PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Parallel Database Management System: Kappa-P

Moto Kawamura   Hiroyuki Sato †
Kazutomo Naganuma   Kazumasa Yokota
Institute for New Generation Computer Technology (ICOT)
21F. Mita-Kokusai Bldg., 1-4-28 Mita, Minato-ku, Tokyo 108, Japan
e-mail: {kawamura, naganuma, kyokota}@icot.or.jp   Tel: +81-3-3456-3069   Fax: +81-3-3456-1618

† Mitsubishi Electric Corporation
Computer & Information Systems Laboratory
5-1-1 Ofuna, Kamakura, Kanagawa 247, Japan
e-mail: hiroyuki@isl.melco.co.jp   Tel: +81-467-46-3665   Fax: +81-467-44-9269

Abstract

A parallel database management system (DBMS) called *Kappa-P* has been developed in order to provide efficient database management facilities for knowledge information processing applications in the Japanese FGCS project. The data model of Kappa-P is based on a nested relational model for treating complex data structures, and has some new data types. Kappa-P has features of both a parallel DBMS on a tightly-coupled multiprocessor and a distributed DBMS on a loosely-coupled multiprocessor. In this paper, we describe the overview of Kappa-P.

1 Introduction

In the Japanese FGCS (Fifth Generation Computer System) project, many knowledge information processing systems (KIPSs) have been designed and developed under the framework of logic and parallelism. Among them, R&D of databases and knowledge-bases[14] 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 a name of the project. The objective is to provide database management facilities for many KIPSs and support efficient processing of the FGCS prototype system as the database engine. In the Kappa project, we have developed a sequential DBMS, *Kappa-II* and a parallel DBMS, *Kappa-P*. Both systems adopt a nested relational model.

Kappa-II, which is a research result of the intermediate stage, is written in ESP, and works on sequential inference machines PSI and its operating system SIMPOS. The system showed us that our approaches based on the nested relational model are sufficient for KBMSs and KIPSs, and has been used as the DBMS on PSI machines by various 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.

A parallel DBMS project called Kappa-P[7] was initiated at the beginning of the final stage. Kappa-P is based on Kappa-II from a logical point of view, and its configuration and query processing have been extended for the parallel environment. Kappa-P is written in KL1 and works on the environment of PIM machines and their operating system PIMOS. The smallest configuration of Kappa-P is almost the same as Kappa-II. Compared both systems on the same machine, Kappa-P works with almost the same efficiency as Kappa-II. Kappa-P is expected to work on PIM more efficiently than Kappa-II, as their environments are different.

We describe the design policies in Section 2 and the features in Section 3. We explain the features of Kappa's nested relational model that are different from others in Section 4. Then, we describe an overview of the Kappa-P system: data placement in Section 5, management of global information in Section 6, query processing in Section 7, and implementation issues of element DBMSs in Section 8.

[1]Knowledge Application-Oriented Advanced Database Management System

2 Design Policies

There are various data and knowledge with complex data structure in our environment. 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, the physical mapping data, and related bibliographic information. Amount of data has been increasing exponentially. Furthermore, the length of values is extremely variable. For example, the length of sequence data ranges from a few characters to 200,000 characters and becomes longer for genome data. Conventional relational model is not sufficient for efficient data representation and efficient query processing. Since the data is increasing rapidly, more processing power and more secondary memory will be required to manage it. Such situations require us to have a data model which can efficiently treat complex structured data and huge amount of data.

Parallel processing enables us to improve throughput, availability, and reliability. PIM-p is a hybrid MIMD multi-processor machine which has two aspects, a tightly-coupled multi-processor with a shared memory, called a *cluster*, and a loosely-coupled multi-processor connected by communication networks. Disks can be connected to each cluster directly. The architecture can be that of typical PIMs. Both applications and Kappa-P are executed on the same machine. Both KBMSs and KIPSs need a lot of processing power to improve their response, so Kappa-P should be designed to improve the throughput. The system should use resources effectively, and be adapted for the environment.

