A locking facility for parallel systems

by N. S. Bowen
D. A. Elko
J. F. Isenberg
G. W. Wang

Clustered and parallel architectures provide the ability to construct systems of high capacity, scalable features, and high availability. In order to provide high throughput in a shared-disk architecture, fundamental advances in multisystem locking technologies are required. This paper describes a locking architecture and operating system support provided for the locking services in a clustered environment. Although initially targeted toward database systems, the functions are general enough for use in many other environments. The paper also describes the products that have deployed this technology.

Synchronization is a fundamental function required in nearly any aspect of a computer system. It has been studied in various research communities including distributed database systems, basic operating system principles, parallel processing systems, and centralized database systems. In parallel processing the synchronization techniques must be extremely efficient to allow frequent synchronization of parallel application programs (e.g., barrier synchronization used in a parallel DO loop[1]). A survey of various techniques can be found in Reference 2. Concurrency control in database systems has also been a well-studied problem in both centralized databases[3] and decentralized systems.[4] Kohler reviews many techniques for concurrency control such as locking, time stamps, circulating permits, tickets, conflicts, and reservations.[4] Although many of these techniques are quite different, the fundamental goals of high-performance locking services are similar. This paper describes a locking facility designed for the S/390* Parallel Sysplex*; this facility is general purpose in nature and has specific implementations that support database concurrency control systems.

The objectives of this facility are high availability, scalability, and high transaction throughput for workloads that are both update-intensive (high write-to-read ratios) and read-intensive (low write-to-read ratios). Although there are many emerging systems with strong claims for availability, scale, and throughput (see Reference 5 for a comprehensive and recent survey and analysis), these systems do not generally support high write-to-read ratios unless the workloads are first partitioned across systems. It is claimed that a "shared-disk" function (that is, all disks are accessible from all processors) combined with a high-speed locking facility are essential functions. Note that additional functions are also required (e.g., high-speed cache-like functions and shared-memory functions),[6,7] but this paper focuses on locking.[1]

This paper is organized as follows. The next section describes the system model and design objectives, and is followed by a section on an overview of the locking services. The last section justifies many of the technical claims through performance modeling and measurements.

System model and objectives

Parallel processing is increasingly being used in commercial systems. The limitations of symmetric mul-

0018-8670/97/$5.00 © 1997 IBM

tiprocessors (SMPs) have long been noted,[8] with practical limits on the order of ten processors. However, demands for processor capacity greatly exceed this capability and the advent of advanced coupling technology[9–13] and commodity technology (e.g., processors, interconnects, disks) is driving these systems into acceptance in commercial markets. An overview of these systems, in addition to covering several key technical issues, can be found in Reference 5.

There are four key objectives to the S/390 Parallel Sysplex, specifically related to the use of parallel technology. These are:

1. High availability
2. Scalability
3. High transaction throughput
4. High write-to-read ratios in the workloads

Underlying the parallel architecture is a shared-disk architecture. In Reference 5, various topologies for connecting disks to processors are discussed and it is observed that many shared-nothing architecture (i.e., disks partitioned among the processors) implementations are emerging. It is claimed that to achieve a high write-to-read ratio, not only must a shared-disk architecture be used, but in addition, advances must be made in critical functions to support data coherency among processors. These functions, embodied in the S/390 coupling facility,[6,7] contain functions for buffer sharing and invalidation, shared queue structures, and locking. Not only does this system support high write-to-read ratios but it can also achieve higher levels of success with respect to availability, scale, and throughput. There are also benefits in areas such as workload management, systems management, and similar issues that are further discussed in Reference 6.

The primary workload is high-throughput transaction-processing systems. In addition, the workloads contain relatively low database contention. Low response time is critical for high-throughput transaction processing, and locking is a critical component. This can be illustrated with a simple use of Little's Law[14] that states the multiprogramming level is a product of the arrival rate and the response time. For example, assume an arrival rate of 100 transactions per second with an average response time of 0.5 seconds. This would lead to a multiprogramming level of 50 transactions; that is, on the average at any instant there would be 50 transactions at some point in their execution in the system. The lock contention seen by any given transaction is a function of the current locks held by other transactions. Thus, the multiprogramming level has a major impact on the lock contention. Now consider the transition from a database running on one processor to one being accessed across multiple processors. Obviously, the response time to obtain a lock will increase. Using the above discussion, this effect would increase the multiprogramming level, which could impact the lock contention. This highlights the key performance objective, which is to provide a high-speed locking facility that has a minimal impact in transaction processing performance in a parallel system and has the additional property that more processors can be added without further affecting the lock response time (i.e., the algorithms are not a function of the number of processors in the parallel server).

For historical perspective we also highlight the evolution of the multisystem locking capabilities of clustered S/390 processors. Shared disks, initially introduced in 1969, provided multisystem serialization controls by allowing one system to reserve the device (that is, lock out other systems from accessing the disk) for a period of time. The global resource serialization (GRS) component provided shared or exclusive named locks across multiple systems.[15] This was used by operating system components to serialize file access across multiple systems. This gave the capability to concurrently share files on a single disk drive. The resource lock manager was introduced to provide record-level locking and buffer management for IMS* (Information Management System) in a two-system configuration.[16] Finally, the coupling facility introduced the locking model that is the basis of this paper.[6] Table 1 summarizes this discussion.

S/390 Parallel Sysplex and the locking model

The system consists of multiple operating systems with a shared-disk architecture. This is referred to as the S/390 Parallel Sysplex. The base operating system contains a rich set of services to support the clustering of these systems. These include membership services, high-speed signaling, and shared-disk support.[6,17] The system, as shown in Figure 1, consists of up to 32 processing nodes (each of which can be up to a 10-way SMP) connected to shared disks. Each SMP runs a single copy of the OS/390* operating system. For the remainder of this paper, we use the term *system* to describe one of the nodes. The sysplex timer serves as a synchronizing time reference source for systems in the sysplex, so that local processor time

Figure 1 Parallel Sysplex system model

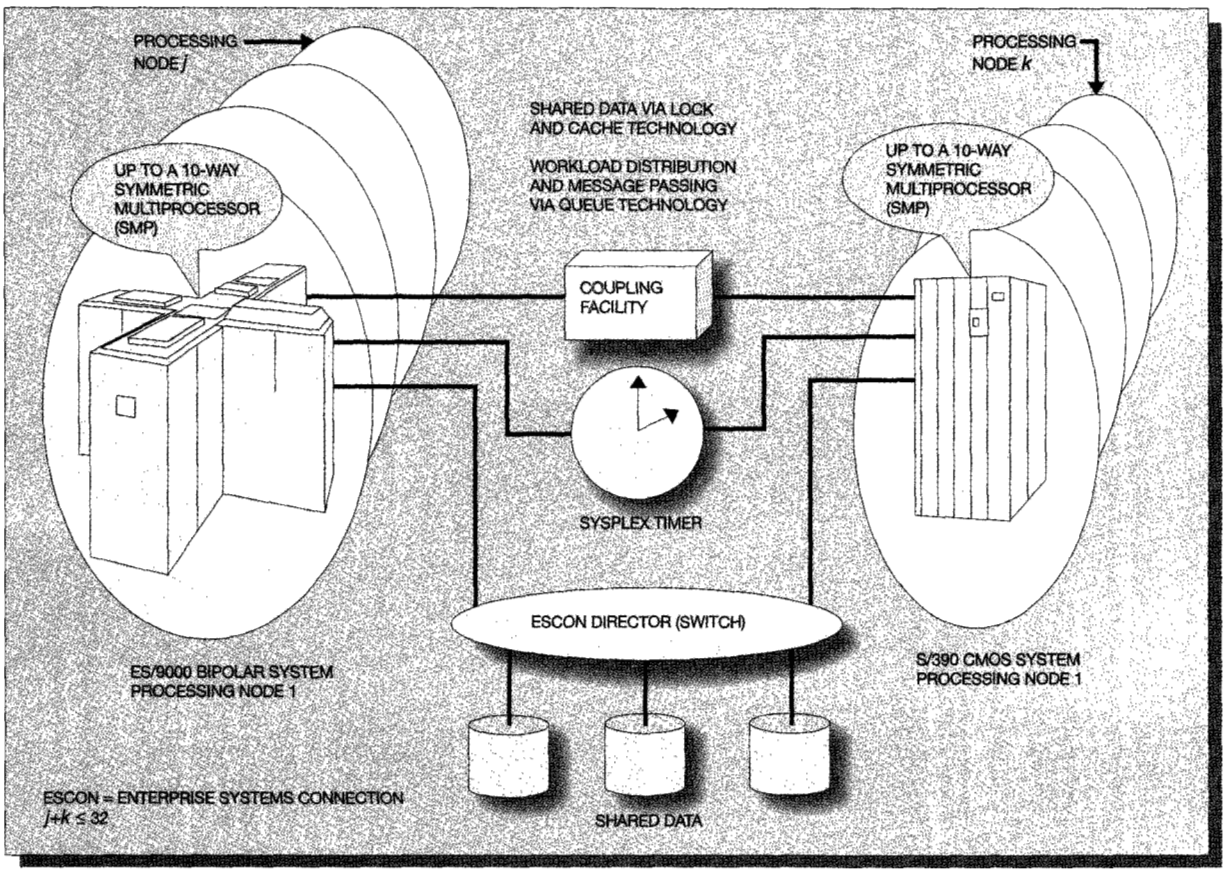


Table 1 History of serialization techniques for the S/390, where *M* = number of systems in the cluster

Year	Type	Relative Performance	Granularity
1969	Device reserve	I/O operation	Disk drive
1980	GRS	*M* − I/O operations	Files, general purpose
1981	IRLM	2 − I/O operations	Database records
1992	Coupling facility	Context switch	General purpose

stamps can be relied upon for consistency with respect to time stamps obtained on other systems. The coupling facility (CF) is a key Parallel Sysplex technology component providing multisystem data-sharing functions. An overview of the various functionalities is described in Reference 6. It is important to point out that the CF is implemented using an S/390 processor with special links to the other systems.

