병렬 데이타베이스 컴퓨터 구조의 성능 분석

이 용 규†

요 약

현재 병렬 데이타베이스 컴퓨터가 광범위하고 성공적으로 활용되고 있다. 이의 구조로는 주기억 장치와 디스크를 공유하지 않는 구조, 두가지를 모두 공유하는 구조, 디스크만을 공유하는 구조, 그리고 절충형 구조 등의 네가지 구조가 있다. 이 논문에서는 데이타베이스 컴퓨터 구조의 성능을 비교 분석하기 위하여 데이타베이스 컴퓨터 구조를 추상적인 모형으로 정의하고, 각각의 모형에 대하여 절충형 해쉬 조인 연산의 수행시간을 수식화한 성능식을 구하여 여러 가지 데이타베이스 컴퓨터 구조 모형의 수행시간을 비교 분석한다.

Performance Analysis of Parallel Database Machine Architectures

Yong Kyu Lee†

ABSTRACT

The parallel database machine approach is currently widely and successfully used. There are four major architectures which are used in this approach: shared-nothing architecture, shared-everything architecture, shared-disk architecture, and hybrid architecture. In this paper, we use an analytical model to evaluate the performance of these database machine architectures. We define an abstract model for each type of database machine design to obtain performance equations describing the execution times with respect to the hybrid hash join operation. Using the performance equations, we evaluate the execution times of the various database machine design models.

1. Introduction

One of the problems that conventional database management systems face is the I/O bottleneck, caused by slow disk access time with respect to main memory access time. Initially, database machine designers tried to solve this problem by reducing the amount of data to be moved through the I/O channel. They designed special-purpose database machines, such as intelligent secondary storage devices and database filters. The intelligent secondary storage device was specially designed to eliminate the limitations of the conventional secondary devices, and the database filter achieved the same filtering effect by using an intelligent secondary storage controller [22]. However, their efforts have failed due to a poor performance/cost ratio compared to the software solution that uses conventional processors, memories, and disks [8, 24, 25].

※ 본 연구는 동국대학교 신임교수 연구지원으로 이루어짐
† 종신회원 : 동국대학교 컴퓨터정보통신공학부
논문접수 : 1997년 4월 15일, 심사완료 : 1998년 2월 16일

A general solution to the I/O bottleneck is to increase the I/O bandwidth by exploiting parallelism [25]. The parallel database machine approach, which uses a large number of smaller machines, is currently widely and successfully used. There are two main reasons for the dominance of the parallel database machines. One is the remarkable progress of the multiprocessor technology which made it possible to build high-performance database systems at a much lower price than equivalent mainframe computers. And the other is the widespread use of the relational data model which is well suited to parallel execution.

There are four major architectures which are used in the parallel database machine approach: shared-nothing architecture, shared-everything architecture, shared-disk architecture, and hybrid architecture. It seems that the shared-nothing architecture attracts many database designers, but there are still many open problems and issues to be solved before deciding which architecture is best for database machines [16, 24, 25]. Among them, one important issue is to know which one shows better performance in various conditions.

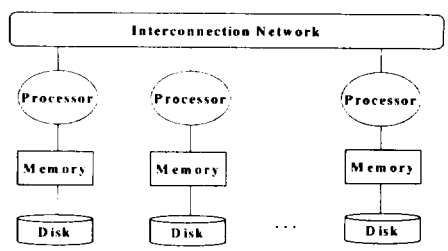
There have been a number of efforts to analyze the performance of different database machine architectures. In [10, 22], analytical models have been used to evaluate the performance of various database machine architectures. In [3, 4], simulation models have been used to compare the performance of parallel database machine architectures in transaction processing. These studies, however, have not fully reflected the current trends of database machine design.

In this paper, we use an analytical model to evaluate four parallel database machine architectures. We classify presently available database machines into these four classes, and define an abstract model representing each class. Using performance equations, we analyze the performance of each model with respect to the relational join operation.

2. Parallel Database Machine Architectures

2.1 Shared-Nothing Architecture

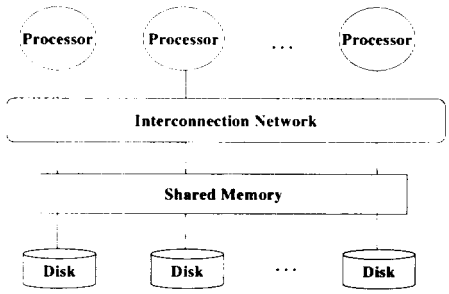
In the shared-nothing architecture (Fig.1), each processor has a portion of the database and only that portion can be directly accessed by the processor. Processors communicate with one another only by sending messages via a high speed interconnection network. There are no shared storage devices in this architecture. Examples of the shared-nothing parallel database machine architecture include Teradata's DBC/1012 [6], Tandem's NonStop SQL [23], Gamma [7], Bubba [5], EDS [1],and Arbre [14].