For the above requirements, the system is designed as follows:

- In order to treat complex structured data efficiently, a nested relational model is adopted. The model is nearly the same as Kappa-II's data model, which shows us efficient handling of complex structured data. New data types and new indexed attributes should be added to handle huge amounts of data efficiently.

- The system should use system resources effectively to improve throughput. System resources are processing elements, shared memories, disks, and communication networks.

  - The system should use hybrid multi-processors effectively.

  - Main memory database facilities should be provided for effective utilization of (shared) main memories. Because the data structure of the nested relation with variable occurrences and strings is complex, such a structure can be handled more efficiently on main memory than secondary memory.

  - The system should provide parallel disk access to reduce disk access overheads.

  - The system should actively control communication among clusters in order to reduce communication overheads in query processing.

- The system should be adapted for the environment. Though Kappa-P may be similar to database machines[1], the difference between Kappa-P and database machines is that both applications and a DBMS work together on the same machine.

  - The system should provide functions to reduce communication overheads between applications and the system, because they work together on the same machine. The functions execute part of applications at clusters which produce input data.

  - The system should provide a mechanism to process some queries in an application, because internal processing of an application is parallel, and some queries can occur in parallel.

  - The system uses the PIMOS file system, a part of which is designed for Kappa-P. The file system provides efficient access to large files, mirrored disks, and the sync mechanism for each file.

3 Features

According to the policies mentioned in the previous section, Kappa-P has been implemented. The system has the following features:

- Nested Relational Model

Already mentioned, conventional relational model is not appropriate in our environment. In order to treat complex structured data efficiently, a nested relational model is adopted. The nested relational model with a set constructor and hierarchical attributes can represent complex data naturally, and can avoid unnecessary division of relations. The model is nearly the same as Kappa-II's data model, which shows us efficient handling of complex structured data. Because Kappa-P is also the database engine for the KBMS of the FGCS project, the semantics of nested relations matches the knowledge representation language, *QUIXOTE*[10] of the KBMS.

Term is added as a data type to store various knowledge. The character code of the PIM machine is based on 2-byte code, but the code wastes secondary memory space. In order to store a huge amount of data, such as a genome database in the near future, new data types and new indexed attributes are added.

- Configuration

The configuration of Kappa-P corresponds to the architecture of the PIM machine, and distinguishes inter-cluster parallelism from intra-cluster parallelism. Kappa-P is constructed of a collection of *element DBMSs* located in clusters. These element DBMSs co-operate to process each other's queries.

Figure 1 shows the overall configuration of Kappa-P. The global map of relations is managed by some element DBMSs called *server DBMSs*. Sever DBMSs manage not only global map but also ordinary relations. Element DBMSs, with the exception of server DBMSs, are called *local DBMSs*. *Interface processes* are created to mediate application programs and Kappa-P, and receive queries as messages.

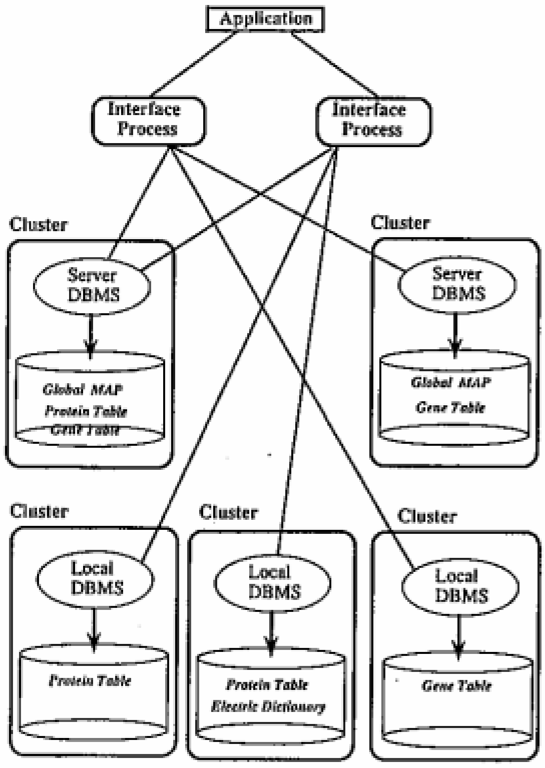

Figure 1: Configuration

• Data Placement

The placement of relations also corresponds to parallelism: inter element DBMS placement and intra element DBMS placement.

