

Optimizing HP Servers with Microsoft SQL Server 2008



SQL Server instances.....	2
Processor affinity	4
Degree of parallelism	8
Memory management	8
Hot addition of memory.....	11
Hot addition of CPUs	11
Windows System Resource Manager	12
Resource Governor	15
Defining the workload	16
Creating the classifier user-defined function	16
Registering the classifier function in Resource Governor	17
Testing a resource pool with Resource Governor	17
NUMA architecture.....	22
Windows and NUMA	24
SQL Server 2008 and NUMA	24
NUMA software	29
Storage and NUMA architecture	31
Partitioning and virtualization.....	33
Hard partitioning	33
Pay per use	33
Virtualization	34
Conclusion.....	36
For more information.....	37

Overview

When implementing a hardware consolidation project, one major step involves consolidating corporate data, which is mostly managed by multiple database engines. This activity requires high-performance servers with multiple CPUs and addressable memory capacity. Experience shows that the average size of databases grows exponentially, with the result that business processing applications like data warehouse and business intelligence become more demanding and complex.

Some studies on consolidation of corporate data demonstrate that stand-alone SQL Server implementations are, on average, underused. It is easy to understand why we have this situation in many data centers. A few years ago, SQL Server was running only on 32-bit platforms. The resources available on these servers were not sufficient to host multiple instances, especially in regard to the memory requirements. Because we had to be sure that the resources were available during the peak period, servers were being underused most of time.

With the introduction of SQL Server 2000, then with SQL Server 2005, and now with SQL Server 2008, Microsoft is providing its database engine in a 64-bit version. Processors, such as the Xeon and the Itanium from Intel, or the Opteron from AMD, provide 64-bit capabilities that can be used by Windows Server and SQL Server.

Combined with Windows Server 2008, SQL Server can now be deployed on enterprise-class servers. Today Windows Server can scale up to 64 threads and 2 TB of physical memory. These numbers might even increase in the upcoming years.

With 64-bit configurations, consolidating the corporate data on one single device is now a valid option. The financial benefits are significant: a lower total cost of ownership and a higher return on investment. From the technical standpoint, having multiple databases exposed through multiple database engines presents new challenges: first, the data isolation must be guaranteed between the applications. Grouping all the data onto one server also requires a very high level of availability. Another challenge is to guarantee suitable response times for all the applications that will be accessing their data from the same server. Resource management technologies are essential features to ensure that the service level agreement (SLA) is fulfilled and the resources are well optimized and dispatched according to the need.

The objective of this whitepaper is to show how Microsoft SQL Server 2008, specifically on 64-bit platforms, together with the resources and technologies available on HP servers offer unique solutions for consolidating corporate data.

SQL Server instances

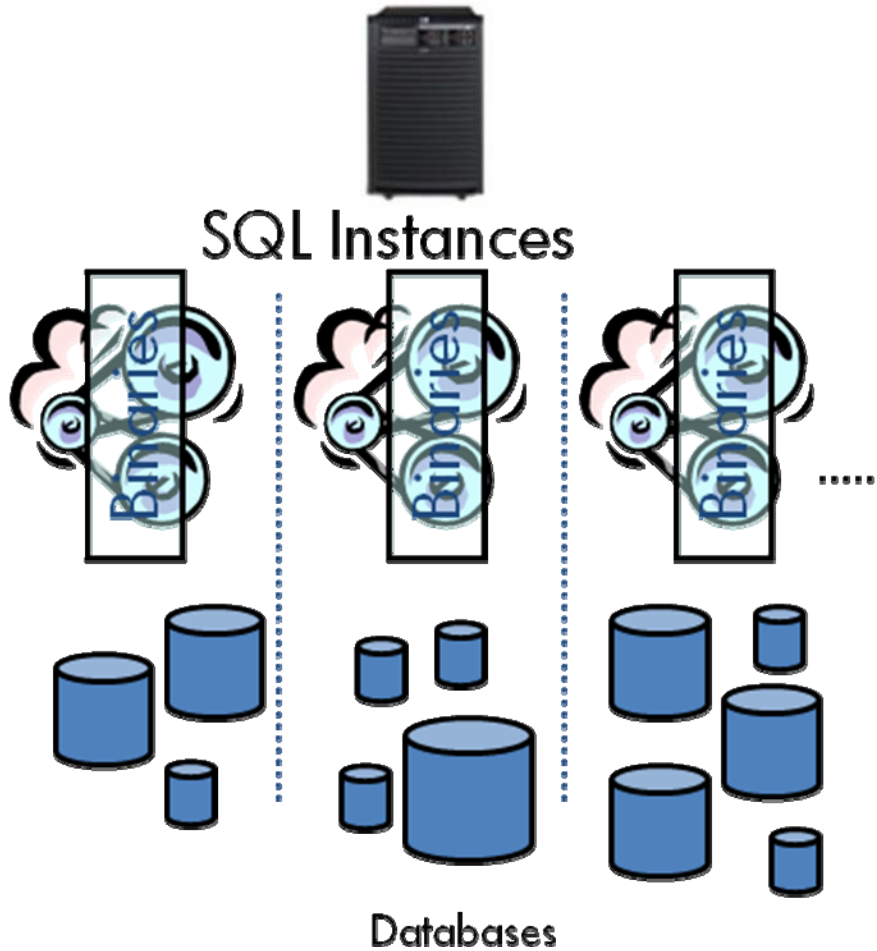
One of the significant SQL Server capabilities is the ability to run multiple database engines (instance) on a single server. The concept of instances was introduced with SQL Server 2000. It enables a server to manage multiple database engines, each having its own dedicated binaries, databases, and settings. This feature could not be implemented on a 32-bit platform because of limited resources. When SQL Server 2000 was released for the Itanium II-based platform (64-bit), multiple SQL instances became feasible. Previously limited to 16 instances, SQL Server 2005 extended this number to 50 with the release of Enterprise Edition. SQL Server 2008 will provide the same capabilities.