(Fig.1) Shared-Nothing Architecture

2.2 Shared-Everything Architecture

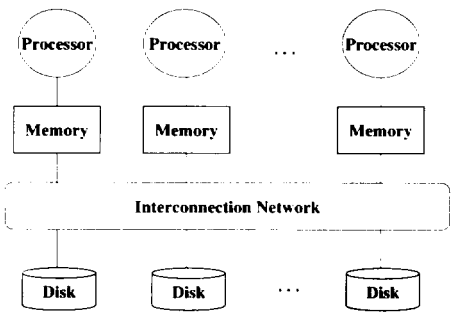
In this model (Fig.2), processors share main memory as well as disks through a high speed interconnection network. Examples of the sha-red-everything database machine architecture include XPRS [21], Volcano [9], Symmetry S81 [12], and DBS3 [2].



(Fig.2) Shared-Everything Architecture

2.3 Shared-Disk Architecture

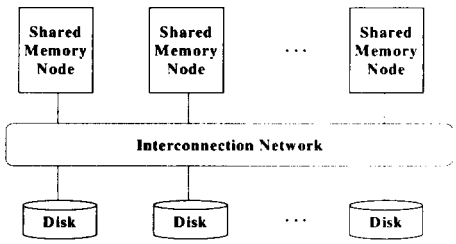
In the shared-disk architecture (Fig.3), all the disks containing the databases are shared among the different processors, and each processor has its own exclusive memory. Examples of the shared-disk parallel database systems are IMS/VS [20], Amoeba [19], ORACLE on VAXcluster [13], and TPF [17].



(Fig.3) Shared-Disk Architecture

2.4 Hybrid Architecture

In the hybrid architecture (Fig.4), each shared memory unit consists of a small number of shared-everything processors. However, the whole system may adopt shared-nothing architecture or shared-disk architecture. Examples of this architecture are SDC [11], P-90 [6], and NCR 3700 [26].



(Fig.4) Hybrid Architecture

3. Performance Analysis

In this section, we describe an abstract model for each database machine architecture which will be used to analyze performance. We also include an abstract model representing conventional single processor systems since many database systems are still running on conventional systems. Based on the model, we have obtained performance equations using the notations similar to those in [22].

3.1 Abstract Models

3.1.1 Conventional System Model

This model has a processor, a main memory, and a disk. We have assumed that this model supports compiled user queries and sophisticated query execution strategies for relational operations. This model will also be used as a host computer in the definition of other database machine models. Throughout this paper, we have made the following assumptions:

- There are no special-purpose hardwares for database operations such as sorting devices and joining devices.
- Each relation does not fit entirely in the main memory of a processor.
- The pages of a relation are stored in

consecutive disk tracks.

- The processor execution time and the disk access time are not overlapped.
- When the result of an operation is generated, it is consumed immediately by the user application program.
- Each track is the same size as a page.
- The basic processing and moving unit is page.
- There is no cache memory.

3.1.2 Shared-Nothing Architecture Model

In the shared-nothing architecture, each processor has its own exclusive memory and secondary storage device. The processors can communicate with one another only by sending messages through the interconnection network. Additionally, we have assumed the following characteristics of this model:

- The processors are conventional Von Neumann processors with identical capabilities.
- The front-end host computer issues database commands and collects the results from the back-end database machine.
- Each relation is horizontally and evenly partitioned in the secondary storage devices.
- The interconnection network provides broadcasting capability.
- There is no contention on shared resour -ces.

These assumptions are applied to other database machine models as well as the shared-nothing model.

3.1.3 Shared-Everything Architecture Model

In the shared-everything architecture, all memory modules and disks are shared by the processors. We added the following assumptions to this model:

- The main memory consists of a number of memory modules, and the different modules can be accessed simultaneously by more than one processor.
- The disk consists of a number of disk drives, and the different drives can be accessed simultaneously.
- The operation of the entire system is controlled by one operating system.
- A global concurrency control mechanism for the shared memory and disk is provided.
- The host computer can access the shared memory.

