Chapter 1

Introduction

Mahdi Abdelguerfi and Kam-Fai Wong

1.1 Background

There has been a continuing increase in the amount of data handled by database management systems (DBMSs) in recent years. Indeed, it is no longer unusual for a DBMS to manage databases ranging in size from hundreds of gigabytes to terabytes. This massive increase in database sizes is coupled with a growing need for DBMSs to exhibit more sophisticated functionality such as the support of object-oriented, deductive, and multimedia-based applications. In many cases, these new requirements have rendered existing DBMSs unable to provide the necessary system performance, especially given that many mainframe DBMSs already have difficulty meeting the I/O and CPU performance requirements of traditional information systems that service large numbers of concurrent users and/or handle massive amounts of data [13].

To achieve the required performance levels, database systems have been increasingly required to make use of parallelism. As noted in [1], the traditional approach to parallelism for conventional DBMSs which use industry-standard database models such as the relational, can take one of two forms. The first is through the use of massively parallel general-purpose hardware platforms. As an example of this, commercial platforms such as nCube and SP1 are now supporting Oracle's parallel server [5]. Also, the Distributed Array Processor marketed by Cambridge Parallel Processing is now being used to produce a commercial massively parallel database system [4]. The second approach makes use of arrays of off-the-shelf components to form custom massively parallel systems. For the most part, these hardware systems are based on MIMD parallel architectures. The NCR 3700 [1] and the Super Database Computer II (SDC-II) [27] are two such systems. The NCR 3700 uses a high-performance multistage interconnection network known as Bynet and RAIDS (Redundant Arrays of Inexpensive Disks [11]). This system can now run a parallel version of Sybase relational DBMS [5, 26]. The SDC-II consists of eight data processing modules, where each module is composed of seven processors and five disk drives. The data processing modules communicate through an omega interconnection network.

The use of clusters of workstations as virtual parallel systems is a more recent approach that is already impacting the DBMS industry. These networks of workstations provide enormous amounts of aggregate computational power, often rivaling that of tightly coupled multiprocessor systems. They provide a viable high-performance computing environment and have several benefits over large, dedicated parallel machines, including cost, elimination of central point of failure, and scalability. Their use as a virtual parallel machine does not preclude the use of individual machines in more traditional ways. As an example of this, parallel versions of relational DBMSs, such as Oracle, are now even available on clusters of PC-compatible systems [2], thereby providing high performance at a relatively low cost.

The number of general purpose or dedicated parallel database computers is increasing each year. It is not unrealistic to envisage that all high-performance database management systems in the year 2010 will support parallel processing. The high potential of parallel databases in the future urges both the database vendors and practitioners to understand the concept of parallel database systems in depth.

1.2 Parallel Database Systems

The parallelism in databases is inherited from their underlying data model. In particular, the relational data model (RDM) provides many opportunities for parallelization. For this reason, existing research projects, academic and industrial alike, on parallel databases are nearly exclusively centered on relational systems. In addition to the parallel potential of the relational data model, the worldwide utilization of relational database management systems has further justified the investment in parallel relational databases research. It is, therefore, the objective of this book to review the latest techniques in parallel relational databases.

The topic of parallel databases is large and no single manuscript could be expected to cover this field in a comprehensive manner. In particular, this manuscript does not address parallel object-oriented database systems. However, it is noteworthy that several projects described in this manuscript make use of hybrid relational DBMSs. The emergence of hybrid relational DBMSs such as multimedia object/relational [10] database systems has been made necessary by new database applications as well as the need for commercial corporations to preserve their initial investment in the RDM. These hybrid systems require an extension to the underlying structure of the RDM to support unstructured data types such as text, audio, video, and object-oriented data. Towards this end, many commercial relational DBMSs are now offering support for large data types (also known as binary large objects, or BLOBs) that may require several gigabytes of storage space per row.

1.2.1 Computation Model

In relational databases, the mutual independence between two tables as well as between two tuples within a table, makes simultaneous processing of multiple tables and/or tuples possible. A database request often involves the processing of multiple tables. The ways in which a request accesses these tables and combines the intermediate results are defined by *the computation model*. Based on this model, one can understand the potential parallelism embedded in a database request.

