OVERVIEW OF THE JASMIN DATABASE MACHINE

Daniel H. Fishman Ming-Yee Lai W. Kevin Wilkinson

Bell Communications Research

ABSTRACT

The Jasmin database machine is being implemented as part of a research project in distributed processing and database management. A primary goal of the work is to demonstrate the feasibility of a practical multiprocessor database machine suitable for large database, high transaction-rate applications. Key features of Jasmin are its configurable performance, its use of off-the-shelf parts, and its ability to handle distributed databases. A uniprocessor prototype of Jasmin has already been completed and the multiprocessor version is planned for later this year. In this paper we describe Jasmin's architecture and discuss the performance observed in the uniprocessor prototype.

1. INTRODUCTION

The Jasmin database machine is being implemented as part of a research project in distributed processing and database management. The project is an outgrowth of earlier work in this area [BUR83]. A primary goal of the work is to demonstrate the feasibility of a practical multiprocessor database machine that is suitable for large database, high-volume applications. A uniprocessor prototype of Jasmin has already been completed and the multiprocessor version is planned for later this year.

The implementation of Jasmin uses both conventional processors and disks. While some database machines use content-addressable disks [OZK75, SU75, BAN78, LIN76], we decided against their use because of their need for special hardware. This requirement limits one's ability to use the latest disks to emerge from this rapidly changing technology. It is also presently unclear how to make effective use of such disks in general purpose applications [HAW81]. For similar reasons, we also decided against the use of special-purpose processors. While a variety of such processors are used in some database machines [KUN80, SON80, BEN79, MAL79, DEW81, IDM81], we will not use them in our prototype because they are not generally available. However, our architecture is flexible enough to incorporate special hardware devices when they are available, provided they meet whatever standard interface we adopt, e.g., multibus, Q-bus, etc. In some ways, Jasmin resembles the IDM-500 [IDM81]. Both use conventional processors and disks in a bus-based architecture. (The IDM-500 also uses a special-purpose hardware device called a "database accelerator.") Whereas the IDM-500 is essentially a uniprocessor machine, Jasmin is composed of software modules specifically designed to run in various combinations on multiple processors, much in the spirit of SABRE [GAR83]. In fact, Jasmin can accommodate multiple copies of any component in configurations that satisfy a wide range of throughput requirements.

Our design goals for Jasmin are that it be: implemented in a machine independent manner using off-the-shelf parts; configurable for a range of application sizes, small to very large; able to process distributed databases efficiently; and suitable as a testbed for research in distributed concurrency control and query processing, reliability and recovery in multiprocessor database machines, and performance evaluation.

To satisfy these goals, we have devised a software architecture (Figure 1) with three layers of database services. Each successive layer represents a higher level of abstraction. The layers are implemented as processing modules (servers) that communicate with each other with messages. Because intermodule communication is restricted to messages, the module-to-processor assignments can be changed with no effect on the software. Further, the modules are written in a fashion that allows each to coexist with clones of itself. Thus, Jasmin can be configured with many modules and processors, as processing requirements dictate. Our initial prototype will consist of eight processors on which we will experiment with a variety of module assignments.

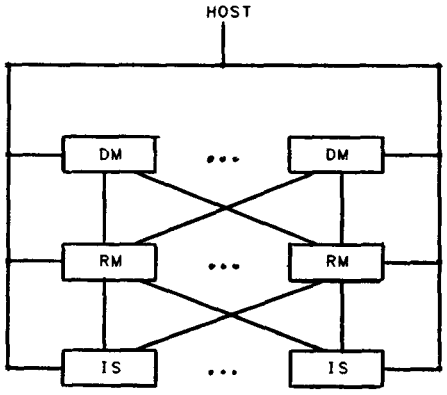

Figure 1. Software Architecture of Jasmin

The database software consists of three modules. The Intelligent Store (IS) [ROO82] performs page and physical block management, transaction management, concurrency control, and crash recovery. The Record Manager (RM) performs record and index management, and single relation query processing. The Data Manager (DM) performs relational query processing and schema management using

the RM for single relation subqueries. Since each module is implemented as a server, it offers a flexible and well-defined interface. It is possible to implement new database servers using existing modules, e.g., we plan to implement a robust file server that uses the IS. User applications on host machines can also use these servers. Applications are not constrained to use higher level services when they are not needed. Furthermore, since concurrency control is managed by the IS, the RM and DM can be used simultaneously on the same database with no danger of corruption.