3.1.4 Shared-Disk Architecture Model

In the shared-disk architecture, each processor has its own exclusive memory, but each processor can directly access any disk. The following assumptions are added to this model:

- The disk consists of a number of disk drives, and the different drives can be accessed simultaneously.
- A global concurrency control mechanism for the shared disk is provided.

3.1.5 Hybrid Architecture Model

We have assumed that the hybrid architecture model is hybrid of the shared-nothing architecture and the shared-everything architecture. That is, the overall architecture adopts the shared-nothing architecture, but each node (shared-memory node) adopts the shared-everything architecture. We added the following assumptions to this model:

- All the shared-memory nodes have identical capabilities.

- The main memory in a node consists of a number of memory modules, and the different modules can be accessed simultaneously by more than one processor.
- The disk in each unit consists of a number of disk drives, and the different drives can be accessed simultaneously.
- In each node, the operation of the system is controlled by one operating system.
- A global concurrency control mechanism for the shared memory and disk is provided in each node.

3.2 Join Operation

We use the hybrid hash join operation to evaluate the performance of database machine architectures. The hybrid hash join algorithm for a multiprocessor environment is described in [18]. The parallel algorithm for the equijoin of two relations R and S is as follows.

Initially, both relations R and S are horizontally and evenly distributed among the processors. Suppose there are t processors in the system. To perform the hybrid hash join, each processor executes the following steps in parallel:

1. Each processor scans its fragment of R. As each tuple is processed, the processor computes a hash function h_1. This hash value is used to determine to which processor the tuple should be sent by a lookup in a split table. The tuples of R received at processor p_i form the partition R_i.
2. As a processor p_i receives an incoming tuple of R_i, p_i applies another hash function h_2, which determines to which local hash bucket the tuple belongs. The hybrid hash join algorithm keeps one local hash bucket in memory, and spools the rest to its local disk. An in-memory hash table is built out of the tuples that fall into local bucket zero, for use in the probing phase of the join.
3. Each processor scans its fragment of S. As each tuple processed, the processor com- putes the hash function h_1 and looks up this value in a split table to determine which processor the tuple should be sent.
4. When a processor p_i receives an incoming S tuple, it applies the hash function h_2 to determine to which local bucket the tuple belongs. If this local bucket is bucket zero, p_i immediately probes the local bucket hash table for any matching tuples; otherwise, the tuple is spooled to disk into the appropriate local bucket of S_1.
5. After S has been redistributed, the join of the local bucket zero has been completed. Then each processor repeatedly reads a local bucket of R_1 into memory, then scans the corresponding local bucket of S_1 to find all joining tuples, until all local buckets have been processed.

4. Performance Comparison

This section presents the performance comparison of the database machine architecture models. The parameter values are shown in <Table 1>.

In order to make the comparison fair, we have assumed that all the database machine models have the same number of processors, memory modules, and disk drives.

Each model consists of 16 processors and 16

〈Table 1〉 Parameter Values

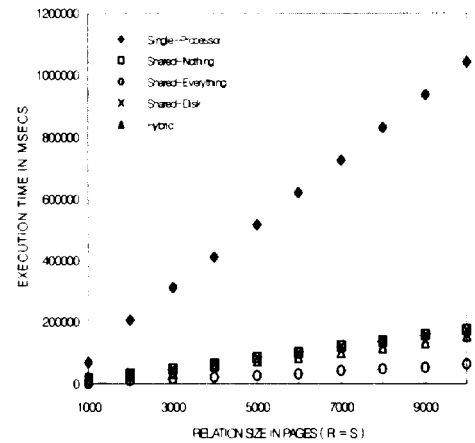
Parameter	Value
CPU speed	4 MIPS
page network latency	5.6 ms
simple hash	10 instructions
start an i/o	1000 instructions
disk page size	16 kilobytes
hash	20 instructions
initiate join	40000 instructions
write tuple into buffer	100 instructions

disk drives. In the hybrid model, there are four shared-memory nodes, each of which consists of four processors and four disk drives. The values for the processor, disk, and network parameters are based on the measurements from the Intel iPSC/2 Hypercube and Fujitsu Model M2266 (1 GB, 5.25") disk drive [15]. The disk page size has been modified to make the evaluation easy. Other software parameters are based on instruction counts taken from the Gamma database machine [15]. Since we have assumed that relations are evenly partitioned among the disk drives, we have not considered memory contention, disk contention, and network contention in our evaluation.

