Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers

by J. Jann
L. M. Browning
R. S. Burugula

A logical partition in an IBM pSeries™ symmetric multiprocessor (SMP) system is a subset of the hardware of the SMP that can host an operating system (OS) instance. Dynamic reconfiguration (DR) on these logically partitioned servers enables the movement of hardware resources (such as processors, memory, and I/O slots) from one logical partition to another without requiring reboots. This capability also enables an autonomic agent to monitor usage of the partitions and automatically move hardware resources to a needy OS instance nondisruptively. Today, as SMPs and nonuniform memory access (NUMA) systems become larger and larger, the ability to run several instances of an operating system(s) on a given hardware system, so that each OS instance plus its subsystems scale or perform well, has the advantage of an optimal aggregate performance, which can translate into cost savings for customers. Though static partitioning provides a solution to this overall performance optimization problem, DR enables an improved solution by providing the capability to dynamically move hardware resources to a needy OS instance in a timely fashion to match workload demands. Hence, DR capabilities serve as key building blocks for workload managers to provide self-optimizing and self-configuring features. Besides dynamic resource balancing, DR also enables Dynamic Capacity Upgrade on Demand, and self-healing features such as Dynamic CPU Sparing, a winning solution for users in this age of rapid growth in Web servers on the Internet.

One of the cardinal features of an autonomic component in an information technology (IT) infrastructure is the ability of the component to adapt itself smoothly to changes in its environment. Endowing a computing system with this self-management feature often translates to the implementation of self-protecting, self-healing, self-optimizing, and self-configuring algorithms and subcomponents. Because the primary role of an operating system (OS) is to manage the physical resources of a computer system so as to optimize the performance of its applications (including middleware, which consists of applications from the perspective of the OS), an OS supporting autonomic computing[1] needs to handle the changes in the amount of physical resources allocated to it in a smooth fashion. Some of the most prominent physical resources of an OS are processors, physical memory, and I/O devices.

The current tendency among the noncommodity symmetric multiprocessor (SMP) system vendors is to develop systems that are increasingly large in terms of the number of processors, number of I/O slots, and memory size. Although advances in the design of hardware continue to provide rapid increases in the sizes of these physical resources, a number of major applications and subsystems often lag behind in scalability; hence, the trend in high-end SMPs is to support partitioning of large SMPs and to use these systems for effective server consolidation. Partitioned SMPs typically come in two kinds: systems with physical partitions (PPARs) and systems with logical partitions (LPARs). In a physically partitioned system, the granularity of partitioning is typically coarse, because the partitioning occurs at physical boundaries such as system boards. In a logically partitioned system, the granularity of partitioning is typically much more fine-grained, such as a single CPU or even a fraction of a CPU, a small block of memory, or an I/O-slot instead of an entire I/O-bus. Hence a given SMP can be subdivided into many more LPARs than PPARs.

IBM first provided LPAR support in the Advanced Interactive Executive (AIX*) operating system with the introduction of the pSeries* 690 system in December 2001. This first release of LPAR support was static in nature, that is, the reassignment of a resource from one LPAR to another LPAR cannot be made while AIX is actively running, and both the donor LPAR and the receiver LPAR must be rebooted to enable a reassignment. For such a system to provide support for various resource-related autonomic computing features, such as dynamic resource balancing across LPARs, Capacity on Demand, Dynamic CPU Sparing, and hot swapping, it needs to augment the static partitioning capabilities with dynamic LPAR (DLPAR) capabilities. As of 2002, the pSeries 690 supports the dynamic reassignment of resources across LPARs running AIX. In AIX, this functionality is referred to as dynamic reconfiguration (DR). Since AIX is an enterprise UNIX** operating system that has been designed to be robust, high in performance, rich in functions and support of platforms, and hence monolithic, the addition of a valuable autonomic computing feature such as DR has to be carefully morphed into the existing semantics, code base, and structural organization of the operating system. These challenges found in adding autonomic computing capabilities are encountered by most of the large systems with a significant installation base. Later in this paper, we briefly describe the design of DR in AIX and show that with carefully developed designs, autonomic computing capabilities can be added to an enterprise quality OS, while preserving its performance, semantics, and structural organization. Besides describing the designs within AIX that enable the smooth migration of physical resources, we also describe how these designs are being exploited to provide a variety of valuable autonomic computing features to an IT establishment.

