WebSphere software

IBM

Redbooks Paper

Paul DiMarzio
Brent Watson
Patrick C Ryan

Scaling for High Availability: WebSphere XD and WebSphere Application Server for z/OS

Introduction

The objectives of this IBM® Redpaper are to address several existing issues with respect to helping customers and IBM personnel decide when it is advantageous to deploy applications using z/OS®-based solutions and when to deploy using distributed IBM @server® pSeries® solutions. Thorough analysis of the scalability of WebSphere Application Server on all IBM server systems (xSeries®, iSeries™, pSeries, zSeries®, and BladeCenter™) and competing products from Sun Microsystems and Dell would be ideal, but unfortunately is beyond the initial scope of this paper. As zSeries and pSeries solutions have been in frequent contention for similar Web-serving environments, this paper will deal primarily with them.

The audience for this paper includes but is not limited to software information technology architects and WebSphere® Application Server information technology specialists. Also, it can be used to assist customers in deciding what platform is best for their applications.

Evaluating non-functional behavior

An application written to the WebSphere Programming Model will, by definition, produce consistent business results regardless of the platform on which it is deployed. For example, the end result of executing the business process for a funds transfer will repeatedly result in the funds being transferred or not, as specified by the business rules for that process, on any deployment platform.

However, the non-functional behavior exhibited by the application is tightly linked to the choice of deployment platform. Non-functional behavior is the ability of an application to operate within the parameters of a given set of business goals or objectives. That is, the application is available to those who need it, when they need it, with the expected degree of safety and responsiveness.

Non-functional behavior *may be specified* as part of the development process—via deployment descriptors or annotations, for example—but it is *always managed* as part of the deployment process, requiring the IT staff to focus on a number of infrastructure issues such as:

- Providing infrastructure reliability and availability that supports business operations
- Mitigating business risk through protection of information assets, management of privacy, and maintenance of data integrity
- Capturing, analyzing, and utilizing information effectively to support business decision making
- Integrating existing processes within the company and maximizing utilization of existing computing resources while simplifying and streamlining core business processes

Scaling with high availability

While all of these issues are important, they represent the more traditional aspects of IT management and as such tend to be fairly well understood, with clear processes for evaluation.

A focus on resilience and responsiveness

Becoming an on demand business, however, introduces a new set of issues for which evaluation is still more an art than a science. Among these issues are the concepts of resilience and responsiveness; this adds the following concerns, which must be addressed without sacrificing those previously discussed:

- Providing continuity of business operations under extremely variable conditions

- Anticipating and forecasting changes in the marketplace; responding quickly to opportunities, competitors, and regulations

The focus of this material is on the evaluation of deployment platforms with respect to resilience and responsiveness. From a technological point of view, this primarily boils down to a deployment platform's ability to provide continuous availability of business processes at high levels of scale.

Availability at scale

Availability at scale requires an IT infrastructure to be resilient enough that capacity can be autonomically assigned and re-assigned, added and subtracted, in response to changing business conditions, while ensuring that business processes consistently meet business goals and objectives.

Availability in this context is somewhat different from the traditional view of availability, which is typically taken as the avoidance of or recovery from failures.

Availability at scale includes the traditional view of failure avoidance and recovery but adds the additional requirement that delivery of expected service as defined by the business is consistent regardless of spikes in demand (whether anticipated or unexpected), temporary resource constraints (whether due to saturation or unavailability of resources), or the intervention of outside elements (such as a data mining process running against live data).

Note: No platform decision should ever rest on a single data point; this material should only be used as one resource in an overall platform assessment.

Primary WebSphere offerings for scaling with high availability

IBM offers several different eServer™ brands, and the WebSphere Application Server is available in several different configurations, providing many options for deploying solutions that satisfy a wide variety of business requirements.

For supporting high availability at high levels of scale, two notable IBM solutions are WebSphere Extended Deployment (XD), deployed to pSeries servers running AIX®, and WebSphere for z/OS, deployed to zSeries servers running z/OS.

Note: IBM offers other options that can provide different degrees of availability at scale: BladeCenters, iSeries, xSeries, and zSeries Linux® solutions. However, these are outside the scope of this initial material.

Both pSeries and zSeries are enterprise-class servers. Although there are many similarities between the two, there are also significant differences by design.

pSeries

pSeries is designed to compete in the Linux/UNIX® space by offering high performance and excellent RAS (Reliability, Availability, and Serviceability – “qualities of service”) and system resource management capabilities.

zSeries

zSeries systems are designed to sustain high end-user service levels through quality and proven systems management facilities, outstanding RAS, and security features.

Evaluation of the ability to scale with pSeries and zSeries begins by examining each brand’s architecture of a single, shared-memory, symmetric multiprocessor (SMP) server. Within the boundary of a single SMP machine, we need to look at the processor architecture and processor effectiveness in light of how we can scale up the machine by increasing the number and capacity of processors.

The class of application that we are addressing in this analysis nearly always requires the capacity, flexibility, and availability characteristics of a multiple server solution deployed in a *scale-out* configuration.

Scale-out

Scale-out has traditionally been defined as the deployment of applications on a collection of small but tightly-packaged and interconnected computers as seen in many pSeries implementations. However, an equally valid scale-out scenario is the deployment of applications on a collection of large, independently packaged and tightly interconnected computers such as the zSeries mainframe. Deciding which to use relies, to a great degree, on the nature of the workload.

The importance of application patterns

The tradeoffs necessary to optimize pSeries for performance and zSeries for quality of service has left each platform with a unique design point that has a direct effect on how each solution scales. The effect of these design points is additive and is experienced all the way up the WebSphere middleware stack. The net result is that each deployment favors a particular and distinct set of application patterns, so it is necessary to understand what these patterns are and how the respective platforms are tuned to one pattern or another.

An application pattern, with respect to achieving availability at scale, involves an understanding of the following characteristics:

- Is the application processor-intensive or data-intensive?

- Are the data access patterns fairly static or do they vary widely?
- Will the pace of scale be slow (ramp up) or fast (rapid deployment)?
- Will the need for scale be steady or sporadic?
- Will scale be consistently positive or up-and-down?
- Are these characteristics well understood in advance or will change be unpredictable?

Frequently, the initial cost of acquisition is a factor in driving pSeries-style scale-out computing. The appeal of zSeries-style scale-out computing is the quality of service that the platform provides and the ability to tightly integrate with legacy zSeries resources such as CICS® and IMS™.

Note: Cost is an important decision point. From a single box perspective, a zSeries server with its associated software is more expensive to acquire than a pSeries alternative. However, which is right for any given deployment depends on whether the focus is on cost of acquisition or ownership or both. Debate over the merits of total cost of acquisition and total cost of ownership, and how they are calculated, is beyond the scope of this material.

Our intent is to show that a critical factor in analyzing cost is a thorough understanding of platform architectures and the nature of the application patterns. The ability to match the pattern to the architecture is an important factor in obtaining a favorable result.

In general, applications that exhibit patterns that can be deployed in large replicated environments will benefit from the lower price point of pSeries servers. Replicated environments tend to lead to higher costs for management software and staffing due to the larger number of physical servers, but the aggregate cost savings of the servers can sometimes offset these “soft” costs and lead to a very cost-efficient solution.

Massive replication works best in situations where applications are inherently parallel and the application has segmentation of data access or the individual work streams do not require simultaneous access to a single data resource.

The efficiency of a replicated environment is diminished when applications are highly skewed from a business perspective, where multiple applications or components are vying for the same data resource (and not easily parallelized), or the requirement for a single data resource leads to overprovisioning servers to avoid bottlenecks.

Applications that have no segmentation of data access or segmentation that is very random in data access are typically highly skewed and can sometimes be more efficiently deployed on zSeries servers.

The cost of ignoring patterns

Given enough time, talent, and resource, any of today's top-tier platforms could be utilized to solve any given business problem. Very few organizations have an unlimited supply of time, talent or resource, so great care must be taken to thoroughly understand application patterns and match them to the platform best optimized to handle those patterns.