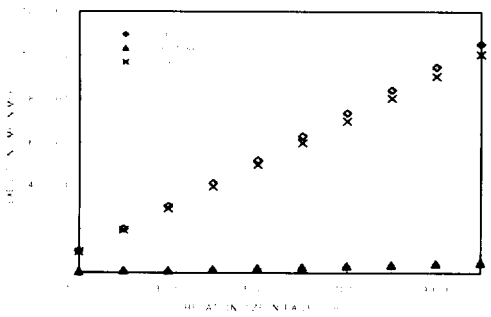
Using the performance formulas, we have quantified the hybrid hash join performance of the various database machine architecture models through a number of different tests. (Fig.5) presents the execution times for the various relation sizes when the join selectivity factor is 0.00001. We have assumed that the bucket size is 100 disk pages. The shared-everything architecture model shows the best performance. The difference in the execution times between the shared-everything model and the other parallel database machine architecture models is quite large. This reflects the fact that there is no communication overhead during join processing in the shared-everything model, while most of the execution time is spent communicating between processors in the other models. The hybrid architecture model shows somewhat better performance than the shared-nothing and shared-disk architecture models. This is because the communication overhead is not required within a shared-memory node in the hybrid model, while the relations have to be distributed among the processors through the interconnection network during join processing in the shared-nothing and shared-disk models. The shared-nothing model and shared-disk model have the same execution times (same curve). We have obtained similar results with other join selectivity factors (0.0001, 0.001).

4.1 Comparison between Time Components

We compare the execution times between major time components to process the hybrid hash join operation when the join selectivity factor is 0.0001. In each model, the execution time consists of three major time components: CPU time, disk access time, communication time through the interconnection network.



(Fig.5) Join Execution Times: Selectivity Factor = 0.00001



(Fig.6) Single-Processor: Join Selectivity Factor = 0.0001

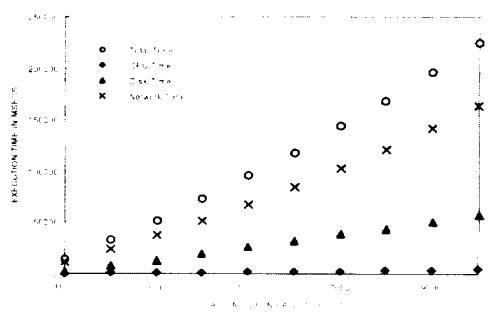
(Fig.6) presents the execution times of the major time components in the conventional single-processor model. Since the communication time is not required in this model, the execution time for the join operation consists of two time components: CPU time and disk access time. In this model, the disk access time accounts for the greater portion of the join execution times. This shows that most of the execution time is spent accessing the disk during the bucket partitioning and joining phase. When the size of each participating relation is 1000 pages (16 megabytes), the disk access time accounts for 96.09 % of the total execution time. As the relation size grows, the portion of the disk access time somewhat decreases. The disk access time occupies 95.30 % of the execution time when the relation size is 10000 pages (160 megabytes). When the join selectivity factor is 0.001 and the relation size is 10000 pages, the disk access time accounts for 87.79 % of the total execution time.

When the size of each participating relation is 1000 pages, 82.55 % of the total time is used for the communication, while 15.59 % is used for the disk access. As the relation size grows, the portion of the communication time decreases. When the relation size is 10000 pages, the communication time accounts for 72.58 % of the execution time.

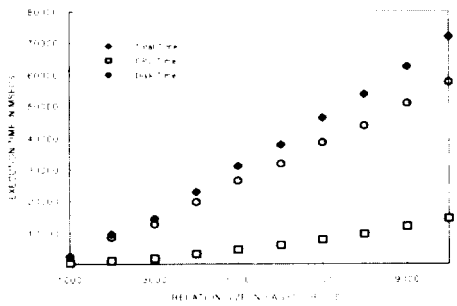
In the shared-everything model, there is no communication overhead during join processing. Thus, the execution time consists of two major time components: CPU time and disk access time. The time components of this model are presented in (Fig.8). When the size of each participating relation is 1000 pages, the disk access time accounts for 85.85 % of the total execution time. The percentage of the disk access time decreases to 80.10 % when the relation size grows to 10000 pages. Since this model does not suffer from communication overhead, it outperforms all the other models.

The execution times of the major time components in the shared-disk model are the same as those of the shared-nothing model.

