, we describe a parallel DBMS Kappa on KLIC.

In order to provide KBMSs and KIPSs with efficient database management facilities, the system adopts a nested relational model, and is designed to use parallel resources efficiently by using various parallel processing. We intend to experiment on the efficient utilization of parallel resources, to show that the system provides KBMSs and KIPSs with efficient database management facilities.

References

[1] P. Dadam, et al, "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies", *ACM SIGMOD*, 1986.

[2] "GenBank/HGIR Technical Manual", *LA-UR 88-3038*, Group T-10, MS-K710, Los Alamos National Laboratory, 1988.

[3] M. Kawamura, H. Sato, K. Naganuma, and K. Yokota. "Parallel Database Management System: Kappa-P", *Proc. FGCS'92*, Tokyo, 1992.

[4] A. Makinouchi, "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model", *VLDB*, 1977.

[5] H.-J. Schek and G. Weikum, "DASDBS: Concepts and Architecture of a Database System for Advanced Applications", *Tech. Univ. of Darmstadt, TR*, DVSI-1986-T1, 1986.

[6] J. Verso, "VERSO: A Data Base Machine Based on Non 1NF Relations", *INRIA-TR*, 523, 1986.

[7] H. Yasukawa, H. Tsuda, and K. Yokota, "Object, Properties, and Modules in $\mathcal{QUIXOTE}$", *Proc. FGCS'92*, Tokyo, June 1-5, 1992.

[8] K. Yokota, M. Kawamura, and A. Kanaegami, "Overview of the Knowledge Base Management System (KAPPA)", *Proc. FGCS'88*, Tokyo, Nov.28-Dec.2, 1988.

[9] K. Yokota and H. Yasukawa, "Towards an Integrated Knowledge-Base Management System — Overview of R&D for Databases and Knowledge-Bases in the FGCS project", *Proc. FGCS'92*, Tokyo, June 1-5, 1992.