This paper focuses on the locking model. The complete details of this model are beyond the scope of a single paper; our objective is to provide an overview of the architecture. In order to simplify the presentation of the model, the architecture is described from the perspective of three layers. We first describe the basic underlying architecture and the functions within the coupling facility. Next we describe the additional services and functions that are added by the operating system. Finally we describe the view from the perspective of a user of the operating system services. This is a unique aspect of the scheme—that various database lock managers can tailor their use

to create a multisystem lock manager with unique locking semantics.

Level 1: The locking architecture. This section describes the underlying locking architecture. This primarily describes functions within the CF. Nevertheless, the functions described are actually part of both the CF and the operating system.

Conceptual model. Viewed in isolation, this aspect can be described as a centralized lock manager that provides simple shared (SHR) and exclusive (EXC) semantics. It does not provide queuing. A user can operate against a named lock table. Multiple tables are supported where each table contains N lockable entries ($0 \ldots N - 1$). Table 2 shows the semantics of this model.

A key aspect of this system that sets it apart from conventional approaches is that the requests to the locking facility are done synchronously with respect to the processors that execute the request. That is, when a lock request is made, the requesting processor logically stalls until the request is completed. This approach is taken because the performance characteristics of the locking service make it technically feasible. The issue of the performance impact on the design is discussed in more detail in a later section of this paper. This means that a context switch is not needed, thus avoiding issues such as the overhead of suspending the requestor and the complexity of asynchronous locking protocols.

Physical model. Figure 2 shows the key structures within a locking table. It contains N entries and each entry has information to track the exclusive or shared state. For each entry, the first byte is used to contain the system identifier for exclusive or globally managed locks (these terms are explained in detail shortly). The second field in the entry is a bit vector with one bit for each possible system that may have interest for this entry. These bits represent the interest of a particular lock manager on a particular system. The concept of individual lock managers is clarified in the next two sections.

Figure 3 shows the overall structure of the multisystem locking model. There are up to 32 nodes connected to one or more coupling facilities. Each system may have multiple links to the coupling facility for both availability and performance reasons. The performance of the link is critical to achievement of the synchronous behavior and is explored in further detail in a later section. There can be multiple coupling facilities, again for both performance and availability reasons. And finally, within each coupling facility there can be one or more named lock tables.

Figure 2 Structure of a locking table

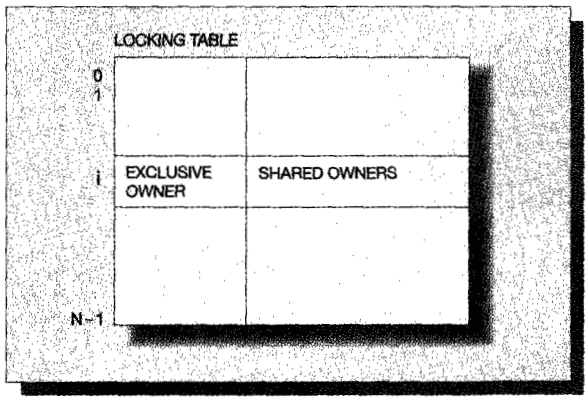


Table 2 Locking semantics from the Level 1 perspective

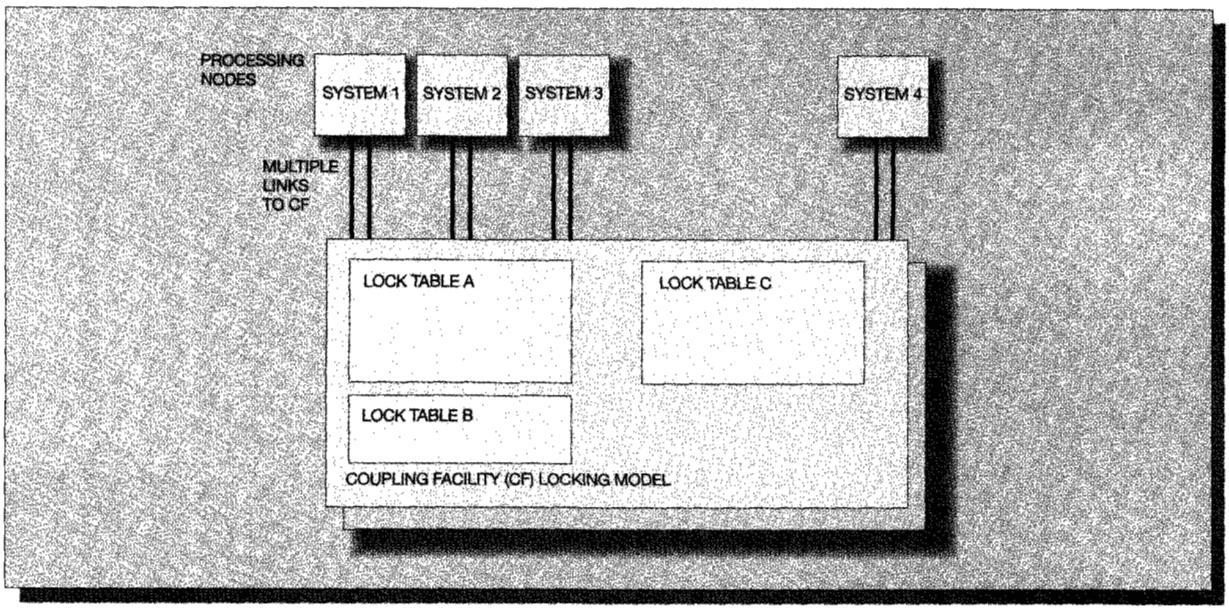
Current State	Request	Outcome
Free	SHR	Granted in share mode
Free	EXC	Granted in exclusive mode
SHR	SHR	Granted
SHR	EXC	Granted with warning about share holders
EXC	SHR	Rejected but told who is the owner
EXC	EXC	Rejected but told who is the owner

Level 2: The operating system. This section describes the operating system support for the model. The focus is a simplified description of the locking capabilities. The actual capabilities go far beyond the simple shared and exclusive model described here. The complete set of services in the product is defined in Reference 10 and an overview is provided in the next section on system-level components.

Conceptual model. The model presented in Level 1 would have limited use in any real system. This section discusses the enhancements in the operating system to provide a richer set of locking semantics while exploiting the basic services provided in the Level 1 model.

The operating system provides the capability to name a lock request. The name consists of a character

Figure 3 General locking model



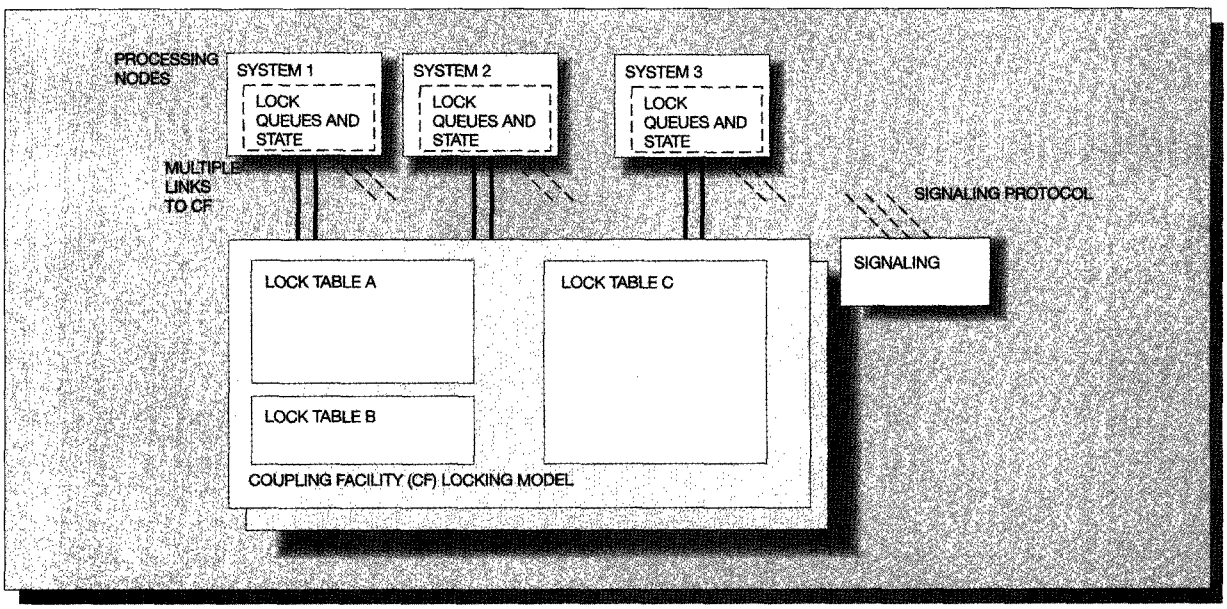
string, to be referred to as the lock name, and an integer, to be referred to as the hash class value. In addition, queuing of conflicts and additional semantics to support multiple lock states (this aspect will be discussed in the Level 3 description) are also provided. The conceptual model has aspects of both a distributed lock manager and a centralized lock manager. The set of operating system services that support this model is referred to as the system lock manager (SLM). The operating system views the locking architecture as a high-speed lock-contention detector. By exploiting the locking hardware, the operating system is actually able to operate in two distinct modes. When there is no contention, the Level 2 model is that of a high-performing centralized lock manager. When contention exists, the Level 2 model is a distributed lock manager that uses signaling protocols to resolve contention and perform notifications.