Autonomic benefits of DLPAR. DLPAR in a pSeries 690-AIX system offers a great deal of flexibility to users, allowing resources to be shifted to where they are most needed without impacting system availability. The DLPAR technologies that have been developed provide the basic building blocks on which many self-optimizing, self-configuring, self-protecting, and self-healing features of the system are built. These features enable the implementation of autonomic system management and goal-oriented policies to optimize the performance and usage of system resources. DR also improves the levels of resource utilization and the reliability and serviceability (RAS) characteristics of the SMP, that translate into real cost savings for the IT establishment.

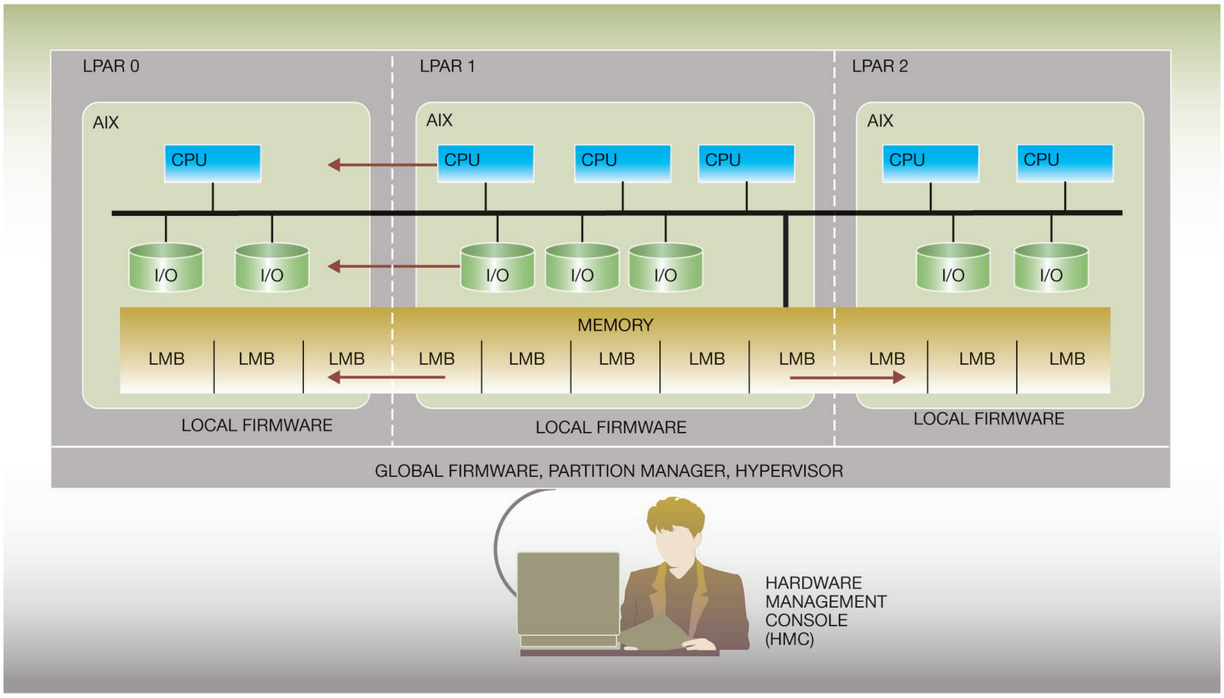
Some of the benefits offered by an SMP with LPAR capabilities are:

1. Servers can be consolidated by simply placing the workloads of several smaller servers into separate LPARs of a big SMP, hence reducing and unifying systems administration tasks.
2. Workloads can be separated by designating separate LPARs to run different workloads, for example, one LPAR for development work, one LPAR for testing, and several LPARs for production workloads.
3. The running of an application/subsystem/OS at its optimal performance and scalability can be obtained on an LPAR with optimal amounts of physical resources for that specific instance of application/subsystem/OS.