The software is implemented on a sparse operating system kernel that provides message-based intermodule communication, multitasking, and scheduling. The kernel is an excellent runtime environment, providing for efficient execution with no extraneous function.

The hardware and software architecture of Jasmin are described in Section 2. Section 3 follows with a discussion of preliminary uniprocessor performance results and our performance goals for the multiprocessor. Section 4 concludes the paper with a summary of the project and our plans for the future.

2. JASMIN ARCHITECTURE

In this section, we outline the software and hardware architecture of Jasmin. We first discuss the software components. Then we discuss the hardware architecture. Our design is not tied to any particular hardware, however we list some hardware features that will improve performance. We also describe the hardware we intend to use for our multiprocessor prototype. Finally, we describe alternative configurations of software modules on our proposed hardware and discuss how those configurations affect performance.

2.1 Software Components

Jasmin is designed as a collection of cooperating software modules, each with a well-defined interface. To support our design goals of incremental growth and transparent distribution, we imposed the constraint on the implementation of modules that they be able to coexist with clones of themselves in arbitrary configurations. The Jasmin kernel provides a simple but powerful environment on which to implement such software.

2.1.1 The Kernel. The kernel provides a minimal set of facilities for building distributed software. It offers three types of services: tasking, scheduling, and message passing. It includes no database-specific features and even device drivers are not included. Thus, the kernel is a sparse environment with little of the overhead found in general purpose operating systems [STO81]. It gives the database system implementor complete control over how memory is scheduled, swapped, and paged. The kernel is accessed via subroutine calls.

The kernel supports multitasking within **modules**. A module consists of one or more associated tasks, on the same processor, that share an address space. In addition, each task of a module has its own stack and may have a private data segment. Tasks represent parallel computations that are created and destroyed dynamically by explicit user control. A module facilitates parallel activity within a shared data area (for example, a buffer cache in a file manager).

Many modules may run simultaneously on the same cpu. A single message mechanism is used for communication between and synchronization of tasks (both intra- and inter-module). However, tasks within the same module may also communicate via their shared data area. Because the kernel provides non-preemptive scheduling, synchronization of tasks within a module is simple and inexpensive. That is, since tasks running at the same priority level cannot preempt each other, they can't interfere with each other within critical sections.

Messages are small (16 byte) fixed-length objects. They are sent along one-way communication channels, called **paths**. Our paths are similar to the links used in Roscoe [SOL79] and DEMOS [BAS77]. Messages are copied and queued in kernel space on the machine of their intended receiver. They remain queued until the receiver asks for them. Large amounts of data are passed by associating a buffer with a path. The kernel moves data in path buffers from one address space to another, whether on the same machine or on separate machines.

Additional supporting software includes device drivers, and Name and File Manager modules. The Name Manager provides a locator service by dispensing paths to other modules. The File Manager implements a file system similar to the one in the UNIX* operating system. These modules, and the database modules described below, are implemented directly on the kernel.

2.1.2 Intelligent Store. The Intelligent Store (IS) is a sophisticated page manager that maps pages into secondary storage. The IS underlies the Jasmin database management facilities, providing data consistency, concurrency control, and crash recovery. The IS is transaction oriented. It permits multiple transactions to access and update pages stored on one or more disk subsystems. It implements an optimistic concurrency control method, similar to [KUN81], maintaining a consistent version of the database for each active transaction. There are no hard locks on pages. If two update transactions access the same page, and this page is declared to be "important" (see [ROO82]) to both transactions, then the first transaction to **commit** its updates succeeds, and the other transaction is rolled back. Page requests are based on (logical) page identifiers. The IS translates between logical page id's and physical block addresses. Since the concurrency control scheme involves page shadowing, e.g., updates are not done in place, the IS user need not be concerned with the precise physical location of pages. The IS also implements rollback (transaction abort) and recovery from system crashes (committed updates are preserved, and uncommitted updates are discarded). In addition, the IS provides a priority caching scheme to enable some classes of pages to remain in cache longer than others. This is used to give priority to index pages over data pages. Though not yet implemented, algorithms have been devised to accommodate transactions that span multiple IS's. data managed by several IS's. A detailed discussion of the IS appears in [ROO82].