Physical model. The implementation of these services is very complex and a complete description is beyond the scope of this paper. Instead, several sample lock requests are used to illustrate the key functions provided. The actual programming interfaces contain much more than described here. Figure 4 shows the overall model with respect to this level. Lock queues have been added at each system and there is a signaling protocol used to resolve lock conflicts. First we want to highlight where lock state information is stored. The coupling facility contains the state of each lock table entry, while each processor contains additional state information.

The notation used in these scenarios is first described. Figure 5 shows three systems connected to a coupling facility. The examples used all refer to lock table entries i and k. The lock table entries are used through a concept called *hash classes*, which are described in subsequent sections. The relative point is that lock names are mapped to hash classes and all lock managers that are using a common lock table must also use a common hashing algorithm.

Within the lock table entry the exclusive field (the column "EXC") will refer to the system that currently has exclusive control of that lock entry. The share string is a bit mask that positionally refers to a system that has shared interest in the lock entry (if set to "1"). For example, a share string of "011" means that systems 2 and 3 have shared interest. Within each system four distinct areas of state and queue information are shown. These are used to clarify the examples and do not necessarily indicate how it was implemented. The first area, *local lock state*, is where each system "remembers" its own view of the state of each lock entry. This is either no knowledge (0),

Figure 4 Operating system model



shared knowledge (S), exclusive knowledge (E), or that some other system is an exclusive holder (Gx).

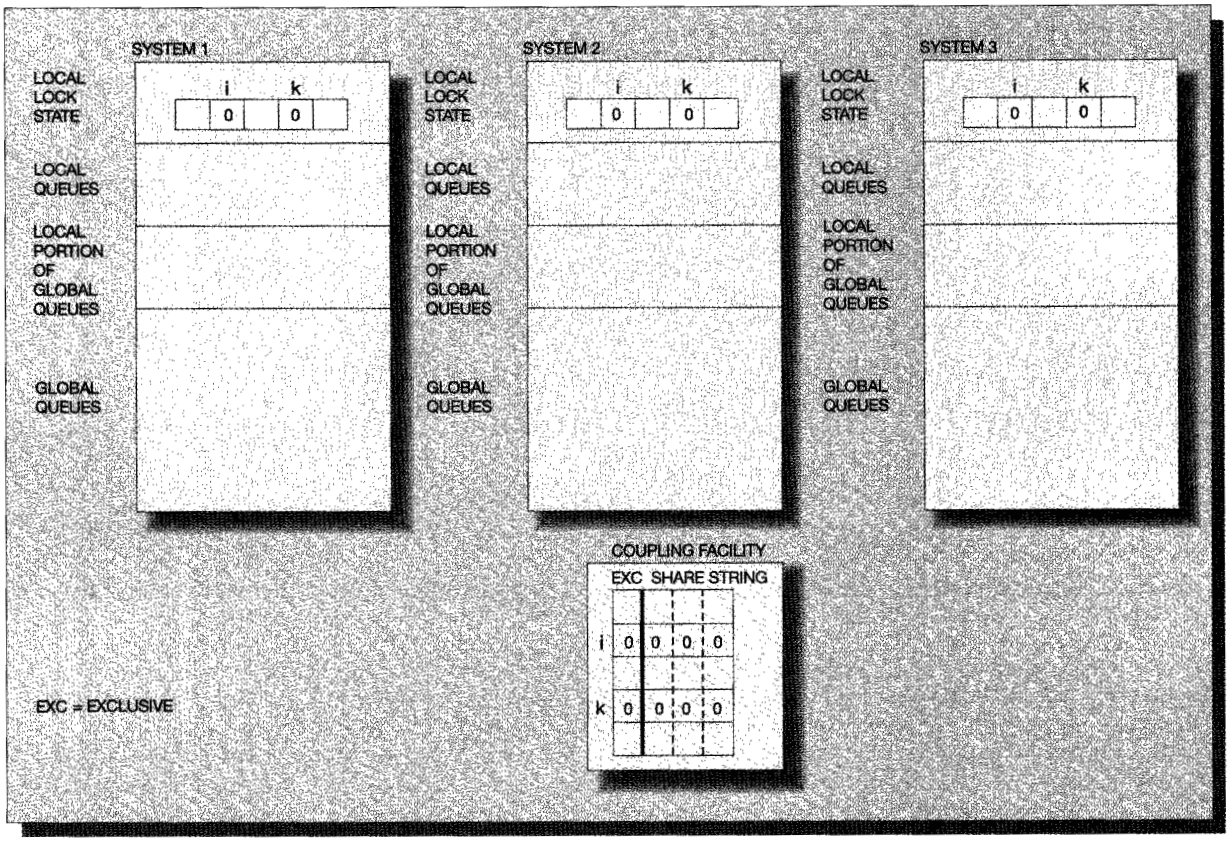
The second area, *local queues*, shows the software queues that are maintained when a lock is granted without the involvement of other systems. This shows the lock name and the hash class value (e.g., "A,i"), and the current requestor, state, and ownership (e.g., "P_1 *(Own, EXC)*" means that Process P_1 owns the lock in exclusive mode). The third and fourth areas show the same type of information as in the local queues but with a different perspective. Requests are moved into the third area when another system becomes the global manager (i.e., a single system provides overall management of the hash class). That is, this area shows the view of *this system* of a global queue. Requests are moved to the fourth area when this system becomes the global manager of the queues. Therefore, the locks in this area represent the complete global state.

The first example (see Figure 6) shows a simple request from Process P_1 for a lock named A in hash class i. Upon receiving this request the local lock state is checked for hash class i, and since it has no knowledge of the state it makes a lock request for hash class i in exclusive state (shown with action 1 in Figure 6). The coupling facility receives this request and since there are no prior requests for this hash class, it sets the exclusive entry with that of System 1 and returns a positive response. System 1 then grants the request and sets the appropriate information in the local queues. It also sets an indicator in the local lock state and records the full lock name in the local queue area. Process P_1 is able to obtain a global lock without interacting with any other systems.

Figure 7 shows the resulting state after Process P_2 on System 2 obtains a lock named C in hash class k. This also results in a single trip to the coupling facility where the share entry is now positionally set for System 2. The request is then recorded in the local queues and granted.

The next example (Figure 8) shows the case where a process makes a request that is compatible at the hash class level. The key point in this example is that the global lock is obtained without even having to visit the coupling facility. Figure 8 shows an example of Process P_3 on System 1 making a request for a lock named B in hash class i. This time when the local lock state is checked the system determines that it already has exclusive interest in the hash class so it does not need to make a coupling facility access. It can simply check the local queues and determine

Figure 5 Notation

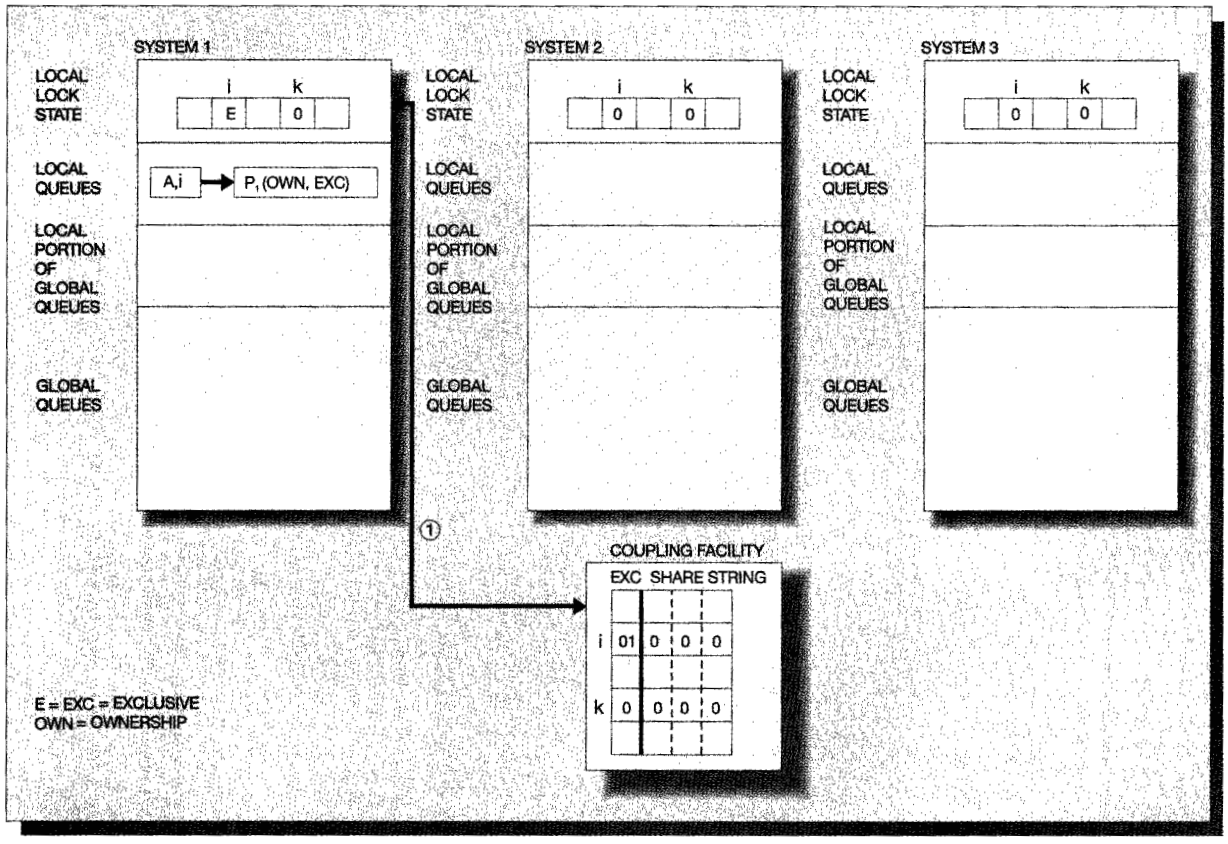


that the request is for a different lock than what is currently held. The request can then be granted.