Our experience shows that more than 10 instances running on one stand-alone server has an overhead in terms of resource utilization. From a performance standpoint, a best practice is to have one instance per type of workload: for example, one instance to consolidate online transaction processing (OLTP) databases and one instance for the data warehouse. From a resource management

standpoint, an instance provides a good level of granularity. This is discussed in detail in the following sections.

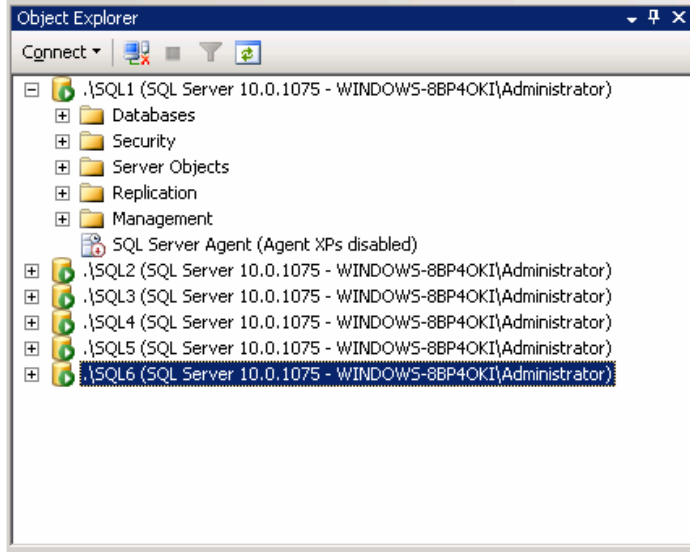
Figure 1 shows a server running multiple instances: the databases are dedicated to a given instance and the level of isolation (security) is guaranteed with the SQL credentials.

Figure 1. SQL Server instances



Multiple instances can be managed from a single SQL Server Management Studio console as shown in Figure 2.

Figure 2. SQL Server 2008 instances registered in SQL Server Management Studio



Note that in an active/active cluster configuration, a node failure will result in two SQL Server instances running on the same device. This scenario could generate some challenges regarding resource management that are addressed in the section on Memory management.

Processor affinity

By default, SQL Server can potentially use all of the server's memory and CPU resources. You can edit the instance properties and check the box for the "Automatically set processor affinity mask for all processors" option to allow the CPU's resources to be shared when running multiple instances on one server. This could, however, impact the response time of one of the database engines because some of the resources could be used by another instance. The quality of service is then very difficult to guarantee.

Instead of allowing the access to all the CPUs for all the SQL Server instances, you can limit the number of CPUs assigned. With this feature, you can dedicate CPUs for each instance. A benefit of CPU affinity is the ability to change the number of CPUs assigned to an instance dynamically without having to restart the instance. If you are not satisfied with the average response time of an instance, you can improve performance by removing CPUs from another instance and adding them to the slower instance. This configuration change can be achieved with SQL Server Management Studio by editing the instance properties (see Figure 3) from the command line or by using a SQL Agent (see Figure 4).

Figure 3. Processor Server properties

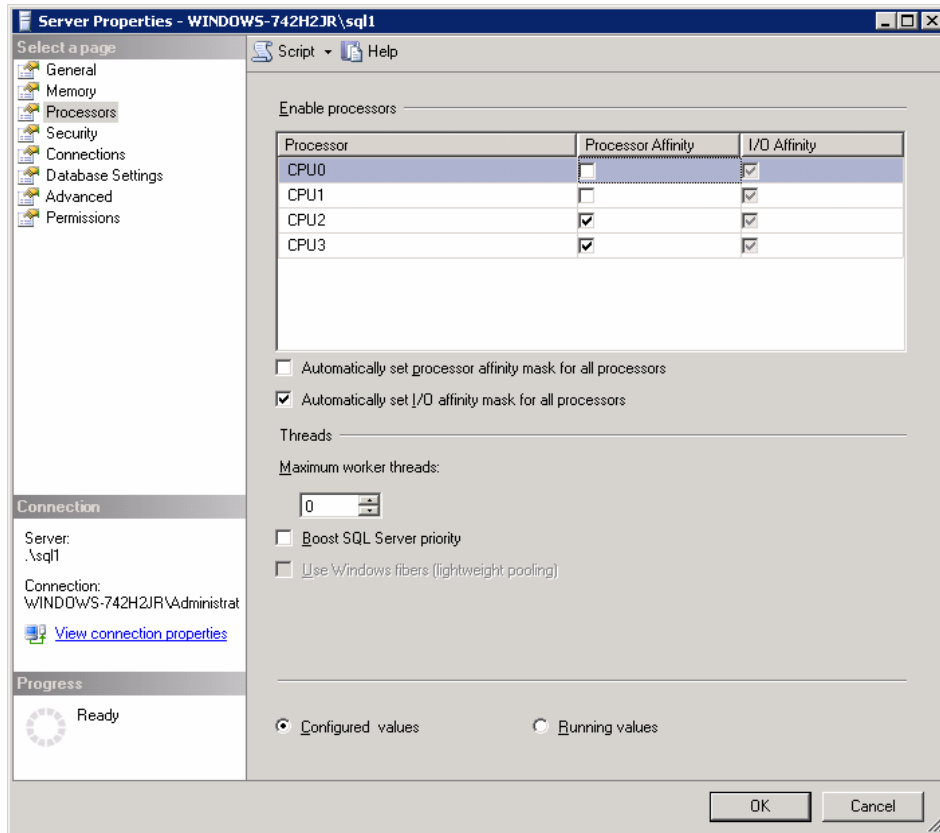
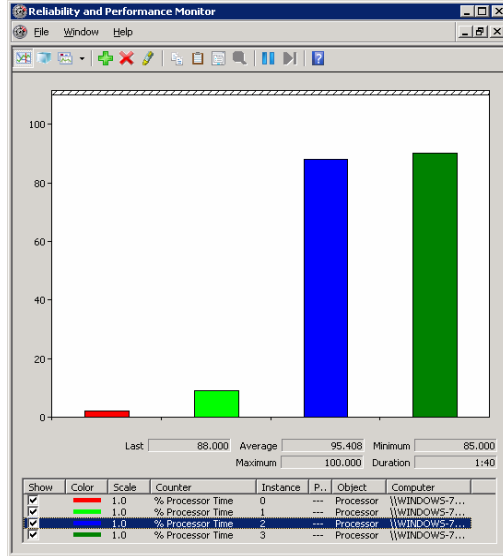