Computation of a database request is modeled by an extended dataflow graph (EDG) [29]. The idea is to extend a conventional dataflow graph (that is, data operation nodes interconnected by data communication arcs) with partition parallelism; as such, one data operation node may be made up of multiple subnodes. As a result, complex and time consuming database operations, such as join, can be executed concurrently in a divide-and-conquer manner. For example, a join operation between (A join B) can be executed by performing N subjoins (Ai join Bi, where i = 1 to N) in parallel and later combining the results to produce the final answer of the join operation. The EDG model is the basis of FAD [12] and LERA [8], the query languages of the BUBBA and EDS parallel database servers. To better understand the concept of EDG, let us consider the SQL request below:

```
SELECT  *
FROM    employee, department
WHERE   (employee.dept_no = department.dept_no) AND
        (employee.position = "manager")
```

Assuming that the employee relation is partitioned in three fragments (that is, E1, E2, and E3) and the department relation in two (that is, D1 and D2), this request can be represented in the EDG shown in Figure 1.1(b). The control of the execution of the EDG is completely data driven. Referring to the example, when a START signal is received by the SW (single wait), it will initiate the query execution process. It will send a trigger signal to the two SCAN operators which then commence the scanning of the employee table seeking for the tuples whose position matches with the string "manager." Those tuples which satisfy the condition are distributed, by hashing on the dept_no attribute, to the JOIN operators. Conceptually, as soon as one of the employee tuples appears at any one of the JOIN operators, the join operation with the department tuples can proceed without delay. The results of the join operation are assumed to be stored on the same nodes which store the department table. Finally, once the last tuple on each JOIN operator is processed, a signal from each of them will be sent to the GW (global wait) operator. Upon receiving these three end signals, GW will terminate the execution process. It is shown from the above that the EDG is a self-scheduling structure such that once the START signal is sent, the execution of the EDG will run to completion by itself without any external control such as a program counter.

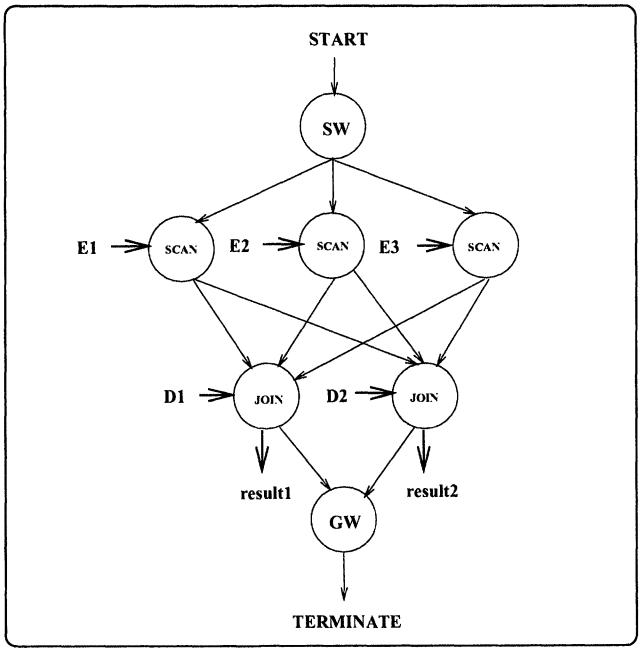
Furthermore, if the underlying execution platform is comprised of multiple processing units, a set of similar database requests can be executed in parallel as follows:

Interquery parallelism. More than one database request can be executed at one time. This will lead to increased throughput—that is, the number of requests processed per second, an important metric in on-line transaction application.

Interoperation parallelism. Within one request more than one operation can be performed simultaneously. In this example both the SCAN and JOIN operations can be executed in parallel. Notice that this form of parallelism is also referred to as pipeline parallelism since the result tuples from the SCAN operator can be processed by the JOIN operator as soon as they are available.

```
SELECT    *
FROM      employee, department
WHERE     (employee.dept_no = department.dept_no) AND (employee.position = "manager")
```

(a) SQL Request



(b) An equivalent Extended Dataflow Graph.

Figure 1.1: An Extended Dataflow Graph example.

Intraoperation parallelism. This is the parallelism offered by data partitioning. An operation can be split into smaller suboperations if the data that it processes are partitioned. This form of parallelism is the key to response time reduction.

An EDG only shows the potential parallelism of a database request. The actual parallelism depends on the implementation. Realization of the parallelism is a complicated engineering task whose primary objective is to achieve high performance in a cost-effective manner.