Often, pSeries solutions will favor workloads where consistent data access patterns can take full advantage of the pSeries' fast CPUs, and growth is fairly predictable and consistent. zSeries solutions are more geared toward workloads where data access patterns are more random, multiple applications are vying for the same resources, and capacity needs are unpredictable.

A deployment mismatch can prove to be very costly over the life of an application. Overprovisioning a solution leads to white space (capacity that must be configured to handle peak load but sites idle most of the time), which requires the acquisition of more hardware, software, and administrators. Nonetheless, this can be a cost-effective environment for deployment of your application.

All things being equal, applications that rely on tight integration with existing z/OS resources such as CICS or IMS transactions, DB2® databases, or VSAM files are often deployed to zSeries, for reasons such as the elimination of network latency, ability to use optimized access protocols, stronger affinity for the z/OS style of security and transaction management, and so forth.

For larger customers that already have skills in both platforms, the best solution may be to deploy portions of the workload to both pSeries and zSeries in cases where the business process can be segmented by application pattern. Such a multiplatform deployment exploits the strengths of each platform and may result in a lower cost overall. There are no hardened rules for determining the appropriate platform for your organization. All factors, including your organization's existing IT skills, strategic goals, and comfort level, should be taken into account.

Important: IBM intentionally offers multiple platforms, each optimized for different design points. These offerings complement each other and, in support of scenarios where it makes sense to segment deployment, they will become increasingly integrated over time.

Do you manage an application or an enterprise?

One final point before we dive into the detail is that an application should never be evaluated in a vacuum; it is critically important to understand the overall context within which the application will execute.

Unless an application is a standalone solution to an isolated business process that does not interact with other processes, some degree of integration must be assumed. Virtually every enterprise-class application is a piece of the overall architecture that is running in an enterprise computing environment, and it must be made to fit without negatively affecting its neighbors.

The degree of interaction and the level of integration can change the dynamics of the aggregate access pattern. For example, a given application when assessed on its own may exhibit a pattern that favors a pSeries deployment.

Another application, when assessed on its own, may also exhibit a pattern that favors pSeries. However, if the business requires that these applications work together, the overall pattern may change to one that favors a zSeries deployment.

For example, an online stock lookup application may have an access pattern that favors a pSeries deployment. But a business requirement that the batch trade resolution processes be able to access the same data concurrent with the online application may skew the overall pattern to favor zSeries.

Evaluating the applications individually produces a different result than evaluating the applications as part of an interconnected enterprise.

The true value of zSeries—per its design point—is in the consolidation of multiple, diverse applications that must interact with one another against a common data source.

A truly optimal solution can be achieved only if each application is evaluated in the enterprise context to which it will ultimately be dispatched.

It is not always easy to ascertain an application's pattern, especially when the application is more coarse-grained and therefore exhibits multiple patterns. Furthermore, there is no question that understanding the interrelationship of applications in an enterprise is a complicated task, and is sometimes actually suboptimized by the selection engagement process.

Many WebSphere engagements begin as projects—targeted applications taken in a very narrow context—and the most pressing requirement of these projects is the overwhelming need for rapid deployment. What may look like an ideal decision in a standalone project environment may lead to a protracted

implementation cycle and higher-than-expected cost structure when the project is finally brought into the enterprise environment (at which point it may be too late to make adjustments, leading to costly overruns).

The bottom line is that the most satisfactory results will be achieved when decision makers have access to the right skills and resources to evaluate the consequences of their choices in light of the overall enterprise on demand plan.

Scaling-up: anatomy of a server

Although the class of application that we are addressing with our pSeries and zSeries WebSphere deployments requires the capacity of more than one server in a scale-out configuration (which will be addressed later), the basic organization of the server and effectiveness of the operating system are fundamentally important in determining the application patterns that best fit a given architecture.

If a mismatch results in unutilized capacity on a single server, the cost of this “white space” is seen at each step of the scale-out configuration. Depending on the situation, this cost can be acceptable or not. Understanding which application patterns make the most efficient use of the capacity of a single server in a scale-up configuration is an important starting point for later evaluating scale-out scenarios.

The capacity of a single server is mainly a factor of:

Processor speed (CPU time) The fundamental chip technology that dictates how much hardware remains available for other characteristics.

Processor effectiveness (memory time) Also processor utilization; the way in which caches, buses, and I/O devices get data to the processor to keep it busy, as well as the ability to partition a machine via virtualization to keep all resources as busy as possible.

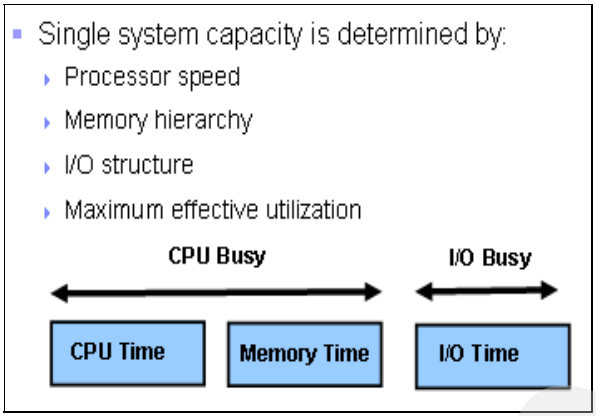

Figure 1 Single system capacity

A basic chip consists of circuitry that forms the processor, cache, and RAS elements. Because of constraints in chip space, it is not possible to fully maximize each of these variables. Design tradeoffs will be made in all aspects, and all components affect performance. If a particular application pattern does not match the design point of a particular server, resource utilization is less efficient (for example, processor cycles are wasted), extra capacity is required to compensate for the inefficiency, and the cost of the deployment increases.

When architecting a processing chip, one immutable law is that the more silicon that is set aside for speed, the less is available for other characteristics. Achieving the fastest possible processor speed uses the most real estate for the processor, and as a result limits space for cache and RAS.

A design that maximizes processor effectiveness focuses on keeping the processor as busy as possible by ensuring that the right data is available for processing when needed. This design point focuses on caches, bus speeds, and I/O capabilities at the expense of processor speed.

If a data-intensive workload is matched to a machine that maximizes processor speed, that super-fast CPU may sit idle while waiting for data to move around, limiting its effective speed and increasing the capacity required for the application. To the contrary, if a computation-intensive workload is matched to a machine that maximizes processor effectiveness, the large areas devoted to cache and RAS may be underutilized while the CPU runs at maximum speed.

Most modern benchmarks focus on measuring processor speed. They are less effective at measuring processor effectiveness. Benchmarks are worthwhile data points but do not fully reflect how an application will deliver availability at scale in an on demand setting.

As we will see in the following sections, the pSeries design point is processor speed and the zSeries design point is processor effectiveness. These design decisions have a direct impact on the application patterns that run best on each server. We will demonstrate that maximizing processor speed forces on-chip cache to be limited; in effect, minimizing CPU time has the effect of elongating memory access time.

Context switching

Our analysis will focus very heavily on the concept of context switching, which is the process of storing and restoring the state of a CPU—the context—so that multiple threads of execution can share CPU resources in a multitasking, multiprocessing environment.

While there are many reasons why context switches are processed, the condition that we are most concerned with in this material occurs when the CPU requests data that is not immediately available on the chip. In these circumstances the CPU will switch contexts and process some other, ready thread rather than busy-wait until the data is made available (at which point the CPU can be interrupted and switch contexts again to complete the operation).

Context switching is busy-work during which the processor is managing overhead rather than processing application logic, so it is something to be reduced.

If an application pattern does not cause the processors to switch context very often, the working set of the on-chip cache will not change much and a smaller cache can provide a sufficient solution. Workload patterns that rely primarily on processor power will most likely favor pSeries because the pSeries devotes more space to the processor. pSeries is more tuned to a stable and homogeneous workload pattern.

Workloads that move lots of data have lots of users that generate lots of transactions (online transaction processing or OLTP) and workloads that are heavily virtualized force more context switching and therefore are favorable to the zSeries design because the zSeries devotes more space to the cache. zSeries is more tuned to dynamicism and heterogeneity.