In order to use inter-cluster parallelism, relations can be located in some element DBMSs. The simple case is the distribution of relations like distributed DBMSs. When a relation needs a lot of processing power and higher bandwidth of disk access, the relation can be declustered as a horizontally partitioned relation and can be located in some element DBMSs. When a relation is frequently accessed in any query, some replicas of the relation can be made and can be located in some element DBMSs. However, in the current implementation, the replicated relation can be used for the global map only, that is, for server DBMSs.

Relations can be located in main memory or secondary memory in an element DBMS. Relations in main memory are temporary relations with no correspondent data in secondary memory. This means relation distribution in an element DBMS. A *quasi main memory database*, which guarantees to reflect those modifications to secondary memory, contains a relation in secondary memory and a replica of the relation in main memory.

• Query Processing

There are two kinds of commands for query processing: *primitive commands* and *KQL*, a query language based on extended relational algebra. Primitive commands are the lowest operations for relations, and can treat relations efficiently. The KQL is syntactically like KL1. New operations can be defined temporarily in a query.

A query in KQL is translated into sub-queries in intermediate operations for extended relational algebra, and is submitted to relevant element DBMSs. A query in primitive commands is submitted to relevant element DBMSs. The query is processed as a distributed transaction among relevant element DBMSs, and is finished under the control of two phase commitment protocol.

• Parallel Processing

Parallel processing of Kappa-P corresponds to the architecture of the PIM machine: inter-cluster parallelism among element DBMSs and intra-cluster parallelism in an element DBMS. The trade-off is processing power and communication overheads.

There are two kinds of parallel processing depending on data placement. Distribution of relations and horizontal partition of relations give us inter-cluster parallelism. In this case, a query is translated into sub-queries for some element DBMSs. Replication of a relation decentralizes access to the relation and improves availability.

• Compatibility to Kappa-II

The PIM machine is used via PSI machines acting as front-end processors. In order to use programs developed on PSI machines, such as terminal interfaces or application programs on Kappa-II, Kappa-P provides a program interface compatible to Kappa-II's primitive commands.

4 Nested Relational Model

A nested relational model is well known to reduce the number of relations in the case of multi-value dependency and to represent complex data structures more naturally than conventional relational model. However, there have been some nested relational models[8, 9, 2] since the proposal in 1978[6]. That is, even if they are syntactically the same, their semantics are not necessarily the same. In Kappa, one of the major problems is which semantics is appropriate for the 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 in 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:

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

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

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

$$\begin{array}{l} \{r[l_1 = a, l_2 = \{b_1, \cdots, b_n\}]\} \\ \qquad \Leftrightarrow \quad \{r[l_1 = a, l_2 = b_1], \cdots, r[l_1 = a, l_2 = b_n]\} \end{array}$$

The operation of "$\Rightarrow$" corresponds to an *unnest* operation, while "$\Leftarrow$" corresponds to a *nest* or *group-by* operation. "$\Leftarrow$", however, is not necessarily congruent 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 take such semantics is to retain the first order semantics for efficient processing and to remain compatible with widely used relational model. Let a nested relation