Figure 4. Changing CPU affinity with SQL

```
sp_configure 'show advanced options', 1
reconfigure
go
sp_configure 'affinity mask', 3
reconfigure
go
```

Figure 5 shows the CPU utilization of a server running SQL Server under load where only the two CPUs assigned with CPU affinity are in use.

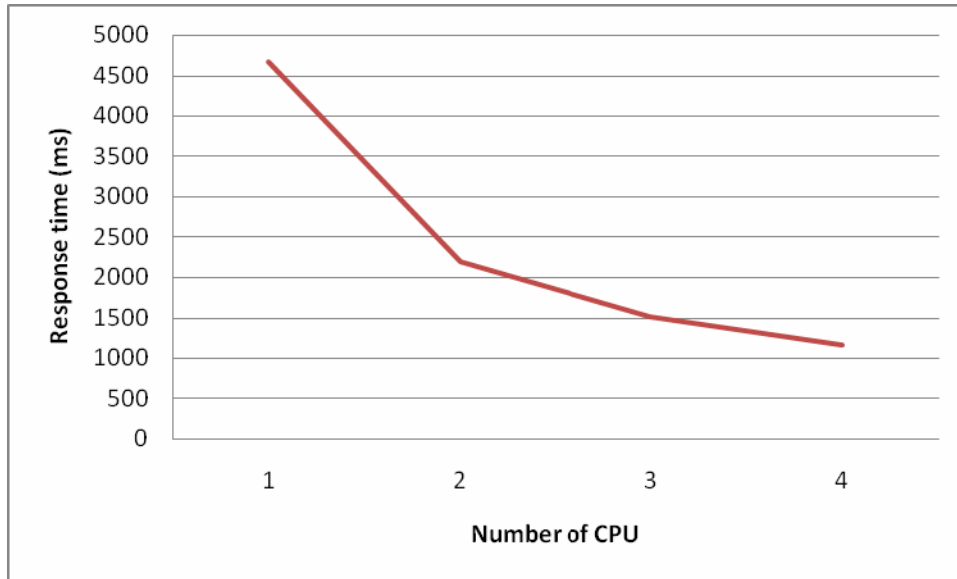
Figure 5. Performance monitor showing the SQL workload on the last two CPUs



Note that if SQL Server is licensed per processor (see Processor Licensing Model), the addition of CPUs with the CPU affinity feature may impact your licensing agreement.

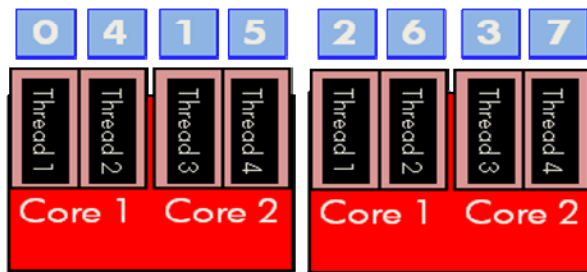
In Figure 6 we show the effect of the number of CPUs assigned to the SQL Server instance on the response time of a given query. Obviously the response time decreases while we add CPUs. This statement is true only if the SQL Server engine considers that the ideal number of CPUs to process the query is equal to the number of CPUs available on the server (degree of parallelism equals 0 (default) or equals the number of CPUs). For an OLTP workload that is executing one single thread, the CPU affinity will have no influence on the response time. In that case, we need to consider executing multiple threads concurrently so that the overall response time will be improved.

Figure 6. Response time of a given query based on the number of CPUs assigned with CPU affinity



Ideally, the CPU affinity should take into account the hardware layout of the server. The Non-Uniform Memory Access (NUMA) architecture is used on some HP Integrity servers; therefore, it does not make sense to set the affinity for a SQL Server instance to two processors that do not belong to the same NUMA node. (This is discussed in greater detail in subsequent sections.) Note that when hyperthreading technology (HT) is enabled on some Intel processors, the processor enumeration performed by Windows Server is in a *logical* order. This fact should also be considered when you are deciding whether to use CPU affinity. Windows Server 2008 will enumerate a server with two Itanium II dual-core and HT-enabled processors as shown in Figure 7.

Figure 7. Threads enumeration on a two Intel Itanium II dual-core server with HT enabled



In a cluster, you will have to adjust the CPU affinity settings to accommodate failover. The remaining nodes will have more workload to manage, especially if you have an active/active configuration. You may need to dispatch the remaining resources to fulfill your SLA after the failover event.

Degree of parallelism

Each time a query is executed, the SQL Server engine, by default, will detect the best degree of parallelism (DOP), that is, it will determine the ideal number of processors required. The DOP is based on the estimated elapsed time or cost of the different tasks the engine will have to process. In most configurations the DOP is not controlled by the database administrator. For high-end configurations, it is informative to force the number of processors to be used for two reasons:

- To determine the ideal DOP for a given query, the engine needs to calculate the cost of the query. This process could generate an extra cost, especially with an OLTP workload where typically only short T-SQL statements (inserts, updates, and deletes) are found.
- To control the resources assigned to a workload, for example, if you do not want complex queries to use all the CPU resources of your SQL Server.