1.2.2 Engineering Model

To achieve a high-performance parallel database system, one must provide an efficient environment for EDG execution. This will require a thorough understanding of the parallel database systems engineering model [29]. The engineering model defines how an EDG is

physically executed on a parallel database environment. The main target of this book focuses on the engineering model and presents a number of state-of-the-art techniques which can either directly or indirectly lead to high-performance parallel database implementation.

In some practical situations, due to various physical constraints, certain forms of parallelism in an EDG are simply "restricted." In the above example, if there are only four processing units in the underlying parallel platform, two of the five data fragments, E1, E2, E3, D1, and D2 (see Figure 1.1(b)), must then be placed together on the same processing unit. This, inevitably, limits parallelism; but for cost-effectiveness reasons, the parallel database designer may, however, choose to do this. In other situations, parallelism may be "reduced." In the above example, if the communication cost (for example, in setting up a message packet) is high, the SCAN operator may bundle several tuples together before sending them out to the downstream JOIN operator. Effectively, this reduces the pipeline parallelism. In yet other situations, parallelism may be "eliminated" deliberately by the system. In the above example, if the sizes of the employee and the department relations are small, the communication cost between the SCAN and JOIN operators may undermine the processing time of the two operations. In that case, one may group the operators together to avoid the communication overhead. The above are just a few examples depicting how the execution environment can hinder the exploitation of the potential parallelism embedded in an EDG.

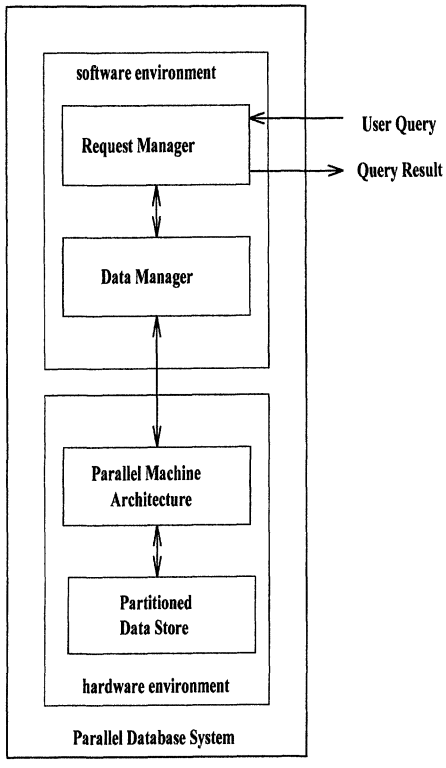


Figure 1.2: Overall internal architecture of a parallel database system.

Software Execution Environment. The top-level architecture of a typical parallel database system (such as the EDS parallel database server [8]) consisting of the software and hardware execution environments is shown in Figure 1.2. The software part is comprised of the request manager and the data manager.

Request Manager. The role of the request manager is (a) to transform a user's request into a set of semantic equivalent EDGs; (b) to select the "best" EDG from the set; and (c) finally, to translate the EDG into an executable object file. For a single request, there is usually more than one way of execution. Query optimization is the process of determining the best execution path. This is a complicated process even for sequential databases. Conventional parallel database systems (such as XRPS) adopt a two-phase optimization approach. In the first phase, the "best" sequential execution plan is determined and in the second, this plan is parallelized. Recent query optimization techniques are presented in [23].

Data Manager. This can be regarded as the parallel database operating system whose major responsibility is to execute the object file produced by the request manager. As pointed out by Stonebraker [24], conventional operating systems (OS) are inefficient for database applications. Some of the traditional features of OS that, generally, do not meet the needs of database applications are buffer management, load placement, and transaction processing. For instance, KEV is a special OS designed for the BUBBA parallel database machine [28]. In [2], it was observed that well-known load balancing methods at the OS level were generally unsuitable and that database-specific schemes were needed. Parallel transactions processing exploits interquery parallelism extensively. Due to its importance, the kernel of a number of existing commercial parallel database systems such as NonStop SQL [5, 9], Oracle Parallel Server [5, 3], and DB2 [5, 20] already support interquery parallelism. Furthermore, parallel transaction processing has opened up many new technical issues, such as recovery management. Data partitioning in a parallel database system renders data more vulnerable to failure. For the same reason, recovery in parallel database platforms is more difficult than in conventional database systems. Researchers are actively investigating techniques to provide reliable yet fast recovery environments (see, for example, [30, 22]).

