S/390 cluster technology: Parallel Sysplex

by J. M. Nick
B. B. Moore
J.-Y. Chung
N. S. Bowen

This paper describes a clustered multiprocessor system developed for the general-purpose, large-scale commercial marketplace. The system (S/390® Parallel Sysplex™) is based on an architecture designed to combine the benefits of full data sharing and parallel processing in a highly scalable clustered computing environment. The Parallel Sysplex offers significant advantages in the areas of cost, performance range, and availability.

Parallel and clustered systems initially found in numerically intensive markets are gaining increasing acceptance in commercial segments as well. The architectural elements of these systems span a broad spectrum that includes massively parallel processors that focus on high performance for numerically intensive workloads[1] and cluster operating systems that deliver high system availability.[2] This paper describes new clustering functions that are implemented by IBM's S/390* processors and OS/390* operating system.

The S/390 cluster (parallel system complex, or Parallel Sysplex*) contains innovative multisystem data-sharing technology, allowing direct, concurrent read/write access to shared data from all processing nodes in the parallel configuration, without sacrificing performance or data integrity. Each node is able to concurrently cache shared data in local processor memory through hardware-assisted cluster-wide serialization and coherency controls. This in turn enables work requests associated with a single workload, such as business transactions or database queries, to be dynamically distributed for parallel execution on nodes in the sysplex cluster, based on available processor capacity. Through this state-of-the-art cluster technology, the power of multiple OS/390 systems can be harnessed to work in concert on common workloads, taking the commercial strengths of the OS/390 platform to improved levels of competitive price/performance, scalable growth, and continuous availability.

In this paper we review the S/390 Parallel Sysplex architecture, its core technology components, and the customer business objectives that shaped the overall system structure. In Part I we discuss the objectives that guided the Parallel Sysplex designers and introduce the technology components of the Parallel Sysplex cluster. Part II presents an overview of the coupling facility (CF) and coupling support facility architectures, and discusses the scalability of the S/390 Parallel Sysplex. A concluding section summarizes the key points contained in the paper.

0018-8670/97/$5.00 © 1997 IBM

PART I

Design overview

This section summarizes the key design points for the S/390 Parallel Sysplex and relevant design rationale. We begin with a set of objectives that guided the overall system structure. This is followed by a description of design benefits derived from its data-sharing capabilities. Then, alternative cluster architecture models are discussed. Finally, Parallel Sysplex technology functions are introduced.

Customer business objectives. One key customer business objective was to reduce the total cost of computing for S/390 systems. There are many examples of systems that use low-cost microprocessors as a building block for a large system. In order to obtain the same cost advantages as these systems, the most dramatic change for S/390 meant replacing the S/390 bipolar processor technology with complementary metal-oxide semiconductor (CMOS) microprocessor technology and clustering multiple systems together to meet aggregate capacity requirements. This strategic decision enabled the S/390 systems to leverage industry-standard CMOS technology for price/performance advantage, both in terms of reduced base manufacturing costs and significant ongoing customer savings in reduced power, cooling, and floor space requirements.

A closely related objective was to provide a commercial platform that would support the nondisruptive addition of the scalable processing capacity, in increments matching the growth of workload requirements for customers, *without requiring re-engineering of customer applications or repartitioning of databases*. Satisfying this objective was critical to the design of the Parallel Sysplex shared-data cluster architecture, which will be discussed later in this paper. Prior to Parallel Sysplex, S/390 customers had been forced to contain the capacity requirements of a workload within the technology limits imposed by the size of the largest single symmetric multiprocessor system available. Workload growth beyond these limits required splitting the workload and repartitioning the database between the nodes—a complex, resource-intensive process not supportive of customer business objectives.

A third key business objective was to address the increasing customer demands for improved application availability, not only in terms of failure recovery, but for the more important reduction of planned outage times. Today, there is less opportunity for planned systems shutdowns in the global economic environment. Here again, meeting this objective was key to the Parallel Sysplex cluster design.

Another key business objective was to protect investments customers have in existing applications. There were two aspects to this objective. First, the Parallel Sysplex technology had to be introduced in a compatible manner with existing applications. Second, the benefits of parallel processing had to be transparently applied to applications through exploitation of the technology by application subsystems and database managers. With few exceptions, these objectives have been met. The Parallel Sysplex technology extensions to the S/390 architecture (introducing new CPU instructions, new channel subsystem technology, etc.) are fully compatible with the base S/390 architecture. The IBM subsystem transaction managers in the Customer Information Control System (CICS*) and the Information Management System (IMS*), and the key subsystem database managers such as DATABASE 2* (DB2*) and IMS-DB, have exploited the data-sharing technology while preserving their existing interfaces.

A final objective was to logically present a single-system image to users, applications, and the network, and to provide a single point of control to the systems operations staff. Meeting this objective was key to controlling the overall cost of managing a multisystem configuration. In a Parallel Sysplex environment, many cluster technology components, both hardware and software, have been developed to meet this objective. New data-sharing technology hardware enables multiple-system nodes to serve common workloads with the appearance of a single large computing resource. Base operating system cluster services[3–5] provide robust intersystem communication, system monitoring, and automatic failure takeover mechanisms. Shared consoles are provided for managing multiple operating systems and multiple underlying hardware system nodes with a single point of control. Key system profiles, catalogs, and other resources can be shared across the clustered systems to enable efficient "cloning" of system definitions. Through these and other means, systems management costs do not increase linearly as a function of the number of systems in the sysplex. Rather, total cost of computing efficiencies of scale accrue through the coordinated management facilities of the Parallel Sysplex cluster.

Data-sharing design benefits. Given the customer business objectives outlined above, the Parallel Sysplex shared-data architecture and technology was critical to delivering the following system benefits: dynamic workload balancing, continuous availability, and continuous operations.

Dynamic workload balancing. A key aspect of being responsive to changing business needs in a commercial parallel processing environment involves the ability to dynamically adjust system resources to best satisfy workload performance objectives in terms of throughput and response times. In the S/390 Parallel Sysplex environment, the high-performance data-sharing technology provides the means for OS/390 and its subsystems to support dynamic workload balancing across the collection of systems in the configuration. Functionally, workload balancing can occur at two levels. During initial connection to the cluster, clients can be dynamically distributed and bound to server instances across the set of cluster nodes to effectively spread the workload. Subsequently, work requests submitted by a given client (such as transactions) can be executed on any system in the cluster based on available processing capacity. The work requests do not have to be directed to a specific system node due to data-to-processor affinity, which is typically the case with alternative data-partitioning parallel systems, wherein buffer coherency and serialization controls are not cluster-wide in scope. In a Parallel Sysplex cluster environment, work will normally execute on the system on which the request is received, but in cases of "over-utilization" on a given node, work can be directed for execution on other less-utilized system nodes in the cluster. For both on-line transaction processing (OLTP) and decision-support workloads, dynamic workload balancing across systems can be made transparent to the customer applications or users.

Continuous availability. Within a Parallel Sysplex cluster it is possible to construct a parallel processing environment with no single points of failure. Parallel Sysplex hardware components such as sysplex timers and coupling facilities (to be discussed in detail later) can be redundantly configured. The sysplex timer serves as a common time reference source for systems in the sysplex, distributing synchronizing clock signals to all nodes. The coupling facility (CF) is the key Parallel Sysplex technology component providing state-of-the-art cluster data-sharing functions. If a coupling facility fails, critical data contents can be "rebuilt" into an alternate CF under OS/390 system and subsystem control. Since all systems in the Parallel Sysplex can have concurrent access to all critical applications and data, the loss of a system due to either hardware or software failure does not necessitate loss of application availability. Peer instances of a failing subsystem executing on remaining healthy system nodes can take over recovery responsibility for resources held by the failing instance. Alternatively, the failing subsystem can be automatically restarted on still-healthy systems using automatic restart capabilities to perform recovery for work in progress at the time of the failure. While the failing subsystem instance is unavailable, new work requests can be redirected to other data-sharing instances of the subsystem on other cluster nodes to provide continuous application availability across the failure and subsequent recovery.

Continuous operations. The same availability characteristics associated with handling unscheduled outages are applicable to planned outages as well. A system can be removed from the Parallel Sysplex for planned hardware or software reconfiguration, maintenance, or upgrade. New work can be dynamically redistributed across the remaining set of active systems. Once the system is ready to be brought back on line, it can be reintroduced into the sysplex in a nondisruptive manner and participate in dynamic workload balancing as described earlier.

New system nodes can be introduced into the Parallel Sysplex in a similar fashion. That is, the already-running systems continue to execute work concurrent with the activation of the new system node. Once the new system is active, it can become a full participant in dynamic workload balancing. New work requests are naturally driven at an increased rate to that system until its utilization has reached steady state with respect to the demand for overall processor resources across all system nodes in the Parallel Sysplex. This capability eliminates the need to shut down the entire cluster to repartition the databases and retune workloads for each system to distribute work evenly after introduction of the new system into the configuration, as is typically required with a data-partitioning parallel processing system.

A further design objective for the Parallel Sysplex was for new releases of OS/390 and its key subsystems to support the current and the next release migration coexistence. This allows new software product release levels to be rolled through the Parallel Sysplex one system at a time, providing continuous application availability across the systematic migration install process.

Cluster architecture models. Clustering, as a way of organizing computer systems, was surveyed by Pfister[6] who identified a cluster as "a type of parallel or distributed system that consists of a collection of interconnected whole computers and is utilized as a single, unified computing resource." The individual cluster nodes can be either uniprocessor or symmetric multiprocessor (SMP) systems. Although the computers may be connected by a high-speed communication mechanism, they do not share any central (main) storage.

Another viewpoint[7,8] classifies parallel systems based on conformance to one of the following architecture models, each having its own strengths and weaknesses: the shared-nothing model, the shared-disk model, and the shared-everything model.

The shared-nothing (data-partitioning) model. Each system owns a portion of the database, and each portion can only be read or modified by the owning system. Data partitioning enables each system to locally cache its portion of the database in processor memory without requiring cross-system communication to provide data access concurrency and coherency controls. Scalability characteristics are excellent with this approach.

However, there are limitations imposed in a commercial processing environment by such a design point.[9,10] Significant capacity planning skills and cost are required to tune the overall system to match the processing capacity for each cluster node with the projected workload access rate to data owned by that node. Real-time workload demand fluctuations can over- or under-utilize processor resources. Repartitioning of the cluster databases to introduce new cluster nodes for additional capacity requires the entire cluster to be shut down.

The shared-disk (shared-data) model. All of the disks containing databases are accessible by all of the systems. The basic strength of this approach is that it allows a workload to be dynamically balanced across nodes of a cluster, which also has potential benefits for availability and continuous operations, as discussed earlier. However, the major drawback to shared-data models prior to the Parallel Sysplex architecture has been poor scalability characteristics.

In shared-data configurations, distributed lock management protocols are employed to provide concurrency (serialization) controls across the cluster, generally involving message passing between the systems on mainline paths to obtain and release locks. This is necessary to ensure that only one system is allowed to modify a given shared-data item at a time. Global (cluster-wide) buffer coherency controls are required in order to ensure that the currency of data items cached in local buffers in the local processor memory for each system can be determined prior to buffer reuse.

One approach to a shared-disk architecture employs broadcast-invalidate mechanisms to provide coherency control, sending cross-invalidate signals to all other nodes whenever a system updates a copy of a shared-data item locally. This is done to inform the other nodes that their locally cached copy of the shared-data item is now "down level." This approach scales poorly as the number of nodes in the cluster increase. An alternative approach avoids the broadcast-invalidate protocol, by continuing to hold the lock on a valid locally cached data item after the transaction ends. This allows the cached copy to be subsequently reused locally with integrity. Ownership of the lock is released only in the presence of contention from other systems. However, with this approach, only one system can maintain a current local cache copy of a given data item in memory at a time, that is, while the lock on that data item is held. Ownership of the current data item copy transfers or "pings" from one system to another as references to those data are made.

The S/390 Parallel Sysplex architecture is characterized as a shared-data model.

Regardless of the global coherency protocols used, these cross-system "ping" effects occur whenever a system determines that it does not have a current copy of a needed shared-data item. This typically results in the data being pushed out to shared disk by the system in the cluster owning a current copy, where the data are then fetched by the requesting system node. These multisystem data transfer I/Os can cause significant performance degradation in the cluster if a high degree of multisystem interest in the shared data is present.

The shared-everything model. Central storage, as well as disks, are shared by all of the processors. This approach is used in structuring an SMP. An SMP is not a clustered system by itself, but can serve as the system building block for individual nodes of a cluster. Shared-everything architectures have processing efficiency advantages when applied across a relatively small number of processors, but do not generally scale well as the number of processors increases. Also, single points of failure compromise the availability characteristics of the processing system.

A more detailed comparison of alternative cluster architectures with respect to performance and scalability is discussed in Reference 10.