The DOP is usually defined at the instance level. It is obviously not the best option because not all the queries will need the same numbers of CPUs to be processed. Another option is to define the DOP in the query itself using query hints:

```
select count(*) from dbo.telco_fact32 option (MAXDOP 1)
```

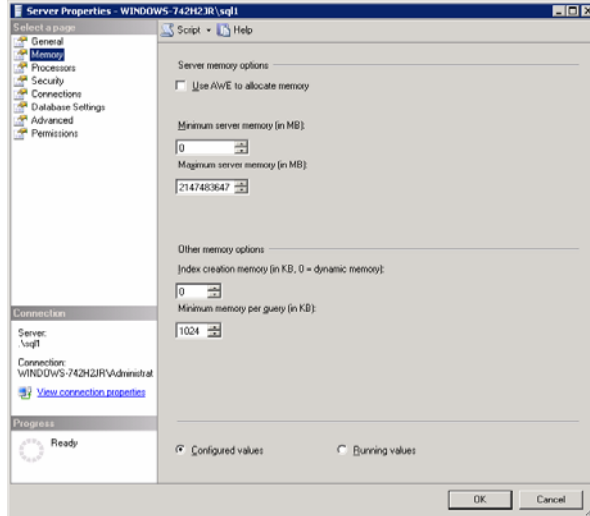
This could be challenging because you must have control of the application. The last option is to use plan guides to force the DOP for previously identified queries.

Memory management

In a multiple instance scenario, memory allocation should be controlled and managed. The SQL Server engine has the tendency to use all the memory available within the server. The presence of multiple instances will lead to a permanent fight between the different dynamic memory managers.

One method to resolve the situation, without considering resource management, is to limit the amount of memory presented to each instance. Two parameters are available in SQL Server to control the minimum and the maximum memory assigned to the buffer pool (see Figure 8). These two parameters can be changed dynamically to reassign the memory, according to the need of the different instances. A best practice recommendation is to leave 10% of the memory available to the operating system. Consequently, in the case of multiple instances, the sum of the maximum memory should remain below or equal to 90% of the entire physical memory available on the server.

Figure 8. Memory settings of the SQL Server instance properties



In order to avoid SQL Server page outs by the operating system when the application is in a state of memory pressure, the “Lock pages in memory” policy should be enabled on the Windows setting for the user defined to run the SQL Server Service. Figure 9 shows the group policy editor and highlights the “Lock pages in memory” policy.