Figure 9 illustrates how requests "bump into" one another at the hash class level from multiple systems but are still compatible. The key point in this example is that the global lock is obtained without having to exchange lock names even when there is some collision at the coupling facility. Figure 9 shows an example of Process P_4 on System 3 making a request for a lock named D in hash class k. The local lock state indicates that the system has no interest in the hash class so a request is made to the coupling facility. This time the coupling facility sees that System 1 also has a shared interest in the request but since the current request is compatible (shared) it sets the share entry for System 3 and returns a positive response. System 3 is never made aware of the other "sharers" and the request can then be granted.

The previous example (Figure 9) illustrated the case of a hashing scheme mapping multiple lock names to the same hash class. A variation on that example is now shown—when two different lock names map to the same hash class but they are incompatible at the hash class level. Figure 10 shows an example of Process P_5 on System 1 making a request for a lock named E in hash class k. The local lock state indicates that the system has no interest in the hash class so a request is made to the coupling facility. This time the coupling facility realizes that several systems have a shared interest in the hash class. The coupling facility sets the exclusive entry and returns the shared string to System 1. System 1 has now accepted responsibility to sort out the global state of this hash class. System 1 then begins a process called *escalation* in which a global queue for the hash class must be built. It first parses the shared string to determine that Systems 2 and 3 have interest in the hash

Figure 6 Initial request

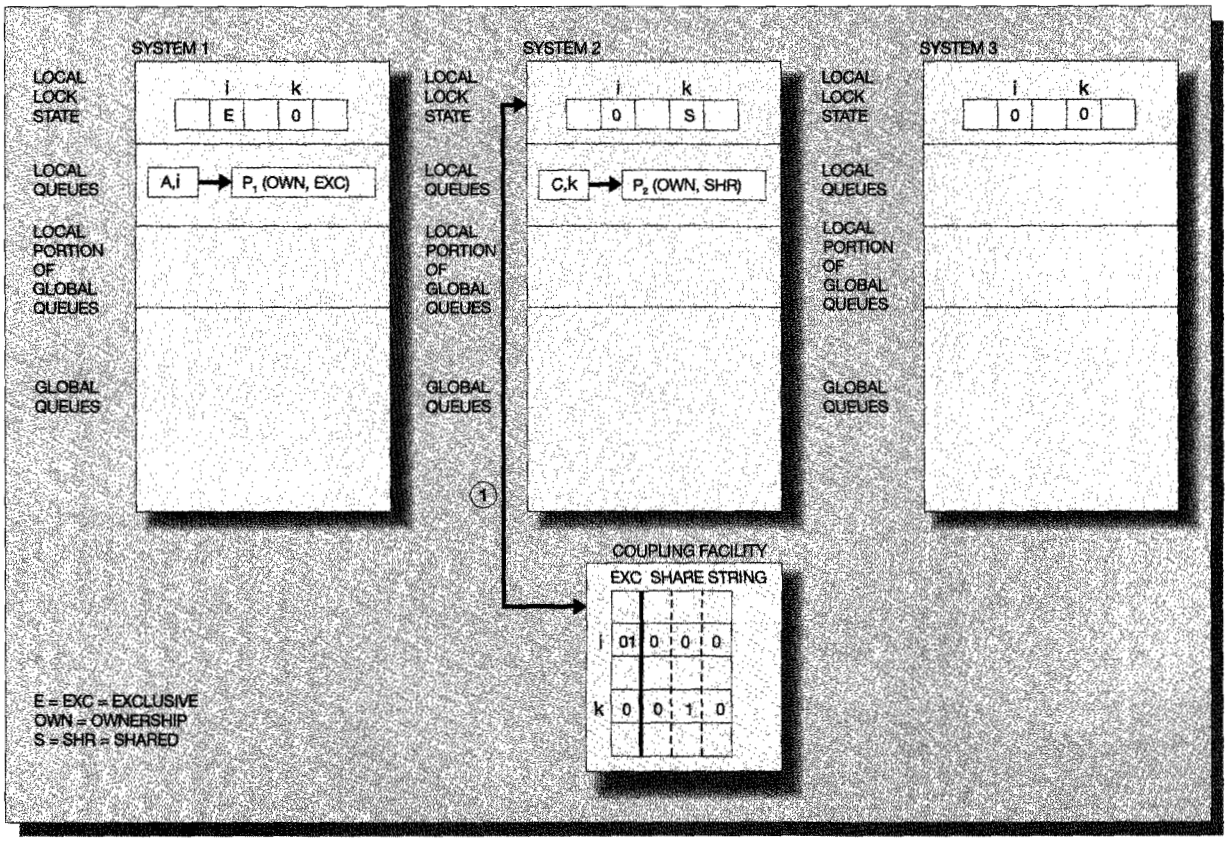


class and signals those systems to return their lock information. There are two important points here. First, these signals are done in parallel over a high-speed signaling facility. Second, only the systems that have a current interest in the hash class are signaled. This is an important aspect of the scalability of the design—if there were 32 systems in this example configuration only two systems would be interrupted for lock information. Once System 1 receives the local information from the other systems, it builds a global picture and realizes that there is no contention for any lock name, just for a hash class. This situation is called *false contention* and the process P_5 can be granted the lock. The example also shows a movement of the queue information on Systems 2 and 3 to the local portion of global queues. This is to illustrate that the process now has a responsibility to communicate with the global manager on future state transitions (e.g., unlocks). Also note that the local lock state was changed from S to G_1 indicating System 1 is the global manager for this hash class.

Our final example (Figure 11) looks at real contention. Figure 11 shows an example of System 2 making a request for Process P_6 for a lock named A in hash class i. The local lock state indicates that the system has no interest in the hash class so a request is made to the coupling facility. This time the coupling facility realizes that System 1 has exclusive interest in the hash class. The coupling facility returns an indicator that System 1 is the exclusive owner of the hash class. When System 1 receives the message from System 2 it then builds a set of global queues. Since it was the exclusive owner it does not have to signal other systems. Once the queues are built it determines there is real contention. There are elaborate facilities for handling contention that are described in the next section.

Level 3: System level components. This section completes the description of the locking services by highlighting how a system level component (e.g., a database manager) could use these services. The goal

Figure 7 Initial shared request



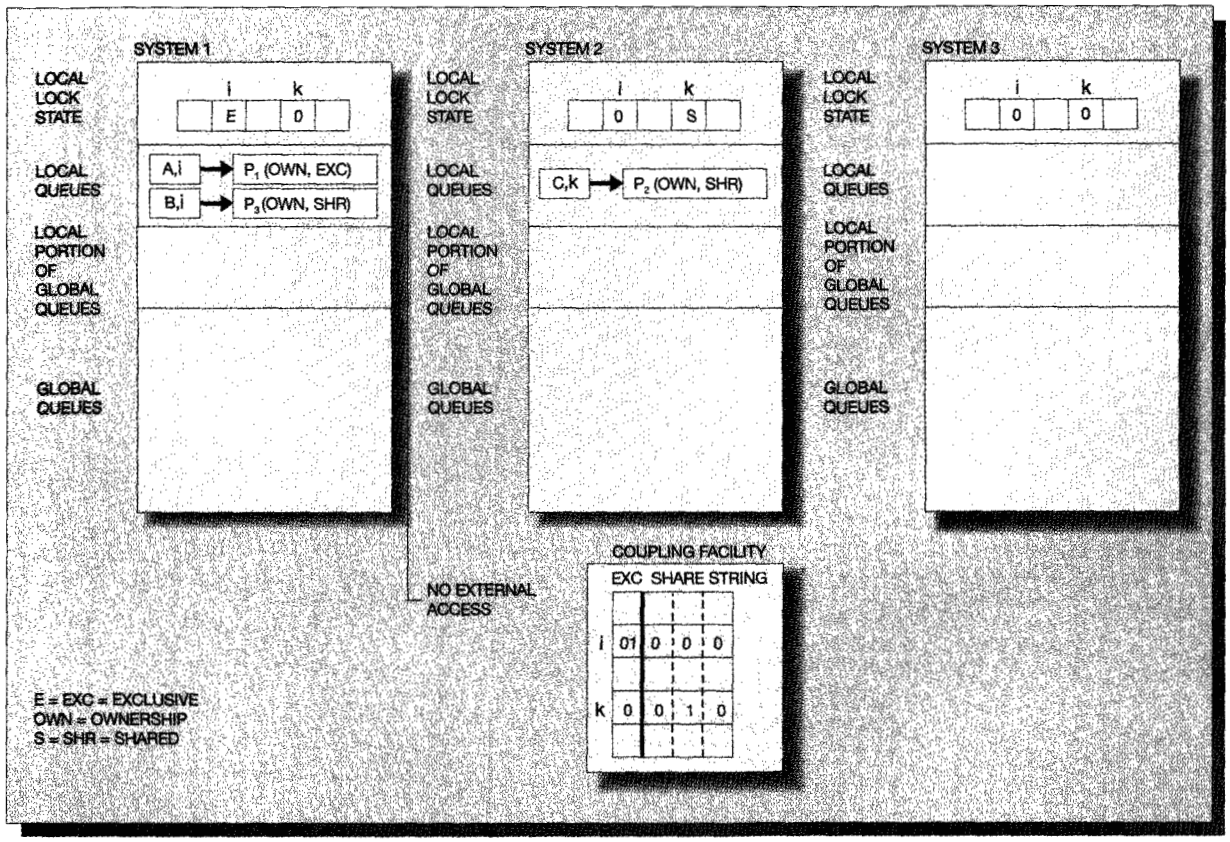
of this section is to illustrate the flexibility of the model and not provide a comprehensive description of all capabilities. The model has been used by several database managers (e.g., IMS, DB2*, and Virtual Storage Access Method, or VSAM) and details of their experiences can be found in related papers.

