Parallel Database Management System: Kappa

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.

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[10] 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*[9] and a parallel DBMS, *Kappa-P*[4]. 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. Section 3 describes those features of Kappa's nested relational model which are different from others. We give an overview of operations on nested relations in Section 4, and Section 5 covers implementation issues.

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[3] 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

[1] Knowledge Application-Oriented Advanced Database Management System

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 the knowledge representation language, *Quixote*[8] of KBMS. There have been other nested relational models[6, 7, 1] since the proposal in 1978[5]. 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 data model which proved 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 provides 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. Both systems provide special indices to handle a huge amount of data such as genome database in the near future. While an ordinary index for an attribute consists of pairs of the value of the attribute in a tuple and all tuple identifiers which point to tuples having the value of the attribute, a special index for an attribute consists of pairs of a modified value and tuple identifiers. There are several ways for the modifications: concatenating specified values to an attribute value, getting a sub-string of an attribute value, and getting multiple sub-strings of an attribute value by scanning the value. The last way can be used to find some fragments in amino acid sequences.

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. Figure 1 shows the overall configuration of Kappa. This is called shared-nothing architecture[2]. 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 to receive queries in primitive commands 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.

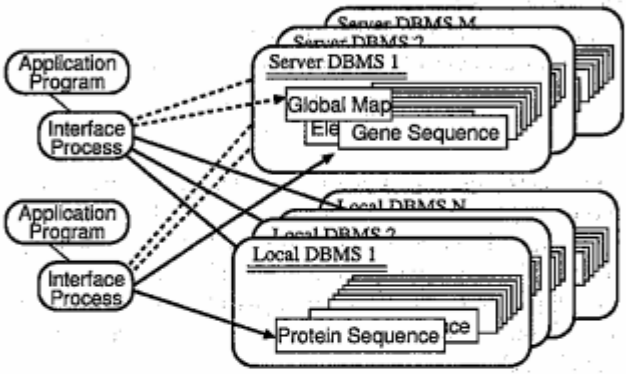

Figure 1: Configuration

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 distributed relations as in distributed DBMSs. When some queries to a relation need large processing power or a relation is too large to treat in an element DBMS, the horizontally partitioned relation relation can be declustered to subrelations located in different element DBMSs according to some declustering criteria. An example is a molecular biological database including sequence data which increases rapidly. If a relation is frequently accessed in 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. The horizontally partitioned relation is logically one relation.

3 Nested Relational Model

A nested relational model is known to reduce the number of relations in the case of multi-value dependency and represents complex data structures more naturally, compared with the the relational model. However, there have been other nested relational models[6, 7, 1] since the proposal in 1978[5]. Specifically, although syntactically the same, their semantics are not necessarily the same. In Kappa, one of the major problems is which semantics is appropriate for many applications in our environment. Another problem is which part of *Quixote* should be supported by Kappa as a database engine because enriched representation is a trade-off for efficient processing. In this section, we explain the semantics of the Kappa model.

Intuitively, a *nested relation* is defined as a subset of a cartesian product of domains or other nested relations:

$$\begin{aligned} NR &\subseteq E_1 \times \cdots \times E_n \\ E_i &::= D \mid 2^{NR} \end{aligned}$$

where D is a set of atomic values [2]. That is, the nested relation may have hierarchical structures and set values of other nested relations. This corresponds to introducing tuple and set constructors as in complex objects. Corresponding to syntactical and semantical restrictions, there are various subclasses, to each of which extended relational algebra are defined.

In Kappa's nested relation, a set constructor is used only as an abbreviation for a set of normal relations as follows:

$$\begin{aligned} &\{r[l_1 = a, l_2 = \{b_1, \cdots, b_n\}]\} \\ &\quad \Leftrightarrow \quad \{r[l_1 = a, l_2 = b_1], \cdots, r[l_1 = a, l_2 = b_n]\} \end{aligned}$$

The operation "$\Rightarrow$" corresponds to an *unnest* operation, while the opposite operation ("$\Leftarrow$") corresponds to a *nest* or *group-by* operation, although "$\Leftarrow$" is not necessarily congruent as for the application sequence of nest or group-by operations. That is, in Kappa, the semantics of a nested relation is the same as the corresponding relation without set constructors. The reason why we adopt these semantics is to retain the first order semantics for efficient processing and to remain compatible with the widely used relational model. Let a nested relation be

$$\begin{aligned} &NR = \{nt_1, \cdots, nt_n\} \\ &\text{where } nt_i = \{t_{i1}, \cdots, t_{ik}\} \text{ for } i = 1, \cdots, n, \end{aligned}$$

Then the semantics of NR is $\{t_{11}, \cdots, t_{1k}, \cdots, t_{n1}, \cdots, t_{nk}\}$. Extended relational algebra for this nested relational database is defined in Kappa and produces results according to the above semantics. This guarantees to produce the same results as the corresponding relational database, except for the treatment of the attribute hierarchy.