2.1.3 Record Manager. The Record Manager (RM) [LIN82] provides access to data stored in the form of records. The RM maps records into IS pages. Variable size records with missing and repeated fields are supported. Records are grouped into sets (record types, relations), each of which must have an associated primary index. Record types may also have an arbitrary number of secondary indexes. Although all indexes are currently implemented as B-trees, provision has been made to accommodate other access methods as well. Retrieval requests are associative: they specify only the record type and a boolean search expression that selects the desired records. Search expressions may include exact, prefix, and range matches. The physical aspects of storing and retrieving index and data pages are handled by the IS. The RM accommodates multiple concurrent users (transactions), both updates and retrievals, and relies on the concurrency control provided by the IS. Note that the RM operations are limited to those that can be computed in one pass over one record type, thus including select and project, but excluding sort and join. The RM interface is accessible to applications that only need to process single relations. More complicated query processing will require use of the Data Manager.

2.1.4 Data Manager. The Data Manager (DM) provides a relational view of data, mapping relations into RM records. It offers a QUEL [STO76] interface and read/write protection of data down to fields within records. Planned features include parameterized stored commands, views, automatic transaction restart, and a heavy dose of query optimization. The DM uses the RM to process one or more single relation queries. Query processing is accomplished in a pipeline. One or more streams of output from the RM are sorted, joined, aggregated, and projected by an optimized network of tasks set up to handle the specific query. The pipe between two tasks is implemented in shared memory and is very efficient. Use of the pipeline eliminates the need to create temporary relations during query processing. We expect the set orientation to be helpful in distributed query processing. We also believe the pipeline orientation will accommodate special

* UNIX is a Trademark of Bell Laboratories

purpose processors for selected operations. Like the RM, the DM is supported by the concurrency control and recovery mechanisms provided by the IS.

2.2 Hardware Components

Jasmin will run on any hardware that supports the kernel. In the past, this has included a DEC PDP 11/45 and an AT&T 3B20S (roughly equivalent to a DEC VAX 11/780). However, these implementations made incremental growth rather expensive. Further, lacking a readily available inter-connect scheme for these machines, we never experimented with distributed processing. Henceforth, a guiding principle for choosing a hardware base for Jasmin will be to select inexpensive, readily available, off-the-shelf components.

The Jasmin kernel is designed to make effective use of a multilevel system architecture, as depicted in Figure 2. At the lowest level of the architecture, processors are grouped together on a high speed memory-addressed bus in a configuration called a **node**. At the next higher level, nodes of machines are interconnected by a message communication fabric to form a **local area network (LAN)**. At the top level, LANs are connected together by gateways to form a **wide area network (WAN)**. The WAN level will not be further discussed here. Given a fast enough message medium, two levels of hardware interconnect (Figure 2) form a suitable hardware base for a Jasmin database machine.

2.2.1 Processors. The first "requirement" on Jasmin processors is that they support a large address space, at least 24 bits. Our experience with 16-bit processors convinced us that it is a bad idea to build large software systems in small address spaces. A second requirement is that the processor support a shared memory bus that makes each processor's memory accessible to other processors on the bus. This allows several processors to be grouped together into a node.

2.2.2 Message Network. The message network is the transport medium for messages between nodes. Many existing LANs are suitable, but they must preserve the order of messages and they must be fast. Further, it should be possible to add or remove nodes without disturbing active nodes.