Figure 9. Group policy editor to edit the “Lock pages in memory” setting

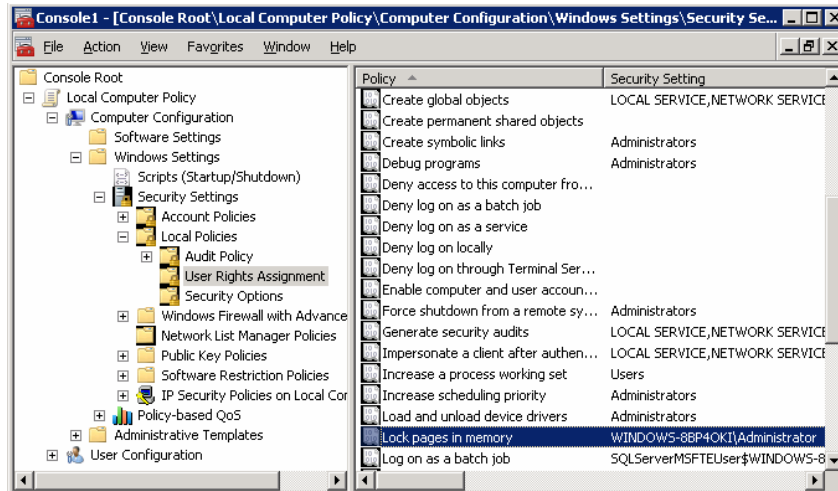


Figure 10. Impact of maximum memory on I/O

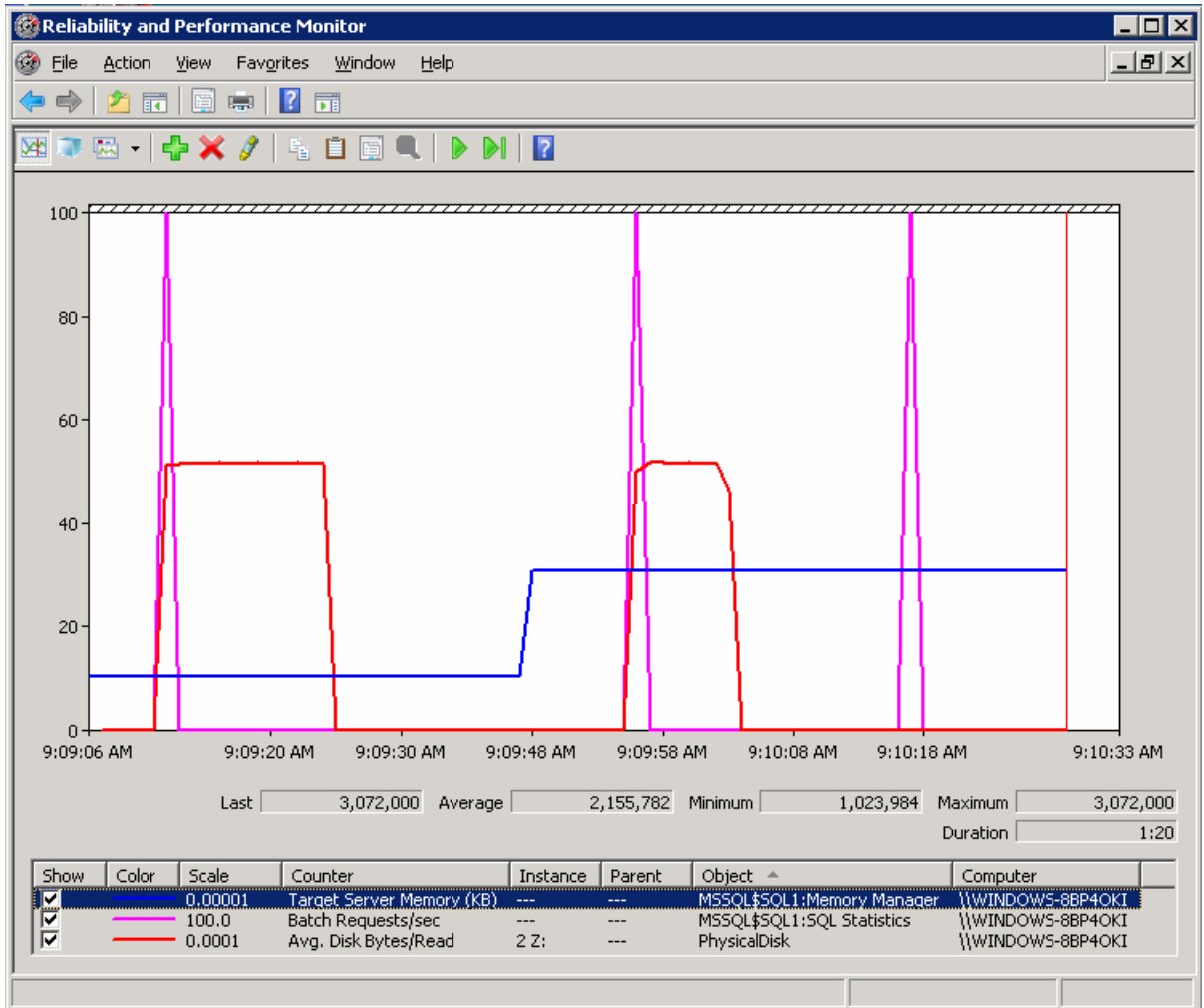


Figure 10 shows the impact on the disk I/O when changing the SQL Server maximum memory setting (Target Server Memory counters). In this test, we ran a query (SELECT statement) targeting a 2-GB database table. Table 1 gives the response times obtained. The first result was collected with a maximum memory setting of 1 GB. Since the table needs to be loaded from disk to memory, we noticed some I/O activity (average disk bytes/second). We then changed dynamically the maximum memory to 3 GB (Target Server Memory), and executed the same query again. We still noticed some I/O activity because the previous memory setting (1 GB) was not enough to load the target data. The third execution showed no I/O activity. The database table is entirely loaded into the memory (buffer cache) prior to the query execution and can therefore be processed much faster.

	Response time	Maximum memory
First execution	10 ms	1 GB

	Response time	Maximum memory
Second execution	6 ms	3 GB
Third execution	0 ms	3 GB

Table 1. Impact of maximum memory on query response time

Memory management in a cluster environment could also be challenging. If you allow the dynamic memory manager to operate, a failover of a SQL Server instance in an active/active configuration may lower performance. The SQL Server instance that is running on the surviving node has to make memory available to the failover instance. In the case of a heavy workload, this process might take a while, meaning that the failover instance will provide poor response time due to the lack of memory. In addition, if the Lock Pages privilege has been set to the SQL Server user who is running the service, none of the pages will be paged out.

To correct the situation, one option is to set the maximum memory to both SQL Server instances in order to be able to manage them as running on one single node. The sum of the maximum memory should then be set to the available physical memory (minus 10% for the OS). However, this means that most of the time half of the memory will not be used. A second option is to change the maximum memory settings for both instances when a failover occurs.

Hot addition of memory

The hot addition of memory is supported in Windows Server 2008 for the x64 and Intel Itanium 2 platforms. Windows Server 2008 can therefore offer this new resource to different running processes. Of course, this feature is available only when the underlying hardware is capable of supporting it without stopping the server.

HP ProLiant DL580 G4 Server and ML570 G4 Server take advantage of this feature (see Figure 11). By means of modules on the server front panel, the operator can, at any time, remove a cartridge and add memory banks. SQL Server 2008 Enterprise Edition is capable of detecting this method of adding memory without restarting the instance.



Figure 11. HP ProLiant DL580G4 and ML570G4: two HP servers with hot-plug RAID memory

Hot addition of CPUs

Windows and SQL Server 2008 also support the hot addition of CPUs. As long as the hardware supports this capability, you can add CPUs while the OS is up and running. SQL Server 2008 can take advantage of these new CPU resources while it is running. Note that hot removal is not supported.