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

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

As a relation among facts in a database is conjunctive from a proof-theoretic point of view, we consider a query as the first order language: that is, in the form of rules. The semantics of a rule constructed by nested tuples with the above semantics is rather simple. For example, the following rule

$$\begin{array}{l} r[l_1 = X, l_2 = \{a, b, c\}] \\ \qquad \Leftarrow B, r'[l_3 = Y, l_4 = \{d, e\}, l_5 = Z], B'. \end{array}$$

can be transformed into the following set of rules without set constructors:

$$\begin{array}{l} r[l_1 = X, l_2 = a] \Leftarrow \\ \quad B, r'[l_3 = Y, l_4 = d, l_5 = Z], r'[l_3 = Y, l_4 = e, l_5 = Z], B'. \\ r[l_1 = X, l_2 = b] \Leftarrow \\ \quad B, r'[l_3 = Y, l_4 = d, l_5 = Z], r'[l_3 = Y, l_4 = e, l_5 = Z], B'. \\ r[l_1 = X, l_2 = c] \Leftarrow \\ \quad B, r'[l_3 = Y, l_4 = d, l_5 = Z], r'[l_3 = Y, l_4 = e, l_5 = Z], B'. \end{array}$$

That is, each rule can also be unnested into a set of rules without a set constructor. The point of efficient processing of Kappa relations is how to reduce the number of unnest and nest operations, that is, how to process sets directly.

Under the semantics, query processing to nested relations is different from conventional procedures. For example, consider a simple database consisting of only one tuple:

$$r[l_1 = \{a, b\}, l_2 = \{b, c\}].$$

For a query ?-$r[l_1 = X, l_2 = X]$, we can get $X = \{b\}$, that is, an intersection of $\{a, b\}$ and $\{b, c\}$. That is, a concept of unification should be extended. In order to generalize such a procedure, we must introduce two concepts into the procedural semantics[11]:

1) *Residue Goals*
   Consider the following program and a query:

$$\begin{array}{l} r[l = S'] \Leftarrow B. \\ ?\text{-}r[l = S]. \end{array}$$

   If $S \cap S'$ is not an empty set during unification between $r[l = S]$ and $r[l = S']$, new subgoals are $r[l = S \setminus S'], B$. That is, a residue subgoal $r[l = S \setminus S']$ is generated if $S_1 \setminus S_2$ is not an empty set. Otherwise, the unification fails. Note that there might be residue subgoals if there are multiple set values.

2) *Binding as Constraint*
   Consider the following database and a query:

$$\begin{array}{l} r_1[l_1 = S_1]. \\ r_2[l_2 = S_2]. \\ ?\text{-}r_1[l_1 = X], r_2[l_2 = X]. \end{array}$$

[2]The term "atomic" does not carry its usual meaning. For example, when an atomic value has a type *term*, the equality must be based on unification or matching.

Although we can get $X = S_1$ by unification between $r_1[l_1 = X]$ and $r_1[l_1 = S_1]$ and a new subgoal $r_2[l_2 = S_1]$, the succeeding unification results in $r_2[l_2 = S_1 \cap S_2]$ and a residue subgoal $r_2[l_2 = S_1 \setminus S_2]$. Such a procedure is wrong, because we should have an answer $X = S_1 \cap S_2$. In order to avoid such a situation, the binding information is temporary and plays the role of the constraints to be retained.

The procedural semantics of extended relational algebra is defined based on the above concepts. According to the semantics, a Kappa database is allowed not necessarily to be *normalized* also in the sense of nested relational model, in principle: that is, it is unnecessary for users to be conscious of row nest structure. On the other hand, in order to develop deductive databases on Kappa, a logic programming language, called CRL [11] is developped on the semantics and is further extended in *QUIXOTE* [10].

There remains one problem such that unique representation of a nested relation is not necessarily decided in the Kappa model, as already mentioned. In order to decide a unique representation, each nested relation has a sequence of attributes to be nested in Kappa.

Consider some examples of differences among some models. First, assume a relation r consisting of two tuples:

$$\{r[l_1 = a, l_2 = \{b, c\}],\\ r[l_1 = a, l_2 = \{c, d\}]\}$$

By applying a row-nest operation on l_2 to R, we get two possible relations:

$$\{r[l_1 = a, l_2 = \{b, c, d\}]\}\\ \{r[l_1 = a, l_2 = \{\{b, c\}, \{c, d\}\}]\}$$

According to the semantics of Kappa and Verso[9], we get the first relation, while, according to one of DASDBS[8] and AIM-P[2], we get the second.