In addition to the classical database requirements, special features must be provided by the OS for high-performance parallel database implementations. These include interprocessor communications, distributed/shared memory management, multithreading, and group communications.

Hardware Execution Environment. To realize parallelism, suitable hardware must be employed. This includes both a machine with a parallel architecture and a data store supported with the data partition model.

Parallel Machine Architecture. For database applications, machine architectures are classified according to the way in which resources are shared. The classification scheme, first introduced by Stonebraker, includes: shared nothing, shared memory, and shared disk. In a parallel database system based on the shared nothing concept, each system's node has its own main memory and secondary storage devices. Communica-

tion between the different nodes is achieved through message-passing across an interconnection network. This lack of resource sharing reduces nodes contention and permits a high degree of scalability. A parallel version of IBM's DB2 [7] running on the RS/6000-based POWERparallel multiprocessor system [14] is one such machine. This version of DB2 has the added advantage of providing advanced features such as support for binary large objects (for compressed video, images, and audio), and character large objects (for text documents) [5]. Despite their popularity [25], shared nothing parallel database systems suffer, in general, from a load balancing problem. In shared disk systems, each node has its private main memory but secondary storage, usually a disk array, is a shared resource. Because the disk data is shared by all nodes, these systems tend to have limited load imbalance but require a global lock manager to preserve the data consistency. Shared memory systems make use of a global main memory shared by all of the system's nodes. The use of a global memory eliminates the load balancing problem. The XPRS running Postgres [17, 15] is an example of such systems. However, these systems do not scale up well, as an increase in the number of nodes leads to more contention over the shared memory. Commercial database systems such as the NCR 3700 tend to have a hybrid architecture that combines some of the above features.

It is noteworthy that in recent years, due to the advancement in operating system and high-speed networking technologies, the differences between distributed and parallel computing are gradually disappearing. Many parallel processing techniques can be applied to distributed computer systems. For example, a shared nothing parallel database system could be developed on a cluster of high performance workstations interconnected by an asynchronous transfer mode (ATM) network by using a parallel OS kernel which supports physical virtual machine (PVM), multithreading, remote procedural call, as well as efficient message passing, and so forth.

Partitioned Data Store. Multiple disks are often used to minimize the data I/O overhead in a parallel database system. In a shared nothing architecture, each processing node is mounted with a local disk and in the shared disk environment the multiple disks are shared among the processing nodes. In the former case, the process of partitioning a database across the processing nodes is a performance-critical design issue. Data partitioning and allocation determine how much work needs to be performed by each node (for example, each subjoin operator), as well as how much communication overhead is incurred during the transfer of intermediate results from one node to another. Common partitioning techniques include range, round-robin, and hash partitioning [13]. These partitioning techniques generally perform the declustering based on a single attribute and, as a result, cannot support multiattribute declustering [6]. Additionally, they do not provide an effective solution for the imbalance caused by insertions and deletions [6]. The grid file declustering (GFD) technique [18] elevates these shortcomings. Other data partitioning techniques are presented and compared in [31]. Similarly, in a shared disk environment, improper partitioning will lead to high data contention rates. This causes reduced concurrency and as a result the system performance can be drastically affected.

1.3 About this Manuscript

Database technology is now rapidly expanding into new application areas such as multimedia, digital libraries, and data warehousing. These new applications areas bring with them many challenging new requirements—such as increased functionality and the efficient handling of very large heterogeneous databases—that need to be addressed by next-generation database management systems (DBMSs).

This manuscript describes how the use of parallel processing technology has made it possible to meet these new challenges. It includes nine invited papers that emphasize techniques for achieving high performance in parallel database systems. This will be beneficial to designers in the areas of database and parallel processing who intend to design their own parallel database products as well as to database practitioners who are contemplating migrating their sequential database systems to parallel platforms.

Because of its increased importance, the body of research in the area of parallel databases has become a large one. As a result, several manuscripts, such as [1, 16, 19], dealing with this topic have appeared recent years. In particular, [1] focuses on the architectures of various parallel database systems such as DBC/1012, NCR 3700, IDIOMS, MDBS, IFS2, and the Datacycle project. Consequently, [1] is a good complement to this book.

The body of this manuscript is structured according to the overall architecture of a parallel database system (see Figure 1.2). The nine chapters in this book present a number of state-of-the-art techniques which can be adopted in the design of parallel database software and hardware execution environments. Chapters 2, 3, and 4 are related to the request manager; Chapters 5 and 6 to the data manager; Chapters 7, 8, and 9 to the parallel machine architecture; and finally, Chapter 10 addresses concerns about different techniques for the partitioned data store.