Parallel Sysplex cluster technology. The S/390 Parallel Sysplex architecture is generally characterized as a shared-data model. Its fundamental distinguishing characteristic over traditional shared-disk architectures is that the Parallel Sysplex technology enables multiple systems to cache the same data concurrently in local processor memory with full read/write access control and globally managed cache coherency, *with high-performance and near-linear scalability.*

Specialized hardware and software cluster technology is introduced to address the fundamental performance obstacles that have traditionally plagued data-sharing parallel-processing systems. The core hardware technologies are embodied in the CF (for data sharing) and the coupling support facility (for communication between processors and the CF) components of the system and are discussed in detail later in this paper. Some of the most critical functions provided are outlined below:

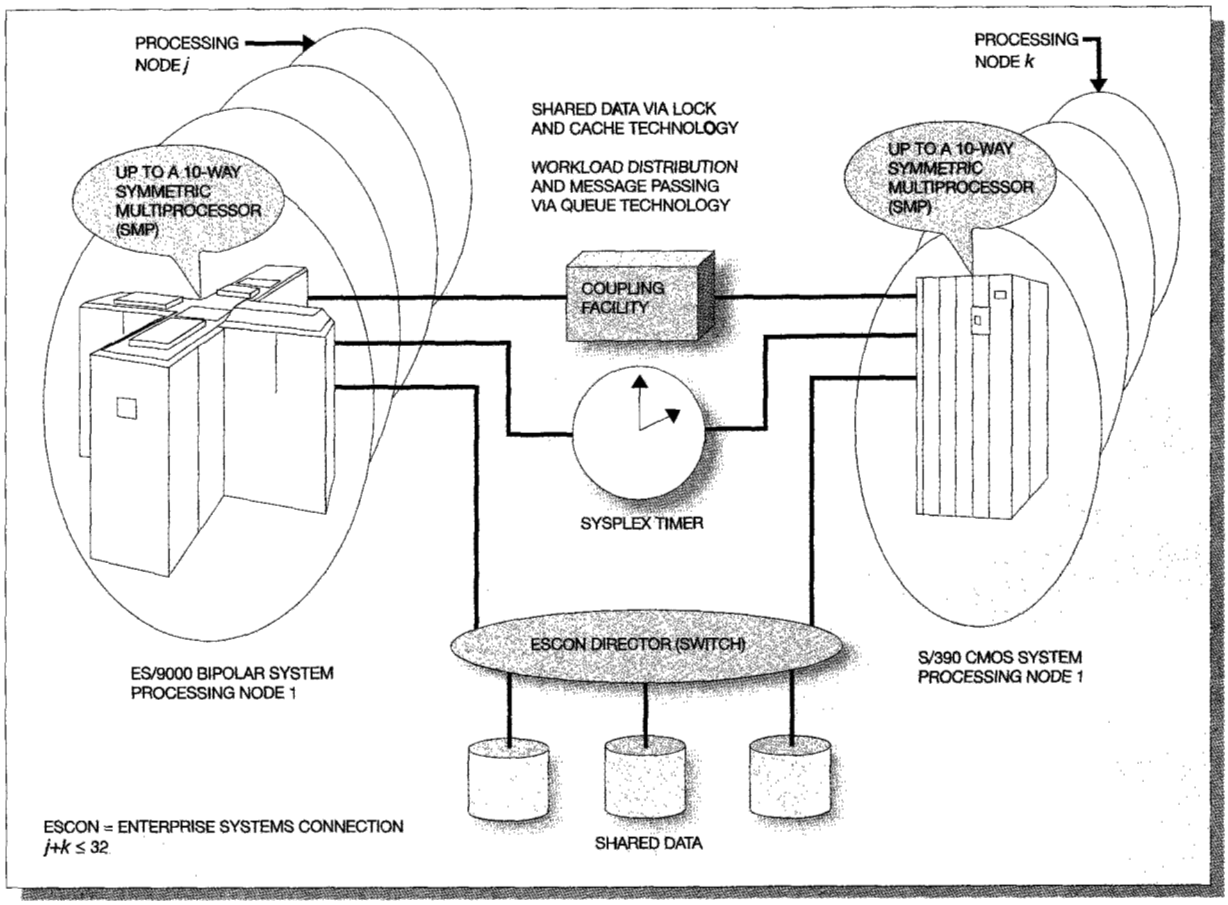
- Hardware-assisted global concurrency controls. Specialized hardware is provided to support low-overhead, fine-grained global lock management with hardware-assisted lock contention detection. In the absence of lock contention, locks can be efficiently granted and released without intersystem software message passing.
- Hardware-assisted global buffer coherency controls. The CF and coupling support facilities combine to track the locally cached shared-data items for each system, providing low-overhead mechanisms for global buffer cross-invalidation. The cross-invalidate operations do not involve software message passing, nor do they interfere with normal processor instruction execution. Cross-invalidate signals are only sent to nodes with registered interest in a data item being updated—not broadcast to all nodes in the cluster. Further, local buffer coherency can be checked by the program buffer managers on each node via new CPU instructions that access local processor memory.
- Synchronous locking and buffer coherency request handling. High-speed, low-latency links using streamlined protocols are provided, allowing locking, caching, and queuing operations directed to a CF to generally be completed instruction synchronously. That is, in certain cases, delaying further CPU instruction processing while the CF executes an operation *CPU-instruction-synchronously* consumes fewer machine cycles than would otherwise be consumed by allowing CPU instruction processing to continue while the CF executes the operation *asynchronously*, thus forcing the software to perform a task switch to suspend and later resume the requesting unit of work (after the CF completes the operation). This can be contrasted with the intersystem software message-passing costs to obtain and release a lock in typical distributed software lock management protocols, or with the several milliseconds that are required for a typical disk operation.
- Global shared-buffer cache. The CF has its own processor memory that can serve as a global cache to enable high-speed local buffer refresh following a local cache miss. The operation to retrieve data from the coupling facility can be performed CPU-synchronously if the requested data item is up to 4 kilobytes in size. Data transfers of up to 64K are performed asynchronous to the initiating CPU. In either case, the cost of a disk I/O or intersystem message passing to "ping" ownership of the data item from one system to another is avoided when the data are resident in the coupling facility.
- Hardware-assisted shared queuing constructs. The CF supports general-purpose data-sharing queuing functions that are applicable for a wide range of cluster-wide uses, including workload distribution, intersystem message passing, and the maintenance of shared control block state information.

S/390 Parallel Sysplex cluster

This section provides an overview of the technical capabilities of the S/390 Parallel Sysplex. It covers the overall system structure, the basic operating system support for parallel processing, and the advanced technology introduced to enable efficient clustering or "coupling" of system nodes.

System model. An S/390 Parallel Sysplex[11,12] is a cluster of interconnected processing nodes with attach-

Figure 1 Parallel Sysplex system model



ments to shared storage devices, network controllers, and core cluster technology components, consisting of coupling facilities, coupling support facilities, and sysplex timers. (See Figure 1.) A coupling facility (CF) enables high-performance read/write sharing of data by applications running on each node of the cluster through global locking and cache coherency management mechanisms. It also provides cluster-wide queuing mechanisms for workload distribution and message passing between nodes. Another component, a coupling support facility, resides on each of the processing nodes and is responsible for communications between the nodes and the coupling facility. A sysplex timer serves as a common time reference source for systems in the sysplex, distributing synchronizing clock signals to all nodes. This enables local processor time stamps to be used reliably on each node and synchronized with respect to all other cluster nodes, without requiring any software serialization or message passing to maintain global time consistency. The synchronized time reference source facilitates real-time or post-processing merges of transaction manager logs across systems, for example, to provide coordinated transaction and database recovery across the cluster for a shared workload.

The Parallel Sysplex currently supports up to 32 processing nodes where each node is a symmetric multiprocessor containing between 1 and 10 processors. The nodes do not have to be homogeneous; that is, mixed configurations supporting both S/390 CMOS processor systems and traditional ES/9000* bipolar systems can be deployed. The basic processor design has a long history of fault-tolerant features.[13] The disks are fully connected to all processors. The I/O

architecture has many advanced reliability and performance features (e.g., multiple paths with automatic reconfiguration for availability). The basic I/O architecture is described in Reference 14 and one aspect of the dynamic I/O configuration is described in Reference 15.

The cluster is organized in this fashion to increase the number of processors that can be applied effectively to large business problems, on-line transaction processing, extensive queries, and applications on different systems that need to concurrently access and update a single database. For example, a cluster with three ten-way SMP nodes can utilize 30 processors to work on a problem, with effective performance increasing nearly linearly with the number of processing nodes.[10,16] On the other hand, if an attempt is made to include more than ten processors in an SMP, incremental effective capacity diminishes rapidly. This is due to increasing interprocessor communication to provide interlocked-update access to memory, processor cache invalidation, and operating system overhead to manage processor resources.

Base OS/390 cluster services. A set of operating system services are provided as building blocks for construction and management of multisystem applications, subsystems, and components. These are described in detail later; here we only briefly cover some of the most relevant aspects.

First, a set of cluster group membership services are provided. These allow processes to join or leave multisystem logical groups, communicate with other group members, and be notified of events related to the group.

Second, the ability to provide efficient, shared access to operating system resource state data is provided. These state data are located on *coupling data sets* and many advanced functions are provided, including serialized access to the data (with special time-out logic to handle faulty processor nodes) and duplexing of the disks containing the state data. In addition, there are availability enhancements for planned and unplanned changes to the coupling data sets (e.g., "hot-switching" of the duplexed disks).

Third, processor "heartbeat" monitoring is provided. In addition to standard monitoring of the health of each node, functions are also provided to automatically terminate a failing node and disconnect the node from its externally attached devices. This enables other multisystem components to be designed with a "fail-stop" strategy (performing peer recovery for a failing node with assurance that the faulty processor does not suddenly resume processing and interfere with recovery of shared resources). This system isolation function is system *fencing* and is exploited by OS/390 as part of *sysplex partitioning* actions. Sysplex partitioning is the term used to describe the set of actions peer systems take to remove another system node from the cluster, including physical isolation, freeing of shared resources, and cleanup of state information related to the system being removed. More information is provided in the section on system fencing.

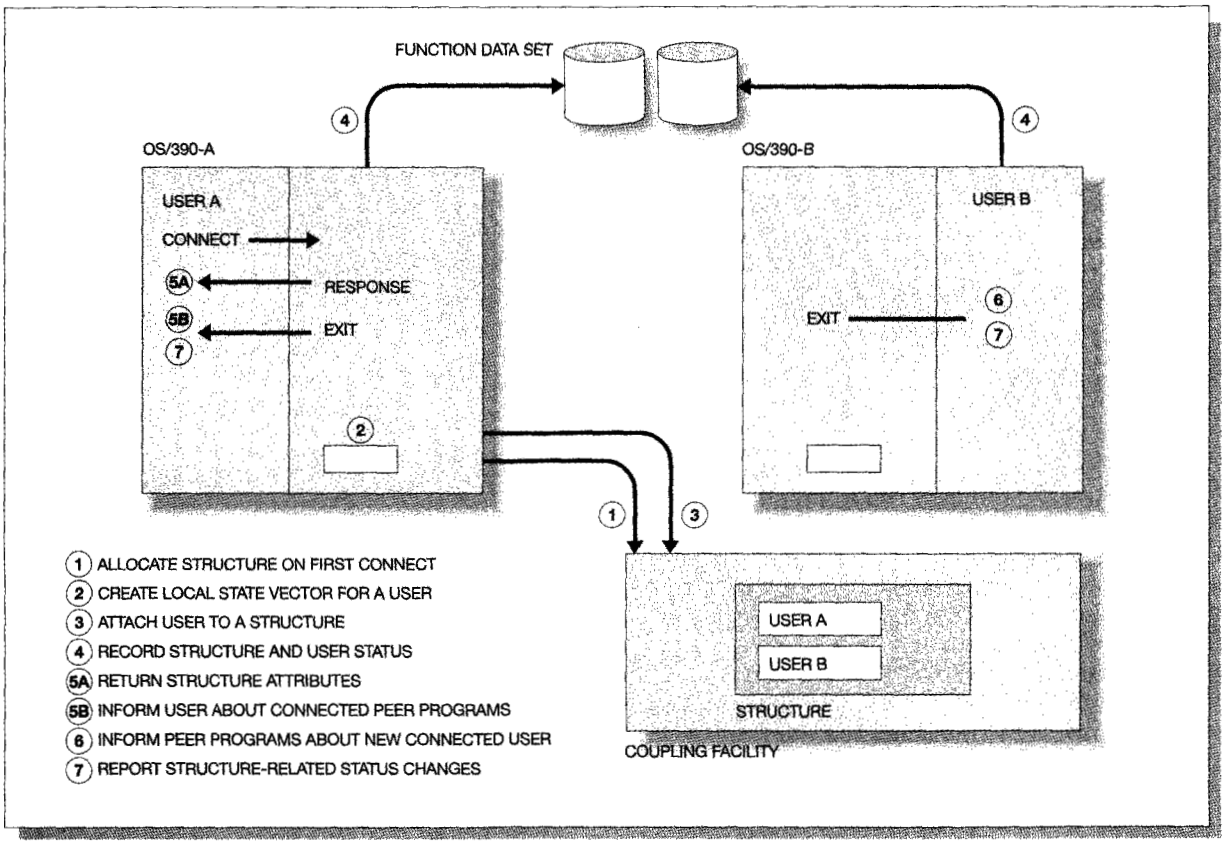
Although the use of multiple interconnected microprocessors can aggregate large amounts of processing power, low cost can only be achieved if the processors are efficiently utilized. Therefore, the ability to dynamically and automatically manage system resources is a key objective. A new component, the workload manager,[17] was designed to meet this objective.

A multisystem automatic restart manager (ARM) facility is provided as a base operating system cluster component. The ARM component is fully integrated in the Parallel Sysplex structure and provides significantly more functions than a traditional "restart" service. First, it utilizes the shared-state support previously described so that at any given point in time the ARM is aware of the state of processes on all systems (i.e., even of processes that "exist" on failed nodes). Second, the ARM is tied into the processor heartbeat functions so that it is immediately made aware of node failures. Third, the ARM is integrated with the workload manager so that it can provide a target restart system based on the current resource utilization across the available nodes. Finally, the ARM contains many features to provide improved restarts such as affinity of related processes, restart sequencing, and recovery when subsequent failures occur. These services are described more fully in Reference 3.

Coupling facility. At the heart of the Parallel Sysplex coupling technology is the coupling facility (CF), a new component providing hardware assists for a rich and diverse set of multisystem data-sharing functions. The coupling facility architecture provides three behavioral models to enable efficient clustering protocols:

- Lock model: supports high-performance, fine-grained global locking and contention detection
- Cache model: provides global coherency controls

Figure 2 CF connection processing



for distributed local processor caches and a high-performance shared data cache

- Queue (list) model: provides a rich set of queuing constructs in support of workload distribution, message passing, and sharing of state information.

Physically, the CF consists of hardware and specialized microcode (control code) that implements the S/390 Parallel Sysplex architecture extensions. The CF control code runs on the latest generations of S/390 processors. CFs are attached to other S/390 processors running the OS/390 or MVS operating system via high-speed *coupling* links. The coupling links use specialized protocols for highly optimized transport of commands and responses to and from the CF. The coupling links are fiber-optic channels providing 100 megabyte per second data transfer rates. Commands to the CF can be executed synchronously or asynchronously to further CPU instruction processing, with CPU-synchronous command completion times measured in microseconds, thereby avoiding the asynchronous execution overheads associated with task switching and processor cache disruptions. Multiple CFs can be connected for availability, performance, and capacity reasons.

Logically, the CF storage resources can be dynamically partitioned and allocated into CF *structures*, subscribing to one of the three defined behavioral models: lock, cache, and queue models. Specific commands are supported by each model and, while allocated, CF structure resources can only be manipulated by commands for that structure type as specified at initial structure allocation. Multiple CF structures of the same or different types can exist concurrently in the same coupling facility.