Although the Level 2 model is a general-purpose lock manager, many database lock managers have much richer locking semantics such as multistate support (e.g., more than just the two basic lock states; shared and exclusive). Other functions such as lock promotion or demotion (i.e., changing the state of a currently held lock) are critical for the overall performance in a clustered environment. Using the semantics of the Level 2 model, the Level 3 model can be tailored to support virtually any lock model. For example, the lock states supported by many database systems are far more complex than the simple shared or exclusive state. Using the following features this can be accomplished.

This section describes some of the key capabilities that can be constructed using the Level 1 and Level 2 models previously described. Since the paper describes a general set of functions that could be constructed by any lock manager, we use the term *tailored lock manager* (TLM) to represent its name. A TLM can be specific to any environment such as a database manager or a shared file system. The key point is that the TLM must be able to optimize its performance based on its unique environment. The functions in the lower levels provide these building blocks. These points are illustrated by presenting a partial list that highlights some of the unique capabilities of this model.

Lock names and hash classes. In general, database managers (DBMs) lock on names that are meaningful in their particular data structure. For example, the IMS system uses a lock with 19 bytes (or 152 bits) that is representative of the data in the IMS hierarchical data structure. The DBM can thus optimize the

Figure 8 Compatible requests—local

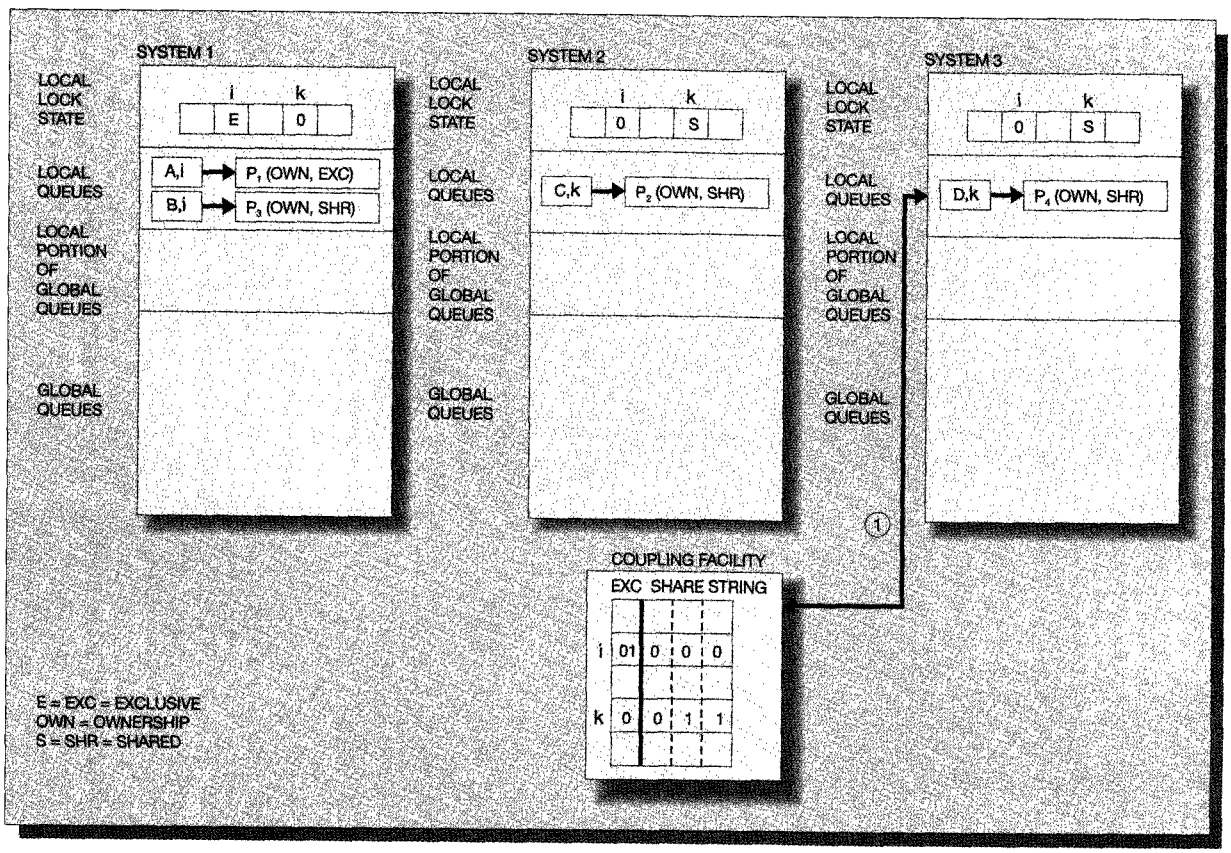


locking structure around the data structure. Obviously the number of locks held at any given point in time is an extremely small fraction of the 2^{152} locks that are possible. In order to provide for efficient contention detection between systems, a hashing algorithm is employed to map the lock names into a hash class table. As long as the tables are of a size that is many times larger than the number of locks held, any false contention on a hash class is kept small. An initialization process is used by the DBM and the TLM to allocate resources in the system lock manager (SLM) and the coupling facility to provide for the appropriate contention detection. At initialization time, the first TLM that is started calculates an appropriate size for the coupling facility hash tables (out of the total storage made available for lock tables and record tables by the customer policy) and requests the SLM to allocate structures for the hash tables using a predefined name of a sharing group. Subsequent TLMs that are started share the same data join in the use of the facilities allocated by the first TLM by using the same sharing group name.

The TLM can also specify a 64-byte "user-data" parameter with the lock requests. One use of this information is to contain the lock states when the TLM lock protocol supports states other than just share and exclusive.

Contention detection. A lock can be held with one of several ownership privileges. Ownership can be granted when the privileges of the holders (if any) are compatible with the privileges needed by a new requestor. For example, a lock may be requested with either share or exclusive privilege. Contention is detected when a share privilege is requested and a lock holder has exclusive privilege, or when an exclusive privilege is requested and there are existing holders.

Figure 9 Compatible requests—remote

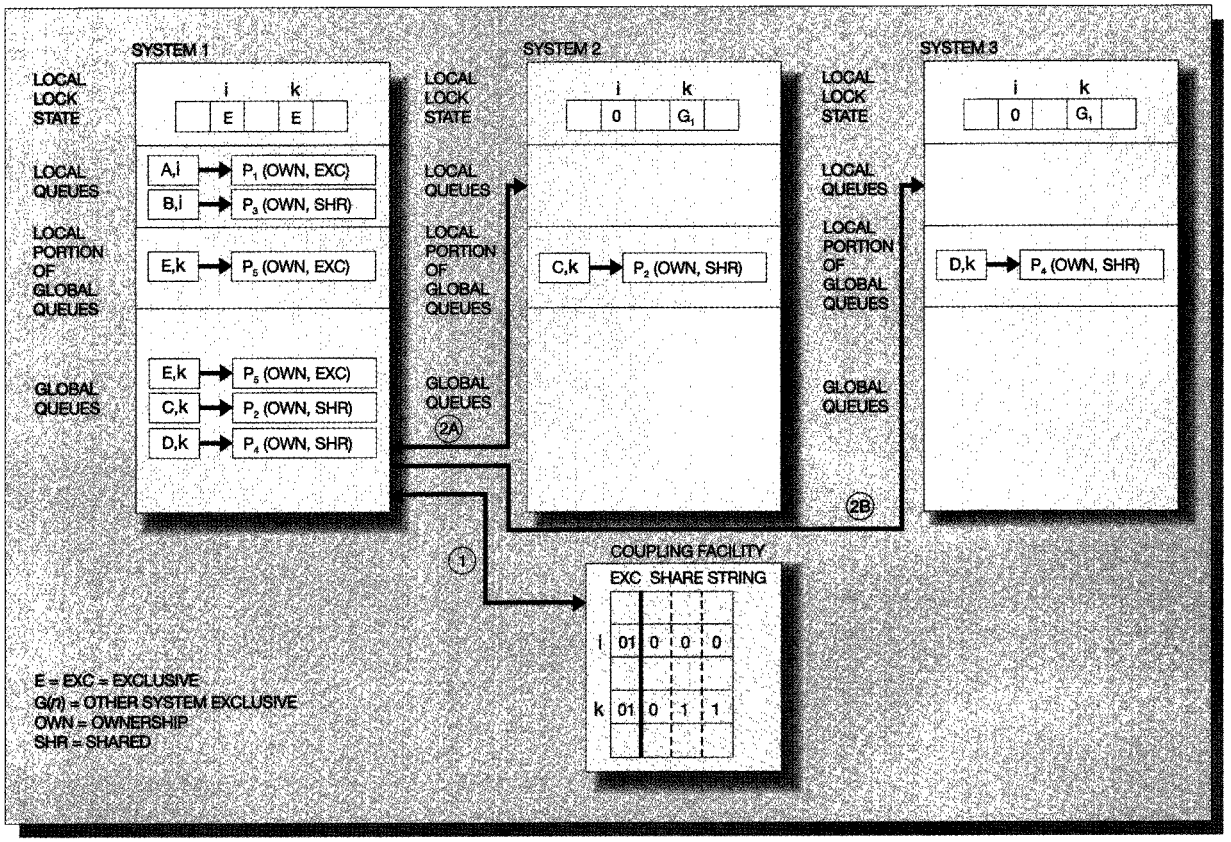


Contention and notify exits. The SLM communicates to the TLMs via a mechanism called *exits* in OS/390.[18] Two of these exits, the contention and notify, are the means by which the TLMs resolve contention for shared resources. In these exits, the 64-bytes of user data can be inspected or modified. One key value is that complex lock protocols can be implemented using this structure.