Processor speed

An examination of processor chip circuitry shows that pSeries devotes 70% of the space to the processor, and the remaining 30% to L1 cache. zSeries, by contrast, devotes only 20% to processor and 50% to L1 cache. (The remaining space is allocated to RAS items and crypto.) As was previously stated, the pSeries is designed to achieve very high performance, and the size of the L1 cache must be sacrificed to get it.

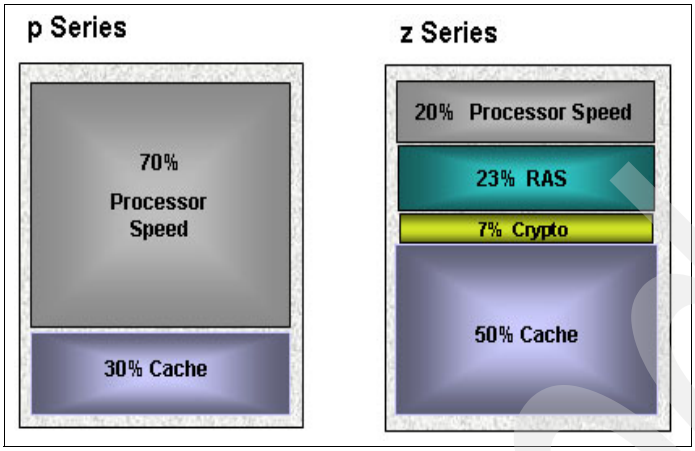

Figure 2 Comparison of processor speed

Standard benchmark

pSeries excels at most standard performance benchmarks. The typical benchmark uses a small working set of active data that fits easily in the pSeries cache. Benchmarks do not cause very much context switching, so the data is typically available in cache and the processor speed is fully leveraged. zSeries, by sacrificing processor speed for cache and RAS (which are not exercised by a benchmark) typically does not score as well as pSeries for this type of workload.

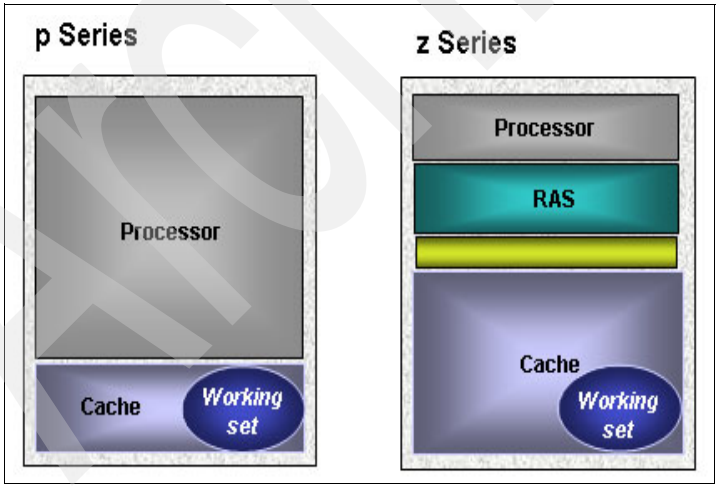

Figure 3 Typical benchmark

Mixed workload

By contrast, a mixed or data-intensive workload generates a lot of context switching, which produces a large working set that fits comfortably in the larger cache of the zSeries. Because there are fewer cache misses, the processor is more fully utilized. Furthermore, for this type of workload the RAS circuitry tends to be highly valued, validating the design decision to devote chip real estate to function that does not directly translate to performance.

The smaller cache of the pSeries cannot contain a larger working set so more data must be moved in and out of memory, causing its processor to lose cycles while data is made available and therefore lowering the effective speed of the processor.

Note: There are different levels of heterogeneity of the application environment and the size of the cached data. Depending on the magnitudes, either platform could perform better. There certainly are instances where a very heterogeneous environment could favor a pSeries deployment due to very small data sizes.

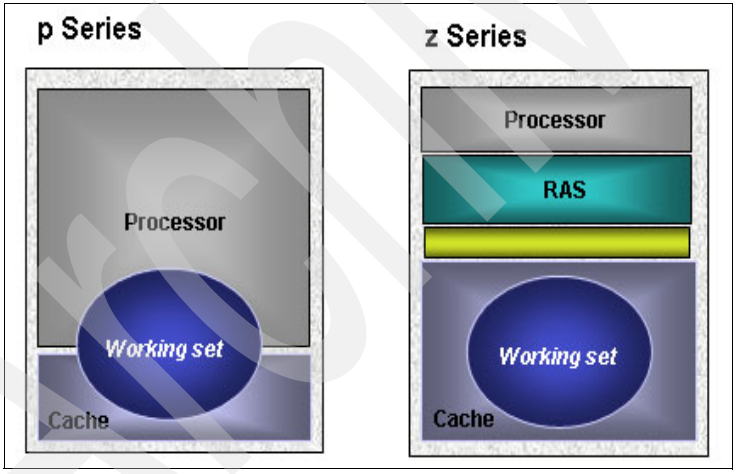

Figure 4 An example of mixed workload

Execution time is a function of pathlength, cycles per instruction, and cycle time. pSeries is designed to minimize cycles per instruction. zSeries is designed to minimize pathlength by avoiding cache misses and reloads and handling page faults in parallel.

Note: Neither design is necessarily better than the other; they illustrate different optimization points that tend to favor different workloads.

Processor effectiveness: utilization

A significant amount of server design work goes into minimizing cache misses and pathlength in order to keep the CPU busy. Different workload patterns stress different elements of the server, so only a perfect match of the correct pattern to the correct server design will lead to optimal utilization.

Underutilizing processors because of a mismatch requires the acquisition of more processing power than necessary. This is not necessarily a huge factor in a strictly scale-up configuration, but as we will see, this inefficiency factor can become costly when many multiple servers are deployed in a scale-out solution.

Processor effectiveness is the ability to achieve a very high degree of utilization, keeping the processors busy at all times. Because of the tradeoffs involved, all servers are designed to optimize utilization of a given set of application patterns. We will review four key technologies:

- Cache design and management
- Assist processors
- Workload management
- Virtualization through logical partitioning

Cache design and management

In a perfect world, the data that the processor needs is always in cache, enabling the processor to run at its rated cycle speed. In an imperfect world the data must always be retrieved from memory, causing the processor to wait, reducing its effectiveness. Cache size and cache design play a role in determining the real amount of work that a processor can process.

Cache size is a direct factor of the space available on the chip and is the result of tradeoff decisions regarding the amount of processor, cache, and RAS circuitry to place on that chip.

We have already seen that the reduced instruction set (RISC) design point of pSeries favors a homogeneous workload with relatively small working set due to the smaller L1 cache. pSeries does very well with workloads that have longer execution path lengths because the processor is kept busy.

The complex instruction set (CISC) design point of the zSeries favors a more varied workload requiring a large working set due to more space for L1 cache on zSeries chips. Neither is right or wrong; they are simply different. The key is to match a workload to the type of processor that fits the application pattern.

Effectiveness is not just a function of the size of cache, but also of how the cache is managed among the processors in an SMP. The goal of an SMP is for all processors to be busy and productive all the time, and as we have seen,

processor effectiveness is impaired when data is not in the cache and must be retrieved from memory. In an SMP, the organization of cache is important relative to context switching. There are two factors to consider: *set-associativity* and *cache sharing*.

Because the cache is smaller than memory, many memory locations must share the same cache location. For very large caches and small working sets this is usually not an issue, but when the processor needs data from two locations that are mapped together at or near the same time, the cache thrashes and has a negative rather than positive impact on performance.

Set-associativity cache

A set-associative cache strives to reduce thrashing by mapping cache locations to a set of memory locations; when there are N locations available for each set we call the cache N-way set associative. Higher way set associativity leads to higher hit ratios and lower probability of thrashing, giving the effect that the cache is larger than it really is.

Higher way set associativity also incurs a cost in terms of the circuitry required to manage the associations, so there is a design tradeoff to be made. The pSeries L1 cache is 10-way set associative; the zSeries cache is 16-way.