CF connection processing. A CF structure is allocated when the first attempt is made by a program to connect to that structure by name (see Figure 2). CF al-

location commands (Step 1 in Figure 2) are provided to specify the type of structure to allocate, the amount of storage to assign, and optional structure attributes that depend on the intended usage of the program. The location of the structure (given multiple coupling facilities to choose from) and its size are determined by OS/390 based on customer-supplied coupling facility resource management policy information. As part of the connection request, the operating system creates a local state vector via the DEFINE VECTOR instruction (Step 2), if warranted. Local state vectors are described in the next section on coupling support facility.

The vector token returned by DEFINE VECTOR, which serves as an identifier for the vector, is passed to the CF in an attach command (Step 3). The command establishes a binding between the program and the CF structure; the token is subsequently used by the CF to deliver secondary commands (not shown in the figure) targeting the vector during execution of specific other CF commands. At the completion of the allocation and attach processes, the operating system records information concerning the structure and user status in a function data set (Step 4), returns structure attributes to the requesting program (Step 5A), and informs the program about all current peer programs connected to the CF structure (Step 5B). Other connectors are similarly informed about the presence of the new connector (Step 6). Two of the notifications are presented by OS/390 to user program event exits, which were specified on the OS/390 connection service interface, and which are used to inform programs about any subsequent status changes (Step 7) related to the CF structure. The structure persists as long as there are connectors to it, and can optionally persist even in the absence of any attached program users. Related services and CF commands are provided for disconnect and structure deallocation.

General CF characteristics. In Part II on architecture, the CF models will be discussed in some detail; however, it is worthwhile to introduce some general behavioral characteristics as a frame of reference. The CF supports a number of key functions to facilitate reliable resource management and communication with attached system processing nodes. Some of the functions are:

1. Global commands are provided to control CF resource management and ownership, to ensure that resource management policies are cohesively administered by the systems comprising a single sysplex cluster. Assignment of CF resources by the attached operating system nodes is predicated on authority-based conditional execution of commands requesting resource allocation.

2. A set of pathing commands are provided that enable each attached system to establish reliable communications with an attached CF. Information is exchanged as part of path validation that uniquely identifies the CF and each processing node so that reliable pathing configuration tables can be constructed and reverified across link failures. These mechanisms ensure that commands directed from attached systems to a CF or vice versa (such as cross-invalidate commands) are not inadvertently executed on the wrong target processor due to miscabling of physical links.

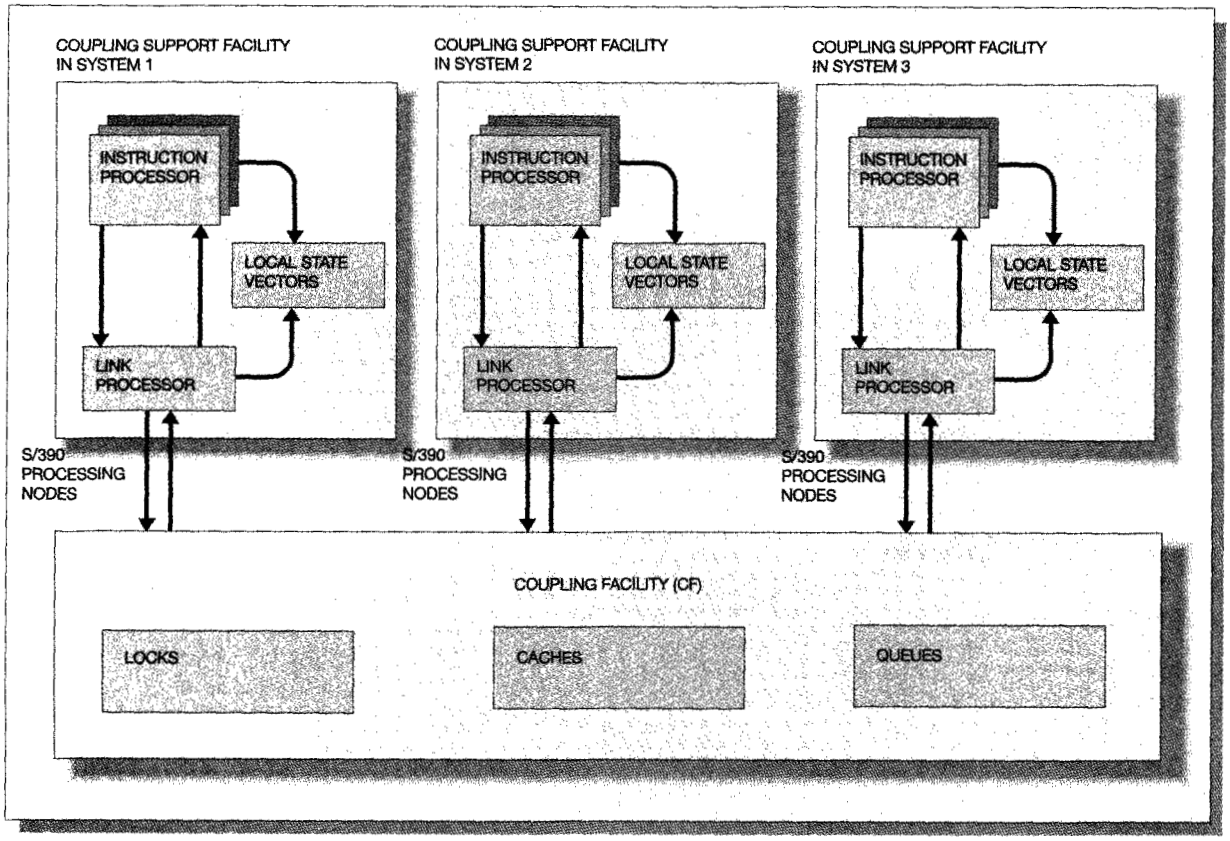
3. Specialized hardware and operating system software protocols are supported to guarantee the integrity of command delivery, even in the presence of link failures, without introducing sympathy sickness across nodes in the cluster.

   Through these link recovery mechanisms, for example, a write command to the coupling facility initiated by a program on one node of the cluster does not have to fail, even if the resultant cross-invalidate signal cannot be delivered to another target node caching a down-level version of the data item. The target system node is guaranteed to observe the fact that its link to the CF was impaired prior to reliance on the integrity of its local state vectors. Upon detection of such a failure, the affected operating system takes recovery actions to cause data-sharing programs, on that node only, to reregister their interest in shared resources with the CF. This is accomplished by over-indicating the invalid state of local cache vectors (or the nonempty state of list-notification vectors) when loss of connectivity is detected.

4. Commands to the CF are executed atomically, i.e., they are completed in their entirety or they are backed out at the CF in the event of failure. They never complete with partial results being stored. This greatly simplifies the recovery logic for systems attached to the CF.

5. Further, this behavior extends to the execution of concurrent commands in parallel at the CF. Partial results of a command execution *are not* observable to other commands while that command

Figure 3 Coupling support facilities



is still in progress. These atomicity properties enable programs connected to the CF to rely on the implicit serialization of command execution. This eliminates the need for programs to obtain explicit multisystem software serialization in order to execute a single command, such as inserting a work element onto a shared queue.

6. While the ensuing discussions focus on one or more systems connected to a single CF, it is generally anticipated that two CFs will be configured to provide redundancy. OS/390 provides a recovery service to exploiting programs to coordinate the repopulation of the contents of a CF structure into an alternate CF, for either failure or planned reconfiguration.

Coupling support facility. Specialized hardware provided on each processing node in the Parallel Sysplex cluster is responsible for controlling communication between the processor and the CF. This specialized hardware is called a *coupling support facility*, as depicted in Figure 3. The coupling support facility consists of new S/390 CPU instructions, high-speed links, and link microprocessors. It also utilizes processor memory to contain *local state vectors*. These vectors are used to locally track the state of resources maintained in the CF. As will be seen, these local state vectors are key to avoiding unnecessary communication between the processing node and the CF to observe critical state information.

The coupling support facility provides several critical functions, discussed next.

Coupling facility command delivery. The coupling support facility provides the means by which a program sends commands to the CF to request that locking, caching, and queuing actions are to be performed. The coupling support facility supports both synchronous and asynchronous modes of command delivery. Synchronous commands are completed at the

end of the CPU instruction initiating the command, based on highly optimized, low-latency transport protocols. Asynchronous commands are completed after the CPU instruction initiating the command is ended, with the completion notice being sent to the operating system via a new notification mechanism that avoids the necessity of raising a processor interruption.

Secondary command execution. The coupling support facility executes secondary commands that are sent by a CF to the processing node as part of performing certain command operations at the CF. With one exception, the secondary commands direct the coupling support facility to update state information in the local state vectors to reflect updated resource status at the CF. A secondary command may, for example, store an invalid-buffer indication at a processing node to signal that the node no longer has the latest version of a locally cached data item.

Local state vector control. The coupling support facility introduces a set of CPU instructions that interrogate and update local state vectors. A DEFINE VECTOR instruction dynamically allocates, deallocates, or changes the size of a local state vector. The vectors are in protected storage and are only accessible via a coupling-support-facility-assigned unique token. This ensures that programs do not inadvertently overlay vectors for which they have no access authority. Instructions are provided to test and manipulate bits in the state vectors conveying the state of associated resources, and are described in the context of their use. There are three kinds of local state vectors used: (1) Local cache vectors are used in conjunction with CF cache structures to track local buffer coherency; (2) list-notification vectors are used with CF list structures to provide notification of CF list empty/nonempty state transitions; and (3) list-notification vectors are also employed by the coupling support facility to indicate the completion of asynchronous command operations. Usage scenarios for each of these types of vectors are described later in sections on cache structures, list structures, and command delivery.

Hardware-assisted system isolation. The coupling support facility also provides a system *fencing* function that isolates a failing system node from being able to access shared external resources during cluster fail-over recovery scenarios. This capability is discussed further later in this paper.

We discuss a detailed architectural review of the coupling support facility functions later in this paper.

PART II

Coupling facility architecture

This section introduces three types of CF storage structures that are used to enable high-performance, highly scalable, read/write data sharing across a Parallel Sysplex cluster. We discuss the features of CF lock, cache, and list structures and outline the software-controlled caching protocols that are implemented using CF cache structures.

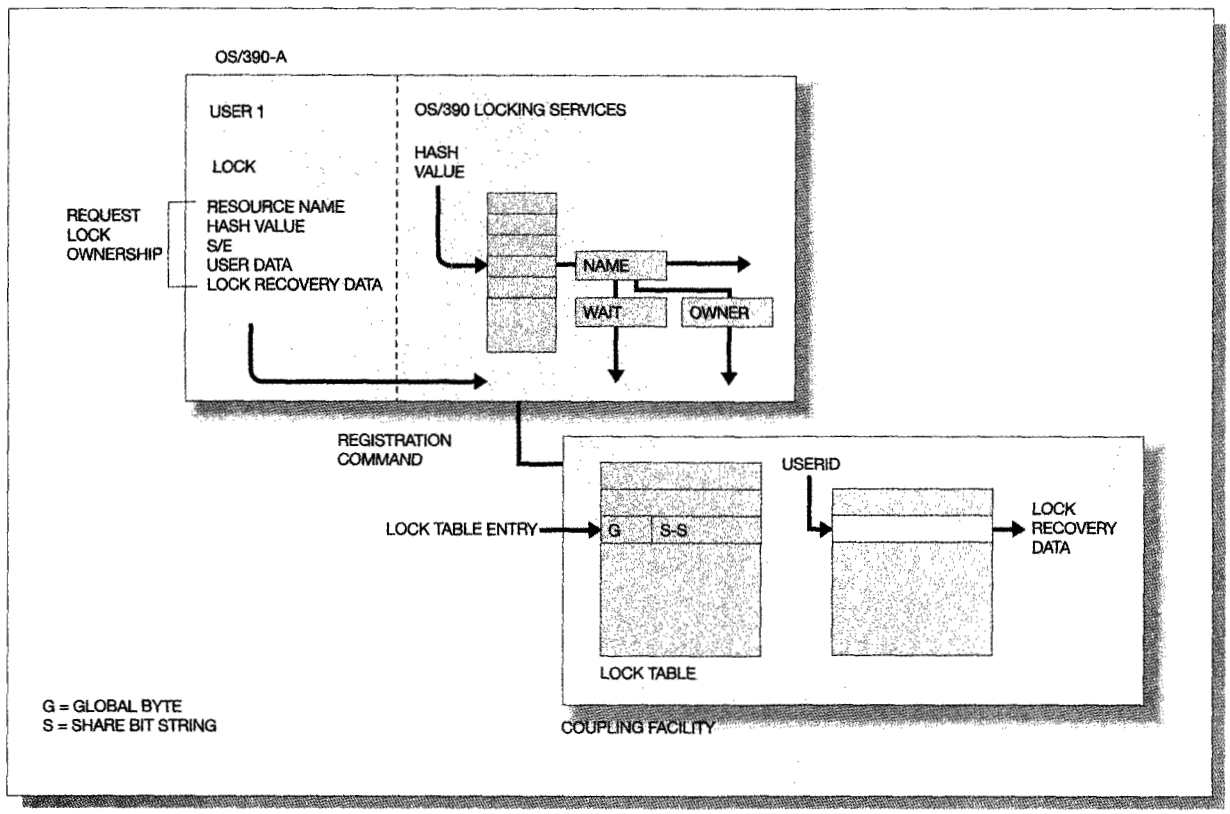
Lock structures. The CF lock model supports high-performance, finely grained lock resource management, maximizing concurrency and minimizing communication overhead associated with multisystem serialization protocols. This model enables a specialized lock manager (e.g., a database lock manager) to be extended into a multisystem environment.

The CF lock structure provides a hardware-assisted global lock contention detection mechanism for use by distributed lock managers, such as the IMS Resource Lock Manager. The lock structure supports a program-specifiable number of lock table entries that are used to record shared or exclusive interest in software locks that map via software hashing to a given CF lock table entry (see Figure 4). Interest in each lock table entry is tracked for all peers connected to the CF structure across the systems in the sysplex. Each entry has a global byte to contain the system identifier of the first system to register exclusive interest in any of the lock resource names that hash to that lock table entry, and a share bit string that identifies, by position, systems that have share interest in that hash class.

OS/390 provides locking services to obtain, release, and modify lock ownership state information for program-specified lock requests. To request lock ownership, a program passes the software lock resource name, the hash class value (to use as the index to the coupling facility lock table entry), the shared or exclusive interest, user data (used to negotiate protocol-specific hierarchical lock ownership states), and program-specified lock information (recorded in the entry for use in recovery processing). If the system does not already have a registered compatible interest in the specified lock table entry, OS/390 will issue a command to the CF to perform the registration.