DLPAR additionally enables the following autonomic features in a system:

- Dynamic Capacity on Demand (DCOD)—DLPAR enables cross-partition workload management, which is particularly important for server consolidation, in that it can be used to better leverage system resources across partitions, thereby achieving higher levels of resource utilization, resulting in enhanced system throughput. Here is a possible usage scenario: The LPARs on an SMP are the servers for workloads originating from users in different time zones of the country, or even from different cities around the globe. While one LPAR "sleeps," its spare resources can be shifted to another LPAR that "wakes up" to do its work for the day. This shifting can be done manually via oper-

Figure 1 Logical partitioning in pSeries SMPs



ator command, and then later can be automated via the Global Resource Manager (GRM, an automated resource balancer across a specified group of LPARs in an SMP, based on OS utilization and needs) or the enterprise WorkLoad Manager (eWLM, an end-to-end response-time-based load balancer for "instrumented" applications spanning LPARs in an SMP, or even across SMPs).

- Dynamic Capacity Upgrade on Demand (DCUoD)—DLPAR enables the upcoming DCUoD feature of the pSeries 690 by allowing customers to purchase a server with extra unlicensed resource capacity, and later license and add this capacity dynamically to running AIX LPARs as their resource requirements increase.
- Dynamic CPU Guard and Dynamic CPU Sparing—DLPAR allows systems to smoothly replace processors that show intermittent, but correctable, errors. This self-healing feature will continue to become important with reduction in the silicon device size along with greater and greater integration on a chip. The Dynamic CPU Guard feature is an improved and dynamic version of the existing RAS feature named CPU Guard in earlier AIX versions. The older CPU Guard feature predicts the failure of a running CPU by monitoring certain types of transient errors and dynamically takes the CPU off line, but it does not provide a substitute CPU, so that a customer is left with less computing power. Additionally, the older feature will not allow an SMP to operate with less than two processors. The DLPAR technologies allow the OS to function even with one processor. In addition, the Dynamic CPU Sparing feature allows the transparent substitution of a good unlicensed processor for one that is suspected of being defective. This on-line switch is made seamlessly, so that applications and kernel extensions are not impacted. The new processor autonomously replaces the defective one. Dynamic CPU Guard and Dynamic CPU Sparing work together to protect a customer's investments through their self-diagnosing and self-healing software. Both features are planned to be available on pSeries 690 servers in AIX 5.2.

The IBM pSeries DLPAR system architecture

The initial release of DR will be supported on the POWER4 pSeries 690 and 670 servers. Figure 1 illustrates the RS/6000* system architecture for DLPAR. In the diagram, LMB stands for logical memory block and is the granularity of physical and logical memory assigned to an LPAR. In AIX 5.2, an LMB will consist of 256 MB of contiguous memory. The size of an LMB is expected to decrease in future releases of AIX.

In this section we list some of the pSeries system components that had to be modified to become DR-aware in order to implement DLPAR. Some of these components were introduced during the implementation of static LPAR (e.g., the hardware management console, hypervisor, global firmware, and two registers for partition memory management), and some components existed even before LPAR existed (e.g., local firmware and AIX). We did not have to introduce any new components for the implementation of DLPAR; we only made changes to existing ones.