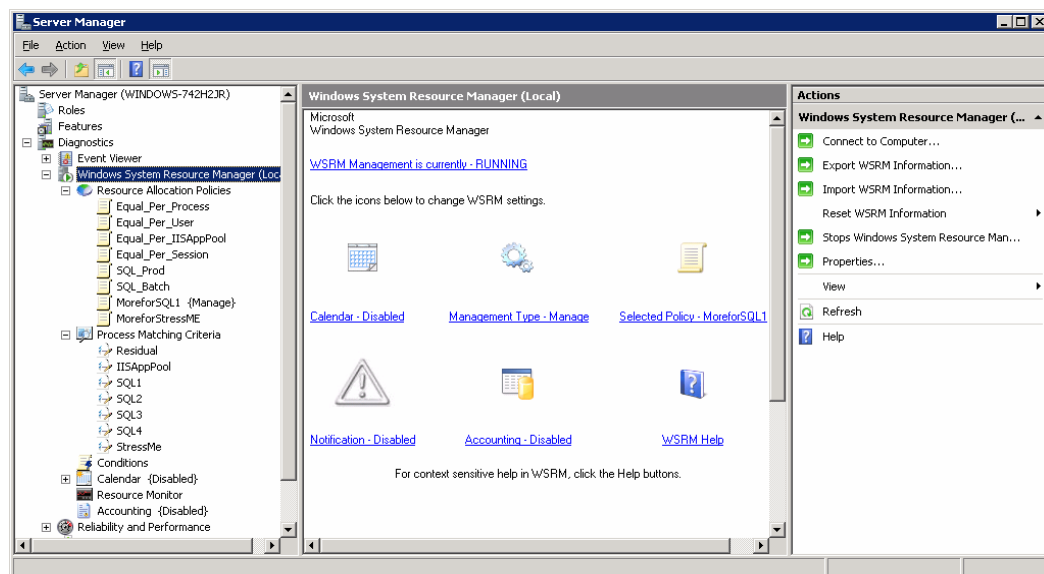
Windows System Resource Manager

Windows System Resource Manager (WSRM) has been fully integrated in Windows Server 2008 (see Figure 12). With WSRM, you can manage server processor and memory usage with resource policies. New features include the concept of conditions, which is a way to enable a Policy when an event occurs. The events currently covered are the following:

- Processor added (hot add CPU)
- Memory added (hot add memory)
- Cluster event (node goes down, node comes up)

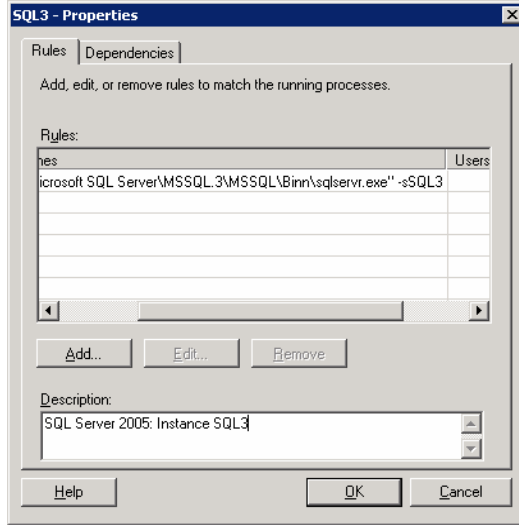
Note: To install WSRM, open the Server Manager and then open Features. Select WSRM from the list of features. The WSRM console will be accessible from the Administrative Tools folder.

Figure 12. New WSRM console in Windows Server 2008



In addition to its use with SQL Server, this tool allows you to allocate resources when running multiple instances on a single server. With WSRM, you can define a policy to be scheduled automatically with the built-in orchestrator. Within a policy, you can define the different processes you want to control. Figure 13 shows a SQL instance defined as a Process Matching Criteria.

Figure 13. Example of SQL instance defined as a Process Matching Criteria



For each process, you then define the resources (CPU and memory) you want to assign. There is one active policy at a time and moving from one policy to another is transparent to the application.

Note that SQL Server does not support all the features provided by WSRM. Consequently, if WSRM offers the ability to add or remove CPUs to an application, the only way to control the CPU allocation on a SQL Server instance is to use CPU affinity. WSRM allows some mechanisms to control the memory utilization, but these features are not supported with SQL Server. The minimum/maximum memory instance properties can be used to control the memory allocated to a SQL instance.

The only resource you can control through WSRM for SQL Server is the CPU percentage. For example, you can specify that a given instance will have 50% of the CPU resources guaranteed under load.

Figure 14. Two Resource Allocation policies

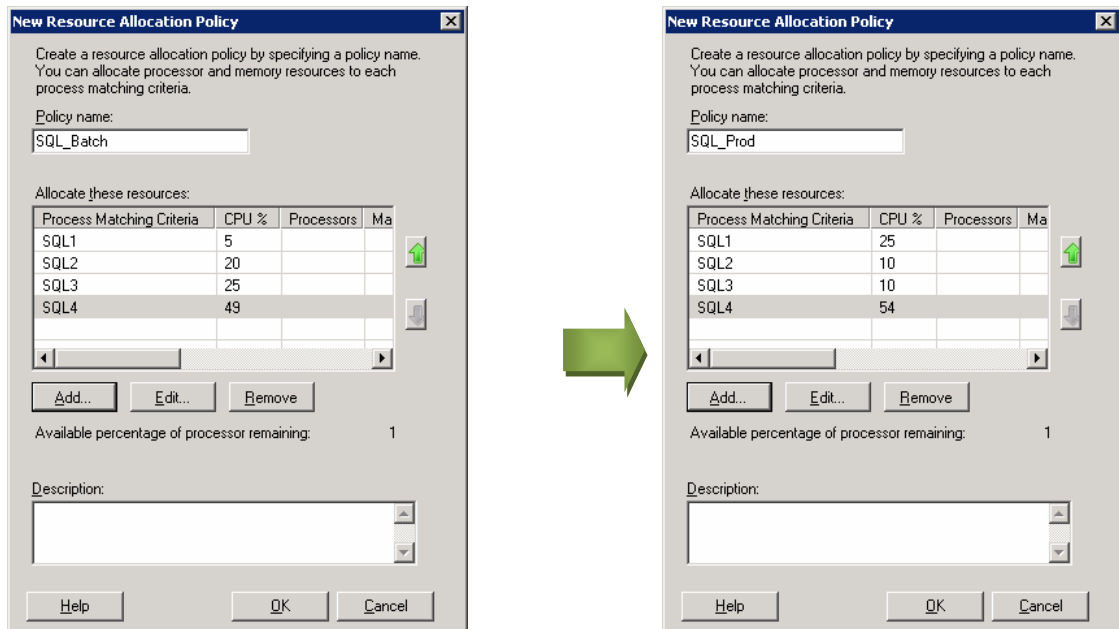
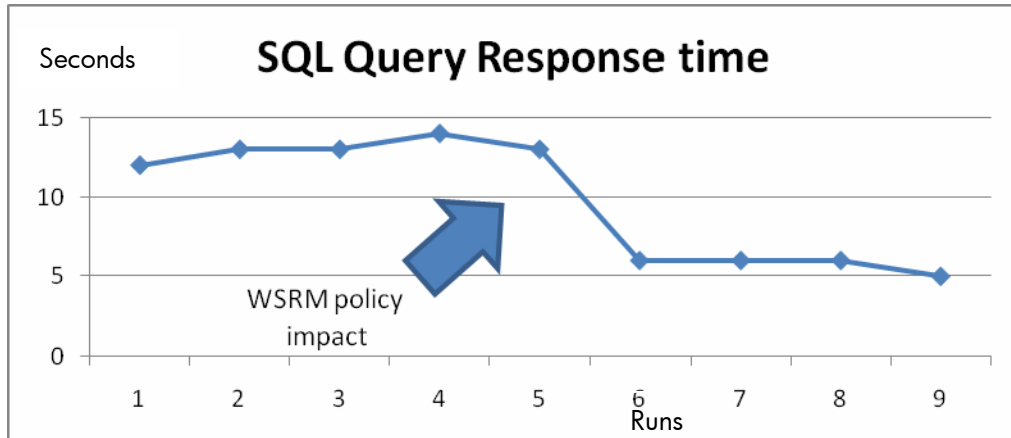