Through use of efficient hashing algorithms and granular serialization scope, false lock resource conten-

Figure 4 Lock structure



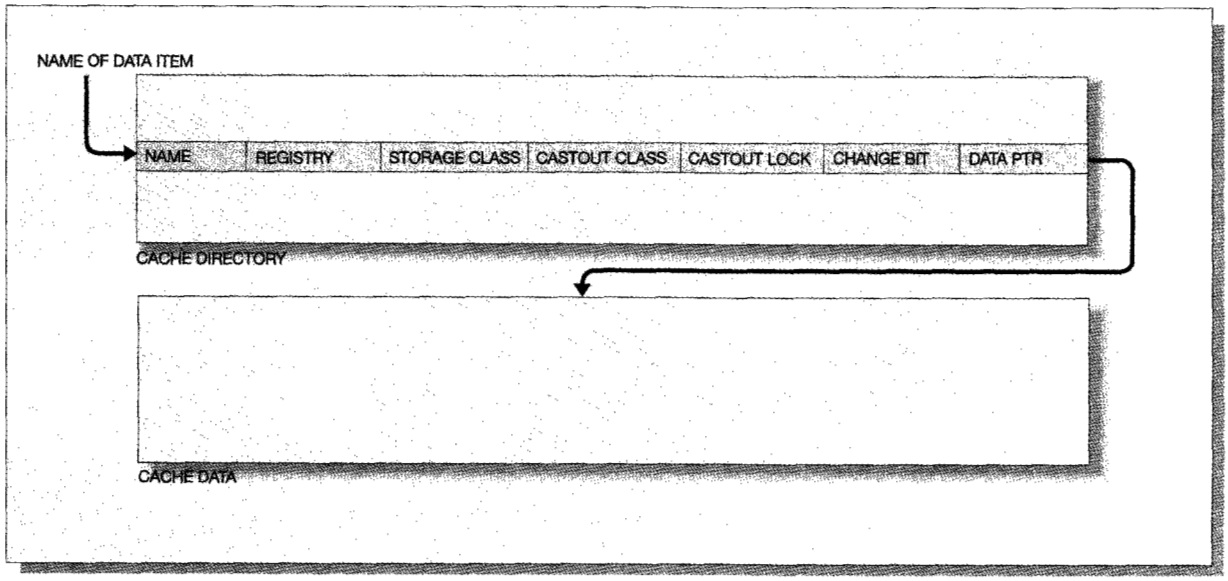
tion is kept to a minimum. This allows the majority of requests for locks to be granted synchronously (CPU-instruction-synchronously) to the requesting system, where synchronous execution times are measured in microseconds. Only in exception cases involving lock contention is lock negotiation required, wherein the CF returns the identity of the systems currently holding locks in an incompatible state with the current request to enable selective cross-system communication for lock negotiation.

OS/390 provides a rich set of cross-system lock management services to coordinate lock contention negotiation and resolution, lock request suspension and completion, and recording of persistent lock information in the CF. In the event of system or lock manager failure, other systems can interrogate the recorded recovery information for the failing system to quickly determine the set of locks held at the time of failure, enabling efficient lock recovery. The CF lock structure and supported protocols are discussed in detail in Reference 18.

Cache structures. A CF cache structure provides the functions needed for multisystem shared-data cache coherency management. The purpose of this model is to enable an existing buffer manager (e.g., a database buffer manager) to be extended into a clustered system environment. It permits each system node to locally cache shared data in processor memory with full data integrity and optimal performance. Additionally, data can be optionally cached globally in the CF cache structure for high-speed local-buffer refresh. As a global shared cache, the CF can be viewed as a second-level cache between local-processor memory and shared disk in the storage hierarchy.

A CF cache structure contains a global buffer directory that tracks multisystem interest in shared-data items cached in one or more system local buffer pools. A separate directory entry is maintained in the CF structure for each uniquely named data item. A directory entry is created the first time a command requests registration of interest in the data item or a write operation is performed to place the data item

Figure 5 Cache entry



into the shared cache. The directory entry contains control information for the data item used in execution of CF commands targeting that entry. For example, the directory entry contains the program-provided unique name of the data item (which serves as the means for finding the directory entry via internal hash on the name on cache structure commands). Also, the directory entry contains a user registry identifying each system that has a valid registered interest in that data item, along with the local cache vector index being used to track the interest each database manager has in the data item cached in its local buffer pool. The directory entry contains an internal pointer to the CF-cached version of the data item if present, as well as a bit indicating whether the data item is cached in a changed or unchanged state with respect to the permanently stored version of the data item on shared disk (see Figure 5).

The CF cache structure architecture was designed to support three basic caching protocols:

- Directory-only cache. A directory-only cache utilizes the global buffer coherency tracking mechanisms provided by the CF, but does not store data in the cache structure. This allows read/write sharing of data with local buffer coherency, but refresh of down-level local copies of data items is via access to the shared disk containing the data item, and all updates are written permanently to disk as part of the write operation.
- Store-through cache. When used as a store-through cache, in addition to the global buffer coherency tracking, updated data items are written to the cache structure as well as to shared disk. The directory entries for these data items are marked as unchanged, since the version of the data in the CF matches the version hardened on disk. This enables rapid buffer refresh of down-level local buffer copies from the global CF cache, avoiding I/Os to the shared disk.
- Store-in cache. When used as a store-in cache, the database manager writes updated data items to the CF cache structure synchronous to the commit of the updates. This protocol has additional performance advantages over the previous protocols as it enables fast commit of write operations. However, here the data are written to the cache structure as changed with respect to the disk version of the data. The database manager is responsible for *casting out* changed data items from the global cache to shared disk as part of a periodic scrubbing operation to free up global cache resources for reclaim. Further, an additional recovery burden is placed on the database manager to recover changed data items from logs in the event of a CF structure failure.

Reclaim processing. The CF cache architecture provides commands and processes to efficiently manage shared cache directory and data resources. Each directory entry in the CF cache (and related data when present) is associated with a program-specified storage class when the directory entry is created. When read or write command references are made to a named data item being tracked or cached in the CF, that entry is marked by the CF as being the most recently referenced entry for the storage class. Directory entries are maintained in the storage class in least-recently-used (LRU) order for purposes of reclaiming unchanged directory and data resources from the cache to satisfy new resource requests. Multiple storage classes in the CF cache allow programs to group data sets being cached according to performance class priority, and commands are provided to direct CF resource reclaim algorithms in accordance with the priorities established for the storage classes.

CF directory and data reclamation for unchanged data items is performed automatically by the CF in response to demand. If it is necessary to reclaim an aged directory entry to satisfy a new request and there is registered interest being actively tracked for one or more connected programs in the targeted entry, cross-invalidate signals are directed to the local cache vectors for those programs to reflect the fact that their interest is no longer being tracked. Note that the CF does not perform reclaim processing for changed data items in the cache structure.

Castout processing for changed data items. To facilitate use as a store-in cache, the CF mechanisms allow efficient retrieval of changed data items from the cache so that they can be written to disk rendering them unchanged and available for subsequent reclaim. The directory entry contains a castout class field used to group changed data items together on common castout class queues (program-specified) so that physically coresident data items can be retrieved and written to the same disk volume in a single I/O operation. Refer again to Figure 5.

Further, each directory entry contains a castout lock that prevents multiple program processes from casting out the same data item to disk concurrently. Failure to provide this mechanism could result in interleaved write updates being cast out to disk out of sequence with respect to the order in which the updates were made to the CF cache entry. The castout lock is set during execution of a read-for-castout command that marks the data item unchanged and returns the data to the program. Note that the data item is *not* eligible for CF reclaim while the castout lock is held. When the program completes the disk I/O, it issues an unlock-castout-lock command to cause the CF to release the castout lock, rendering the data item eligible for reclaim.

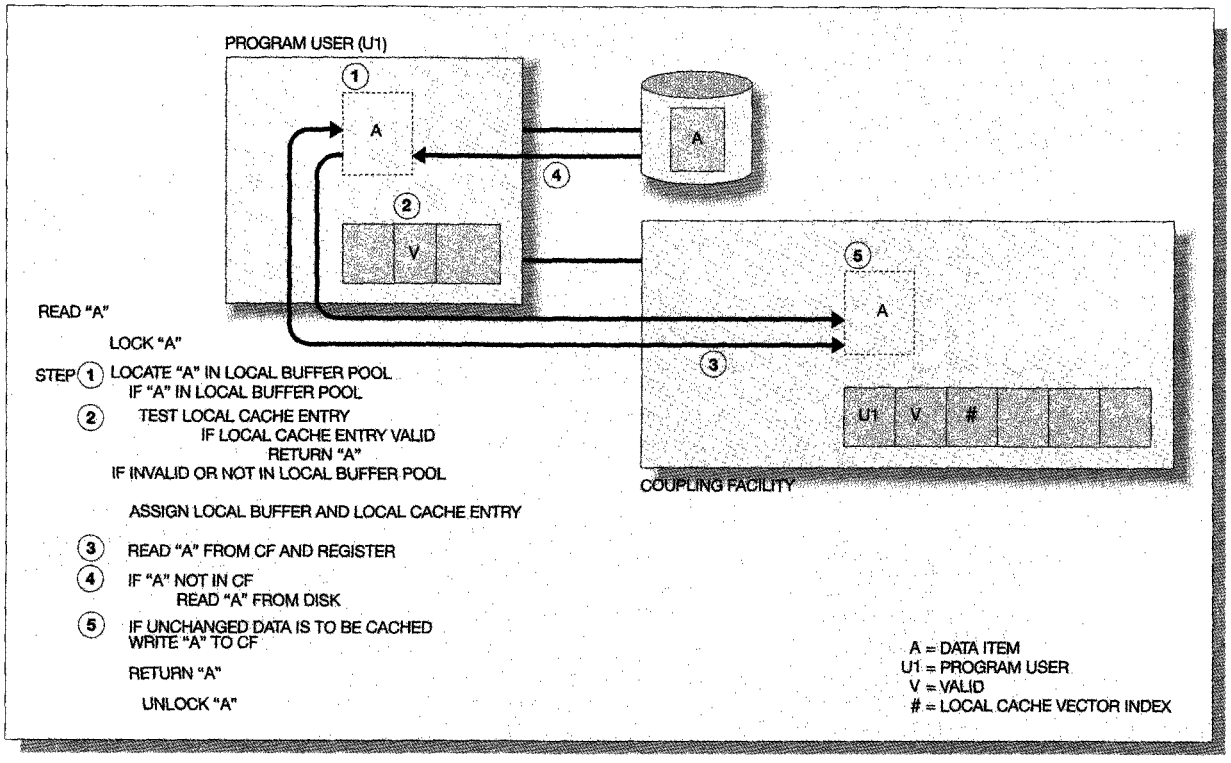
However, it is desirable to allow new write operations to continue to make updates to a CF directory entry concurrent with castout processing for that entry. Thus, the architecture enables writes to an entry to store updated data while the castout lock is held by another program process. Data integrity is preserved by setting the change bit for the entry on again, which will persist when the castout process releases the castout lock (i.e., the data item will not be eligible for reclaim when the castout lock is released).

Reference 19 contains greater detail about these processes and how they relate to exploitation of a coupling facility by IBM's DB2.

Read scenario. In order to describe how the CF supports protocols enabling distributed local caches to maintain coherency with respect to one another, it is best to walk through two scenarios. First a read scenario is discussed, followed by a write scenario.

Refer to Figure 6 for the following discussion. When a database manager, such as IBM's DB2, first connects to a CF cache structure via OS/390 system services, the operating system allocates a local cache vector in protected processor storage on behalf of the database manager. The local cache vector is used to track the coherency of data cached in the local buffer pool. OS/390 passes the local cache vector token to the CF as part of attaching the program user (DB2) to the cache structure, as previously described in the section "coupling facility." The database manager associates each buffer in the buffer pool with a unique bit position in the local cache vector. When the database manager receives a request for access to a data item (named "A" in this scenario), it acquires a lock on the data. The lock may be a global lock obtained through access to a CF lock structure, for example. Next, the program attempts to locate "A" in the local buffer pool at Step 1. If "A" is located, then the currency of the locally cached copy of "A" needs to be determined. This is accomplished using a TEST VECTOR ENTRIES instruction in Step 2, passing the vector token and the local cache vector index for that local buffer as input to the instruction. The TEST VECTOR ENTRIES interrogates the vector in protected processor storage and sets a condition code

Figure 6 Read scenario

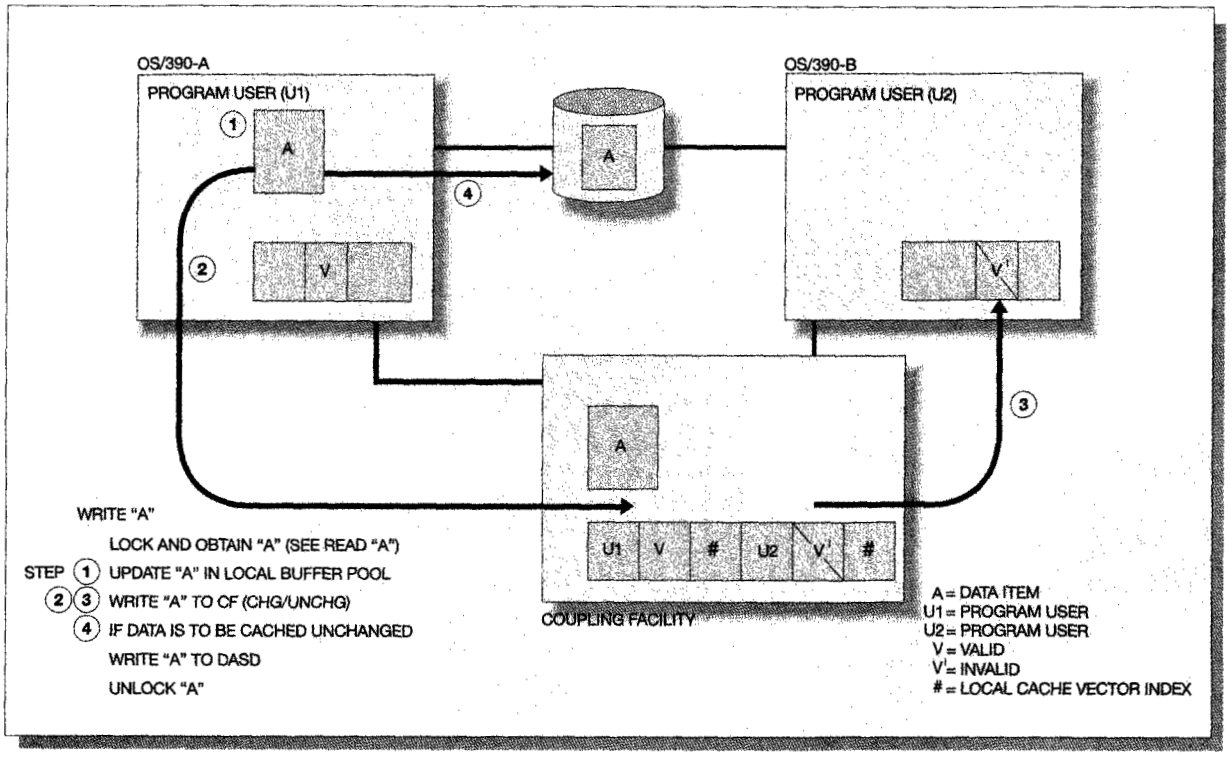


indicating whether the local copy of "A" is valid or invalid (down level). Note that this check is a processor storage reference and involves no communication with the CF. If the locally cached copy of "A" is valid, it is returned to the requestor from the local buffer pool and the lock on "A" is released.