In an SMP many threads of execution operate simultaneously, all vying for the limited processor resources. Any time a thread is interrupted it is usually set aside so that another, waiting thread can utilize the processor. When an interrupted thread is resumed it may be restarted on the same or a different processor. The way in which threads are started, stopped, and restarted is a function of workload management (which will be discussed later), but this characteristic of SMP systems forces another tradeoff point in cache design.

Cache sharing

If each processor maintains its own, private cache, starting a thread on a different processor requires a movement of data through memory from one cache to the other. A cache-to-cache bus is faster but also involves a data movement. In this type of private cache architecture, the only way to limit the cache-to-cache movement of data is through processor affinity: ensuring that a piece of work is scheduled to the same processor all the time. The downside of processor affinity is the possibility of reduced processor effectiveness if a piece of work is available to be scheduled and a processor is free, but that processor does not have access to the cache data required by the work (leaving the processor idle).

A cache working set has two components: the part that is common for all threads and the part that is unique to each thread. When working sets have a large

common part and relatively small unique part, building a shared cache provides advantages:

- Many cache-to-cache migrations are eliminated, in effect providing infinite bandwidth relative to the bandwidth between two private caches at the same level.
- The cached access to common data supports a multiple-server, single-queue dispatching model, which is the best way to do things according to queuing theory, and in essence enables the scheduler to ignore processor affinity and simply schedule the next available processor from the shared pool.

zSeries L2 cache

The zSeries L2 cache is shared among all processors, giving zSeries a very high hit ratio with respect to the cache and establishing the zSeries architecture as ideal for a multiple-server, shared-queue cluster (to be discussed later) because z/OS always schedules work on the next available processor.

pSeries L2 cache

pSeries shares L2 cache among two processors. To avoid moving data from cache to cache through memory as much as possible, the AIX scheduler establishes an affinity between a thread of execution and a processor pair. AIX will wait to see whether the processor that ran the thread the last time becomes available in a timely manner (with SMT there are actually four threads competing for the two processors, making it less likely that a thread will immediately execute when dispatched).

Workload skew introduces its own set of challenges, reducing the overall average by affecting memory time; different cache designs respond differently. Mixed workloads put greater stress on caches; more context switches make processor affinity difficult to maintain.

Assist processors

One way to improve processor efficiency is to augment the main processors with supplemental, task-specific engines that relieve the processors of the burden of many housekeeping chores. A zSeries server has a number of these engines.

The zSeries offloads much of the work of processing I/O to other processors:

- The System Assist Processor (SAP), channels, and storage controllers. For instance, on zSeries, I/O path failure and recovery is handled by the SAP, with little or no involvement by the OS other than to log and report the error and initiate a repair action.
- HW crypto, compression.

- zAAP, a specialized processing unit that provides a z/OS Java™ execution environment.

Workload management

To this point we have focused on architectural tradeoffs that must be made in hardware: how processor real estate is laid out and how caches are organized and managed. Moving up a layer in the platform stack into the system software, we need to understand how the operating system utilizes the resources (such as processors, I/O paths, and memory) of an SMP to maximize efficiency.

In many respects the way in which the OS allocates resources to workloads is a direct consequence of the decisions made at the hardware layer. For example, we have already seen how processor affinity dispatching can be so important to cache efficiency, if the cache is not shared.

The way in which decisions are made regarding which waiting work unit to dispatch when a processor becomes available—and how to share the limited physical resources of the machine among the various work units that are executing simultaneously—has a direct effect on the type of application patterns that are processed most efficiently.

Workload management is composed of two parts: operating system dispatching and the platform's Workload Manager (WLM). Both the pSeries AIX and zSeries z/OS dispatchers are preemptive, but the similarities end here.

z/OS Workload Manager (zWLM) is an important component in WebSphere Application Server for z/OS scale-out deployments as it is tightly integrated into the infrastructure. pSeries WLM is not a valid point of comparison because it serves a different purpose than zWLM. The worthy comparison is that of the zSeries solution, including zWLM to the WebSphere XD solution, which uses the On Demand Router to provide its load balancing and workload management. (See "The WebSphere XD On Demand Router" on page 22.)

Note: Enterprise WorkLoad Manager (EWLM) is a new product that broadens the scope to the full enterprise, across different platforms. EWLM is built on the Application Response Monitoring 4.0 standard. It currently provides end-to-end topology views and statistics, helping with performance management, capacity planning, and problem diagnosis.

AIX dispatcher

Because the pSeries hardware design point favors workloads that do not cause the data in the cache to change much, pSeries solutions tend to be deployed as clusters of single-application machines. Within any given machine, therefore, AIX dispatcher on that machine assumes that all work to be dispatched is of more or

less the same business priority (same application, same priority). The priority of work to be dispatched is set per process and can be changed dynamically.

This leads to a workload management design point that is based on fairness in which the dispatching priority is the inverse of CPU utilization; the more a unit of work uses the CPU, the less likely it is to get the next available slice. As supported by the AIX dispatcher, this time slice is a single length time slice.

z/OS dispatcher

The z/OS dispatcher provides numerous methods of establishing priority and long-term and short-term weighting. These prioritizations can be changed dynamically. z/OS dispatcher supports three levels of declining time slices, which can be important when balancing work in order to achieve the business goals set by zWLM.

z/OS extends this management to storage and the network with Intelligent Resource Director. Resources are balanced and reassigned dynamically and automatically.

pSeries Workload Manager

pSeries WLM controls CPU time, memory, and I/O bandwidth with its fairness algorithm. In AIX, the work is assigned to classes that are, in turn, assigned to tiers. pWLM uses “waterfall” allocation of resources to tiers. The management is based on the time of day, providing automatic operator notification when resource limits are reached on a particular class of work.

pWLM has the ability to dynamically modify the fairness algorithm used by the AIX dispatcher, but does not do so automatically.

z/OS Workload Manager

The zSeries hardware was designed to support application heterogeneity, favoring workloads that move a lot of data in an inconsistent fashion. It has been matched with a workload manager in z/OS, zWLM, that has similarly been designed to meet the needs of a heterogeneous workload while maximizing the investment in system resources. This is an important factor in zSeries deployments due to the higher initial cost of the hardware.

This combination of hardware and OS optimization enables zSeries servers to routinely run at or near 100% CPU utilization. zSeries z/OS workload management can ensure that long-running tasks do not monopolize system resources; interactive tasks execute within consistent and guaranteed response times; and resources are allocated according to business goals.

zWLM manages the resources of the server to a set of service-level goals that reflect the relative business importance of a variety of workload types to the

organization. User requests are classified, prioritized, and allocated to system resources based on operational policies that are bound to business goals.

zWLM enables the organization to specify a set of work classification rules that identify all incoming requests. These rules can be established using a variety of business-oriented parameters that include transaction name or type, specific user name or user type, and time of day. Each work unit is assigned to a service class, which represents a group of work with similar performance requirements that are expressed in terms of a service-level goal and relative business importance.

Service level goals can be expressed as a factor of response time (which guarantees predictability), execution velocity (which prevents a single application from monopolizing system resources), or discretionary status (enabling low-priority work to execute on a best-can-do basis).

zWLM constantly monitors the health of the overall system, using feedback metrics to continuously manage delays and assess their impact on attaining goals. Resources are balanced and reassigned dynamically and autonomically to ensure that operational policies and business importance objectives are met.

A z/OS image is routinely populated with a heterogeneous mix of applications that typically are managed to different business priorities. The needs of online ordering transactions, databases, batch processes, interactive processing, data mining applications, transaction processors, and so forth are handled by zWLM according to the specified business policies and priorities.

There is no need to constrain the amount of work fed to the zSeries server; as long as important work runs effectively, other tasks can enter the system and utilize excess resources. There is no need to partition the available resources for each separate workload. The system administrator does not have to specify the resource demands of work in advance; effective use of capacity is provided by the management algorithms. zWLM is responsible for reallocating resources to match current requirements.

Note: As with other comparison points made so far, neither approach to workload management is necessarily right or wrong; both pSeries WLM and zWLM were designed with certain assumptions about the optimal workload for their respective servers.