Secondly, consider another relation r' consisting of only one tuple:

$$\{r'[l_1 = a, l_2 = \{b_1, b_2\}, l_3 = \{c_1, c_2\}]\}$$

By applying selection operations $\sigma_{l_2=b_1}$ and $\sigma_{l_3=c_1}$, we get the following two relations, respectively:

$$\{r'[l_1 = a, l_2 = b_1, l_3 = \{c_1, c_2\}]\}\\ \{r'[l_1 = a, l_2 = \{b_1, b_2\}, l_3 = c_1]\}$$

If we apply a union operation to the above two relations, we get two possible relations. According to the semantics of Verso, we get the following (original) relation:

$$\{r'[l_1 = a, l_2 = \{b_1, b_2\}, l_3 = \{c_1, c_2\}]\}.$$

That is, although the combination of $l_1 = a$, $l_2 = b_2$, and $l_3 = c_2$ is not selected after two selections, it comes back to life in the result of the union. On the other hand, according to the semantics of Kappa, we have one of the following:

$$\{r'[l_1 = a, l_2 = b_1, l_3 = \{c_1, c_2\}],\\ r'[l_1 = a, l_2 = b_2, l_3 = c_1]\}, \text{ or}$$

$$\{r'[l_1 = a, l_2 = \{b_1, b_2\}, l_3 = c_1],\\ r'[l_1 = a, l_2 = b_1, l_3 = c_2]\}$$

Which relation is selected depends on nested sequence defined in the schema.

According to the above semantics, the Kappa model guarantees more efficient processing by reducing the number of tuples and relations, and more efficient representation by the complex construction than relational model.

5 Data Placement

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

- Distribution

Distribution of relations is a simple case like distributed DBMSs. When relations are distributed in some element DBMSs, larger processing power can be obtained, but communication overheads may be generated at the same time. A database designer should be responsible for distribution of relations, because how to distribute relations relates to relationships among relations and kinds of typical queries to the database. In typical queries, strongly related relations should be in the same element DBMS, and loosely related relations might be in different element DBMSs.

A query to access these relations is divided into subqueries for some element DBMSs by an interface process (Figure 1), and each sub-query is processed as a distributed transaction.

- Horizontal Partition

A horizontally partitioned relation is a kind of declustered relation. It is logically one relation, but consists of some sub-relations containing distributed tuples according to some declustering criteria. A horizontally partitioned relation is effective when the relation needs a lot of processing power and higher bandwidth of disk access. For example, it is effective in a case of a molecular biological database which includes sequence data which requires homology search by a pattern called motif. A database designer is also responsible for horizontal partition of relations, because horizontal partition does not always guarantees efficient processing if it does not satisfy declustering criteria.

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

- Replication

Replication of a relation in some element DBMSs enables us to decentralize to access the relation, and to improve availability. Only the global map held in server DBMSs is replicated with a voting protocol in the current implementation of Kappa-P. The replication avoids centralizing access for server DBMSs, and even if some server DBMSs would stop, server facilities can work on.

6 Management of Global Information

Metadata of Kappa-P is divided into two kind of information: global information and local information. The global information consists of logical information, such as the database name and relation names, and physical information about element DBMSs, such as start-up information, current status, and stream to communicate. The local information also consists of logical information, such as the local database name and schema, and physical information, such as file names and physical structures of relations. Each element DBMS manages local information, and server DBMSs manage global information in addition ordinary relations. The role of server DBMSs is management of global information, especially, management of relation names for query processing and establishment of communication path between an interface process and element DBMSs.

- Management of Relation Names

It is necessary to guarantee the uniqueness of relation names. The simplest way is that a relation name forces to contain the relevant element DBMS name. Such a name is not suitable for Kappa-P, because Kappa-P treat logically one database. Server DBMSs manage relation names centrally, and provide location independent relation names. The information consists of a relation name and an element DBMS name in which the relation exists. This information is referred in order to find relevant element DBMSs from relation names at the beginning of query processing. When a relation is created, a message to register the relation name information is sent from the transaction. When a relation is deleted, a message to erase the relation name information is sent from the transaction.