If "A" is not in the local buffer pool or the cached copy was invalid, the program assigns a buffer in the pool to contain the data item. Then, at Step 3, the program issues a read-and-register command to the CF to register its interest in those data with the CF, passing the program-specified data item name and the local cache vector index associated with the local buffer where the data item is being cached. In addition, the program can provide the name of the old data item that was cached in the assigned buffer before it was reassigned to contain "A" as input to the command, for example "B." OS/390, as part of passing the command to the CF, first sets the specified local cache vector bit optimistically to the valid state via a SET VECTOR ENTRY instruction. Upon receipt of the read-and-register command, the CF finds or assigns a directory entry for data item "A" and updates the user registry for the requesting connected program user (U1), saving the local cache vector index and marking the user as having a registered interest in "A." If the data for "A" are present in the CF cache from a prior write operation, the data are returned to the program and stored in the local buffer pool as part of the command execution. Also, if the "old" named data item "B" has a current directory entry present in the cache structure and it still reflects U1 as being validly registered for that data item with the same local cache vector index being tracked, then U1's interest in "B" is deregistered, as the local cache vector index is now being used to track interest in "A." If the read-and-register command fails for any reason, the operating system issues a SET VECTOR ENTRY instruction to reset the target local cache vector bit to the invalid state.

If the CF did not have a copy of the data in its cache, then the program issues an I/O to retrieve the data item from disk at Step 4. If the program desires to place the unchanged data item into the CF cache so

Figure 7 Write scenario



that it may be fetched subsequently for rapid buffer refresh when a local read miss occurs, a write-when-registered or write-and-register command is issued to store the data item at the CF in Step 5. At this point the data item can be returned to the requesting program and the lock on "A" released.

Write scenario. Refer to Figure 7 for the following discussion. Assume here that a request is made to the database manager to update data item "A." As before, the database manager locks and locates "A" in its local buffer pool and tests the validity of the locally cached copy. The program uses the local copy if current or retrieves a current copy if not, as described in the previous section. Then, at Step 1, the program updates the local copy of data item "A."

At Step 2, if it desires to store the updated data in the CF, the program issues a write- when-registered (WWR) or write-and-register (WAR) command to the CF, passing the data and the local cache vector index being used to track interest. If the program intends to write the data to disk as part of a store-through caching protocol, then an indication is specified on the write command to set the change bit as "unchanged" in the directory entry for "A." If the protocol is to use the CF cache as a store-in cache, then the change bit setting is designated to the "changed" state.

The difference between the WWR and WAR commands is that the WAR command will allocate a directory entry for "A" if one is not present and will unconditionally over-write the existing data for "A" if already present in the CF (on the presumption that the program holds an exclusive lock on the data item "A" and knows it has a current copy). The WWR command conditionally performs the write operation only if the writer is currently registered at the CF as having a valid local copy of "A." This capability is important for programs holding a lock on a specific record within data item "A," but not a lock on the entire data item. The validity check at the CF entry on the WWR command ensures that concurrent updates to different records associated with the same data item cached in the CF cannot result in one sys-

tem writing a down-level version of the data item into the CF. Without this validity check, a program could test its local cache vector index contents as being valid and then proceed to update the local copy, missing the cross-invalidate signal issued on behalf of another update to a different record *just after* the test of the local vector bit.

Alternatively, if the CF cache is being used solely to provide cluster-wide buffer coherency tracking as part of a directory-only caching protocol, an invalidate-complement-copies (ICC) command is issued to the CF at Step 2 instead of a write command to cause the cross-invalidate function to be performed without storing data in the CF cache for "A."

At Step 3, as part of execution of the WWR, WAR, or ICC command at the CF, the user registry for "A" is checked to determine whether there are any other connected users who have a valid interest in "A," meaning that they have a locally cached copy of "A" which still reflects the valid state. If so, the CF marks those users as invalid in the user registry and then sends a cross-invalidate command via the coupling links in parallel to those systems having a registered interest in that data item. The CF issues the cross-invalidate command, specifying the local cache vector token and local cache vector index uniquely identifying the specific vector and bit which is to be manipulated on the attached processor node. Specialized coupling link hardware provides processing for buffer invalidation signals sent by the CF to attached systems. The *coupling support facility* link microprocessor receives the cross-invalidate command and updates the CF-specified bit in the data manager's local cache vector to indicate the local copy is no longer valid. This process does not involve any processor interruption or software involvement on the target system. Work continues without any disruption. After the CF has observed completion of all buffer invalidation signals, it responds to the system that initiated the data update process. Again, this entire process can be performed synchronously (CPU-instruction-synchronously) to the updating system, with completion times measured in microseconds.

At Step 4, if the database manager has written the data item to the cache structure as unchanged (store-through) or not at all (directory-only cache protocol), then it will write the data item to disk at this point. This step is bypassed if the CF cache is being used as a store-in cache for fast commit of write updates to avoid incurring disk I/O costs synchronous to mainline program processing.

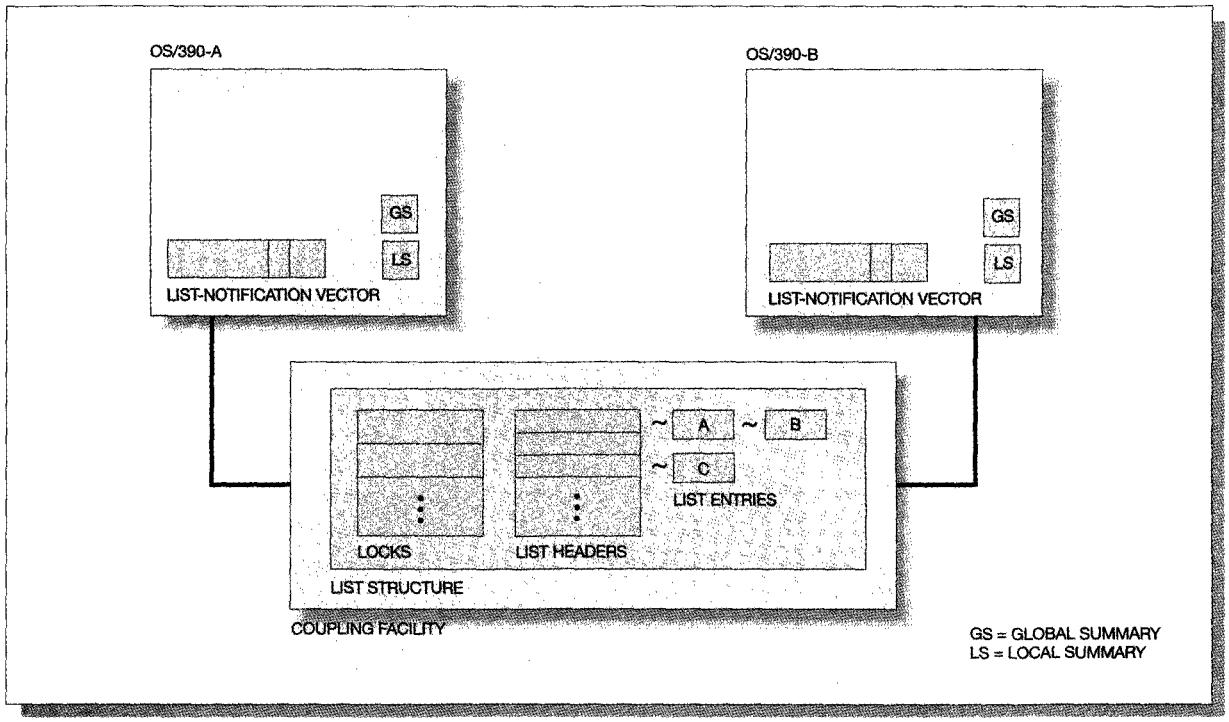
At this point, the issuing database manager is free to release its serialization on the shared-data item.

By exploiting the cache coherency and global buffer cache management mechanisms previously described, it can be seen that the CF and related S/390 Parallel Sysplex cluster technology provide the basis for high-performance, scalable read/write data sharing with integrity across multiple systems, avoiding the message-passing overheads typically associated with data-sharing parallel systems.

Queue (list) structures. The CF queue or list structure supports general-purpose multisystem queuing constructs that are applicable for a wide range of uses, including workload distribution, intersystem message passing, and maintaining shared control block state information. As depicted in Figure 8, a list structure includes a program-specified number of list headers. List structures can support queuing of entries in last in, first out/first in, first out (LIFO/FIFO) order or in collating sequence by key under program control. Individual list entries are dynamically created when first written and queued to a designated list header. List entries can optionally have a corresponding data block attached at the time of creation or subsequent list entry update. Existing entries can be read, updated, deleted, or moved between list headers atomically, without the need for explicit software multisystem serialization in order to insert or remove entries from a list. Compound operations are supported, such as read-and-delete, write-and-move, etc.

Optionally, the list structure can contain a program-specified number of lock entries. When so specified, the structure is referred to as a *serialized list structure*. In the serialized list structure, locks are obtained in an exclusive mode only. The individual locks are solely under software control and do not architecturally map to any other list objects; however, it is common to map a given lock entry to a list header (queue) in the list structure. Lock operations include the ability to obtain ownership of a lock, release the lock, test whether a specific lock is held, and execute a list command only while a given lock is not held. A powerful construct of the list model is the ability to combine a locking operation with a queuing operation to the list structure in a single compound command, using the success of the locking operation as a condition for execution of the queuing action. A common exploitation of the serialized list structure is to request conditional execution of mainline CF commands as long as a specified lock

Figure 8 List structure

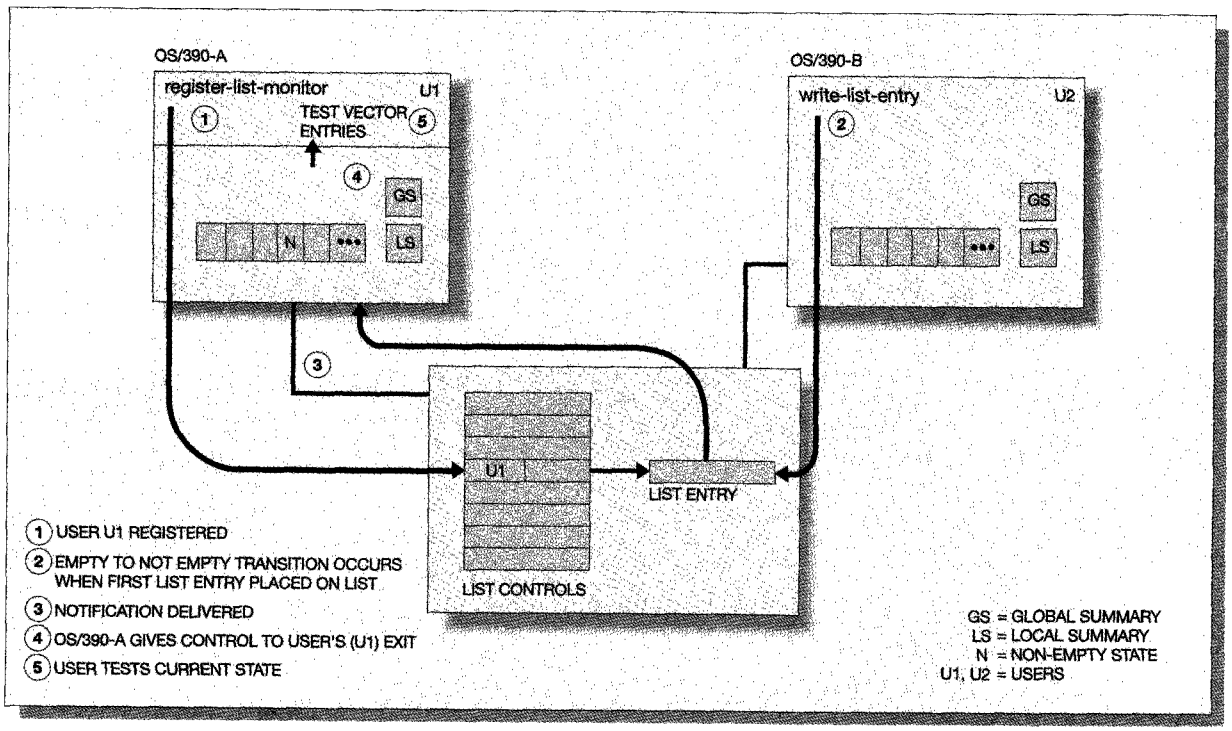


is not held. Recovery operations requiring a static view of a list or the entire structure can set the lock causing mainline operations to be rejected. Such a protocol avoids the necessity for mainline processes to explicitly gain or release the lock for every request, but still allows such requests to be suspended or rejected in the presence of long-running recovery operations. OS/390 supports the ability to either suspend a serialized list request if the requested lock is not available, or to conditionally obtain the lock and return control to the program if the lock is not immediately available.