Hardware management console. The hardware management console (HMC) is the main control point for DLPAR configuration definitions and operations. In the initial release, these operations are controlled mainly through a new resource management graphical user interface (GUI) that runs on the HMC. This GUI invokes a new HMC command that encapsulates the DLPAR request and provides the necessary sequencing, so that the firmware and the OS can act in a coordinated fashion to achieve the desired result. Internally, this new HMC command notifies both the OS and the global firmware of a remove or add request. In time, this command will be used by other programs such as the GRM to dynamically transfer resources based on capacity requirements.

Global and local firmware. Global firmware provides the basic DLPAR enablement through machine-dependent logic that is designed to start, stop, and electronically isolate physical resources within the context of a logical partition. That is, global firmware performs these operations indirectly at the request of the OS, which actually passes these requests to the local firmware of the LPAR. In general, the local firmware does not contain machine-dependent logic, but is used to provide an abstraction to the OS of the logical resources that are currently assigned to the LPAR. The OS utilizes the local firmware interfaces to determine the identity and physical characteristics of logical resources that are present in the LPAR and to control them. For example, local firmware provides interfaces to start, stop, isolate, and unisolate logical resources by invoking functions provided by the global firmware.

AIX. When the OS receives a request to add or remove a resource, it has to take actions, both in the user space and in the kernel, to achieve the desired result. These actions are described in more detail in the next section.

AIX DR components

Dynamic reconfiguration support in AIX 5.2 is provided for three types of hardware resources: processors, memory, and I/O slots. In this AIX release, the granularity of addition or removal for processors is one CPU; for memory, 256 MB; and for I/O, one PCI (Peripheral Component Interconnect) slot. In future releases of AIX, finer granularities are planned for both memory and processor. The following subsections briefly describe the design for the addition and removal of each hardware resource type, and the kernel modifications required for enabling DR in AIX. Although we mostly focus on DR as an enabler of autonomic computing in this paper, we have also used autonomic computing principles even within the DR design, an example of which is given in a subsequent subsection entitled “Dynamic removal of memory.”

DR of processors. Achieving dynamic reconfiguration of processors introduced changes to the base kernel as follows:

1. For consecutive CPU identifiers (IDs), the kernel keeps track of its CPUs by assigning a distinct number, called a logical CPU ID, to each of its CPUs. Prior to DR, the kernel assumed that these logical CPU IDs were consecutive numbers, that is, no holes were allowed. To be able to randomly remove and add CPUs without perceptible disruption to applications, we had to modify the kernel so that it can tolerate missing items in the numbering of logical CPU IDs. Though we could have modified the kernel whichever way we wanted, there are many third-party applications (e.g., device drivers and performance tools) that assume consecutive numbering of CPUs, and they must be provided with binary compatibility, so as to fulfill the backward compatibility objective of AIX. Hence, another layer of numbering, bind CPU IDs, was introduced. The bind CPU ID abstraction provides a consecutive numbering of on-line CPUs for applications, even when the logical CPU IDs in the kernel are randomly removed and added, that is, the list of logical CPU IDs now becomes a list of on-line CPUs and off-line or removed CPUs.

2. For MP/UP locks, some components of the kernel were coded to decide at boot time whether they will acquire uniprocessor (UP) locks or multiprocessor (MP)-capable locks during the lifetime of the OS session. These components were modified so that they will function properly even when DR changes the system dynamically from a UP to an MP (and vice versa).

Dynamic removal of processors. Dynamic removal of a processor involves the following tasks initiated from the OS:

1. Notify DR-registered applications and kernel extensions, if any, so that they will voluntarily remove dependencies on the CPU to be removed. This task typically involves unbinding the threads that are bound to the CPU being removed.
2. Migrate threads bound to this CPU to another running CPU.
3. Retarget pending interrupts, and change the bindings of all interrupts currently bound to the processor to be removed. This task involves changing the interrupt controller data structures.
4. Migrate the timers and threads from the CPU being removed to another CPU within the same LPAR.
5. Notify the hypervisor or firmware to complete the removal task.