2.2.3 S/NET. We plan to build the Jasmin prototype on S/NET [AHU82], a high-speed message bus running at 80 Mbps. The S/NET, as originally configured, consists of 8 nodes connected to the network. Each node is contained in a 5 slot multibus cage; one slot is used for the processor (a Motorola MC68000), one for expansion memory and one for the S/NET interface. We plan to use one additional slot for a high-speed message interface processor. It will implement some of the kernel message passing primitives in hardware. The remaining slot may be used for an additional processor, additional memory, disk and tape controller, or possibly for special-purpose database hardware. S/NET configurations with more nodes and more slots per node are entirely possible. The S/NET matches our hardware needs very closely. The multibus provides a shared-memory bus for several processors to form a node. The S/NET itself provides a high-speed message network. In addition, the processors are inexpensive and the multibus is a standard interface that should make it easy to experiment with new hardware.

2.3 Jasmin Configurations

Since Jasmin is so easily configured, we have the problem of deciding how best to assign software modules to processors. There are actually two problems: we must decide how many modules to run on each processor, and we need to decide which types of modules (i.e. IS, RM, DM) to run on a given processor. There are two extremes: run all modules on each processor or run each module on a separate processor. The first extreme, using a single processor, was the configuration of our initial prototype. While it simplified development and benchmarking, it is the least interesting configuration and it pays the overhead of distribution and message passing without taking advantage of it. The other extreme may obtain the best performance if message passing between processors is very fast.

We expect the IS to be primarily I/O bound and the RM and DM to be CPU bound. This suggests that RM and IS modules might reside

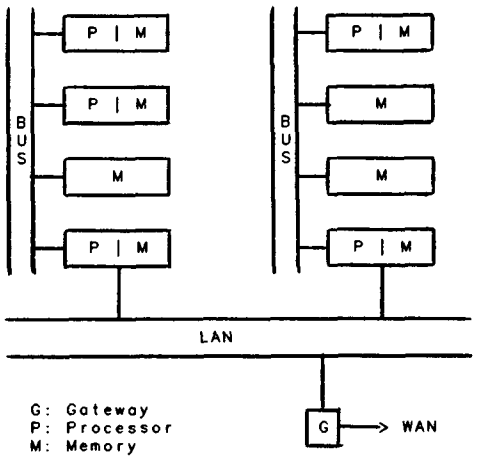

Figure 2. Two-Level System with Two Nodes

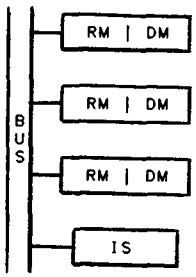

Figure 3. Configuration for Short Transactions

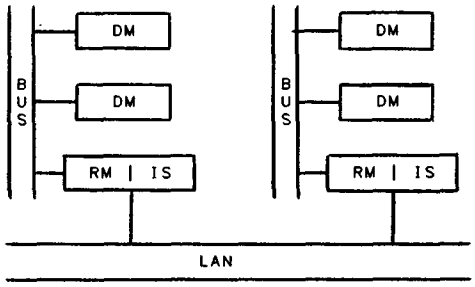

Figure 4. Configuration for Longer Transactions

on the same processor with little or no loss in performance. Another reason these modules may live well together is that the traffic between the IS and RM is likely to be heavier than between the RM and DM. However, the most cost-effective configuration for an application will depend on the processing requirements of that application. For example, consider an application where transactions are short and most references are to a common part of the database. In this case, we might want the database to reside on a single IS, as in Figure 3, to take advantage of caching in the IS, and to avoid distributed transactions. Further, it makes sense to run the DM and RM on the same processor, since the DM does relatively little work.

Another example might be an application where transactions make references to records that are uniformly distributed across the database. In this case, we might want to distribute the data and place an RM and IS on the same processor, as in Figure 4. Such a configuration is likely to result in concurrent processing of transactions.

Given an application and a fixed number of processors we plan to experiment with various module-processor configurations to determine the performance trade-offs.

3. PERFORMANCE ANALYSIS

In this Section, we describe the performance of our initial Jasmin prototype implemented on a single AT&T 3B20S computer. To determine the viability of the Jasmin architecture, we compared its performance with that of a Britton-Lee IDM-500. Performance measurement of the IDM-500, internal software version 20, was conducted by Lidor [LID83]. Britton-Lee introduced numerous performance enhancements in their version 21 software. At the conclusion of this Section we discuss the performance gains we expect from a multiprocessor Jasmin.