There are several mechanisms by which a list entry can be accessed, depending on structure attributes specified as part of list structure allocation. Entries can be accessed by a program-provided key, which is also used to queue the entries collated in keyed sequence on a given list. Note that multiple entries of the same key can reside on the same list. Alternatively, list entries can be accessed by a program-assigned name, which is guaranteed to be unique across the list structure when the entry is created. List entries can always be accessed in LIFO/FIFO order from the head or tail of the list. Further, all list entries are created with a CF assigned list-entry identifier (LEID). The LEID is guaranteed to be unique for the life of the list structure and provides a direct means of locating an individual list entry even if it is not otherwise tagged with a key or name.

Each list header in the structure has a set of list controls associated with it. The controls contain threshold values for the number of list entries or data elements that can reside on a given list header, so that a single program user cannot exhaust all of the list structure resources as a "runaway" rogue program. The list controls also contain a list cursor value, which enables multiple concurrent programs on different systems to cooperatively browse through a list. Each program reads the entry adjacent to the last one read by any peer program, without each system having to communicate with respect to the current cursor position within the shared list. The list controls further contain list assignment key controls, whereby the program can seed the initial and maximum key values so entries created on a list can be assigned a generated key in sequence by the CF, without the pro-

Figure 9 List notification



gram having to know the last key assigned on list entry creation by a peer program on another node.

One of the controls, a list authority value (LAU), can be set by a program dynamically and used as a comparative operand on list structure commands directed to the targeted list, causing commands to be rejected if the comparative check on LAU fails. This is a useful mechanism to change list ownership or state with guaranteed failure of any commands issued by peer programs unaware of the changed ownership or state for that list.

Other list structure objects can be atomically compared or replaced as part of list structure command executions to cause conditional execution only if all comparative checks succeed. In addition to the LAU check, execution can be conditional based on successful compare or replace of lock value, list number, or version number. Every individual list entry supports a version number value that is initialized and modified by the programs and can serve as a means of reflecting any list entry state change (such as update of the list entry data contents).

Refer to Figure 9 for the following discussion of list notification. Programs can register interest in specific list headers used as shared work queues or inbound message queues at the CF, for the purpose of being notified when a monitored list becomes nonempty. This provides initiative to the program to issue commands to retrieve list entries that have been placed on the list. The program registers interest in monitoring a specific list via a list structure command, register-list-monitor, passing the list-notification vector index to be used to track interest in that list, as indicated in Step 1 in Figure 9.

When an entry is added to the list causing it to go from an empty to nonempty state, as at Step 2, the CF sends a list notification command indicating an empty-to-nonempty list state transition to registered users at Step 3. The list-notification (LN) vector token (passed in on the initial attach command when the program connected to the list structure) is provided along with the LN vector index on the list-notification command. The command is received by the coupling support facility link microprocessor on the target system and the specified list-notification vec-

tor bit and associated list vector summary bits are updated to reflect the list-state-transition, as will be described next.

Each list-notification vector has a local-summary (LS) bit that indicates the overall contents of the vector as either inactive (all vector bits are set to ones indicating empty list state) or active (at least one bit is reset to zero indicating nonempty list state). There is also one global summary (GS) bit for the processing node; it indicates the overall contents, either inactive (all vectors are inactive) or active (at least one vector is active), for all of the list-notification vectors at the node.

The coupling support facility first sets the specified LN vector bit to the nonempty state. Then the local summary bit for that vector is set to the active state. Finally, the global summary bit for the node is set to the active state. Setting the local summary and the global summary to the active state serves as the means for the operating system to observe the fact that an LN signal has been received; this is detected during normal dispatcher processing when looking for new work units to dispatch.

As with the cache buffer invalidation signal handling, there is no processor interruption, processor cache disruption, or software task context switch caused as a result of processing the list state transition command.

The program steps in polling for list nonempty state transitions are (1) test the global summary, then (2) test the local summary if necessary, and finally (3) test individual vector bits to identify the specific lists that have transition to a nonempty state.

The first test is made by the dispatcher routine of the operating system; if no vectors are active, normal dispatcher processing continues.

Tests of the summary bits use the TEST VECTOR SUMMARY instruction. TEST VECTOR ENTRIES examines bits in a list-notification vector.

Summary bits are placed in the inactive state using the SET VECTOR SUMMARY instruction in response to observing that one or more vectors has been placed into the active state during dispatcher polling. First the global summary is reset. Then the local summary bit is tested and reset if necessary. This is done by the operating system prior to proceeding with testing of the state of individual list vector entries, so as not to lose dispatching initiative for subsequent list-notification events.

Once the operating system has determined that an LN vector has experienced at least one empty-to-nonempty list state transition, it proceeds to drive each target user's list transition exit at Step 4. The user exit routine then executes the TEST VECTOR ENTRIES instruction to determine which lists have entered the nonempty state at Step 5.

Note that when the last entry on a CF list is deleted, list-notification commands signaling a nonempty-to-empty-state transition are sent to registered connected programs. The GS and LS summary bits are not altered as part of a nonempty-to-empty-state transition. The specified LN vector bit is set to indicate the empty state of the list at the CF.

Given a responsive operating system polling means, the above mechanism avoids the undesired overhead of processor interruptions during program execution and the corresponding cache disruption effects that ensue at points in processing where the dispatcher is not intending to preempt the CPU to dispatch another unit of work.

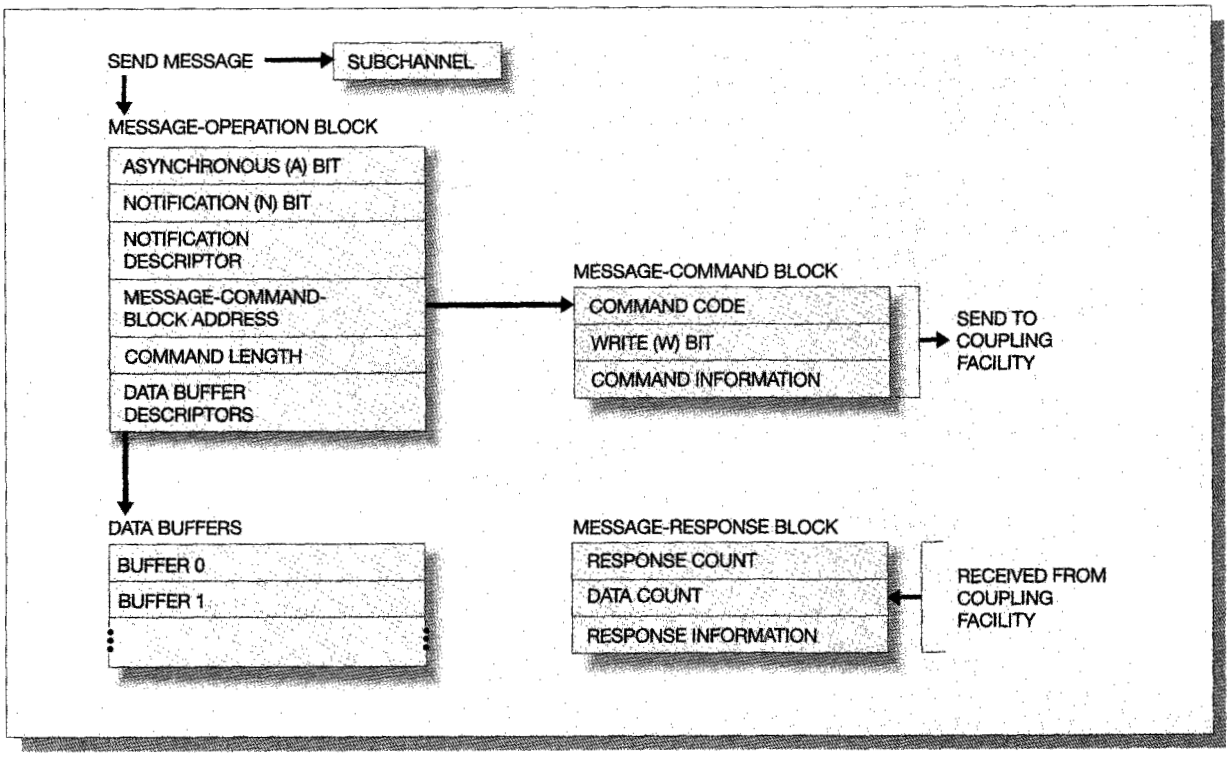
Summary of the CF architecture. From the functions previously described, it can be seen that the CF provides a rich and diverse set of capabilities upon which programs can build efficient, reliable, and scalable protocols for sharing data in a clustered system. Highlighted functions and design characteristics include:

- Global concurrency controls and hardware-assisted lock contention detection
- Global buffer coherency controls for distributed caches
- High-speed shared cache with CPU-synchronous access
- Shared queues for workload distribution and message passing
- Cross-invalidate signal delivery without processor interruption or global broadcast required
- Local processor mirroring of global shared-resource state via local state vectors
- Atomic CF command properties to minimize software serialization requirements and simplify recovery processes

Coupling support facility architecture

This section outlines several aspects of the coupling support facility architecture.

Figure 10 SEND MESSAGE instruction



First, the SEND MESSAGE instruction is described. The instruction is used to deliver a command request to a CF from an attached processor node. Next, links between a coupling support facility and a CF are considered. These links carry command and response information, as well as cross-invalidate and list-notification commands from the CF. Finally, system fencing functions are described.

Command delivery. An exchange of command and response information between a coupling support facility and a CF is called a message operation. It is important to distinguish this mechanism from message-passing protocols between software programs on different nodes of a cluster or communication flows in a networked environment. In the context of this discussion, a *message* is the transport unit for exchanging commands and responses with CF microcode over a high-speed link, with an architecture for the express purpose of supporting efficient data-sharing functions across nodes of the Parallel Sysplex cluster. When an operating system invokes the operation, the command information is specified in main storage; it includes a command code, operands, and output data for a write-to-CF command. Response information is placed in main storage to summarize the results of command execution and input data are also stored for a read command.

The program issues SEND MESSAGE to start a message operation (see Figure 10). The instruction designates a message subchannel and a message-operation block in main storage. The subchannel is associated with a specific CF and identifies the links (there can be several) that may be used for the operation. OS/390 activates as many message subchannels as can be effectively used for parallel execution of multiple CF commands.

After the coupling support facility selects a link for communication, the operation is performed by sending the command to the CF, transferring data as appropriate, decoding and executing the command, formulating a response, and storing response information in main storage. While executing the com-

mand, the CF may send secondary commands to one or more processing nodes.

The message-operation block. Figure 10 illustrates parameters for this operation:

- Asynchronous (A)—When the A bit is one, the message operation is performed asynchronously to continued instruction processing—the SEND MESSAGE instruction is completed before the command reaches the CF. Otherwise, CPU instruction processing is delayed and the entire operation is performed during the execution of SEND MESSAGE.
- Notification (N)—When the N bit is one, the list-notification vector bit designated by the notification descriptor is reset to signal the completion of the operation.
- Message-command-block address and command length—These are the main-storage locations of a coupling command and the number of bytes in the command.
- Data buffer descriptors—These are the main-storage locations and sizes of the data buffers used by the command. The aggregate data area can contain up to 64 kilobytes (KB). The buffer contents are sent to the CF when the write bit in the message-command block is one; the CF returns data to the buffers when the write bit is zero.

The message-command block. This contains information that is sent to the CF:

- Command code—This specifies the command to be performed.
- Write (W)—When the W bit is one, a write operation is performed—information is transferred from the data buffers to a CF structure. Otherwise, a read operation is performed—information is transferred from a CF structure to the data buffers.
- Command information—These are operands that complete the command specification.

The message-response block. This is the destination for information that is returned by the CF. It starts at the location immediately following byte 255 of the message-command block. The following are stored in the block:

- Response count—This is the number of meaningful bytes stored in the message-response block. The count spans information stored starting at byte 0 of the block. The information includes the response count, the data count, and the response field.
- Data count—This is the number of meaningful bytes stored in the data buffers. The data count is zero when the write (W) bit in the message-command block is one.
- Response—This is information summarizing the results of command execution.

Asynchronous vs synchronous operation. In contrast to an I/O operation with a disk or network device, which takes many milliseconds to complete and is always performed asynchronously to continued instruction processing, a coupling support facility message operation is performed synchronously or asynchronously to instruction processing, depending on the option selected by the program.

A general guideline is to use synchronous operation for commands that transfer at most 4 KB of data (not counting the bytes in the message-command block). Most frequently used commands (for example, locking commands), commands that enqueue or dequeue work requests or messages, and commands that read or write 4 KB of data from or to a cache structure, satisfy the guideline.

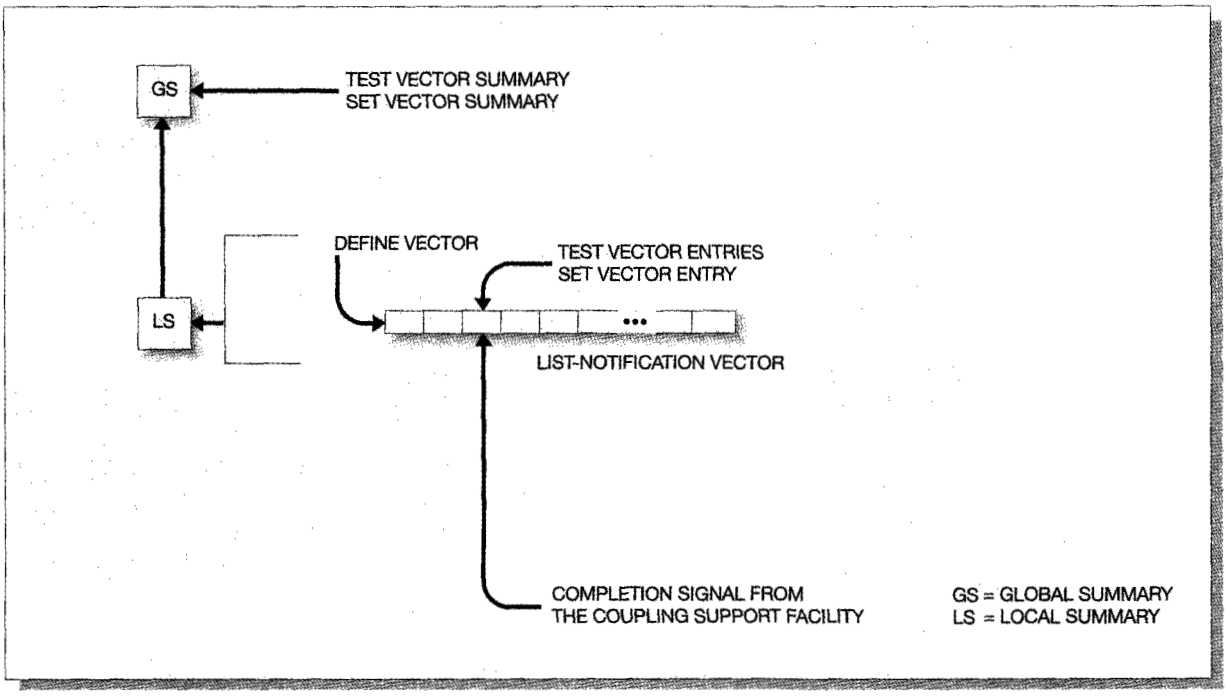
Commands that transfer more than 4 KB of data or are otherwise known to be long-running should use the asynchronous option. Other work can be processed while the command is being executed.