Because the pSeries server is optimized for a workload that favors processor cycles over frequent context switches, the AIX dispatcher naturally uses processor utilization as its criterion for establishing scheduling priority. Because the zSeries z/OS server is optimized for a workload that constantly switches data

context, zWLM focuses on a number of variables that can be set to establish a business goal approach to setting priority.

pSeries dynamically allocates resources in response to internal system utilization rules established by the system administrator. zSeries dynamically allocates resources in order to maintain end user service levels, where minimum acceptable response times and turnaround times are prioritized by business goals and priorities. This results in higher utilization because the system can schedule work on an ad hoc basis to sustain utilization levels approaching 100%.

When facilities run at higher utilization, they need the ability for discretionary workload to be paused autonomically so that the processors can be freed to service higher priority work. zSeries provides this ability.

Note: Higher system utilization does not necessarily translate to lower overall cost or higher performance. Low utilization on a zSeries is costly, and the white space created by low utilization on a pSeries is less costly.

If the deployment platform is cheap enough it may still be more cost effective to run an over-provisioned, less-efficient configuration. However, at times (especially at extremely high levels of scale), the cost of such inefficiency can be prohibitive.

Virtualization: logical partitioning

One final aspect of a scale-up architecture that can have an impact on overall deployment costs is logical partitioning. Logical partitioning of a server's resources among several distinct OS images (or resource virtualization) is another architectural design point that can be employed to improve server utilization by making more workloads available to keep the processors busy. Partitioning is also a means of improving overall availability by eliminating software single points of failure.

Both pSeries and zSeries servers can be logically partitioned. pSeries partitioning technology is delivered by pHypervisor (PHYPE) and is fairly new to the market. zSeries partitioning, delivered via Processor Resource/System Manager (PR/SM™), is a very mature technology that has been deployed in production environments for decades. Some comparisons:

- pSeries CPU virtualization granularity is 1/10 processor, shared/weighted capacity with time-sliced dispatch; zSeries granularity is 1/1000 processor, shared/weighted capacity with preemptive dispatch. Micro-partitioning requires the processors to be in a shared pool, limiting the ability to maintain processor affinity. As a result, micro-partitioning on pSeries is not recommended for applications that use a large amount of memory or are dependent on a high cache hit rate.

- pSeries resource reallocation is weight-driven; zSeries is both weight-driven and goal-based. Both are non-disruptive.
- pSeries logical processors are user-managed; zSeries logical processors are system-managed.
- pSeries physical adapters belong to a dedicated AIX I/O partition, and all virtualized I/O is redirected to the I/O partition with LAN-type latency. In zSeries, the physical adapters can be shared and I/O is sent directly to the adapters with no additional latency.
- The zSeries Intelligent Resource Director (IRD) adjusts logical resources dynamically and autonomically based on WLM and policy. The pSeries Processor Load Manager (PLM) has autonomic levels of control for LPAR adjustment, but this runs separately from the partitioning code itself.
- In a pSeries, virtual processors will run out of their time slice regardless of events. zSeries Start Interpretive Execution (SIE) reduces context switch time during dispatch of virtual processors. SIE exit allows preemption on events.
- zSeries PR/SM recognizes and shares specialty processors, and HiperSockets™ provide the capability for in-memory cross-partition networking.

As with all technologies discussed so far, there are tradeoffs involved with virtualization. Virtualization on its own does not contribute to the ability to scale up, but improper virtualization of a server's resources can actually decrease its ability to scale. By its very nature, virtualization increases the amount of context switching (more work units are vying for the same resources), which in turn puts more stress on caches.

Scaling-out with pSeries

Shared-nothing, massively parallel clusters

In the scale-up discussion, we demonstrated that the pSeries design favors workloads that are fairly homogeneous and do not cause the processors to switch context as often. For this reason, most pSeries deployments limit each server to running a single application. As we look to increase the level of scale by clustering multiple pSeries machines together, this paradigm of one application per machine extends into the cluster as well. The clustering of independent groupings of servers is a *shared-nothing* architecture, as the machines do not share data or state.

Virtualizing WebSphere clusters with a goal-oriented infrastructure

Shared-nothing clustering using banks of replicated servers can be a less efficient means of achieving high availability at scale in certain circumstances. The segregation of servers by application can lead to an imbalance of resources. When one application is busy and another is not, the servers executing the busy application may require additional resources while extra capacity is available on the servers that are lightly utilized.

The design point of the pSeries server, AIX OS and clustering technology, does not support the execution of mixed workloads as a cohesive entity. In order to compensate for this, the WebSphere Application Server Network Deployment (ND) middleware layer has recently been extended with new capabilities, delivered by WebSphere Extended Deployment (XD), which delivers a form of dynamic operations control for WebSphere applications.

In our discussion of scale-up computing we described how zWLM manages z/OS server resources so that a variety of heterogeneous workloads can be integrated within a single z/OS image and managed according to a set of operational policies that are tied to business goals.

The AIX dispatcher manages the resources of the server according to a policy of fairness that does not take business objectives into account. There is no mechanism to manipulate system resources to achieve business goals at the OS level in a pSeries scale-out deployment.

Although it is not possible for middleware to change the way resources are allocated within a given distributed server, it is possible to manipulate workload routing in order to affect the dynamic allocation of physical WebSphere resources by defining pools of servers that can be shared among WebSphere applications. This enables application resources to be allocated to where they are most needed within the context of a grouping of WebSphere applications.

WebSphere XD node groups

Whereas WebSphere Application Server ND applications are deployed directly to a server, WebSphere XD applications are mapped to a server pool, enabling the application to execute on some subset of servers within that pool according to business goals. The collection of machines that can run a given application is called a node group, and one or more dynamic clusters are created within the node group.

The computing power of a node group is divided among its member dynamic clusters. The resources of the node group are dynamically allocated, according to load and policy, to deliver better resource utilization by breaking the physical tie between applications and specific machines.

WebSphere XD provides a means whereby work requests are classified, prioritized according to business importance, and assigned a performance goal.

There are differences from zWLM in the details: for example, for now, WebSphere XD is capable of accepting only HTTP traffic, allows classification based only on transaction type, and does not provide an execution velocity performance goal. Although this scope will likely be broadened in the future, zWLM continues to provide a more granular degree of control.

The WebSphere XD On Demand Router

The XD classification and prioritization mechanism cannot be applied to the management of the resources of a specific server; the AIX dispatcher is simply a fairness-based mechanism. Instead, WebSphere XD manages the queuing and routing of work to the pool of servers available to a set of applications.

The primary mechanism for controlling this routing is the On Demand Router (ODR), a special WebSphere Application Server instance that sits in front of a pool of available WebSphere Application Server servers and represents the entry point into the topology, controlling the flow of requests to the back-end WebSphere Application Server servers.

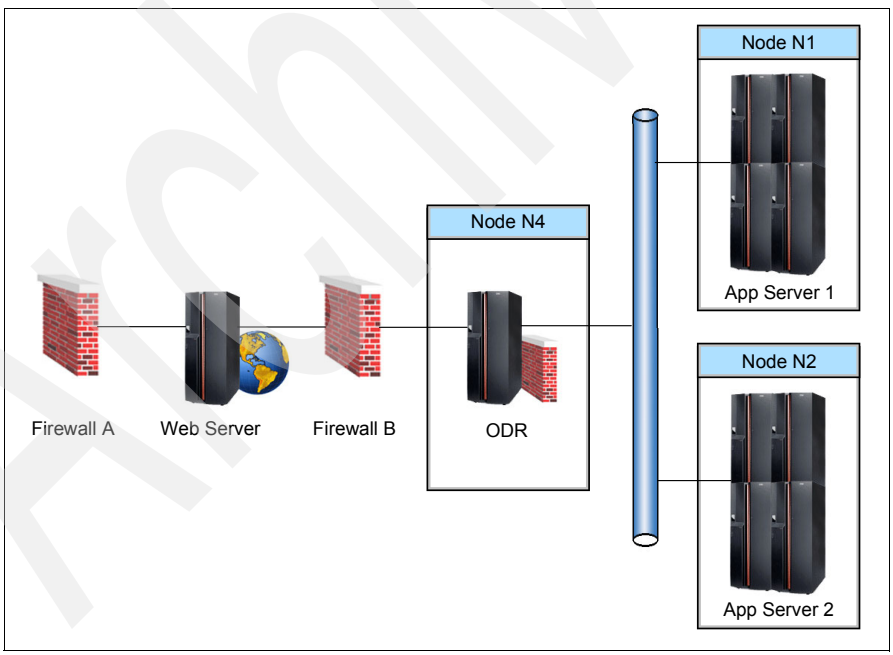

Figure 5 On Demand Router (ODR)

All WebSphere application requests run through the ODR (there can be multiples for redundancy), which classifies, prioritizes, and routes requests to the application servers in the pool, based on operational policies using a weighted round-robin scheduling algorithm. These policies are attached to the applications and reflect application service-level goals and relative importance to the organization.