3.1 Observed Performance

Both Jasmin and the IDM-500 were configured with 2 megabytes of memory. Both were run as a backend database machine connected to a 3B20S host, running UNIX 5.0, by a 9600 baud serial link.

The experimental database consists of three relations, R1, R2 and R3, each containing 1000 records. Each relation has fifteen attributes, A1 through A15. A2 is the primary key and it has an integer domain with distinct values ranging from 1 to 1000. The domain of A4 also contains integers, in random order, ranging from 1 to 1000, but A4 is unindexed. There is a secondary index on A7, and A7 also has an integer domain with distinct integer values, in random order, between 1 and 1000. There is no correlation between the values in R1, R2, and R3. The size of each record is 80 bytes, and the total database size is 240K bytes.

The queries of interest are listed in Table 1. Queries Q1 and Q2 retrieve records in R1 using the primary key, A2. Q1 retrieves 10 records while Q2 retrieves 500 records. Q3 retrieves records in R1 via sequential search since A4 is not indexed. Q4 and Q5 retrieve records via the secondary index Q7. Q4 retrieves 10 records while Q5 retrieves 500 records. Q6 retrieves all records in R1 via sequential search, and does a projection on A2, A3, and A4. Q7 gets the count of all qualified records for Q2. Q8, Q9, and Q10 were repeatedly executed while Jasmin was monitored to obtain an execution profile of each run. The profiling data provides a statistical indication of where, at the subroutine level, the CPU cycles are being spent. Q8 is used to profile keyed retrieval. Q9 is used to profile a join. Q10 is used to profile an aggregate calculation.

In Table 2, we list the performance figures in terms of response times for the above queries in a single-user environment. The measurements were obtained by use of the UNIX "time" utility.

Although Jasmin is mainly designed as a multiprocessor database machine and uses messages as the interprocess communication mechanism, the figures show the response times of a single 3B20S-based Jasmin to be comparable to an IDM-500. From a hardware point of view, one could argue that the 3B20S is much more powerful than the CPU used in the IDM-500. However, the response time figures for Jasmin are taken from an early prototype which, we

Q1: retrieve (R1.A2) where R1.A2 <= 10

Q2: retrieve (R1.A2) where R1.A2 <= 500

Q3: retrieve (R1.A4) where R1.A4 <= 500

Q4: retrieve (R1.A7) where R1.A7 <= 10

Q5: retrieve (R1.A7) where R1.A7 <= 500

Q6: retrieve (R1.A2, R1.A3, R1.A4)

Q7: retrieve (*mcnt* = count(R1.A2 where R1.A2 <= 500))

Q8: 100 commands, key in random order
retrieve (R1.A1, ..., R1.A15) where R1.A2 = key

Q9: 5 commands, join by primary key
retrieve (R1.A2, R2.A2) where R1.A2 = R2.A2

Q10: 20 commands, average on primary key
retrieve (mavg = avg(R1.A2))

TABLE 1. Test Queries

Query	Jasmin	IDM-500
Q1	2.4	3.2
Q2	9.2	10.2
Q3	9.7	6.6
Q4	2.5	5.3
Q5	20.9	15.6
Q6	31.0	39.7
Q7	5.6	3.9
Q8	1.4*	N.A.
Q9	22.7*	N.A.
Q10	7.8*	N.A.

* (average)

TABLE 2. Observed Response Time (seconds)

assume, was not as well tuned as the IDM-500. For example, the buffer size used in Jasmin was less than 128K bytes, as compared to the 466K byte buffer used in the IDM-500. Furthermore, the Jasmin prototype suffered from poor host-backend communications. (The communication line driver was written with several tasks that communicate internally with messages!) Elimination of the use of messages in the communication driver would provide a significant performance improvement.