Global information is replicated in order to decentralize ascesses to server DBMSs. Replication of the information is implemented by using a voting protocol. In order to access server DBMSs, a distributed transaction uses two phase commitment protocol.

- Management of Physical Information

Server DBMSs manage physical information, such as start-up information, current status, and stream to communicate. Sever DBMSs watch the state of element DBMSs. At the beginning of query processing, a server DBMS connects an interface process to relevant element DBMSs.

7 Query Processing

7.1 Query Language

There are two kinds of language for query processing: KQL, a query language based on extended relational algebra, and primitive commands. A query in both primitive commands and KQL is in the form of a message to an interface process, and the result is returned through the tuple stream which dose not have cursors as SQL.

- KQL(Kappa Query Language)

KQL is syntactically similar to KL1. Operations of extended relational algebra are written like predicates, and new operations can be defined temporarily, which take relations only as their arguments. Figure 2 shows a query in KQL.

- Primitive Commands

Primitive commands are the lowest operation for nested relations and a collection of unary operators for a nested relation. Figure 3 shows an example in primitive commands.

7.2 Query Processing

A query in KQL is processed in the following steps.

- Query Translation

```
go(Result::result1, Temp::result2) :- true |
      selection(table2, ''(from = "icot"), Temp),
      difference(table1, table1, EmptyTable),
      transitive_closure(table1, EmptyTable,
                    table1, Result).

transitive_closure(Delta, In, R, Out) :-
      empty(Delta) |
      In = Out.

transitive_closure(Delta, In, R, Out) :- true |
      join(In, In, ''(to = from), In1),
      projection(In1, {'1.from', '2.to'},
              In2::{from, to}),
      union(In2, R, NextIn),
      difference(NextIn, In, Delta),
      transitive_closure(Delta, NextIn, R, Out)
```

Figure 2: Query in KQL

```
ifp:create(pc, off, IFP, Status0),
IFP  = [open([], Status1),
       begin_transaction([table1(read)],[],Status2),
       create_format(table1, ''(*), FMT, Status3),
       read_record(table1,FMT,rid,TupleStream0,S4)
     | IFP1],
TupleStream0 = [Buffer1 | TupleStream1],
               %% Buffer1 = [Tuple1,...TupleN]
TupleStream1 = [Buffer2 | TupleStream2],
               %% Buffer2 = [Tuple21,...Tuple2N]

IFP1 = [ end_transaction(Status5),
       close(Status6)].
```

Figure 3: Primitive Commands

A query in KQL is translated into sub-queries, which is called *intermediate operations* (shortly, *operations* in the following procedures) for extended relational algebra, by an interface process of Kappa-P.

1) Get relation names by parsing a query, and get location information of the relations from randomly selected server DBMSs.

2) Get schemata and supplementary information of the relations from relevant element DBMSs. In case of horizontally partitioned relations and quasi main memory relations, information of sub-relations is assembled into one. Supplementary information is followings:

   - List of indexed attributes
     The algorithm of query processing is dependent on whether an attribute is indexed or not.
   - Uniqueness of attribute value
     The algorithm is dependent on whether an attribute value is unique or not.
   - Kinds of attribute values and the number of attribute values
     The information is used to estimate amounts of intermediate results, and reduce communication costs.
   - The number of tuples and thé average size of tuples
     The information is also used to estimate amounts of intermediate results.

3) Replace an operation for a horizontally partitioned relation with some operations for sub-relations and add merge operations. In case of a quasi main memory relation, replace an update operation with operations both the secondary memory relation and the temporary relation. Replace a non-update operation with an operation for the temporary relation.

4) The executing order of operation in the query is extracted from the query, and operations to control executing order are embedding in the query. Since the query can include update operations, it is impossible to control the data flow graph only.

5) Using basic optimization techniques by supplementary information of relations, the query is translated, and an algorithm for processing extended relational algebra is determined. In this phase, some execution plans are produced.