Dynamic addition of processors. Dynamic addition of a processor involves the following tasks initiated from the OS:

1. Create a process (waitproc) for idle looping on the incoming CPU before it starts to do real work.
2. Set up various hardware registers (e.g., SDR1 = Search Descriptor Register 1, which defines the start physical address and size of the page table in memory, GPR1 (General Purpose Register 1) for kernel stack, GPR2 for the kernel table of contents, etc.) of the incoming CPU.
3. Allocate or initialize, or both, the processor-specific kernel data structures (dispatcher run-queue, per-processor data area, interrupt stack, etc.) for the incoming CPU.
4. Add support for the incoming CPU to the interrupt subsystem.
5. Notify the DR-registered applications and kernel extensions that a new CPU has been added.

DR of memory. The implementation of dynamic reconfiguration of memory in AIX 5.2 enables the removal and addition of 256 MB contiguous sections of memory. This unit is referred to as a logical memory block (LMB). A smaller-sized LMB, for example, 16 MB, will be allowed in future releases of AIX.

Two important challenges were resolved during the implementation of memory DR: In the first one, the physical addresses of some memory are exposed to manipulation by applications over which the kernel does not have direct control. These accesses are allowed for functional reasons in some cases (e.g., direct memory access, or DMA), and for performance reasons in other cases (e.g., pretranslated addresses). Pretranslated addressing is an internal feature of AIX with which virtual-to-physical address translations for a data buffer to be involved in a DMA operation is done only once for the life of the data buffer. DR of page-frames with these uses can be handled in at least two ways: (1) by modifying the kernel and firmware so that such accesses by kernel extensions to the page-frame being removed can be controlled; or (2) by requiring kernel subsystems to register a DR callback function with the kernel DR subsystem, and invoking the callback function at memory removal time.

We have applied both techniques, choosing one over the other, depending on the specific circumstances, while minimizing the impact on system performance as well as kernel changes. In the case of DMA, the kernel and firmware control the access to the memory being removed by selectively disabling the bus traffic while the contents of the memory with DMA are being migrated to a new location. In the case of pretranslated addresses, a callback mechanism is used.

The second challenge is enabling the translation-on execution of the majority of the kernel, which previously was run in translation-off mode and hence required maximally sized data structures to be allocated at boot time for the maximum amount of physical memory that the AIX/LPAR instance can potentially grow into.

Dynamic removal of memory. For the purpose of dynamic removal, we classify the memory page frames of AIX into five categories: unused, pageable, pinned, DMA-mapped, and translation-off memory. The approach taken to remove a page-frame (4096 bytes) in each of these categories is as follows:

1. A page-frame containing a free page is simply removed from its free-list.

2. A page-frame containing a pageable page can be made to either page out its contents to disk or to migrate its contents to a different free page-frame. In the future, an autonomous agent, for example, GRM or eWLM, can choose one of these two approaches, depending on its knowledge of memory availability.

3. A page-frame containing a pinned page will have its contents migrated to a different page-frame; also the page-fault reload handler had to be made to spin during the migration.

4. A page-frame containing a DMA-mapped page cannot be removed or have its contents migrated until all the accesses to the page are blocked. Here, the term "DMA-mapped page-frame" is used generically to mean a page-frame whose physical address is subject to read or write by an external (to kernel) entity such as a DMA engine. The contents of a DMA-mapped page-frame are migrated to a different page-frame with a new hypervisor call (*h_migrate_dma*) that will selectively suspend the bus traffic while it is modifying the TCEs (translation control entries) in system memory used by the bus unit controller for DMA accesses.

   Other page-frames whose physical addresses are exposed to external entities are handled by invoking preregistered DR callback routines and then waiting for completion of the removal of their dependencies on the page.

5. A page-frame containing translation-off pages will not be removed by DR in AIX 5.2. Fortunately, there is just a small amount of these page-frames, and they are usually colocated in low memory. These page frames are intended to be handled in later AIX releases.