Chapter 2 (request manager related). A survey of parallel query optimization techniques for requests involving multiway joins is presented in this chapter. In particular, the two-phase heuristics algorithm designed for the XPRS shared memory parallel database system is emphasized. In the first phase, the optimizer identifies the "best" sequential execution plan; and in the second phase, the optimized parallelized version of the plan is derived.

Chapter 3 (request manager related). In database query optimization, in addition to selecting the "best" execution plan, the optimizer is also required to select the best algorithm for the operations involved in the plan. The former step is often referred to as global optimization and the latter, local optimization. This chapter introduces a new technique for the time-consuming join operation. This can be adopted in the local optimization stage. Page connectivity information [21] is used extensively in this new technique.

Chapter 4 (request manager related.) In the process of designing new parallel query optimization techniques (such as designing a new cost model), it is important that the design choices (for example, communication cost) are valid. This chapter describes the Test Pilot system—a performance evaluation tool for parallel databases. Such a tool can be used to serve the above-mentioned purpose. Furthermore, it can be used

for performance tuning—a critical step to be taken by the database engineers after a database system has been commissioned.

Chapter 5 (data manager related). This chapter gives insight into the behavior of buffering and locking mechanisms for parallel database applications. It includes a case study where traditional load balancing techniques are shown to be unsuitable for such applications. This study stresses the need for database-specific load balancing techniques. The results of this study will be beneficial for the development of intelligent load balancing schemes.

Chapter 6 (data manager related). This chapter introduces a framework for recovery in parallel database systems using the ACTA formalism [22]. This approach enables concise description of the recovery mechanism in a parallel database environment. This is essential for database designers in working out new parallel recovery techniques.

Chapter 7 (parallel machine architecture related). This chapter describes the architecture of the NCR's new Petabyte multimedia parallel system [10]. This is a commercial product. The architectural features of the machine to support multimedia applications as well as the object/relational data model are outlined in this chapter.

Chapter 8 (parallel machine architecture related). The Medusa parallel database system is described in this chapter. It is a transputer-based shared nothing system. It was designed with a new technique for self-managing its own data partitions, with minimal impact to transaction processing [6].

Chapter 9 (parallel machine architecture related). This chapter describes the Super Database Computer, SDC-II. It was designed with a special hardware support for the bucket spreading hash join algorithm. This novel hardware-supported join technique is the key to the high performance rate for complex queries offered by SDC-II.

Chapter 10 (partitioned data store related). This chapter presents a case study for a shared nothing parallel database server. It analyzes and compares the effectiveness of five data placement techniques under different environmental settings, such as number of nodes, the database size, and the relative frequency of queries. The analytical results are helpful to database designers for choosing suitable data placement techniques for his/her environment.

Bibliography

[1] M. Abdelguerfi and S. Lavington, *Emerging Trends in Database and Knowledge-Base Machines: the Application of Parallel Architectures to Smart Information Systems*, IEEE CS Press, Los Alamitos, CA, 1995.

[2] L. Borrmann, B. Shiemann, and E. Born, "Load Placement in Distributed High-Performance Database Systems," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 5, IEEE CS Press, Los Alamitos, CA, 1998, pp. 93–115.

[3] S. Bobrowski, "Bobrowski, S.," *DBMS*, Dec. 1993.

[4] N. Bond and S. Reddaway, "A Massively Parallel Indexing Engine Using dap," *Emerging Trends in Database and Knowledge-Base Machines: the Application of Parallel Architectures to Smart Information Systems*, M. Abdelguerfi and S. Lavington, eds., chapter 9, IEEE CS Press, 1995, pp. 159–179.

[5] C.J. Bontempo and C.M. Saracco, *Database Management Principles and Products*, Prentice Hall, 1995.

[6] G.M. Bryan and W.E. Moore, "The Medusa Project," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 8, IEEE CS Press, Los Alamitos, CA, 1998, pp. 165–182.

[7] C. Baru et al., "An Overview of DB2 Parallel Edition," *Proc. ACM SIGMOD Conf.*, pp. 460–462, 1985.