Figure 14 shows two Resource Allocation policies that can be used to allocate the CPU resources among four SQL Server instances. Again only one policy is active at a time. Either by manual operation or using the scheduler, you can move from one policy to another. Figure 15 shows the impact on the response time of a query targeting a database attached to the SQL1 instance. The response time decreases when we move from the 5% to 25% CPU allocations.

Note that WSRM and SQL Server CPU affinity can be combined. In that case you need to take into account the CPU percentage assigned in the WSRM policy. For example, if you have a four-CPU server and you set the affinity of the SQL Server instances to two CPUs, you will then have 50% of CPU to dispatch in your WSRM policy.

Like most resource management features, WSRM is active only when the server is under load, that is, when the combined processor utilization is higher than 70%. Conversely, when the CPU utilization is low, there is no limitation in terms of resource utilization. So if you have a single application running on a server, there is no reason to limit the resources to this application since the remaining resources will not be used.

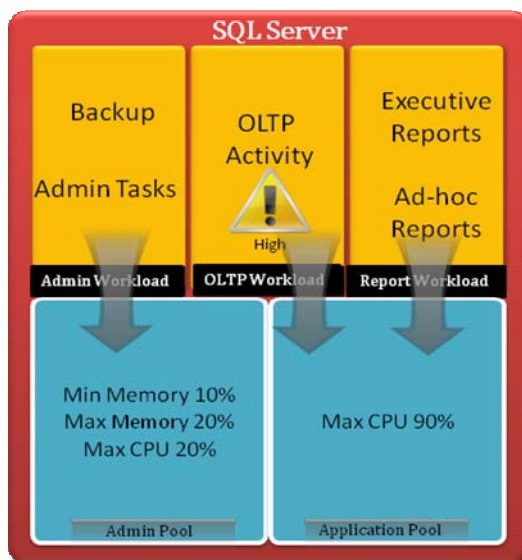
Figure 15. Impact of a WSRM policy on an SQL query response time



Resource Governor

With SQL Server 2005, we did not have a way to prioritize different workloads within the same instance. It would have been ideal to assign more resources to an application with a critical database and fewer resources to administrative or maintenance tasks. With SQL Server 2008, we now have this capability. The Resource Governor is one of the important resource management tools that has been added in the database engine.

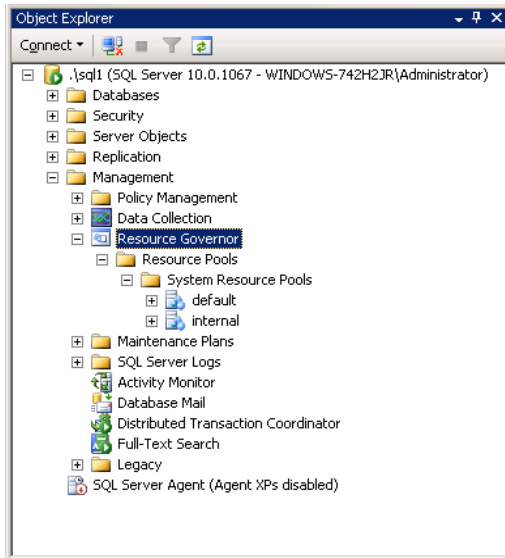
Figure 16. Resource Governor concept



Resource Governor is a way to provide a consistent and predictable response to users. Database administrators can now define resource limits and priorities for different workloads to fulfill their

service level agreements. Resource Governor is disabled by default and can be enabled using SQL Server Management Studio (Figure 17).