Completion of the message operation. No I/O or other interruption is generated for a message operation. This design reduces processor overhead. For example, an interruption at the end of a disk operation normally stops the processing of a higher priority task, invokes an interruption handler to save the machine state, causes a lower priority work request to be placed on a system queue, results in castouts from caches and translation-lookaside buffers, and restores the old machine state to return to the interrupted task. This disruption is avoided using the techniques described next.

When the program selects the synchronous option for a message operation, control is returned at the end of the operation (end-op) of the SEND MESSAGE instruction with the message operation completed. Status of the operation is then determined as indicated in the condition code for a TEST MESSAGE instruction.

When the program selects the asynchronous option, it can designate a list-notification vector bit that is to be reset when the operation is completed. The

Figure 11 List-notification vector used to indicate completions



operating system tests for completion when, in the normal course of events, it is searching for a new unit of work to dispatch.

Notification of asynchronous message completion. The coupling support facility exploits list-notification local state vectors to signal asynchronous message completions to the operating system. List notification vectors were previously introduced. The operating system establishes a separate completion vector for each CF to which the processor is connected. Each bit in a given vector is associated with a different message subchannel used for communication with that CF. The operating system issues a DEFINE VECTOR instruction to set up a list-notification vector in protected processor storage. The coupling support facility assigns a list-notification token to serve as the name for the vector; the token is used in various CPU instructions and coupling commands. A vector that indicates the completion of message operations is shown in Figure 11.

Each list-notification vector has a local-summary (LS) bit that indicates the overall contents of the vector as either inactive (all vector bits are set to ones) or active (at least one bit is reset to zero).

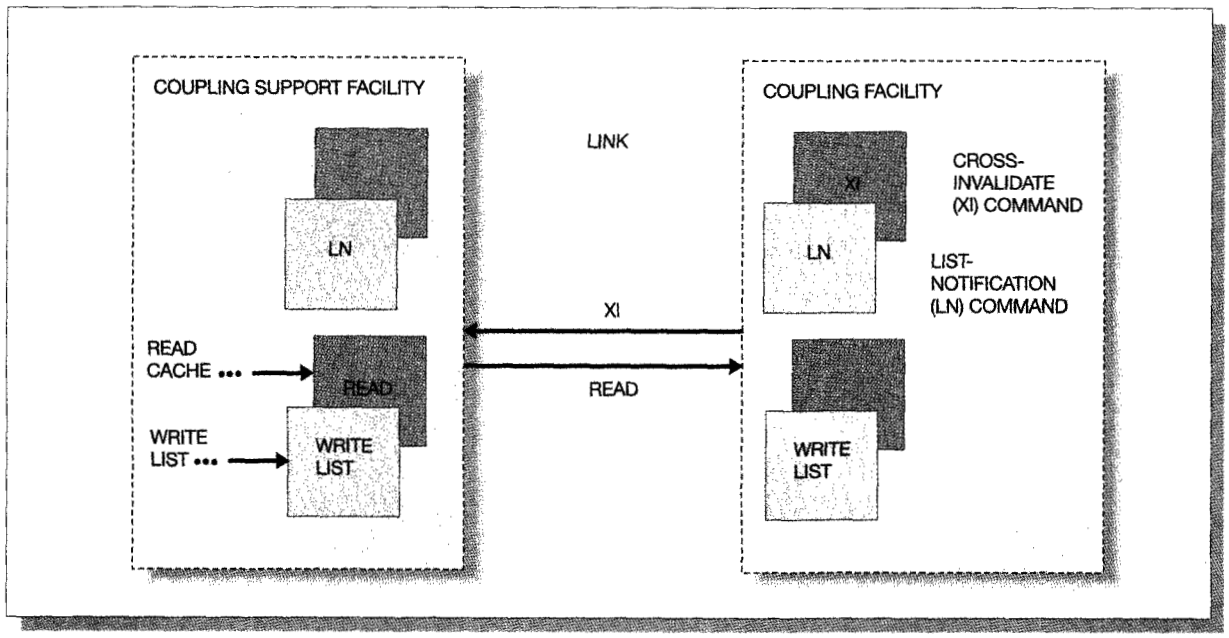
There is also one global summary (GS) for the processing node; it indicates the overall contents, either inactive (all vectors are inactive) or active (at least one vector is active), for all of the list-notification vectors at the node.

The coupling support facility sets the local and global summary bits to the active state after it resets a vector bit to indicate the completion of a message operation.

The program steps in polling for the completion of asynchronous operations are to test the global summary first, then test the local summary if necessary, and finally test individual vector bits to identify the completed operations. The first test is made by the dispatcher routine for the operating system; if no vectors are active, normal dispatcher processing continues.

Tests of the summary bits use the TEST VECTOR SUMMARY instruction. The TEST VECTOR ENTRIES instruction examines bits in a list-notification vector.

Figure 12 Coupling facility link



List-notification vector bits are set using the SET VECTOR ENTRY instruction; this is done by the operating system as part of initiating an asynchronous SEND MESSAGE operation. Summary bits are placed in the inactive state using the SET VECTOR SUMMARY instruction in response to observing that one or more vectors have been placed into the active state during dispatcher polling. This is done by the operating system prior to testing the individual vector bits for completed operations, so as not to lose dispatching initiative. Once the operating system has determined that one or more subchannels have completed execution of a message operation, it proceeds to execute the TEST MESSAGE instruction to observe status for those requests.

As discussed earlier for list notification, the above mechanism avoids the undesired overhead of processor interrupts during program execution and the corresponding cache disruption effects that ensue at points in processing where the dispatcher is not intending to preempt the CPU to dispatch another unit of work.

Links between the coupling support facility and the CF. A connection between a coupling support facility and a CF is called a CF link. The links provide transfer rates of 100 megabytes per second with low-access latency.

Each link is arranged to provide two information flows. Information in one flow is sent from the coupling support facility to the CF. Information in the other flow is sent from the CF to the coupling support facility. The information in the flows need not be associated with the same coupling command.

A number of message operations may be executed concurrently on a single link. The operations are split into short intervals of time during which only a segment of information is transferred over the link. The intervals are sequenced in response to demands made by the coupling support facility and the CF.

Buffers at each end of the link contain areas for command information, data, and response information. They are allocated for use on a dynamic basis to compensate for speed mismatches among the link, the coupling support facility, and the CF. Figure 12 shows an example of a CF link with its two information flows. There are four buffers at each end of the link. The figure suggests that one of the write commands, together with the data to be written, have been sent from the coupling support facility to the CF, which

has not yet completed the command. At the same time, one of the read commands has entered a buffer at the coupling support facility, and has just started to cross the link. Both commands were invoked by the operating system.

Independently of the commands sent by the operating system, the CF has sent secondary list-notification (LN) and cross-invalidate commands (XI) to the coupling support facility as part of the execution of coupling commands that were received from other processing nodes (not shown). The LN command is being executed by the coupling support facility link microprocessor. The XI command is "in flight" over the link to the coupling support facility.

A response for each command will be returned when command execution is completed.

System fencing. Key to cluster availability is the means to "failover" applications to a healthy node when the node on which they are running is deemed to be failing. In order to recover resources owned by the failing node, that node has to be reliably known to be in a terminated state so that it can no longer access shared resources. The CF and coupling support facility provide the means to isolate a processor from accessing any shared resources in the cluster (i.e., to "fence" it) so that cluster recovery can take place.

As part of an availability failover protocol, each OS/390 system periodically broadcasts "heartbeat signals" to the other operating systems of the cluster. When signals are missed, indicating that a system has probably failed, a peer system (any of the remaining healthy nodes) assumes recovery responsibility for any resources held by the failing system. However this does not guarantee that the faulty system is actually in a terminated state. It could be in a temporarily hung state or looping-disabled for an excessive period of time. The recovery system must cause the failing system to become isolated from the cluster before it takes recovery actions, which may include completing or backing out transactions for the failing system and releasing its database locks. Then, the workload of the failing system is distributed to other systems.

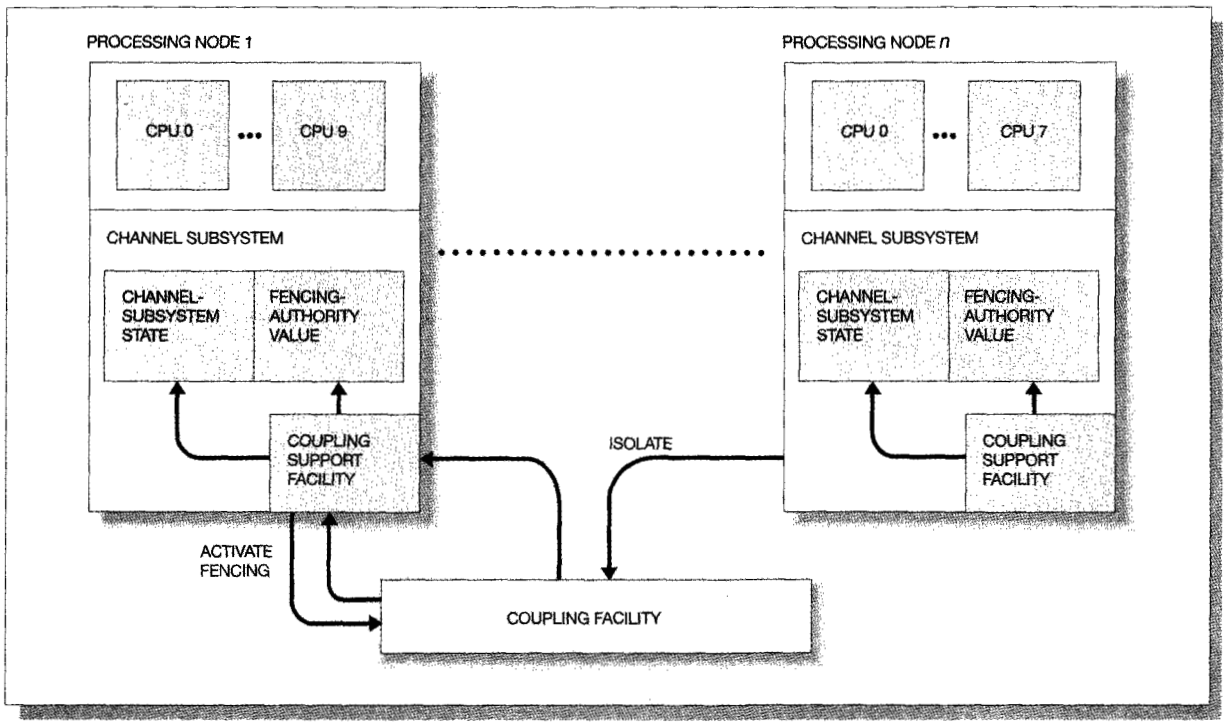
Isolation from the cluster is achieved by establishing a channel subsystem state to screen the I/O and message operations of a processing node. The not-isolated state is set when the node is initialized; when the state changes to isolated, any new I/O or message operations initiated from the isolated node are rejected by its channel subsystem.

Figure 13 shows an isolation scenario. First, an operating system sends an activate-fencing command to initialize the fencing function at its processing node. The command is sent by way of a CF; it stores a nonzero fencing-authority value at the node. The operating system also distributes the authority value to peer systems in the cluster.

When heartbeats are missed for a period of time in excess of a predetermined failure interval, another system in the cluster can take action to partition the failing system from the sysplex. The recovery system issues an isolate command via the SEND MESSAGE instruction to interdict any I/O and message operations attempted by the failing system. The command specifies a fencing-authority test value; it is forwarded by the CF to the coupling support facility at the failing node as indicated on the isolate command.

The coupling support facility executes the isolate command, as follows. When the fencing-authority value at the node is nonzero and matches the fencing-authority test value, the channel-subsystem state is set to isolated and an I/O-termination process is started. A response to the isolate command indicates whether or not all active I/O and message operations have ended; if they have not, the termination process continues at the failing node and the takeover system reissues the command until a response indicates that all operations have ended. If all operations have not completed in a reasonable time period, the recovery system can reissue the isolate command, specifying that the I/O termination process should terminate long-running I/O operations at a channel control word (CCW) boundary. If a program-determined period of time expires again without completion of the isolation process, the recovery system reissues the isolate command specifying immediate termination of any still-outstanding I/O operations. This will terminate any apparently hung CCW operations. In this manner, the system isolation process is executed to allow quiescing of outstanding I/O operations if possible so as to not leave shared resources in an indeterminate state of completion. In addition, the system isolation process causes reset of channel interfaces from the target system so that any serialized state information maintained in shared-disk controllers (such as device reserves, etc.) are released.

Figure 13 System fencing using a coupling facility



Once the response from the isolate command indicates that all I/O operations have been completed or terminated, the failing system has been isolated from the cluster. Resource recovery and workload redistribution can proceed on other systems in the Parallel Sysplex cluster.