The design and implementation of dynamic memory removal has manifested itself as three modular functions, such that one can mix and match these functions in several possible ways, adapting to the state of the system at the time of memory removal, thus achieving the desired end result with the most optimal path. This design adheres to the self-optimizing principles of autonomic computing, as described in Reference 2. These three modular functions perform the following three tasks respectively on the memory (LMB) being removed: (a) remove its free and clean pages from the regular use of VMM (Virtual Memory Manager), (b) page-out its pageable dirty pages, and (c) migrate the contents of each remaining page-frame in the LMB to a free page-frame outside the LMB. Memory removal can be implemented with any one of the sequences: abc, ac, bc, or just c.

For example, the decision to either invoke page-out (task b) or to migrate (task c) all the pages depends on the load in the LPAR at that particular time. If the LMB being removed contains a lot of dirty pages that belong to highly active threads, then it does not make sense to invoke task b, because these pages will be paged back in almost immediately, negatively impacting the efficiency of the system.

As a second example, if there are not enough free frames in other LMBs to migrate the pages to, then the memory removal procedure can invoke task b before invoking task c, so that there will be far fewer pages left that need to be migrated in task c.

Dynamic addition of memory. When a memory-add request arrives at AIX, it has to perform two tasks: allocate and initialize software page descriptors that will hold meta-data for the incoming memory, and distribute the incoming memory among several free-frame pools so as to preserve the behavior of memory management algorithms. The challenges encountered in implementing these two tasks and how they were resolved are now described.

The primary challenge was the allocation of memory for the software page descriptors for the incoming memory. The problem was that, prior to DR, these page descriptors could be accessed in translation-off mode while trying to reload a page mapping into the hardware page table. If the page descriptors are allowed to be accessed in translation-off mode, the memory allocated for those new descriptors has to be physically contiguous with the memory for existing descriptors, which implies that memory has to be reserved at boot time for descriptors for the maximal amount of memory that the OS instance can potentially grow into. This can potentially incur inefficiency and wastage of much memory, particularly if not utilized. We avoided this wastage by changing the kernel so that software page descriptor data structures are always accessed in translation-on mode.

Another challenge that was resolved while implementing dynamic memory addition was the difficulty in distributing the incoming memory across different page replacement daemons, so that each daemon handles a roughly equal load. In AIX, memory is hierarchically represented by the data structures vmpool, mempool, and frameset. A vmpool represents an affinity domain of memory. A vmpool is divided into multiple mempools, each mempool being managed by a single page replacement least recently used (LRU) daemon. Each mempool is further subdivided into one or more framesets that contain the free-frame lists, so as to improve the scalability of free-frame allocators. When new memory is added to AIX, the vmpool that it should belong to is defined by the physical placement of the memory chip. Within that vmpool, we want to distribute the memory across all the available mempools to balance the load on page replacement daemons. However, the kernel assumed that a mempool consisted of physically contiguous memory. Thus, to be able to break up the new memory (LMB) into several parts and distribute them across different mempools, the kernel was modified to allow mempools to be made up of discontiguous sections of memory.

DR of I/O slots. The methods to dynamically configure or unconfigure a device have been introduced as early as AIX version 3. The changes required in the kernel design for DR of I/O slots were not in the same scale as those for DR of processors, and particularly those for DR of memory. The reason is that the onus of configuring or unconfiguring a device lies with the device driver software, which operates in the kernel extension environment. The kernel just acts as a provider of serialization mechanisms for devices accessing common resources and as an intermediary between the applications and the device drivers.