[2]The term "atomic" is not the conventional one. For example, when an atomic value has a type *term*, the equality must be based on unification or matching

4 Operations on Nested Relations

Kappa provides operations on nested relations called primitive commands as an internal model. Primitive commands consist of unary operations based on tuple identifiers. A query in primitive commands is expressed as a sequence of commands to a interface process. The interface process sends each of the commands to an element DBMS or some element DBMSs in which relations accessed in the commands related exist. Primitive commands can be classified into the following operations: restricting a nested relation with/without using indices and getting a collection of tuple identifiers called a *set* as the result, manipulating sets, manipulating tuples in a nested relation, and manipulating meta-data of a database. In primitive commands we call nested relations *tables* and call tuples *records* because tuples of nested relations are not flat data but complex structured data including set values and tuple values.

Operations to Restrict a Nested Relation

Kappa provide two kind of restriction operations: a *search_index* operation and a *search_data* operation. The difference between the two operations is in the use of indices. Both operations return a collection of tuple identifiers: a set as the result. So a set corresponds to a sub-relation of the relation, but does not contain tuples. Most primitive operations can be specified as a set instead of a nested relation. Moreover there are standard operations on sets: union, intersection, and difference. These operations on sets are performed by only using tuple identifiers without using values of tuples, but the sets must correspond to the same relation. As a tuple in a nested relation can contain set values, a tuple identifier consists of a principal tuple identifier, which specifies a whole tuple, and sub-tuple identifiers, which specify some occurrences of set values in the tuple. A set generated by a restriction operation is located on the same machine on which an accessed relation exists, or on all machines on which sub-relations of a horizontally partitioned relation exist (Figure 2).

Operations to Manipulate Tuples in a Nested Relation

Operations to manipulate tuples in a nested relation are a *read_record* operation to extract tuples from a nested relation, an *add_record* operation to put a tuple into a nested relation, a *modify_record* operation to modify a tuple in a nested relation, and a *delete_record* to delete a tuple from a nested relation. We can extract specified attributes from a nested relation for a read_record op-

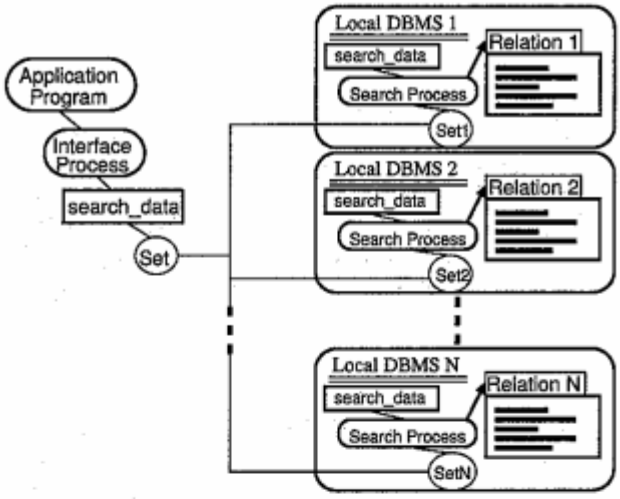

Figure 2: Read operation with a filter

eration like a projection operation in relational algebra. The result of a read_record operation specifying a set as an argument is the same as the result of a operation on a sub-relation restricted by the set.

A read_record operation on a horizontally partitioned relation is performed by read_record operations on element DBMSs in which sub-relations of the horizontally partitioned relation exist, and a merge process to gather all results of the read_record operations on the sub-relations (Figure 3). The result of a read_record operation is not returned in a plain list of tuples but in a *tuple stream* because of efficient processing (SeeSection 5.2).

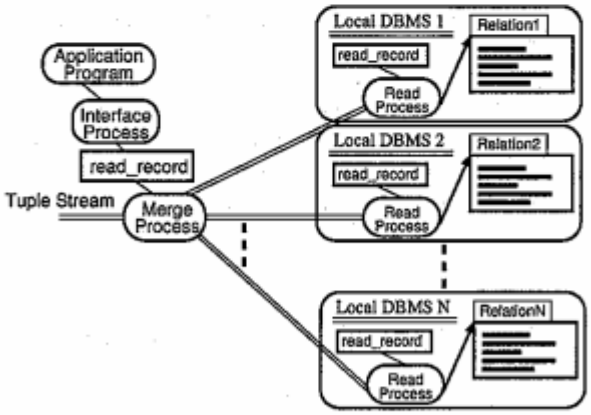

Figure 3: Read operation with a filter

In some cases applications will restrict a nested relation with an application-dependent restriction formula. In order to reduce inter-element DBMS communications, these application-dependent restriction operations should be performed on the same machine on which the accessed relations exist. Kappa provides a *read_record_with_filter* operation to solve this problem. The application-dependent restriction operation is passed as a filter process in KL1 to a read_record_with_filter operation, and is performed on the same machine on which an accessed relation exists, or in all machines on which sub-relations of a horizontally partitioned relation exist (Figure 4).