Waiter queuing. When the lock manager cannot grant a lock because of contention, the SLM preserves a record of the request on a list of *waiters*. The rules for processing waiter queues vary among each pair of DBMs and TLMs. The queuing is done within the SLM; however, the rules for lock compatibility are done by the TLM using the exits mentioned previously.

Availability and recovery recording. In order to meet the demanding continuous availability requirements of many of today's large commercial transaction systems, it is important to allow processing to continue with full integrity of the database while handling recovery from a hardware or software failure. Since the coupling facility is electronically and logically isolated from the systems that are running the DBM/TLM/SLM software, it provides the necessary availability for recovery from a system or software failure. The coupling facility structure provides both a locking function and a recovery recording function. *Modify lock names* (exclusive locks that are used to update database records) are recorded in the list elements in the coupling facility recovery tables. A list is assigned to each instance of the system lock manager participating in the global managed locking protocols. The coupling facility uses the user identification (UID) that specifies the particular system lock manager to access the appropriate list. Atomic operations that manipulate lock table entries and record data elements are provided in the coupling

Figure 10 False contention

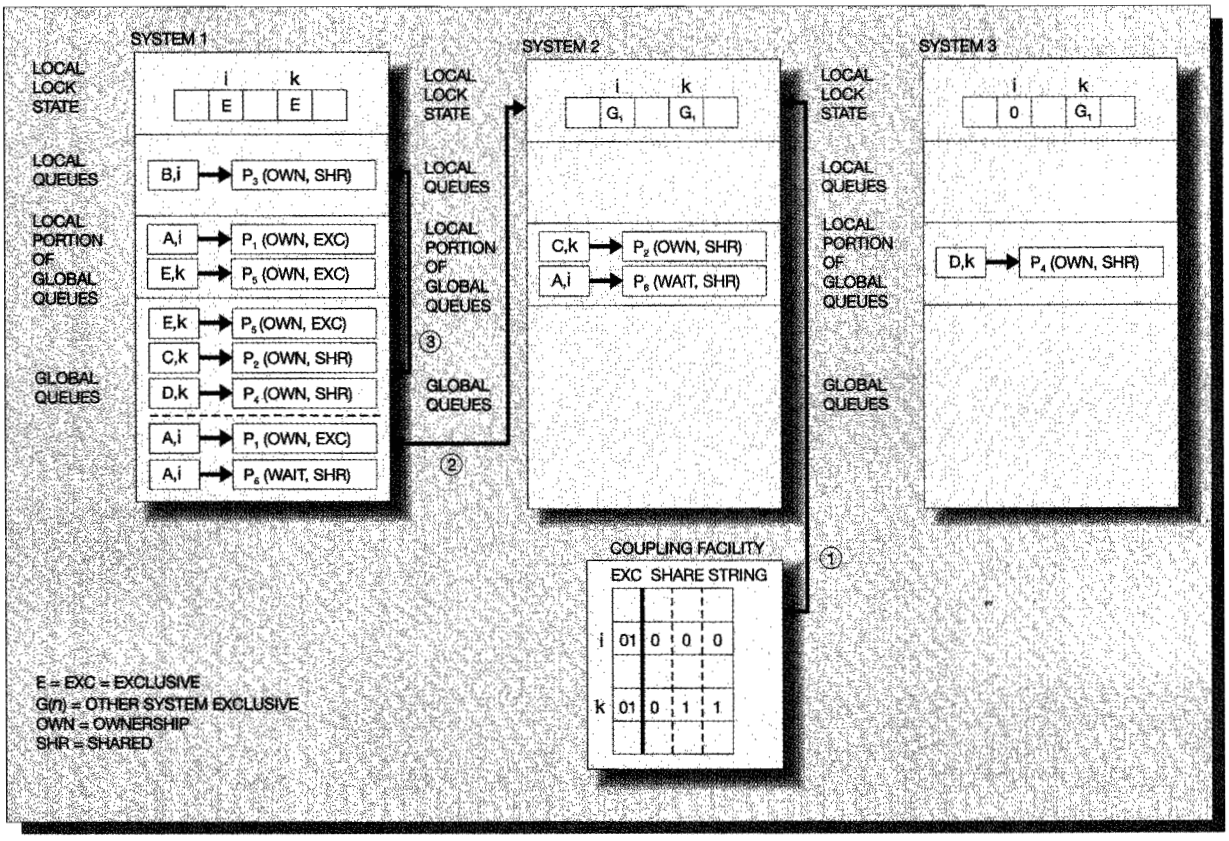


facility architecture. These operations support commands for creating, reading, replacing, and deleting elements. By providing atomic operations, the architecture ensures that the lock structure and the recovery structure are always consistent. Note that in the event of a failure of the coupling facility, no data are lost since all the information in the coupling facility is replicated in the set of TLM/SLMs. Since the architecture supports multiple coupling facilities, a new structure can be allocated in another facility and the TLM/SLMs recreate the coupling facility contents.

Contention resolution. Let us now examine the process used by the TLM/SLM for the resolution of contention. Although the key points are highlighted here, the reader is referred to Reference 9 for a complete description. In order to determine the existence of contention, the coupling facility and the SLM use the lock compatibility matrix shown in Table 2. Any time the requested state is compatible with the existing state, the request is granted and the lock is locally managed by the holder(s). When the requested state is incompatible with the existing state, then the lock becomes globally managed by a chosen TLM/SLM combination, and this request along with future requests are processed by the global manager (TLM/SLM). The chosen SLM does not manage the contention, but rather maintains a queue of holders and requestors for the TLM to use to manage the contention. When contention is detected, the chosen SLM passes the request queue to the TLM by using the contention exit mentioned earlier. The user data information plays an integral role in enabling user-defined lock protocols. At this point the TLM must manage the contention by the use of one of the following actions:

1. Grant a pending request, possibly with a different state than that requested. This will cause the

Figure 11 Contention



requestor to resume, or will control its completion exit if appropriate.

2. Deny a pending request. This will also cause the requestor to resume, or its completion exit to be taken. However, in this case the requestor is also told of the rejection and given any data provided by the denying TLM. The user data can be modified when the request is denied. Since this is presented to the requestor of the denied request, the user data can be used to communicate the reason for denial.
3. Regrant a held request with a different state than it was originally granted (e.g., demotion of a lock that is held exclusive to shared). In this case the holder's completion exit is initiated, informing the holder of the change in state. In addition, the user data can be modified on a regrant.
4. Inform a current resource owner that contention exists for a resource it owns. This is done by passing the notice (through the SLM) to the holding TLM by use of its notify (notification) exit.
5. Leave a request pending. The request will be left in the request queue maintained by the SLM. The TLM may grant the request on a subsequent invocation of the contention exit (e.g., when the unlock of the current owner is presented to the contention exit).

This implementation of TLM/SLM exits and commands provides a powerful method for implementing lock negotiation protocols in the TLM while allowing the system (SLM) to maintain an awareness of the contention, and the state of the holder/requestor queue. In turn, this allows the TLM to implement a process whereby users (DBMs) initially obtain high-level locks, and then negotiate them downward to finer levels of granularity as contention occurs. This process of negotiating and notifying continues iteratively until the TLM determines that no more notifies are required and the state of the holder/requestor queue is correct. The global management of the lock continues with the selected TLM/SLM until contention

no longer exists. Note that due to the flow of requests, including unlock requests, it is possible that the global manager TLM does not have any interest in the lock it is managing.

Locking performance

The performance of the locking function is a critical component of a parallel transaction processing system with shared data. Much effort went into ensuring that the overhead of locking was as small as possible. There were many components of this effort. Several of the most critical are described in this section. It should be noted that this is not a reflection of a completed body of work, but represents work in progress. As most complex designs evolve, the performance bottlenecks become better known and their solutions often affect the original design accordingly. Locking in a Parallel Sysplex is no exception.

Synchronous versus asynchronous locking. Accesses to the coupling facility for obtaining locks require a communication outside of the processor. In that way it is similar to an I/O operation. However, the overhead of an I/O operation is well known to be detrimental to both system throughput and transaction response time. Much effort has been expended on nonparallel transaction systems to reduce the number of I/Os by extensive use of main storage and expanded storage to hold I/O buffers. Such data buffering allows records to be accessed without the need to drive an I/O operation and suspend the transaction. This reduces the path length of the transaction and significantly improves the transaction response time.

It is a design goal in the Parallel Sysplex to minimize the additional unavoidable system overhead introduced by locking without elongating the transaction response time. This is accomplished by (1) building protocols on the coupling facility links that optimize the communications for low latency rather than high bandwidth, and (2) defining the architectural interface to allow for locking commands to be delivered to the coupling facility and responses received within a single CPU instruction. The processor spins in an idle loop in the microcode from the time the command is sent on the coupling facility link to the time when the response is returned and stored in main storage. This is referred to as *synchronous* command execution. The alternative design is to perform a context switch, i.e., the process thread is suspended from execution and a new process thread is dispatched. This is referred to as *asynchronous* command execution.

It is important to validate that this choice is optimal for both design goals: minimizing system overhead and minimizing transaction response time. Access times to the coupling facility are measured in the range of 50 to 500 microseconds, depending on the operation and the speed of the processor, which is significantly better than the several milliseconds that an I/O operation requires. So from the point of view of transaction response time, the synchronous design is clearly superior.

However, what is less clear is whether the design maximizes system throughput. Since the alternative asynchronous design is to suspend the thread and perform a task switch, success of the synchronous design is determined by comparisons against the overhead of this approach. If the round-trip access time to the coupling facility takes less CPU time than the cost of performing a task switch, then it is deemed to be the correct choice.