[8] P. Borla-Salamet, C. Chachty, and B.B. Bergsten, "Capturing Parallel Data Processing Strategies within a Compiled Language," *Parallel Processing and Data Management*, P. Valduriez, ed., Chapman-Hall, 1992, pp. 295–316.

[9] C. Mala et al., *Enhancing Availability, Management, and Performance for Nonstop tm/mp*, tech. report, Tandem Systems Review, Tandem, July 1994.

[10] F. Carino and W. Sterling, "Parallel Strategies and New Concepts for a Petabyte Multimedia Database Computer," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 7, IEEE CS Press, Los Alamitos, CA, 1998, pp. 139–164.

[11] D. Patterson et al., "The Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proc. ACM SIGMOD Conf.*, Vol. 6, 1988, pp. 109–116.

[12] S. Danforth and P. Valduriez, "A Fad for Data Intensive Applications," *IEEE Trans. Knowledge and Data Eng.*, Vol. 4, No. 1, 1992, pp. 34–51.

[13] D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems," *Comm. ACM*, Vol. 35, No. 6, 1992, pp. 85–91.

[14] G. Fecteau, "Database 2 AIX/6000 Parallel Technology," tech. report, IBM Software Solution Laboratory, Ontario, Canada, Mar. 1994.

[15] G. Graefe et al., "Tuning a Parallel Database Algorithm on a Shared-Memory Multiprocessor," *Software—Practice and Experience*, Vol. 22, No. 7, 1992, pp. 495–571.

[16] H. Lui et al., *Query Processing in Parallel Relational Database Systems*, IEEE CS Press, Los Alamitos, CA, 1994.

[17] W. Hong, "Exploiting Inter-Operation Parallelism in XPRS," *Proc. ACM SIGMOD Conf.*, 1992, pp. 19–28.

[18] K.A. Hua and C. Lee, "An Adaptive Data Placement Schemes for Parallel Database Computer Systems," *Proc. 16th VLDB Conf.*, 1990, pp. 493–506.

[19] A.R. Hurson, *IEEE Tutorial: Parallel Architectures for Database Systems*, IEEE CS Press, Los Alamitos, CA, 1989.

[20] *IBM DATABASE 2 Version 3, Administrative Guide Volumes I, II and III*, IBM Corp., SC26-4888, Dec. 1993.

[21] C. Lee, "New Approaches to Join Optimization in Parallel Database Systems Utilizing Page Connectivity Information," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 3, IEEE CS Press, Los Alamitos, CA, 1998, pp. 43–75.

[22] L.D. Molesky and K. Ramamritham, "Modeling Recovery in Client-Server Database Systems," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 6, IEEE CS Press, Los Alamitos, CA, 1998, pp. 119–138.

[23] M. Zait, M. Ziane, and P. Valduriez, "Optimization for Parallel Execution," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 2, IEEE CS Press, Los Alamitos, CA, 1998, pp. 15–41.

[24] M. Stonebraker, "Operating System Support for Database Management," *Comm. ACM*, Vol. 24, No. 7, 1981, pp. 412–418.

[25] M. Stonebraker, "The Case for Shared Nothing," *Database Engineering*, Vol. 9, No. 1, 1986, pp. 4–9.

[26] *Sybase Navigation Server: Delivering the Promise of Parallel High-Performance*, Technical Paper Series, Sybase, 1994.

[27] T. Tamura and M. Kitsuregawa, "System Software of the Super Database Computer SDC-II," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 9, IEEE CS Press, Los Alamitos, CA, 1998, pp. 185–201.

[28] K. Wilkinson and H. Boral, "Kev—A Kernel for Bubba," *Proc. 5th Int'l Workshop on Database Machines*, Japan, Oct. 1987, pp. 109–116.

[29] K.F. Wong, "Parallel Database Systems Engineering," *Tutorial Notes in 1995 EuroPar Conf.*, Stockholm, Sweden, Aug. 29–31, 1995.

[30] K.F. Wong, "Performance Evaluation of Three Logging Schemes for a Shared-Nothing Database Server," *Proc. 1995 Int'l Conf. on Distributed Computing Systems (ICDCS'95)*, Vancouver, Canada, May 30–June 2, 1995, pp. 221–228.

[31] S. Zhou and M.H. Williams, "Data Placement in Parallel Database Systems," *Parallel Database Techniques*, M. Abdelguerfi and K.F. Wong, eds., chapter 10, IEEE CS Press, Los Alamitos, CA, 1998, pp. 203–219.