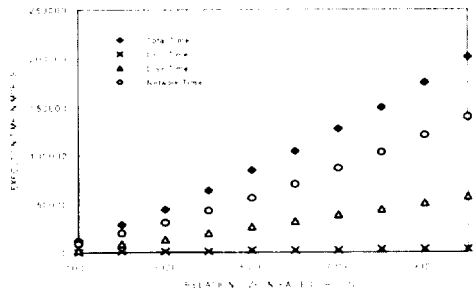
In the hybrid model, most of the time to process the join operation is spent communicating between processors like the shared -nothing and shared-disk models. The results in (Fig.9) are similar to what was seen in (Fig.7). However, the hybrid model shows somewhat better response times. When the size of each relation is 1000 pages, the communication time accounts for 79.68 % of the total execution time. The portion of the communication time increases as the relation size grows. The percentage of the communication time decreases to 69.62 % when the relation size is 10000 pages.



(Fig.7) Shared-Nothing: Join Selectivity Factor = 0.0001

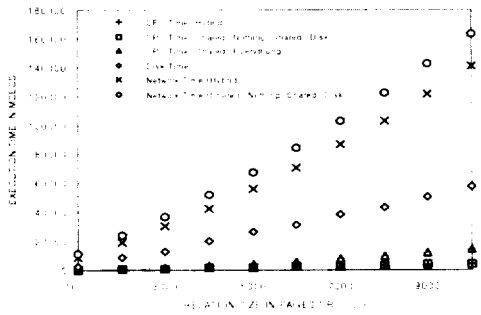


(Fig.8) Shared-Everything: Join Selectivity Factor = 0.0001



(Fig.9) Hybrid : Join Selectivity Factor = 0.0001

In the parallel architecture models, the disk access times for the join operation are the same for all the models. The CPU processing times are almost same for all the models. It is the network communication time that makes the response times of a model different from the other models. This fact is illustrated in (Fig.10). It shows the CPU time and disk access time to process the hybrid hash join operation in the parallel database machine models when the join selectivity factor is 0.0001. It also presents the network communication times of the hybrid, shared-nothing, and shared-disk model. The network communication time accounts for the larger portion of the join execution times in the shared-nothing and shared-disk models. The network times of the shared-nothing and shared-disk model are the same.



(Fig.10) Time Components of the Parallel Models: Join Selectivity Factor = 0.0001

5. Conclusions

We have used an analytical model in order to compare the performance of parallel database machine architectures. We have classified presently available database machine designs into four generic classes: shared-nothing architecture, shared-everything architecture, shared-disk architecture, and hybrid architecture. An abstract model has been defined to represent each class of database machine designs.

Using the performance equations, we have evaluated the execution times of the hybrid hash join for the parallel database machine models. In our evaluation, we have assumed that each model consists of 16 processors. In the experiments, the shared-everything model has shown the best performance, since it does not require the communication overhead between processors. This result is the same as what is described in [4]. The hybrid model has shown somewhat better response times than the shared-nothing and shared-disk models in join processing. This implies that the hybrid of the shared-nothing and shared-everything architectures can be considered as an alternative database machine design.

We have analyzed the execution times of the major time components to process the hybrid hash join operation. In each model, the execution time of a join operation consists of three major time components: CPU time, disk access time, and network communication time. In the parallel architecture models, the CPU time and disk access time for the join operation are almost same for all the models. It is the network communication time that makes the response times of a model different from the other models. The network communication time accounts for the greater portion of the join execution times in the shared-nothing and shared-disk models.

References

[1] F. Andres, et al., "EDS - Collaborating for a High Performance Parallel Relational Dat abase," Proc. ESPRIT '90 Conf., Kluwer Ac ademic Publisher, 1990.

[2] B. Bergsten, M. Couprie, and P. Valduriez, "Prototyping DBS3, a Shared-Memory Parallel Database System," Proc. Int'l Con f. on Parallel and Distributed Information Sys-tems, 1991.

[3] A. Bhide and M. Stonebraker, "A Performance Comparison of Two Architectures for Fast Transaction Processing," Proc. 4th Int'l Conf. on Data Engineering, 1988.

[4] A. Bhide, "An Analysis of Three Trans-actio n Processing Architectures," Proc. 14th VLD B Conf., 1988.

[5] H. Boral, et al., "Prototyping Bubba, A Highly Parallel Database System," IEEE Trans. on Knowl. Data Eng., 2(1), Mar. 1990.

[6] P. Carino and P. Kostamaa, "Exegesis of DBC/1012 and P-90 - Industrial Superco mputer Database Machines," Proc. 4th Int' l PARLE Conf., June 1992, in Lecture Not es in Computer Science 605, Springer-Verl ag, 1992.