Applications are monitored to provide input to a set of autonomic managers, which can dynamically adjust server weights and associated workload routing based on actual server performance. Node agents can manage the number of running application instances to satisfy demand.

XD dynamic cluster managers control an application's footprint within a node group, expanding or contracting dynamic clusters as needed to meet changing demand. When the need for demand distribution exceeds the capacity of a WebSphere node group, the ability to compensate extends beyond the boundary of the XD autonomic managers.

In order to extend the influence of the ODR beyond the pre-defined WebSphere pool, the Tivoli® Intelligent Orchestrator (TIO) can be employed to enable the XD topology to acquire more physical computing resources from some other resource pool (a pool of WebSphere MQ servers, for example).

In a sense the WebSphere ODR becomes a local manager, optimizing the flow of work to the local resource pool, and TIO becomes a global manager, optimizing across heterogeneous resource pools and provisioning resources into the WebSphere pool, if needed.

WebSphere XD introduces a goal-oriented infrastructure that is similar in its externals to that provided by zWLM. While XD improves the utilization of resources in a scale-out configuration of WebSphere servers, zWLM provides more-granular management of resources.

Partitioning applications for high performance

In a distributed, shared-nothing architecture, the back-end database can become a bottleneck to scaling (Figure 6 on page 24), in high-volume applications that intensively read and write to databases and require the utmost in data consistency and availability.

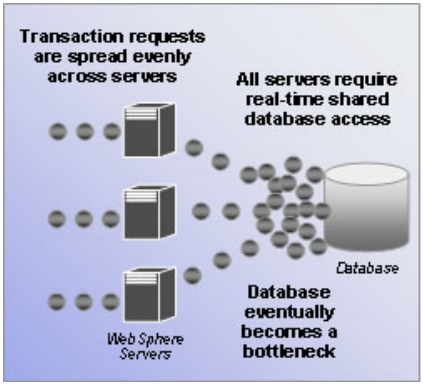

Figure 6 Database bottleneck

Distributed topologies are optimized for read-mostly environments where data caching capabilities must be used to offload the back-end database systems. However, as systems observe increased data-write ratios, these caching systems start to break down because of the strenuous task of maintaining consistency between the cache and database. These schemes can quickly reach a point of diminishing returns where it becomes useless to cache and better to send all traffic back to the database, enabling the database to manage consistency. This strategy can sometimes lead to large and costly database configurations.

Partitionable data

When an application's data access pattern is *partitionable* (such that multiple semi-independent servers can take pieces of the application and operate on the problem without interference between the application servers), employing an application partitioning pattern can offload the database by enabling the application server tier to act as a buffer.WebSphere XD provides a mechanism for specifying and managing dynamic data partitioning and re-partitioning to meet a desire for constant response times, trading off throughput if necessary.

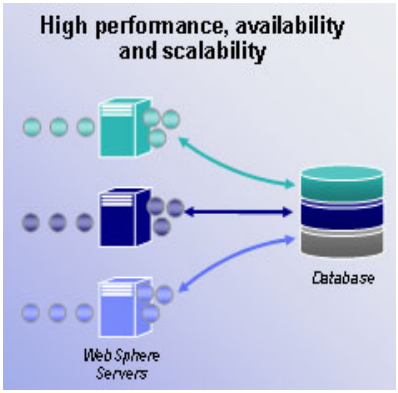

Figure 7 Database partitioning

Exploiting this capability requires that the application programmatically specify its partitioning boundary to WebSphere XD.

Note: This has an effect on the portability of the application, as the XD APIs that enable the specification of a partition boundary are not part of an open standard).

The XD On Demand Router is partition-aware. It uses this information in the scheduling of work.

Applications can be designed to exploit the partitioning pattern in several ways. The most popular uses of the partitioning pattern include:

Caching Leverages the exclusive nature of a partition by aggressively caching data.

Batching Queues requests in local memory, then sends to the server in a batch, reducing remote interactions with the database.

Singleton services Enables applications to be divided by function or service and run across the cluster.

To deal with load imbalances, it is sometimes necessary to physically relocate a partition to a more capable server in order to rebalance the load. Rebalancing is similar to the processing of a server process failure, although it is a controllable process.

The WebSphere XD High-Availability Manager

A partition can be a single source of failure. For example, if a caching partition fails and there is no backup, a considerable amount of cached data can be lost. Failure of a singleton service partition can lead to unavailability of the service.

To compensate, WebSphere XD utilizes the facilities of the WebSphere Application Server ND high-availability manager. Each server process in a WebSphere Application Server cluster maintains an awareness of the availability of the other processes in the cluster. If a server process hosting a partition fails, the surviving processes work together to recover the process, its services, and its partitions.

A core group is a high-availability domain that consists of all Java Virtual Machines (JVMs) in the cluster, including the deployment manager and node agents. Communication among core groups is made possible by a gateway service that is fault-tolerant and scalable. Members of a core group communicate using a proprietary message subsystem that can be configured to use multicast or TCP/IP. The JVMs transmit a heartbeat signal to all other JVMs every second. The proper actions are taken when a signal from a JVM is not received for a configured duration.

Application partitioning is not useful in all situations. This pattern is effective when the number of partitions exceeds the number of servers. The partitioning scheme must be granular enough to spread the partitions among the available server resources; to ensure that the goals for individual partitions are met, a 1-to-1 ratio of partitions to servers is not fine-grained enough.

If the objectives cannot be met with the available servers in the pool, then new servers must be provisioned and added to the pool, at which point the load is redistributed transparently without interruption of service.

Some partitions are inherently single-threaded; they are ultimately limited to a single server instance. In these circumstances if a server becomes saturated it is not possible to provision additional servers to service that partition. When this situation is detected, incoming messages are filtered and rejected in order to maintain the goal for the work already executing in the server (sacrificing throughput in order to maintain a consistent response time).

The partitioning pattern is not a general mechanism for improving scale; it is a very focused solution to problems that are inherent with a specific application pattern that can provide scaling relief for that pattern when used with care. A very careful application study, full understanding of the benefits and risks, and willingness to utilize IBM-specific programming constructs is necessary for success.

Scaling-out with zSeries: the shared-everything cluster

zSeries high-availability clustering: Parallel Sysplex

zSeries z/OS takes a decidedly different approach to clustering, making a multi-server scale-out topology look and behave like a massively parallel SMP scale-up configuration. However, by spreading a large number of processors across multiple SMPs, this clustering technique avoids scalability problems inherent in supporting the same large number of processors on a single SMP where performance is limited by the overhead of interprocessor communication.

zSeries clustering technology, Parallel Sysplex®, is based on a Non-Uniform Memory Access (NUMA) star clustering topology. This multi-system data sharing technology enables up to 32 z/OS systems to be clustered with near-linear scalability.

Coupling facility

The heart of the Parallel Sysplex is a special, high-speed hardware construct, the zSeries Coupling Facility, which provides very fast communication among SMPs with latency of microseconds (close to real-time memory access) versus milliseconds in a message-passing architecture. The Coupling Facility also maintains global cache, global lock manager, and a shared communications area, all of which can be accessed and shared by z/OS subsystems.

The net effect is that every system in a Parallel Sysplex cluster has access to all data resources; the Parallel Sysplex enables direct, concurrent read/write access to shared data from all processing nodes in the configuration without sacrificing performance or data integrity. Each node can concurrently cache shared data in local processor memory through the hardware-assisted cluster-wide serialization and coherency controls provided by the Coupling Facility.