Challenges of adding autonomic features to a mature UNIX OS

The AIX kernel is an industrial strength pageable and pre-emptable kernel that offers high performance and scalability (up to 32-way SMP) and supports many vendor device drivers, databases, and applications. The same kernel source code supports all reasonable combinations of 32-bit or 64-bit kernels, 32- or 64-bit applications, and 32- or 64-bit RISC (reduced instruction-set computer) systems with old and new RISC architectures, uniprocessors, and multiprocessors. Being friendly to its users, AIX goes out of its way to offer backward binary compatibility in a wide variety of situations. Being robust and having high performance, the critical kernel sections are skillfully guarded by a carefully crafted hierarchy of data and code locks. Implementing DR involved careful changes to numerous critical components of the AIX kernel.

Self-healing and self-protecting features

DR also serves as the foundation for a new advanced self-healing and self-protecting technology, especially when it is coupled with the presence of extra unlicensed capacity. This new technology enhances a preexisting self-diagnosing technology, called the CPU Guard feature, that monitors recoverable error rates for processors. At its simplest level, this new technology substitutes a spare processor in a transparent fashion for a processor that the system has internally diagnosed as being defective. Spare resources are only present if the system was shipped with extra unlicensed capacity as defined by the Capacity Upgrade on Demand solution, although it should be noted that there are no license keys that have to be entered to enable this new Dynamic CPU Sparing technology. This Dynamic CPU Sparing feature does not address the case in which a CPU fails so suddenly that a state-save and an orderly live swap of the CPU cannot be performed.

The technical aspects of this new technology are outlined next. Firmware monitors the health of each processor in terms of the number of recoverable errors. If this number exceeds an internal threshold, it raises a repeat guard error to the operating system and makes available an unlicensed processor, assuming that one is available. AIX acts on this error notification by invoking the DR Manager, which guides the self-protecting procedure from the perspective of an operating system. If an unlicensed processor is not available, it proceeds to take the defective processor off line in a fashion similar to that of the existing CPU Guard feature; this procedure constitutes a self-protecting feature of the system. If an unlicensed processor is available, the procedure uses it as a substitute for the defective one, making the switch transparently with the new DR technologies. This action constitutes a self-healing feature of the system.

The DR Manager determines whether an unlicensed processor is available through the use of new firmware routines, and if it finds one, attempts to take ownership of it from the firmware by invoking new firmware routines. These new firmware routines are the same ones that are used for DR processor addition, although a different return code is used to indicate that they are reserved for self-healing. Next, the DR Manager queries the kernel to determine the identity of the defective processor, which was named by the repeat guard error log entry. Finally, it invokes the kernel to perform the live swap.

The new processor is always started before the defective one is stopped, so that this technology can be applied to a single-processor LPAR—an improvement over the old CPU Guard self-protecting technology, which could not remove the last two on-line processors. At a high level, the switch is made by taking over the execution of the defective processor and using it to control the sequence of events that are required to transparently complete the switch, much of which has to be run on the defective processor itself, since the new processor is going to assume the logical identity of the failing one eventually. This is largely a matter of preserving the physical state of the defective processor and atomically reprogramming any funneled hardware interrupts that may be directed to the defective physical processor before allowing the new processor to begin operating.

The kernel actually performs the switch by masking external interrupts on the defective processor, so that it is not expected to respond to external events that may be generated by adapters or other processors in the partition, and by tightly controlling the execution of the new processor. Before starting the new processor, the defective processor saves the state of its registers, so that they can be restored by the new processor once it is functioning. Next, it vacates its logical CPU ID inside the kernel so that the new processor can be brought up in the proper logical CPU ID relative to the kernel. This is important in that logical workloads assigned to the defective logical processor (e.g., its threads and timers) do not need to be migrated, which in large part constitutes the transparency of the switch. There are many logical processor states and very few physical processor states that need to be taken care of. Next, it invokes firmware to start the new processor at a startup routine that is specific to this algorithm, and it waits for the new processor to indicate that it has successfully added itself to the global processor interrupt queue. Once this occurs, the defective processor can stop itself without fear of losing any external interrupts. This synchronization point is required to ensure that the global processor interrupt queue has at least one processor at all times. At this point, the new processor loads the register state that was previously preserved and resumes the execution path previously established by the defective processor.