6) According to the location information of relations, the candidates are divided into sub-sequences to minimize the communication costs estimated by the supplementary information. Sub-sequences with the least communication cost are chosen, and operations to transfer tuples are embedded in the sub-sequences.

7) Each sub-sequence is translated into KL1 program with procedures calling intermediate operations.

- Query Execution

Each sub-sequence is sent to the related element DBMS, and processed. Each sub-sequence is executed as a distributed transaction with two phase commitment protocol. Although processing in an element DBMS is based on tuple streams, data in other element DBMSs are accessed via transfer operations embedded in the query translation phase.

7.3 User Process in Element DBMS

Because both Kappa-P and application programs work together on the same machine, the system cannot only provide higher communication bandwidth between them, but can also reduce communication overheads between them by allocating them in the same cluster.

In primitive commands, a tuple filter are taken as the argument of a read operation. The read operation invocates the filter in the same cluster in which a relation exists. If the relation is horizontally partitioned. filters for each sub-relation is invocated. and the outputs of all filters are merged into one.

In KQL, a filter is specified as one of the new operations.

8 Element DBMS

An element DBMS contains full database management facilities. and accepts intermediate operations for extended relational algebra and primitive commands. We are not concerned with communication overheads in element DBMSs. Kappa-P uses parallel processing on a shared memory only. but doesn't use parallel operations for secondary memory in element DBMSs. in the current implementation.

- Parallel Processing by Tuple Stream

Stream programming is a very typical programming style in KL1. In general. a query can be expressed as a graph, which consists of some nodes corresponding to the operations and arcs corresponding to relationships among operations. In KL1, the graph corresponds to the processing structure of the query. The nodes become processes. and arcs become streams through which tuples are sent. In KL1. the number of tuples in the streams does not only depend on the amount of intermediate results, but also the number of processes to be scheduled. So. it is very important to control the number of tuples. and to drive the streams on demand with double buffering.

Figure 4 shows an example for parallel processing by tuple stream: $Table\,3 = \pi_{[a,b]}Table\,1 \bowtie Table\,2$.

- Parallel Processing of Primitive Commands

Primitive commands process various operations in parallel for nested relations. for instance. operations set for and index operations of temporary relations.

A set is a collection of tuple identifiers. and is obtained by restriction operations. In order to parallelize set operations, a set is partitioned according to the range of tuple identifiers.

Index structure of temporary relations is T-tree[5], which is more sufficient in main memory than B-tree. Range retrieval operations. we are processed in parallel. In general, leaf nodes are connected in order like B^+-tree to trace succeeding leaf node directory. In our experiments. such a structure can't work efficiently in KL1. Range retrieving on a tree. whose leaf nodes are connected. is done following steps: finding the minimum value of the range, and then. tracing through the

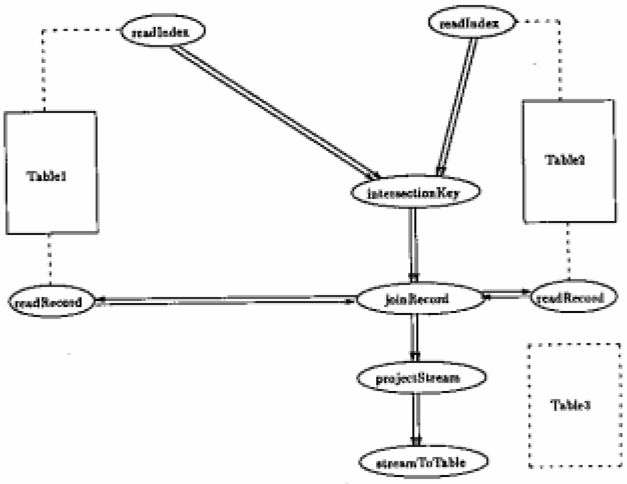

Figure 4: Parallel Processing by Tuple Stream

connection until the maximum value is found. These steps are almost processed sequentially. Assuming that H is the height of the tree. and R is the number of leaf nodes between the range. the number of comparison of values is $H + R$. On the other hand. range retrieving on a tree whose leaf nodes are not connected is done following steps: finding a minimum value of the range. finding a maximum value of the range. and collecting values between them. These steps are almost processed in parallel. The number of comparison is $2H$. The latter has advantages about parallelism. wide range retrieving. and efficient implementation in KL1.