[7] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. I. Hsiao, and R. Rasmussen, "The Gamma Database Machine Project," IEEE Trans. on Knowl. Data Eng., 2, Mar. 1990.

[8] D. J. DeWitt and J. Gray, "Parallel Database Systems : The Future of High Performance Database Systems," CACM, 35(6), June 1992.

[9] G. Graefe, "Volcano : An Extensible and Parallel Query Evaluation System," Oregon Graduate Center Technical Report CS/E 89-006, July 1989.

[10] P. B. Hawthorn and D. J. DeWitt, "Performance Analysis of Alternative Database Machine Architectures," IEEE Trans. on S oftware Eng., 8(1), Jan. 1982.

[11] M. Kitsuregawa, Y. Ogawa, "Bucket Spreading Parallel Hash : A New, Robust, Parallel Hash Join Method for Data Skew in the Super Database Computer (SDC)," Proc. 16th VLDB Conf., 1990.

[12] M. Kitsuregawa, S. Tsudaka, and M. Naka no, "Parallel GRACE Hash Join on Shared-Everything Multiprocessors : Imple-mentation and Performance Evaluation on Symmetry S81," Proc. 8th Int'l Conf. on Data Engineering, 1992.

[13] N. P. Kronenberg, H. M. Levy, and W. D. Strecker, "VAXclusters : A Closely-Coupled Distributed System," ACM Trans. on Computer Systems, 4(2), May 1986.

[14] R. Lorie, J. Daudenarde, G. Hallmark, J. Stamos, and J. Young, "Adding Intra-Transaction Parallelism to an Existing DBMS : Early Experience," IEEE Data Engineering Newsletter, 12(1), Mar. 1989.

[15] M. Mehta, V. Solviev, and D. J. DeWitt, "Batch Scheduling in Parallel Database Systems," Proc. 9th Int'l Conf. on Data

Engineering, 1993.

[16] H. Pirahesh, C. Mohan, J. Cheng, T. S. Liu, and P. Selinger, "Parallelism in Relatio-nal Database Systems : Architectural Issues and Design Approaches," Proc. 2nd Int'l Symposium on Databases in Parallel and Distributed Systems, July 1990.

[17] T. W. Scrutchin, Jr., "TPF : Performance, Capacity, Availability," Proc. IEEE COMPCON Spring '87 Conf., Feb. 1987.

[18] A. Shatdal and J. F.· Naughton, "Using Shared Virtual Memory for Parallel Join Processing," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1993.

[19] K. Shoens, et al., "The Amoeba Project," Proc. IEEE COMPCON Spring '85, Feb. 1985.

[20] J. P. Strickland, P P. Uhrowczik, and V. L. Watts, "IMS/VS : An Evolving System," IBM Systems Journal, 21(4), 1982.

[21] M. Stonebraker, R. Katz, D. Patterson, and J. Ousterhout, "The Design of XPRS," Proc. 14th VLDB Conf., 1988.

[22] S. Y. W. Su, 'Database Computers, McGraw-Hill, 1988

[23] Tandem Database Group, "NonStop SQL : A Distributed, High-Performance, High-Availability Implementation of SQL," Proc. 2nd Int'l Workshop on High Performance Transaction Systems, in Lecture Notes in Computer Science 359, Springer-Verlag, 1987.

[24] P. Valduriez, "Parallel Database Systems : Open Problems and New Issues," Distributed and Parallel Databases, 1(2), 1993.

[25] P. Valduriez, "Parallel Database Systems : The Case for Shared-Something," Proc. 9th Int'l Conf. on Data Engineering, 1993.

[26] A. Witkowski, F. Carino, and P. Kostamma, "NCR 3700 - The Next Generation Industrial Database Computer," Proc. 19th VLDB Conf., 1993.

이 용 규

1986년 동국대학교 전자계산학과 (학사)

1988년 한국과학기술원 전산학과 (석사)

1996년 Syracuse University (박사)

1978년~83년 정보통신부 국가공무원

1988년~93년 국방정보체계연구소 선임연구원

1994년~96년 미국 뉴욕 주립 CASE센터 연구조교

1996년~97년 한국통신 선임연구원

1997년~현재 동국대학교 컴퓨터정보통신공학부 전임강사

관심분야 : 멀티미디어 정보검색, 데이터베이스 시스템, 하이퍼미디어 시스템, 전자도서관, 소프트웨어 공학