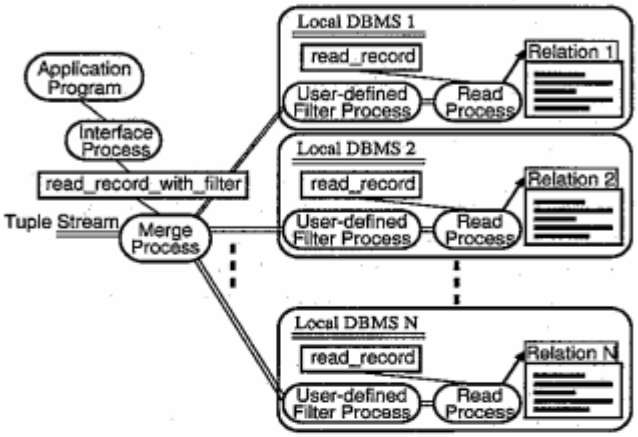

Figure 4: Read operation with a filter

Other Operations

Other operations are controlling jobs and transactions, creating a nested relation, deleting a nested relation, modifying the schema of a nested relation, getting the schema of a nested relation, and getting physical information of a database.

The nested relations to be used in a job and a transaction are specified in the creation operation of the job and the transaction. The specified nested relations are locked exclusively or not exclusively. Although the transaction is logically one transaction, sub-transactions in element DBMSs in which locked nested relations exist, and a transaction coordinator are created for distributed transaction based on the two phase commitment protocol. An argument of the operation to creating a nested relation is the schema of the nested relation. Indices of attributes are specified in a schema. An operation to modify the schema of a nested relation allows attributes to be added or removed under certain restrictions.

5 Implementation Issues

5.1 Management of Global Information

Kappa has two kinds 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. 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. The global map is not managed by one server DBMS but by several server DBMSs with replicated relations to decentralize access to the relation, and to improve availability.

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 under any kind of failure. 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.

5.2 Parallel Processing of Primitive Commands

Kappa is constructed with many modules as processes in KL1. The way tuples are treated among internal modules of a DBMS is important, and influences parallelism and the number of suspended goals. 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.

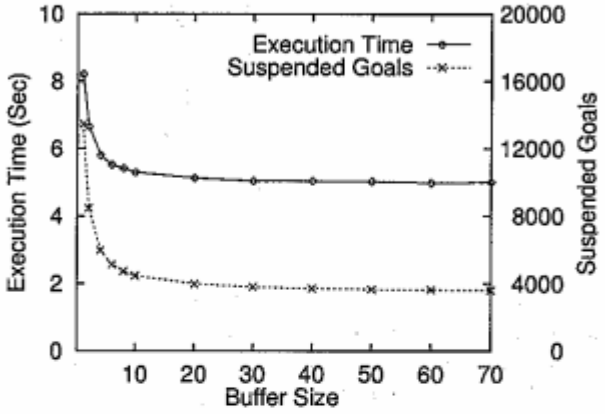

Figure 5: Restriction operation with various sizes of buffers

Figure 5 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 storage. 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, although 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.

5.3 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.

6 Conclusions

In this paper, we describe a parallel DBMS Kappa on KLIC. The KLIC system enables the Kappa to run not only on PIM machines but also on parallel and/or distributed environments on conventional machines. The performance of Kappa for the Wisconsin benchmark program is almost half that of an experimental DBMS based on the relational model on the same machine, although the benchmark program is for the relational model which is a subset of our model, and Kappa is written in KL1. We release Kappa as IFS (ICOT Free Software) to make it easier to adopt our research results.

Acknowledgments

The Kappa project has had many important contributors beyond those listed as authors. The members of the KLIC system of the First Laboratory at ICOT gave important suggestions to improve the performance. The *Quixote* project of the Second Laboratory at ICOT influenced the nested relational model. The members of the biological databases teams of the Second Laboratory at ICOT show us many improvements. Kazumasa Yokota began the Kappa project, and made laid the foundations of the project. The authors are grateful to Shunichi Uchida and Takashi Chikayama for encouragement.

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] D. J. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems", *Communications of the ACM*, Vol.35,No6, June 1992.

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

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

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

[6] 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.

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

[8] Kazumasa Yokota, Hiroshi Tsuda, and Yukihiro Morita, "Specific Features of a Deductive Object-Oriented Database Language *Quixote*", *Proc. ACM SIGMOD Workshop on Combining Declarative and Object-Oriented Databases*, Washington DC, USA, May 29, 1993.

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

[10] 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.