- Main Memory Database Facilities

Each cluster of PIM has hundreds of mega bytes of main memory. In order to use such a large memory effectively. Kappa-P provides temporary relations and quasi main memory database facilities. Because tuples of nested relations with variable occurrences and strings are complex. such a structure can be handled more efficiently in main memory than in secondary memory.

Temporary relations exist only in main memory having no correspondent data in secondary memory. so modifications to the temporary relations are not reflected to secondary memory. But. temporary relations are useful for application programs which create many intermediate relations such as deductive databases. The temporary relations have the same interface as secondary memory relations.

A quasi main memory database. which guarantees to reflect those modifications to secondary memory. is not a pure main memory database. but parallel processing enables the quasi main memory database to work with nearly the same throughput as a main memory database. A quasi main memory relation is a kind of replicated relations consisting of a pair of a secondary memory relation and a temporary relation. Kappa-P guarantees that both relations have the same logical structure, such as tuples, indexed attributes, and the same tuple identifiers, even if the relation is updated. Operations except for update operations can be executed by the temporary relation, because the temporary relation is processed faster than the secondary memory relation. Update operations should be executed by relations in parallel and asynchronously, and synchronization is achieved by two phase commitment protocol, which guarantees the equivalence of their contents.

9 Conclusions

In this paper, we described a parallel DBMS Kappa-P.

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. The smallest configuration of Kappa-P is almost the same as Kappa-II. Compared both systems on the same machine, Kappa-P works with almost same efficiency as Kappa-II. Kappa-P is expected to work on PIM more efficiently than Kappa-I. We will make various experiments for efficient utilization of parallel resources, and show that the system provides KBMSs and KIPSs with efficient database management facilities in the FGCS prototype system.

Acknowledgment

The Kappa project has had many important contributions in addition to the listed authors. The members of biological databases of the third research laboratory: Hidetoshi Tanaka and Yukihiko Abiru, the users of Kappa-II, and Kaoru Yoshida of LBL have shown us many suggestions for improvements. We thank Hideki Yasukawa of the third research laboratory for useful suggestions. The authors are grateful to Kazuhiro Fuchi and Shunichi Uchida for encouraging the projects.

References

[1] D. J. DeWitt and J. Gray, "Parallel Database Systems: The Future of Database Processing or a Passing Fad ?", *SIGMOD RECORD*, Vol.19, No.4, Dec.,1990.

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

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

[4] M. Kawamura and H. Sato, "Query Processing for Parallel Database Management System", *Proc. KL1 Programming Workshop '91*, 1991. (in Japanese)

[5] T. J. Lehman and M. J. Carey, "A Study of Index Structures for Main Memory Database Management Systems", *Proc. VLDB*, 1986.

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

[7] H. Sato and M. Kawamura, "Towards a Parallel Database Management System (Extended Abstract)", *Proc. Joint American-Japanese Workshop on Parallel Knowledge Systems and Logic Programming*, Tokyo, Sep. 18-20, 1990.

[8] 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.

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

[10] H. Yasukawa, H. Tsuda, and K. Yokota, "Object, Properties, and Modules in *QUIXOTE*", *Proc. FGCS'92*, Tokyo, June 1-5, 1992.

[11] K. Yokota, "Deductive Approach for Nested Relations", *Programming of Future Generation Computers II*, eds. by K. Fuchi and L. Kott, North-Holland, 1988.

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

[13] K. Yokota and S. Nishio, "Towards Integration of Deductive Databases and Object-Oriented Databases — A Limited Survey", *Proc. Advanced Database System Symposium*, Kyoto, Dec., 1989.

[14] 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.