From Table 2, we observe that Jasmin performs better than the IDM-500 in primary key retrieval (much faster for Q1 and faster for Q2). For sequential search, Jasmin is faster than the IDM-500 for unqualified retrieval (Q6), but is considerably slower for qualified retrieval (Q3). We attribute this slower performance largely to the use of an inefficient expression evaluator in the RM. The evaluator is invoked for queries involving qualifications on non-primary-key fields. E.g., it is invoked for Q3-Q6, but not for Q1 and Q2. The fact that, in the execution of Q3, the evaluator must be applied to each tuple in R1 distorts that result. For queries on a secondary indexed attribute, Jasmin is twice as fast as the IDM-500 for retrieving a small amount of data (Q4), but is much slower retrieving a large amount of data (Q5). Jasmin always uses secondary indexes when they apply, it does not employ the optimization of sequentially scanning a relation on large retrievals. Jasmin is considerably slower than the IDM-500 on

Q7. We believe the slower speed may be attributable to extra data copying between the RM, where selection and projection are done, and the DM, where the aggregation is done.

The execution profiles for Q8, Q9, and Q10 were obtained by recording "hit counts," for subroutines of interest, using profiling utilities. A high hit count indicates frequent execution. CPU utilization was also determined. For Q8, Q9, and Q10, the CPU was busy 24%, 72%, and 64% of the time, respectively.

In Q8, 55% of the CPU activity was in the kernel for message send and receive. We attribute much of this to the overhead of host-backend communication.

In Q9, 90% of the CPU activity was distributed among the RM (33%), DM (30%), and kernel (27%). Record manipulation, especially projection, and moving bytes were the greatest consumers of time in the RM. Join processing and data formatting dominated the time in the DM. Message send and receive consumed about 30% of the kernel time.

In Q10, 92% of the CPU activity was distributed among the RM (46%), DM (22%), and kernel (24%). Once again, record manipulation, especially projection, and moving bytes dominated the time in the RM. Disk access and message handling dominated the kernel usage. Computing the sum dominated the DM usage. In all cases, CPU cycles attributable to the IS were in the noise level.

3.2 Projected Performance

When Jasmin is implemented on multiprocessor hardware we expect to obtain significant performance improvements. We will explore the performance of a simple configuration consisting of three single processor nodes connected by a LAN. The performance improvement of other configurations can be extrapolated from this example.

In this simple configuration, one node runs the DM, another runs the RM, and the third runs the IS. The database is stored on disks managed by the IS, and the host interacts with the processor running the DM. Note that if there were more than one DM, RM or IS in the system, we would expect system throughput to increase.

Because query processing is spread over three machines, the response time for queries that touch only a few records should increase somewhat. However, queries that require processing lots of data should show improved response times because of the pipeline processing, using double buffering between the DM, RM and IS. Because of this pipelining we will obtain intra-query concurrency. For example, in Q7, the RM can perform projection and selection while the DM computes the aggregate, and the IS does disk access. Pipelining also occurs in a multiplexed fashion, resulting in inter-query concurrency, when more than one query is in process. That is, when running several keyed retrieval queries, the DM can process the third query while the RM does projection for the first and the IS does disk access for the second. The message latency is at least an order of magnitude less than the execution time needed by the DM, RM, and IS to do their functions. Thus the gain from concurrent processing should be much greater than the overhead of inter-node communication.

4. SUMMARY AND FUTURE WORK

In this paper we described the Jasmin database machine architecture and discussed the performance observed in a uniprocessor prototype. Key features of Jasmin are its configurable performance, its use of off-the-shelf parts, and its ability to handle distributed databases. An important goal for Jasmin is that it be useful as a testbed for ongoing research in database machines and database management, in general. Our future work with Jasmin will include:

- Construction of a file system on the IS and its integration with the DM. This will permit additional data types such as text, graphics, and voice in the database.
- Study the reliability of Jasmin configurations and design of enhanced reliability features.
- Study the advantages and disadvantages of optimistic concurrency control for centralized and distributed databases, and comparison of its performance relative to locking. An important advantage of optimistic locking is its non-blocking behavior relative to read-only transactions. This makes it quite useful for interactive and low-conflict applications. It also appears to simplify coordination of distributed transactions.
- Study the performance implications of various configurations of DM, RM, and IS modules.