The cost of a task switch has two components. The first is the path length that is required to suspend the work unit thread, to dispatch a new thread, to field and process the resulting interrupt on the back end of the operation, and finally, to redispatch the original work unit thread. A value of 2000 instructions is used for this path length. The second component is the reduced performance of the processor caches that results from the context switch and the subsequent purging of the working set for the thread from the hardware caches. Much work has been done to estimate this overhead for various processors, and a value of 4000–5000 instructions is used for this effect. So the total overhead of a task switch is estimated at roughly 6000–7000 instructions.

Since the overhead of the synchronous access to the coupling facility is easiest to measure in units of time, it is useful to convert the asynchronous penalty accordingly. The processor speed is clearly a relevant factor since it is more costly to allow a faster processor to wait than a slower processor. The processors under consideration have a MIPS (million instructions per second) rate that ranges from 20 to 50 MIPS. Using these numbers, one obtains a break-even point for the synchronous operation in the range of 6000/50 = 120 microseconds to 7000/50 = 140 microseconds for the faster 50 MIPS processor and in the range of 300 to 350 microseconds on the slower 20 MIPS processor.

The synchronous time for a coupling facility access has several components that can affect the overall time. These are referred to as *elongators*. One critical factor was already described, namely the link protocol. To minimize this impact, special links were developed that were point-to-point links and had minimal handshaking. In fact, the entire link operation consists of a single transfer of the command block to the coupling facility and a single transfer of the response block back to the originating processor.

A second key elongator is distance. Most of the negative effects of distance are eliminated by the link protocol that eliminates end-to-end handshaking. However, distance still remains a factor and elongates an operation by about 12 microseconds per kilometer. Since most distances were originally expected to be within a machine room or, at worst, up to the limits of fiber-optic channels (about 3 kilometers), it was not viewed that distance would be a serious concern. However, it is an intrinsic problem and no amount of additional design will remove it.

A third elongator is the path length needed in the software lock managers that is required to initiate the operation and process its completion. This is needed in both the synchronous and asynchronous designs, but it is still critically important to minimize these path lengths. Several different efforts have been devoted to minimizing these paths and that work is ongoing.

A fourth elongator is the complexity of the processing that is performed by the coupling facility. In this regard, the final design is somewhat of a trade-off. It had been an original goal to define the locking architecture so that a hardware implementation would be possible. But the benefits of the additional functional capability provided by a microcoded implementation moved us away from that goal. However, significant effort was spent in making the command definitions simple and streamlined. One key aspect was the decision to put queuing and waiter notification in the SLM component and minimize the function provided by the coupling facility to simple contention detection and minimal logging for recovery.

The measurements show that a synchronous locking operation on a 50 MIPS processor connected to an equivalent speed coupling facility completes in about 80 microseconds; this is well below the 120–140 microsecond break-even point. Similarly, a synchronous locking operation on a 20 MIPS processor connected to a 20 MIPS coupling facility completes in about 150 to 200 microseconds, again below the break-even point. Where the design is not optimal is when a 50 MIPS processor is connected to a 20 MIPS coupling facility. Then the transfer time is still in the 150–200 microsecond range since the components of time are mostly determined by delays outside of the processor. However, this is no longer better than the overhead of a task switch. For this reason and for similar considerations when data transfers are included, an asynchronous interrupt has been designed (not in the current product). Its use would be managed by a heuristic routine that monitors the synchronous delay and switches modes accordingly. This will be especially useful in addressing the concerns of distance in a sysplex where several factors, such as increased fiber-optic capabilities and the move toward remote site recovery, are changing some of our original views on distance limitations.

System overhead for locking is minimized without elongating the transaction time.

Lock contention. Contention occurs when a lock request for a resource appears to be incompatible with its current lock state. This is a statement that is relative to the given level of locking (as defined in the section on locking models), where each level may have a different view of contention and may act on it in a different manner. At the architectural level, contention is detected on hash classes of locks. Contention, when detected, is reported to the SLM via information stored in the response block. The SLM attempts to resolve the contention by communicating with its other instances that share information on the hash class and comparing the actual lock name with the set of lock names for the current owners and waiters. It may turn out that the lock name that is requested does not match any of these other names. In this case, contention does not exist at this level and the lock can be safely granted. This situation is referred to as *false contention*, i.e., contention detected at the architectural level that turns out not to exist at the SLM level, which resolves the contention.

It may turn out that the SLM finds one or more matching names in the list of names registered for the hash class. This is referred to as *real contention*, although that is actually a misnomer. The SLM reports the real contention to the contention exit of the TLM. The TLM makes the final determination of the contention. If the TLM can decide the contention is real, then the requestor is queued. Alternatively, the TLM can grant some requests that appear (such as from a share/exclusive model) in contention. It may turn out that the lock name is covering too large an area of granularity in the database, and finer granularity locks are required. This results in renegotiating all the relevant locks to a lower granularity, where actual contention may not exist. This hierarchical approach to locking is used to avoid getting locks on areas of the database that are relatively inactive, or tend to be accessed mostly by a single system. This design can significantly reduce the overall locking rate and is a key method for gaining performance by adjusting the locking rate to match the level of contention in the system.

When the granularity of locking is at its finest level, real and actual contention coincide. Workload traces indicate that this level of contention is extremely small in most transaction systems, and it is generally believed to be the case that significantly less than 0.5 percent of all lock requests experience actual contention.

False contention is another matter. False contention is a function of the size of the lock table (as a proportion of the number of active locks in the system), and the hashing algorithm. The size of the lock table can be reconfigured dynamically to make it larger, so there is some amount of tuning that can be done by the system programmer to minimize the occurrence of false contention. The hashing algorithm is more difficult to manipulate and much work has gone into developing uniform hashing functions. The general rule of thumb is that the total contention in the system should be no more than 1.0 percent. If the locking design is good, real contention can be managed to extremely small levels. And if the hashing function is relatively uniform, then false contention can be managed by controlling the size of the lock table. This is the strategy that is employed.

Consider the example cited in the section on system model and objectives, where a transaction rate of 100 transactions per second and response time of .5 second yields a multiprogramming level of 50. Assuming 20 locks held per active transaction, one can predict 1000 current locks held at any time. If the false contention is to be maintained at under 0.5 percent, then the lock table should have this percentage of nonzero entries at any given time. In other words, the table should be 99.5 percent empty on the average. Thus, the number of hash classes defined should be 200 times the number of locks held. So the lock table should be defined with 200 000 entries.

Unlock operations. Special considerations are given to the handling of unlock operations. In particular, two architecture extensions were developed to improve on unlock performance. One is a special interface that allows the operation to proceed asynchronously without the need for an interrupt, and the second is a command that allows a list of lock table entries to be sent to the coupling facility in a single operation.

Asynchronous unlocks. This section describes a capability that is not in the current product but is relevant to an overall understanding of the system design. An unlock operation to the coupling facility is a command that resets an exclusive field or a share bit in a lock table entry to zero. This creates a window in the state of the lock table entry where the system performing the reset views the lock as free, but another system may be viewing the lock table entry just prior to its being reset. Thus an escalation signal may be received by the original owning system after the lock is released. This window is unavoidable and is handled by the SLM chase protocol.[19]

A consequence of the chase protocol is that it is no longer a requirement to perform the unlock operation synchronously. The window exists whether the operation is performed synchronously or asynchronously to the CPU. If the unlock operation should fail without updating the lock table entry, then the mismatch in state is detected by the next requesting system and recovered by the chase protocol.

The locking architecture exploits this by providing for the specification of an asynchronous option that causes the CPU to release control and return to the program as soon as the command has been transmitted on the link and before the response is returned. At the conclusion of the command, the response block is stored in the main storage by the channel, but no interrupt is generated. At this point, the architectural interface is returned to the idle state and new operations can be initiated. So, even though the response is stored, it is not acted upon by the

operating system. This is referred to as the no-response protocol. This allows the overhead of the operation to be reduced to the minimum start-up penalty on the issuing CPU and is on the order of 200–300 processor cycles (or about 20–30 microseconds).

This protocol is used selectively by the operating system and is generally limited to isolated unlock operations (nonlist form) that do not have any associated record table updates.

Batched unlocks. Most transaction systems obtain locks during the execution of a transaction as they are needed and released collectively when the transaction is committed.[20] So while locking operations occur as individual events, unlocks tend to occur in a batch. It is thus reasonable to consider allowing the unlocks to be batched in a single operation to the coupling facility, and this function is provided in the architecture. This allows the overhead required to initiate the operation and the transmission time on the link to be apportioned across the set of lock names in the list of unlock operations.

It turns out, however, that the multilevel design approach to locking makes this a very complex function, and it was added very late in the development cycle. The problem is that the list must be handled at each level, and the state of the locks, as viewed by each lock-manager level, is different. The simplest level is the TLM level, which simply builds a list of lock names with the associated hash-class values that are held in either the shared or exclusive state. The SLM, however, must parse this into two lists: one list of those hash classes that have undergone escalation and are managed, not in the coupling facility, but by some instance of an SLM, and a second list that this SLM views as managed by the coupling facility.

The first list is actually processed individually by the SLM, and no performance gain is realized. The assumption of low contention generally makes this a short list, but the complexity of handling it must, nevertheless, be incorporated in the design.

The second list is bundled in a command and sent to the coupling facility. This is where the performance benefit is realized. Batch sizes vary, but it is not unusual for a transaction to obtain 10–20 locks and release these all at commit time. So, significant savings can be achieved with this scheme.

Considerations for multiple lock managers. A key attribute of transaction processing systems that has already been articulated is that most locks are generally obtained and released within a transaction. So, the lock hold times are reasonably short (less than a second or so). As the example above shows, the number of concurrent locks held in the system can be estimated as the product of the average number of locks per transaction and the multiprogramming level. Both of these can be measured by standard system performance monitors and, from these, the lock table size can be effectively calculated in the manner shown above. Subsequent monitoring of the false contention rate allows the size to be adjusted and tuned to fit the workload.