Summary of coupling support facility architecture. The coupling support facility architecture provides a set of essential functions in the Parallel Sysplex cluster. They are:

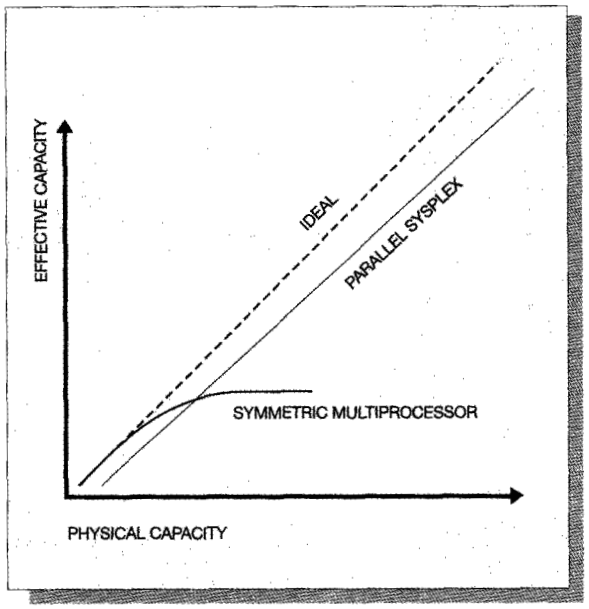
- Efficient command transport for communication with the CF
- CPU-synchronous command delivery and execution
- Asynchronous command completion without I/O interruption
- CPU instructions for manipulation of local state vectors and local tracking of CF resource state to minimize unnecessary signaling traffic between nodes
- System isolation functions to support robust failover protocols

Parallel Sysplex scalability

Figure 14 depicts effective total-system capacity as a function of the number of physically configured CPUs in a processing system. The line labeled IDEAL shows a 1:1 correspondence between physical capacity and effective capacity. That is, as a CPU is added to the processing system, its full uniprocessor capacity would be effectively applied to program execution. Real configurations, of course, do not exhibit this ideal behavior.

The symmetric multiprocessor (SMP) line shows behavior of an SMP as additional CPUs are added to the same single physical system. SMP systems provide maximum effective throughput at relatively small numbers of engines, but as more CPUs are added to the SMP system, incremental effective capacity begins to diminish rapidly, limiting ultimate scalability. This is attributable to the overheads associated with interprocessor serialization, memory cross-invalidation, and communication required in the hardware to support conceptual sequencing of instructions across CPUs, cache coherency, and serialized

Figure 14 Parallel Sysplex scalability



updates to storage performed atomically to CPU instruction execution. These processes are performed in the hardware without the benefit of knowledge of software serialization that may already be held on storage being manipulated at a much more coarse level. In addition, SMP overheads are incurred in the system software due to software serialization and communication to manage common system resources.

The S/390 Parallel Sysplex scalability characteristics are excellent. Physical capacity introduced to the configuration via the addition of more data-sharing systems in the sysplex (where each system can be an SMP or uniprocessor) provides near-linear effective capacity growth as well. Performance studies conducted in a Parallel Sysplex environment consisting of multiple IBM S/390 model 9672 CMOS systems running a 100 percent data-sharing CICS database control facility (CICS/DBCTL) workload demonstrated an incremental overhead cost of less than half a percent for each system added to the configuration. In addition, the initial data-sharing cost associated with the transition from a single-system non-data-sharing configuration to a two-node data-sharing configuration was measured at less than 18 percent.[16]

These results testify to the excellent scalability of the S/390 Parallel Sysplex. This topic is discussed in detail in Reference 10.

Conclusion

Several key design characteristics unfold when considering fundamental properties desired in an ideal large-scale server system capable of handling both current and emerging commercial application workloads. One important attribute is the ability to leverage the power of multiple processors to meet the processing capacity demands of business-critical workloads. This leads to the need to treat these multiple processors as a single large-scale computing resource from several perspectives. Clients of the multiprocessing server want to view the server system as a single node in the network. Applications should be able to be executed seamlessly across the multiprocessing system, accessing processing resources from whichever CPU the application logic happens to reside on. Systems administrators need the ability to manage the multiprocessing system from a single point of control. To maximize system throughput and provide consistent response times to mission-critical applications, it is desirable to be able to direct arriving work requests for execution on any processor having available capacity in a highly responsive and dynamic manner. If the processing compute demands grow and exceed the capacity of the existing server system, it is desirable to add an additional CPU to the existing server system and grow the application workload transparently, without requiring *workload splitting* of customer applications across processors or *repartitioning of databases* to dedicate portions of the database to individual processors of the large-scale server system.

Fundamental to satisfying all of the desired design characteristics outlined is the ability to share data and processing resources across the CPUs of the large-scale server system, without significantly impairing performance in support of resource sharing. This further requires that the multiprocessing server system is designed to provide low-latency, high-performance global serialization controls across its set of CPUs, as well as provide the mechanisms to have multiprocessor coherency controls so that shared data can be cached simultaneously in local processor memory of multiple CPUs with guaranteed coherency properties intact.

Within limits, the symmetric multiprocessor (SMP) is the multiprocessing building block, which has all of these design characteristics, and is in the marketplace today. It has been in existence in various forms in the information technology industry for 25 years,

having evolved considerably in terms of capability and sophistication over that period of time.

Unfortunately, the SMP does have critical limitations that have driven the industry to search for yet a better technology answer. The two fundamental shortcomings of an SMP are its limits in both scalability and availability. As CPUs are added to the SMP, incremental capacity diminishes rapidly beyond a relatively small number of CPUs, due to interprocessor communication in support of concurrency and coherency controls as well as software-related resource management costs. Further, the SMP represents a single point of failure, not only from a hardware perspective, but more significantly from a software view as it runs a single version of the operating system and supported applications.

These shortcomings and the ever-increasing demand for additional processing capacity and improved availability for commercial-processing workloads continue to drive the need to scale capacity beyond the limits of a physical SMP system and exploit multiple system nodes for both scale and availability. This has led to the emerging prominence of clustered systems comprised of multiple SMP or uniprocessor nodes. Clustered systems also offer potential advantages in systems management economies-of-scale given the relative homogeneity of systems within the cluster.

Typically, clustered systems provide high degrees of scalability by partitioning workloads and related databases across the cluster nodes to avoid the need for cross-node buffer coherency and serialization controls, which can significantly compromise scalability beyond a relatively small number of nodes if software-based message-passing mechanisms are deployed to accomplish these functions. However, such "shared-nothing" clustered system environments sacrifice key desired characteristics of an ideal large-scale commercial server in order to meet the scalability and availability objectives. Without data-sharing capabilities characteristic of an SMP server, it is not possible to dynamically balance work based on processor capacity. Nor is it possible, for example, to add a node to the cluster for additional capacity growth without having to split the application or repartition databases, which require a cluster-wide outage.

The S/390 Parallel Sysplex is an advanced commercial processing clustered system, combining many attributes of an SMP in terms of seamless access to multiprocessing resources, with the scalability and continuous availability characteristics of clusters. The Parallel Sysplex supports high-performance multisystem read/write data sharing with local cache coherency, enabling the aggregate capacity of multiple OS/390 systems to be applied against common workloads. This in turn facilitates dynamic workload balancing, maximizes processor utilization, and provides consistent response times. Further, through data sharing and dynamic workload balancing, continuous availability *and* continuous operations characteristics are improved for the clustered system, as nodes can be dynamically removed or added to the cluster in a nondisruptive manner.

The Parallel Sysplex cluster technologies effectively address the overhead issues typically associated with shared-data model architectures, such as: global serialization message-passing protocols, global broadcast cross-invalidate cache coherency protocols, and intersystem "ping" between systems and shared I/O devices. The Parallel Sysplex cluster technologies integrate a comprehensive shared-data architecture model with specialized hardware-assists and optimized software protocols to provide a highly scalable and robust commercial parallel-processing platform.

Key technology functions provided include:

- Global concurrency controls and hardware-assisted lock contention detection
- Global buffer coherency controls for distributed caches
- High-speed shared cache with CPU-synchronous access
- Shared queues for workload distribution and message passing
- Hardware-assisted system isolation for system failover recovery

The Parallel Sysplex cluster is an integral part of the OS/390 platform and is the foundation on which a growing number of new subsystem and operating system enhancements are based. With the maturation of the technology and delivery of sysplex exploitation by the traditional on-line transaction processing and decision support workloads well underway, the Parallel Sysplex focus is shifting to support new application environments, such as commercial parallel Web-server applications, and cluster-enabled object business servers to distributed clients.

The S/390 Parallel Sysplex cluster represents the next step in the evolution of large-scale commercial-processing server systems.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references

1. R. Duncan, "A Survey of Parallel Computer Architectures," *Computer* **23**, No. 2, 5–16 (1990).
2. A. Azagury, D. Dolev, J. Marberg, and J. Satran, "Highly Available Cluster: A Case Study," *Proceedings of the 24th IEEE Symposium on Fault-Tolerant Computing* (June 1994), pp. 404–413.
3. N. S. Bowen, C. A. Polyzois, and R. D. Regan, "Restart Services for Highly Available Systems," *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing* (October 1995), pp. 596–601.
4. M. D. Swanson and C. P. Vignola, "MVS/ESA Coupled-Systems Considerations," *IBM Journal of Research and Development* **36**, No. 4, 667–682 (1992).
5. *MVS/ESA Programming: Sysplex Services Guide*, GC28-1495-02, IBM Corporation (June, 1995); available through IBM branch offices. Chapter 6 describes coupling-facility cache structures, chapter 7 describes list structures, and chapter 8 describes lock structures.
6. G. F. Pfister, *In Search of Clusters: The Coming Battle in Lowly Parallel Computing*, Prentice Hall, Upper Saddle River, NJ (1995).
7. A. Bihde, "An Analysis of Three Transaction Processing Architectures," *Proceedings of the Fourteenth International Conference on Very Large Data Bases* (Los Angeles, CA), Morgan Kaufmann Publishers, Inc., Palo Alto, CA (August, 1988), pp. 339–350.
8. C. Mohan, H. Pirahesh, W. G. Tang, and Y. Wang, "Parallelism in Relational Database Management Systems," *IBM Systems Journal* **33**, No. 2, 349–369 (1994).
9. P. S. Yu and A. Dan, "Performance Analysis of Affinity Clustering on Transaction Processing Coupling Architecture," *IEEE Transactions on Knowledge and Data Engineering* **6**, No. 5, 764–786 (October 1994).
10. G. M. King, D. M. Dias, and P. S. Yu, "Cluster Architectures and S/390 Parallel Sysplex Scalability," *IBM Systems Journal* **36**, No. 2, 221–241 (1997, this issue).
11. *Sysplex Overview—Introducing Data Sharing and Parallelism in a Sysplex*, GC28-1208-00, IBM Corporation (April 1994); available through IBM branch offices.
12. J. Nick, J.-Y. Chung, and N. Bowen, "Overview of IBM System/390 Parallel Sysplex—A Commercial Parallel Processing System," *Proceedings of the 10th IEEE International Parallel Processing Symposium*, Hawaii (April 1996), pp. 488–495.
13. L. Spainhower, J. Isenberg, R. Chillarege, and J. Berding, "Design for Fault-Tolerance in System ES/9000 Model 900," *Proceedings of the 22nd Symposium on Fault-Tolerant Computing* (July 1992), pp. 38–47.
14. S. A. Calta, J. A. deVeer, E. Loizides, and R. N. Strangwayes, "Enterprise Systems Connection (ESCON) Architecture—System Overview," *IBM Journal of Research and Development* **36**, No. 4, 535–552 (1992).
15. R. Cwiakala, J. D. Haggar, and H. M. Yudenfriend, "MVS Dynamic Reconfiguration Management," *IBM Journal of Research and Development* **36**, No. 4, 633–646 (1992).
16. *S/390 MVS Parallel Sysplex Performance*, SG24-4356-01, IBM Corporation (March 1996); available through IBM branch offices.
17. J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive Algorithms for Managing a Distributed Data Processing Workload," *IBM Systems Journal* **36**, No. 2, 242–283 (1997, this issue).
18. N. S. Bowen, D. A. Elko, J. F. Isenberg, and G. W. Wang, "A Locking Facility for Parallel Systems," *IBM Systems Journal* **36**, No. 2, 202–220 (1997, this issue).
19. J. W. Josten, C. Mohan, I. Narang, and J. Z. Teng, "DB2's Use of the Coupling Facility for Data Sharing," *IBM Systems Journal* **36**, No. 2, 327–351 (1997, this issue).

Accepted for publication December 20, 1996.

Jeffrey M. Nick *IBM S/390 Division, 522 South Road, Poughkeepsie, New York 12601 (electronic mail: jeff_nick@vnet.ibm.com).* Mr. Nick is a Senior Technical Staff Member working in the OS/390 System Architecture and Design area. He joined IBM in 1980 as a developer in the S/390 MVS operating system. During his career at IBM, he has held positions in MVS system design and development, and as a large systems technical specialist focused on continuous availability issues. Mr. Nick has lead architecture responsibility for the design of S/390 parallel processing technology and is presently focused on leveraging that technology for new application environments on the OS/390 platform. He is widely recognized as a leading technical expert on S/390 Parallel Sysplex. He received a Corporate Award for his contribution in the design and development for the Parallel Sysplex coupling facility. Mr. Nick currently has 18 patents in the field of operating systems technology and has published several papers in technical journals. He has also given tutorials on the Parallel Sysplex worldwide.

Brian B. Moore *IBM S/390 Division, 522 South Road, Poughkeepsie, New York 12601 (electronic mail: bbmoore@vnet.ibm.com).* Dr. Moore is a Senior Technical Staff Member and member of the IBM Academy of Technology. He joined IBM in 1962, where he has had assignments in processor development, systems architecture, and operating system design. He holds 17 patents and is an inventor on two other patent applications; all are in the area of data processing. He has received a Sixth-Level Invention Achievement Award and two Outstanding Contribution Awards. Dr. Moore received the B.E.E. degree in electrical engineering from Rensselaer Polytechnic Institute in 1961, the M.A. and Ph.D. degrees in mathematics from Syracuse University in 1969 and 1974, and the M.B.A. degree from Marist College in 1981.

Jen-Yao Chung *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: jychung@watson.ibm.com).* Dr. Chung has been with the Thomas J. Watson Research Center, Hawthorne, NY, as a research staff member since June 1989. He currently is the manager of the data intensive computing department and is working on parallel Web server World Wide Web access to database and transaction systems, and parallel systems performance management. His research interests include Web server, database performance, parallel processing, job scheduling and load balancing in real-time systems, object-oriented programming environments, and operating system design. He has published papers in these areas and filed two patent applications. Dr. Chung received the B.S. degree in computer science and information engineering from National Taiwan University in 1982, and the M.S. and Ph.D. de-

grees in computer science from the University of Illinois at Urbana-Champaign in 1986 and 1989, respectively. He has received an IBM Technical Achievement Award, a Research Division Technical Group Award, a Research Division Award, and one IEEE Outstanding Paper Award. He served as industrial chair, program committee member, and session chair in several workshops and conferences. Dr. Chung is a senior member of IEEE and a member of ACM.

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 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 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.

Reprint Order No. G321-5640.