This *shared-everything* approach enables workloads to be dynamically balanced across all servers in the Parallel Sysplex cluster. Because all data required by an application is shared among all members of a cluster, critical business applications such as WebSphere transactions or database queries can take advantage of the aggregate capacity of multiple servers to help provide maximum system throughput and performance during peak processing periods.

A Parallel Sysplex is not a failover environment where extra resources sit idle; the shared data design enables all capacity to be brought to bear on productive work. In the event of failure, high availability managers work in conjunction with zWLM to ensure that business objectives continue to be met within the limits of remaining capacity.

In terms of achieving availability at scale, these same mechanisms work together to deliver resources to the workloads that need it at any given moment, autonomically adjusting the balance according to policy and without user intervention or application awareness. By taking direct advantage of this highly scalable and available clustering architecture, the WebSphere Application Server for z/OS is fully aligned with the heterogeneous, data-intensive design point of zSeries z/OS Parallel Sysplex clusters.

Virtualizing WebSphere applications with a goal-oriented infrastructure

As has already been described, z/OS includes a time-tested, goal-oriented infrastructure for workload management in zWLM. This infrastructure is already available at the zSeries server and z/OS layers, so it was not necessary to build these capabilities into the WebSphere middleware for z/OS. WebSphere Application Server for z/OS was architected specifically to take advantage of the zSeries shared-everything Parallel Sysplex clustering and z/OS goal-oriented workload management environment.

By choosing to rely on what already existed in the base hardware and OS, WebSphere Application Server for z/OS was further able to exploit the heterogeneous nature of zSeries scaling technologies so that a tight integration could be effected between WebSphere and the traditional z/OS resource managers (CICS, IMS, DB2, MQ, and so forth).

There is no need for a workload manager at the WebSphere middleware layer as with WebSphere XD; goal-oriented workload management is already incorporated into the z/OS and is exploited by the WebSphere Application Server for z/OS product.

A key difference in the architecture of the distributed WebSphere Application Server and the WebSphere Application Server for z/OS product lies in the process structure of a WebSphere Application Server server. On distributed platforms, each WebSphere Application Server server is a single process. On z/OS, a WebSphere Application Server server consists of a controller process and some number of subsidiary servant processes.

Within the context of this multi-process design, the controller performs the function of an on demand router (among many other tasks), accepting work requests and using the goals-directed infrastructure of zWLM to queue work to the servants that it manages. When queued, zWLM autonomically balances the work among the servants according to policy and business importance.

A loose analogy can be drawn to the WebSphere XD ODR. There are some very important distinctions, however.

XD is a workload routing technology that balances the utilization of a number of physical servers, using a special WebSphere Application Server server instance (the ODR) to route work requests to back-end WebSphere Application Server servers residing on some number of machines configured in a server pool. There is no way to fine-tune the resource utilization of a single box; after workload reaches a specific physical server, the workload management capabilities of the base OS take over.

XD manages where work gets scheduled, but when work is dispatched to a server, XD has no control over the resources that are available to execute that work other than to control how much additional work gets sent to that box.

WebSphere Application Server for z/OS uses the zWLM goal-oriented infrastructure to route work requests to different processes within a server, providing a fine-grained approach to workload management.

The goal-oriented infrastructure of zWLM is uniformly extended to all workload types that run within a z/OS system. This effectively provides two levels of resource management within the OS:

Within WebSphere, work requests of different types (long and short-running transactions, for example) are properly managed against each other.

Within a z/OS system, the WebSphere work is properly managed against other workloads that may be running in that system (CICS transactions and DB2 database requests, for example).

Because the concept of z/OS workload management is consistent from the server through the OS and middleware, workload routing and rebalancing is only one aspect of this solution. When the work lands on a particular server, autonomic server resource adjustments that are needed to react to workload spikes are coordinated across WebSphere and the resource managers. This is made possible because WebSphere application service levels and goals are integrated along with other applications and workloads that are already running on z/OS.

zWLM has direct control over system resources, such as I/O, storage, and CPUs, and can reallocate resources to the workloads that need them. zWLM provides the goal-oriented advice about where work should run, then balances available system resources across multiple types of work based on service requirements and importance to the business.

zWLM provides a mechanism for heterogeneous workload management, and WebSphere Application Server for z/OS was designed to work within this framework.

This goal-oriented infrastructure is not confined to a single z/OS system. zWLM is sysplex-aware, so that the same goals are applied consistently across an entire Parallel Sysplex cluster.

To further extend the basic concepts of zWLM described earlier into a multi-server cluster, resource groups can be used to carve up the resources in the Parallel Sysplex among different applications or groups of users.

A resource group guarantees that a type of work always gets a specific minimum percentage of all available resources in the cluster, and prevents that group from exceeding a specified maximum of resource usage. A service class can be assigned to a resource group, and zWLM satisfies service class goals within the bounds of the associated resource group.

As was the case in a single system environment, intelligent resource management in the Parallel Sysplex extends beyond the concept of workload routing. zWLM works with a component called the Intelligent Resource Director (IRD) to extend the single-system, goal-oriented infrastructure management of system resources, throughout the shared-everything cluster.

This provides a third level of resource management: the goal-oriented and autonomic control of resources across multiple physical servers in support of a heterogeneous workload.

By fronting a WebSphere Application Server for z/OS installation with a Sysplex Distributor or other zWLM-enabled router, incoming WebSphere work is routed to the zSeries server and z/OS image that is best able to complete that work according to business objectives. Then within the WebSphere Application Server server, it is routed to the servant process that is best able to complete the work within the specified goal.

The net effect is that the shared-everything design point of the Parallel Sysplex enables up to 32 z/OS images to be clustered in a computing facility that can be viewed as a single logical resource to end users and business applications. The mapping of WebSphere Application Server and the z/OS resource managers to the same goal-oriented infrastructure enables this architecture to support the tight binding between tiers of an application (such as the business logic and critical resource and data managers that are already deployed on z/OS).

zWLM and IRD collaborate to direct work to any node in a Parallel Sysplex cluster that has available capacity, and resource balancing permits a diverse set of applications to be run across a Parallel Sysplex cluster while maintaining response levels and meeting business goals.

The zWLMm style of workload management is optimizing the hardware resource, as it is moving work to where resources are available and then manipulating the resources to meet business goals.

Although continuous availability is not the focus of this material, it must be noted in passing that in the event of a hardware or software outage, either planned or unplanned, workloads can be dynamically redirected to available servers, thus providing near-continuous application availability.

Because all systems can be doing work simultaneously, in a failure scenario the other instances can absorb the work because in a truly heterogeneous application environment the zWLM and IRD can distinguish and dynamically adjust resource deployment to meet the goals of critical work.

Note: It is possible that at some time in the future the WebSphere XD ODR technology could be made available on z/OS because it might add some incremental value to a z/OS solution.

An XD-like ODR would provide an alternative front-end routing mechanism that would further enhance the fine-grain capabilities already provided in the z/OS application server, and add a level of systems management consistency with distributed WebSphere XD deployments.

Sharing data for high performance and availability

The application partitioning pattern implemented by WebSphere XD was developed in response to an inherent weakness of distributed application and data topologies. In distributed topologies, large databases tend to be partitioned across banks of dedicated physical servers, and application server pools do not have direct access to the data. (All database access is achieved through remote calls.)

As was discussed earlier, caching is utilized extensively to offload work from the database and minimize network traffic. This technique works well in read-mostly scenarios, but becomes a liability as the level of database writes increases; hence the need for a partitioning pattern to relieve scaling constraints.

A WebSphere Application Server z/OS Parallel Sysplex configuration with DB2 data sharing does not experience this type of constraint to scale. Data sharing enables applications executing against multiple DB2 subsystems to have full, concurrent read/write access to the same set of data while maintaining data integrity.

Because all WebSphere Application Server z/OS application servers within the sysplex configuration have full and direct read/write access to all data, there is no

need to partition data or applications among individual nodes in the cluster or to replicate databases across multiple servers.