In order for this process to be successful, the class of locks mapped to the lock table must be limited to the set of transaction locks associated with a particular TLM. Otherwise there is too much unpredictability in the mix of locks. Unfortunately, there are several multisystem locking components in S/390 systems. Aside from the various transaction managers, IMS, DB2, and CICS* (Customer Information Control System), there are system-locking components such as global resource serialization (GRS). In the case of GRS, the locks are often obtained and held for very long durations.[21] Since the TLM components have control of the hashing algorithms, special locks, such as the allocation locks with very long hold times, can be separated into unique hash classes. These hash classes would be, essentially, permanently managed by the SLM.

However, in order to remain effective, this separation must be maintained all the way to the coupling facility. This is accomplished in the architecture by providing named lock tables and allocation functions that allow for the creation of multiple lock tables on a single coupling facility with unique attributes. These unique attributes include: the presence or absence of share bits, the number of share bits, the existence of a record table, and of course, the size of the lock table and record table. But most importantly, the existence of multiple named tables allows for the separation of locks into distinct name spaces so that unique management, such as that described here for minimizing false contention, can be provided by the TLMs.

Conditional lock requests. Low contention rates are a basic design premise. Real contention in transaction processing systems is known to be quite small (under 0.5 percent of lock requests), and false con-

tention can be managed to similar levels by increasing the size of the lock table. However, contention at some measurable level does exist in the system. When it occurs, the overhead can be quite high; signaling is required from the requesting system to the system managing the entry. Also, if this is the first indication of contention, the managing system changes the state of the entry to global management. This escalation process can be quite lengthy and requires considerable path length and signaling between systems. There is a corresponding penalty when contention is removed and the entry is de-escalated from global management.

Conditional lock requests were included in the TLM/SLM interface to address this problem. If the lock is requested conditionally and the request to the coupling facility indicates that the lock entry is not available, the SLM returns control to the TLM and takes no further action on the request. In particular, no attempt is made to signal the managing system and the possible global escalation is avoided. Subsequently, following a short delay, the TLM can resubmit the lock request. Assuming the lock hold time is short and assuming the probability of new contention occurring is no greater than normal contention, the second request may succeed at the coupling facility. If this is the case, then the global management overhead in the SLM is avoided. The TLM can always abandon this optimistic protocol after several retries and submit the request unconditionally. Use of the conditional protocol by TLMs has resulted in significant reductions in system overhead.

Conclusion

This paper begins with the premise that shared-disk architectures are better than shared-nothing architectures for clustered systems. Specific benefits include workload balancing and effective utilization of the processors, availability, and scalability. It is also argued that in order to support update-intensive workloads that are often found in database environments, the shared-disk architecture must be augmented with special functions to improve the effectiveness of the sharing of information across the cluster. One such function, locking, was the focus of this paper. This paper described a high-speed locking function for use in a parallel operating system environment and provided a detailed description of the architecture and many critical design trade-offs. These facilities are embodied in the S/390 Parallel Sysplex. The facilities are oriented toward transaction processing systems that are update-intensive and require low response times. The system has many unique features, such as the ability to obtain cluster-wide locks using synchronous protocols, the ability to construct a lock manager that supports a user-defined lock protocol (i.e., richer than a traditional share/exclusive protocol), special features for availability, and rigid performance objectives.

Acknowledgments

The authors would like to acknowledge the large number of people, across many IBM divisions, who worked on the S/390 Parallel Sysplex effort.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references and notes

1. C. J. Beckman and C. D. Polychronopoulos, "Fast Barrier Synchronization Hardware," *Proceedings of Supercomputing '90* (November 1990), pp. 180–189.
2. A. Dinning, "A Survey of Synchronization Methods for Parallel Computers," *Computer* **22**, 66–77 (July 1989).
3. R. Obermarck, "IMS/VS Program Isolation," Technical Report RJ2879, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1980).
4. W. H. Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," *Computing Surveys* **13**, 149–183 (June 1981).
5. G. F. Pfister, *In Search of Clusters*, Prentice Hall, Englewood Cliffs, NJ (1995).
6. J. Nick, J.-Y. Chung, and N. Bowen, "Overview of IBM System/390 Parallel Sysplex—A Commercial Parallel Processing System," *Proceedings of the 10th International Parallel Processing Symposium* (April 1996), pp. 488–495.
7. J. M. Nick, B. B. Moore, J.-Y. Chung, and N. S. Bowen, "S/390 Cluster Technology: Parallel Sysplex," *IBM Systems Journal* **36,** No. 2, 172–201 (1997, this issue).
8. M. Dubois and F. A. Briggs, "Effects of Cache Coherency in Multiprocessors," *IEEE Transactions on Computers* **C-31**, 1083–1099 (November 1982).
9. S. Calta, J. deVeer, E. Loizides, and R. Strangwayes, "Enterprise Systems Connection (ESCON) Architecture—System Overview," *IBM Journal of Research and Development* **36**, No. 4, 535–552 (1992).
10. *MVS/ESA Programming: Sysplex Services Guide*, GC28-1495, IBM Corporation (1994); available through IBM branch offices.
11. C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker, "The SP2 High-Performance Switch," *IBM Systems Journal* **34**, No. 2, 185–204 (1995).
12. W. Baker, R. Horst, D. Sonnier, and W. Watson, "A Flexible Servernet-Based Fault Tolerant Architecture," *Proceedings of the 25th Symposium on Fault-Tolerant Computing* (June 1995), pp. 2–11.
13. T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir, "SP2 System Architecture," *IBM Systems Journal* **34**, No. 2, 152–184 (1995).
14. L. Kleinrock, *Queueing Systems Volume I: Theory*, John Wiley & Sons, New York (1975).

15. J. Ranade, *MVS: Performance Management*, McGraw Hill, Inc., New York (1990).
16. J. P. Strickland, P. P. Uhrowczik, and V. L. Watts, "IMS/VS: An Evolving System," *IBM Systems Journal* **21**, No. 4, 490–510 (1982).
17. M. Swanson and C. Vignola, "MVS/ESA Coupled Systems Considerations," *IBM Journal of Research and Development* **36**, No. 4, 667–682 (1992).
18. *OS/390 MVS Installation Exits*, GC28-1753, IBM Corporation (March 1996); available through IBM branch offices.
19. The SLM chase protocol includes the following sequence of events: First, a response to the escalation signal is sent with a message stating the lock is not managed by this system. Then the requesting system makes another access to the coupling facility where it will observe the now reset state of the lock table entry. If the state continues to appear to be held, the escalation signal is resent and the process is repeated. After a threshold of retries is exceeded, the lock table entry is placed in a recovery state and the lock managers perform a coordinated recovery for the entry. The chase scenario is a fallout of any multisystem locking scheme where management of the locks is not bound to a particular system.
20. IMS is an example of a transaction system that releases most locks at transaction commit time.
21. As an example, GRS provides allocation locks for long-running batch jobs, which may execute for minutes or hours.

Accepted for publication January 8, 1997.

Nicholas S. Bowen *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: bowenn@watson.ibm.com).* Dr. Bowen received the B.S. degree in computer science from the University of Vermont, the M.S. degree in computer engineering from Syracuse University, and the Ph.D. in electrical and computer engineering from the University of Massachusetts at Amherst. He joined IBM at East Fishkill, New York, in 1983 and moved to the Research Center in 1986, where he is currently the Department Group Manager of Servers. He is a senior member of IEEE and a member of ACM. His research interests are operating systems, computer architecture, and fault-tolerant computing.

David A. Elko *IBM S/390 Division, 11400 Burnet Road, Austin, Texas 78758 (electronic mail: david_elko@vnet.ibm.com).* Dr. Elko is currently with the S/390 architecture group in Austin, Texas. He joined IBM Poughkeepsie in 1980, worked in the MVS and S/390 architecture groups, and moved to IBM Austin in 1995, joining the server group staff. Dr. Elko received a B.S. degree from Indiana University of Pennsylvania, and the M.S. and Ph.D. degrees in mathematics from the University of Notre Dame. His research interests are computer architecture, number theory, and topology.

John F. Isenberg, Jr. *Isenberg & Hall, Inc., 29 Thornwood Drive, Poughkeepsie, New York 12603 (electronic mail: jacki@us3.global.ibmmail.com).* Before he retired from IBM, Mr. Isenberg was involved in the design of the System/390 Parallel Sysplex. As part of this design work, he was a coinventor on seven key system patents, was a member of the Parallel Sysplex hardware and software design councils, and led the design efforts to achieve continuous availability in the Parallel Sysplex structure. He received a B.S. in electrical engineering from Carnegie Mellon University and an M.S. in electrical engineering from Syracuse University.

George W. Wang *IBM China Research Laboratory, 4/F No. 26, 6th Street, Shangdi Haidian District, Beijing 10085, P.R. China.* Dr. Wang is currently the director of the IBM China Research Laboratory. He joined IBM's Research Division as a research staff member in 1978 at the Thomas J. Watson Research Center in Yorktown Heights, New York. Dr. Wang's research activity has concentrated on operating systems, database management, parallel processing, and distributed systems. He has held several IBM Research management positions and was named to his current position in December 1994. Dr. Wang has been recognized for his research accomplishments and has earned IBM's Outstanding Technical Achievement, Outstanding Contribution, and Outstanding Innovation Awards, and an IBM Corporate Award. Dr. Wang earned his Ph.D. degree in experimental physics from Columbia University in New York City in 1977 and a master of science degree in computer science from the same university in 1978.

Reprint Order No. G321-5641.