This self-healing technology is built into the AIX base operating system and is automatically enabled by default, although the customer may disable it through system management options. This new technology is initially available on the pSeries model 690.

Self-optimizing and self-configuring features

The next release of AIX will have provisions for the automatic movement of its logical resources between LPARs, allowing the overall utilization of the physical resources of an SMP to be increased at no extra cost to the installation. For example, an agent, such as a Global Resource Manager (GRM), will monitor the utilizations, needs, and Service Level Agreements (SLAs) of a pool of LPARs and autonomously move the DR resources from an underutilized LPAR to a needy one, hence enriching the SMP with self-optimizing and self-configuring capabilities in a timely fashion. GRM also ensures that the DR movements will be orchestrated in a smooth fashion and without undesirable oscillations.

Additionally, as part of the IBM autonomic initiative, work is ongoing to provide optimal end-to-end response time for "instrumented" Web and commercial applications that span the LPARs of an SMP, as well as applications that span SMPs with or without LPARs, based on SLAs.

Conclusion

The DLPAR and DR technologies that have been developed on the IBM pSeries 690 servers have enabled these servers to become truly autonomic computer servers. As described in this paper, these servers have features that are self-protecting and self-healing, and they have the basic building blocks that enable these systems to be self-configuring and self-optimizing. The self-configuring and self-optimizing features are currently planned to be available in the near future. Thus, all four self-managing features that are at the heart of an autonomic server will soon be available on the pSeries 690. DLPAR is a strategic pSeries AIX capability and will also be made available on future pSeries servers.

In a possible future enhancement, virtualization of a physical processor into virtual processors allows the processor resource to be sharable in a fine-grained fashion (instead of one processor at a time) among a pool of LPARs in the SMP.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of The Open Group.

Cited references

1. P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IBM Corporation (October 15, 2001); available at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.
2. K. Ekanadham et al., *Anatomy of Autonomic Server Components*, Research Report RC 22637, IBM T. J. Watson Research Center, Yorktown Heights, NY.

Accepted for publication August 16, 2002.

Joefon Jann *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: joefon@us.ibm.com).* Ms. Jann is a Senior Technical Staff Member at the Watson Research Center, where she leads a small team that conceived and prototyped the notion of DLPAR for AIX. She is currently working on automating DLPAR. Her previous projects in IBM include the design and prototype of a DSM (distributed shared memory) system for the pSeries servers, the design and implementation of the LoadLeveler® hierarchical communication infrastructure, which enables IBM's LoadLeveler product to work in the largest SP™ installations. She coinvented the "Jann MPP Workload Model," was a member of the Deep Blue Computer chess team, a developer of the IBM product VMPRF (VM Performance Reporting Facility), a VM/SNA Area Specialist, and an APL programmer. Ms. Jann was a lecturer in mathematics at Lehman College for three years, and holds B.A. and M.A. degrees in pure mathematics from Wellesley College–MIT and the City University of New York (CUNY), respectively, and an M.S. degree in computer science from Columbia University. She is a member of the IEEE.

Luke M. Browning *IBM Server Group, 11501 Burnet Road, Austin, Texas 78758 (electronic mail: browninl@us.ibm.com).* Mr. Browning is a Senior Technical Staff Member in the AIX Kernel Architecture and Design Department of the IBM Server Group, working on dynamic LPAR, workload management, threads, and process management. He joined IBM in 1984 in Austin after receiving his bachelor of science degree in computer science from the University of Texas at Austin.

R. Sarma Burugula *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: burugula@us.ibm.com).* Mr. Burugula is an advisory software engineer at the Watson Research Center. He received his B.S. degree from Regional Engineering College Warangal and an M.S. degree from the Indian Institute of Technology Kanpur in India, both in computer science. He joined IBM in 1996 and has been working primarily on the IBM pSeries platform, developing various parallel and scalable subsystems.