5. ACKNOWLEDGMENTS

Work through the initial prototype was conducted at AT&T Bell Laboratories. Quite a number of people have made significant contributions to this project. These include, in addition to the authors, Micah Beck, Bill Burnette, Rudd Canaday, Susan Fontaine, Mary Hesselgrave, Phil Karn, Hikyu Lee, John Linderman, U. V. Premkumar, Bill Roome, Edith Schonberg, and Chung Wang. We particularly appreciate the efforts of Mary Hesselgrave in obtaining the performance results described in this paper.

6. REFERENCES

[AHU82] Ahuja, S. R. "A High Speed Interconnect for Multiple Computers," Internal Memorandum, Bell Laboratories, Holmdel, NJ, Dec. 1982.

[BAN78] Banerjee, J., and Hsiao, D. K. "Performance Study of a Database Machine in Support of Relational Databases," *Proc. VLDB*, 1978.

[BAS77] Baskett, F., Howard, J. H., and Montague, J. T. "Task Communication in DEMOS," *Proceedings of the 6th ACM Symposium on Operating Systems Principles*, November 1977, 23-31.

[BEN79] Bentley, J. L., and Kung, H. T. "A Tree Machine for Searching Problems," *Proc. IEEE Int. Conf. on Parallel Processing*, 1979.

[BUR83] Burnette, W. A., Canaday, R. H., and Fishman, D. H. "MAX: A Distributed System," Internal Memorandum, Bell Laboratories, Murray Hill, NJ, 1983.

[DEW81] DeWitt, D. J. "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," *IEEE Trans. Computers C-28*, June 1979, 395-406.

[GAR83] Gardarin, G., Bernadat, P., Temmerman, N., Valduriez, P., and Viemont, Y. "Design of a Multiprocessor Relational Database System," *INRIA Technical Report TR-224*, July 1983.

[IDM81] Britton-Lee Inc., *IDM-500 Reference Manual*, Los Gatos, CA., 1981.

[HAW81] Hawthorn, P. B., and DeWitt, D. J. "Performance Analysis of Alternative Database Machine Architectures," *IEEE Trans. Software Engr. SE-8*, January 1982, 61-75.

[KUN80] Kung, H. T., and Lehman, P. L. "Systolic Arrays for Relational Data Base Operations," *SIGMOD Proc.*, May 1980, 105-116.

[KUN81] Kung, H. T., and Robinson, J. T. "On Optimistic Methods for Concurrency Control," *ACM TODS 6*, 2 (June 1981), 213-226.

[LID83] Lidor, G. Personal Communication.

[LIN76] Lin, S. C., Smith, D. C. P., and Smith, J. M. "The Design of a Rotating Associative Memory for Relational Database Applications," *ACM TODS 1*, 1 (March 1976), 53-75.

[LIN82] Linderman, J. P. "Issues in the Design of a Distributed Record Management System," *Bell System Technical Journal 61*, 9 (Nov. 1982), Part 2, 2555-2566.

[MAL79] Maller, V. A. J. "The Content Addressable File Store -- CAFS," *ICL Technical Journal,* November 1979, 265-279.

[OZK75] Ozkarahan, E. A., Shuster, S. A., and Smith, K. C. "RAP - Associative Processor for Database Management," *AFIP Proc., Vol 44,* 1975, 379-388.

[ROO82] Roome, W. D. "A Content-Addressable Intelligent Store," *Bell System Technical Journal 61,* 9 (Nov. 1982), Part 2, 2567-2596.

[SOL79] Solomon, M. H., and Finkel, R. A. "The Roscoe Distributed Operating System," *Proceedings of the 7th Symposium on Operating Systems Principles,* December 1979, 108-114.

[SON80] Song, S. W. "A Highly Concurrent Tree Machine for Data Base Applications," *Proc. IEEE Int. Conf. on Parallel Processing,* 1980, 259-260.

[STO76] Stonebraker, M., Wong, E., Kreps, P., and Held, G. "The Design and Implementation of INGRES," *ACM TODS 1,* 3 (Sept. 1976), 189-222.

[STO81] Stonebraker, M. "Operating System Support for Database Management," *CACM 24,* 7 (July 1981), 412-418.

[SU75] Su, S. Y. W., and Lipovski, G. J. "CASSM: A Cellular System for Very Large Data Bases," *Proc. VLDB,* 1975, 456-472.