In fact, this approach does not make any assumptions at all about the structure of the application. No application modifications are required in order to benefit from DB2 data sharing, making it a very flexible environment in the types of workloads that it can run.

Data sharing within the Parallel Sysplex also provides a significant availability benefit, protecting against loss of data and loss of availability in light of planned or unplanned outages by enabling workloads to be shifted among the nodes of the Parallel Sysplex as needed to maintain business goals.

Note: Unlike the XD ODR, which could provide incremental benefit if made available on WebSphere Application Server for z/OS, implementation of the application partitioning pattern would be a weak substitute for the data-sharing technology already in place that makes partitioning unnecessary and irrelevant.

While partitioning might slightly improve performance on z/OS by lowering the rate of cache misses, the expected improvement would be small.

Summary

Deciding to deploy a new application to WebSphere Application Server is an easy choice. Deciding where to deploy these applications is a somewhat more complicated endeavor. Unfortunately, there is no universal formula that is easily followed that will lead one to the proper deployment platform because different organizations value different qualities.

In this material we focused on one particular quality, the ability to maintain application availability to the user at high degrees of scale. Our focus on scale stems from the understanding that cost of operations is a critical issue with many IT shops these days and heavily influences deployment decisions.

At the same time, the way in which many organizations go about comparing deployment costs does not always factor in the real cost of achieving scale in an enterprise environment. A cost analysis that focuses on individual, self-contained projects, outside the context of how a variety of projects must interact with each other and with legacy resources within the enterprise, can easily lead to sub-optimal results.

More than any other factor, the characteristics of the application and data access patterns will have a great effect on how well a deployment will scale—and consequently how much it will cost—in a production environment.

We cannot provide a cookbook for accurately measuring the effects of scale in an enterprise environment; the enterprise-to-enterprise variability of today's on demand environments puts such an effort beyond the scope of this material.

Instead, we have written a technological comparison of the processor, operating system, and middleware characteristics that have a direct effect on how well a WebSphere deployment might be expected to scale. We hope that by understanding the underlying factors involved, this material can aid in the development of an effective means of measuring the true cost of providing availability of applications at scale in an enterprise environment.

Our analysis focused on a comparison of capabilities of a WebSphere XD deployment to AIX on pSeries against those of a WebSphere Application Server ND deployment to z/OS on zSeries. These two solutions are significantly different in nature by design, and these differences stem from architectural choices that have been made in the server hardware, operating system, and WebSphere middleware.

pSeries

At the hardware level, pSeries was architected to optimize CPU performance. This architectural choice highly favors workloads that exhibit repetitive, predictable data access patterns: patterns that tend to lead to a single-application server deployment model across banks of distributed replicated servers. Consequently, the AIX approach to workload management views the workloads on a server as having a uniform importance and manages resources based on a policy of fairness.

In an on demand environment where resources must be brought to bear autonomically against an unpredictably changing workload mix, the pSeries AIX architecture will require white space to accommodate for spikes in each individual application. Depending on the expected level of scale and the variability of the workload, the cost of this underutilized capacity can vary.

To improve the utilization of resources in a distributed WebSphere deployment, WebSphere XD introduces technology that lays an importance-based workload routing construct on top of this fairness-based resource management platform to enable WebSphere applications to be deployed to a pool of servers rather than tied to dedicated servers. WebSphere XD also introduces technology to implement an application partitioning pattern to compensate for constraints in distributed data access patterns that are write-mostly, which can help smooth out

the effect of data access hot spots that are inherent in a shared-nothing clustering architecture.

The pSeries WebSphere XD deployment to AIX has been architected to deliver exceptional performance at an attractive acquisition price point for workloads consisting of very focused, highly parallelizable or somewhat stateless applications in which the data tier is shared by multiple logic tiers and that data is easily partitionable. Such a deployment will prove very cost-effective and highly scalable.

zSeries

At the hardware level, zSeries was architected to optimize data access and RAS at the expense of CPU performance. This architectural choice favors workloads that exhibit unpredictable application and data access patterns with high levels of dynamic data sharing and random access memory: patterns that encourage the consolidation of heterogeneous workloads to a single server. Consequently, the z/OS approach to workload management differentiates the workloads on a server according to priority and manages resources based on a policy derived from business goals and relative importance.

The zSeries z/OS architectural design point of optimization for mixed workloads is well suited to the variability and unpredictability of the on demand environment. The shared-everything Parallel Sysplex cluster and importance-based resource management technologies of zWLM and IRD were specifically designed to limit the need to overconfigure resources; resource is autonomically made available to the work that needs it regardless of changes in demand patterns.

Because WebSphere Application Server for z/OS was specifically architected to leverage this environment, the zSeries WebSphere Application Server deployment to z/OS exhibits a tightly linked importance-based workload routing and resource management capability tuned to the needs of a heterogeneous, data-intensive workload.

The responsiveness and stability that result from a tight integration of resource management and feedback paths from the native hardware, OS, and clustering technology right through the application server makes for a deployment that is cost effective and highly scalable for heterogeneous workloads.

Evaluating comparisons

Although the technological review points to clearly differentiated value propositions for these two deployments, actual evaluation is not a straightforward task. pSeries is groomed to the individual application and WebSphere XD is similarly a WebSphere-only solution, so an evaluation process that is isolated in scope (at a project level) will nearly always favor a pSeries deployment from a strict price/performance perspective, although other factors beyond the scope of

this material (such as security) also play a part in platform selection). However, unless the application or WebSphere suite is likely to run in this same environment in production, more work must be done to understand whether the target production environment changes the application patterns.

The zSeries, z/OS, and WebSphere Application Server for z/OS value proposition is not always apparent when applications are evaluated on an individual basis. The optimizations that are targeted to heterogeneous workload deployments are not valued in the analysis of homogeneous projects.

It can be dangerous for an enterprise with complex IT interrelationships to evaluate application deployments in a vacuum; it is critical to take an enterprise view when choosing deployment platforms to ensure that the choice is optimally matched to the enterprise application and data access patterns.

One final point

For the current zSeries customer looking to integrate existing resources into new, WebSphere-based business processes, there is a natural value in deploying the WebSphere layer directly to z/OS. Assuming that an evaluation of application patterns does not overwhelmingly favor one platform over another, the zSeries deployment provides many benefits that were not discussed in this paper, such as:

- The elimination of network calls by having the application close to the data.
- Enhanced data consistency via local, optimized transaction control.
- Leveraging existing investments in zSeries z/OS manageability, reliability, availability, security, and such.
- The ability to take advantage of virtualization technology without reducing overall system effectiveness.
- Increased speed of EJB calls through the ability to *pass by reference*. This eliminates the overhead of serialization and deserialization.

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Rochester Center.

Paul DiMarzio is a Consulting Software Product Design Professional with the IBM Systems and Technology Group zSeries Brand Organization and is the On Demand Business Integration Strategy Manager. He has 20 years of experience with IBM mainframe operating systems development, design, and marketing, with a focus on new and emerging application technologies for the past 10 years.

Brent Watson was an eServer IT Architect with the IBM Client Technology Center (CTC) - zSeries e-business Services team. He has a strong background in IT consulting and entrepreneurship with technical expertise in J2EE and .NET software architecture, application servers, portal implementation, and business intelligence solutions architecture. He holds a BS in Computer Science from Clarkson University. He recently left IBM to work with a small startup company.

Patrick C Ryan is a Senior eServer IT Architect with the IBM Client Technology Center (CTC) - zSeries e-business Services team and currently a Project Leader with the ITSO, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide about all areas of the z/OS WebSphere Application Server. He has more than 30 years of experience with IBM mainframe operating systems, and has spent the past five years as an e-business consultant, dealing with the WebSphere Application Server on z/OS.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	Hypervisor™	Redbooks (logo) ™
BladeCenter™	IBM®	Tivoli®
CICS®	IMS™	WebSphere®
DB2®	iSeries™	xSeries®
@server®	Parallel Sysplex®	z/OS®
eServer™	PR/SM™	zSeries®
HiperSockets™	pSeries®	

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.