Figure 17. Resource Governor in SQL Server Object Explorer



Defining the workload

After Resource Governor is enabled, you can define the different workload names and assign each to a resource pool:

```
create workload group PowerUserGroup using demo
create workload group MidUserGroup using demo
create workload group LowUserGroup using demo
```

Creating the classifier user-defined function

Next you define the classifier function that will assign the SQL Server applications to the different workload groups. For instance, you can use the name of the application as the criteria or the user who was used to connect to SQL Server. In the following example, we used the username to sort the workload:

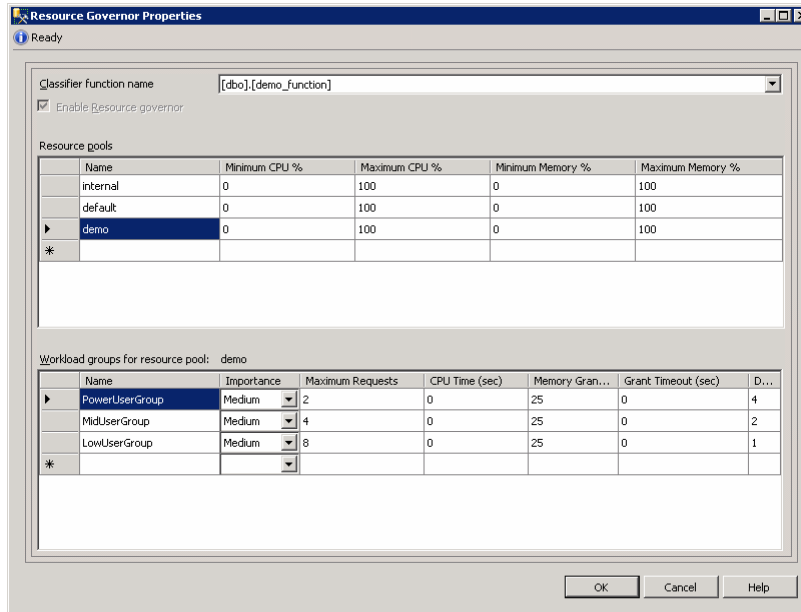
```
create function demo_function() returns sysname with schemabinding
as
begin
    declare @grp_name as sysname
        if (SUSER_NAME() = 'poweruser')
            set @grp_name = 'PowerUserGroup'
        if (SUSER_NAME() = 'miduser')
            set @grp_name = 'MidUserGroup'
        if (SUSER_NAME() = 'lowuser')
            set @grp_name = 'LowUserGroup'
    return @grp_name
end
```

Registering the classifier function in Resource Governor

As a final step, register the classifier in Resource Governor (see Figure 18):

```
ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION=  
dbo.demo_function)
```

Figure 18. Resource Governor properties



In Resource Governor, you can define multiple resource pools. Each resource pool must have an identified workload. Each workload has some dedicated settings to control the maximum number of concurrent requests and the degree of parallelism.

Testing a resource pool with Resource Governor

Figure 19. Resource Governor tree in SQL Server Object Explorer

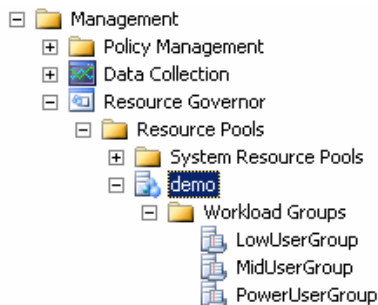


Figure 19 shows the Resource Governor tree with the three workloads created. Figure 20 illustrates the impact of Resource Governor on the degree of parallelism. The load generated corresponds to a query executed by a user belonging to the MidUserGroup: two CPUs are under load. Note that in Figure 18 the value of the last column is 2 for the degree of parallelism.

Figure 20. Two CPUs under load: Degree of parallelism controlled by Resource Governor

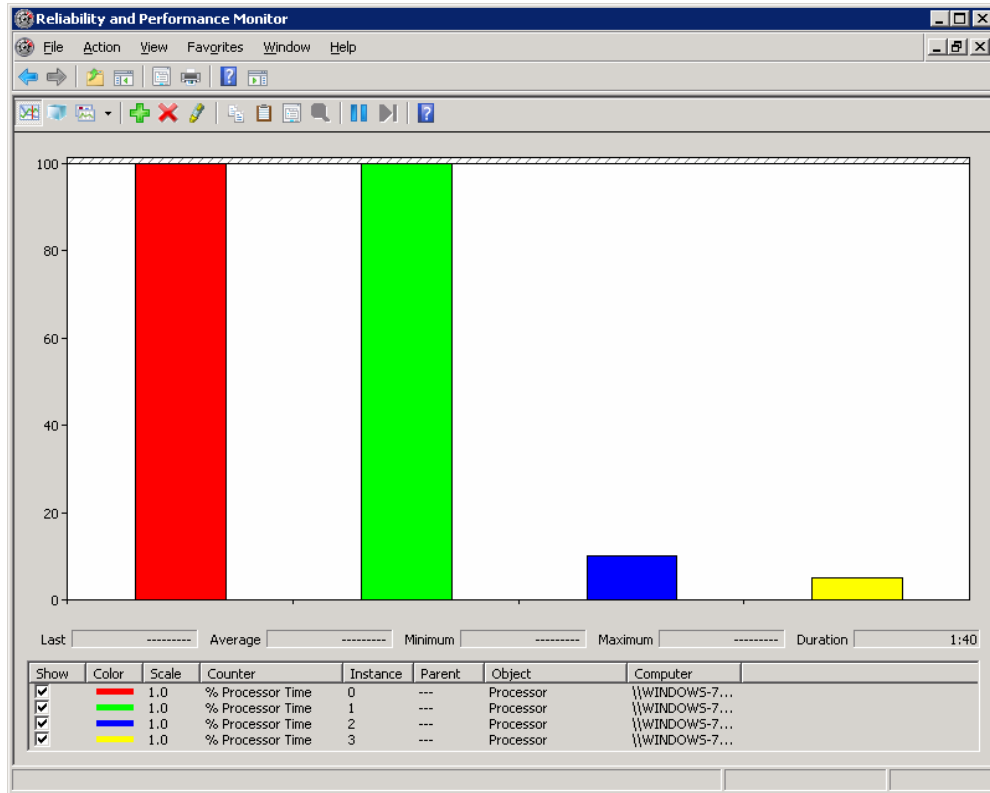
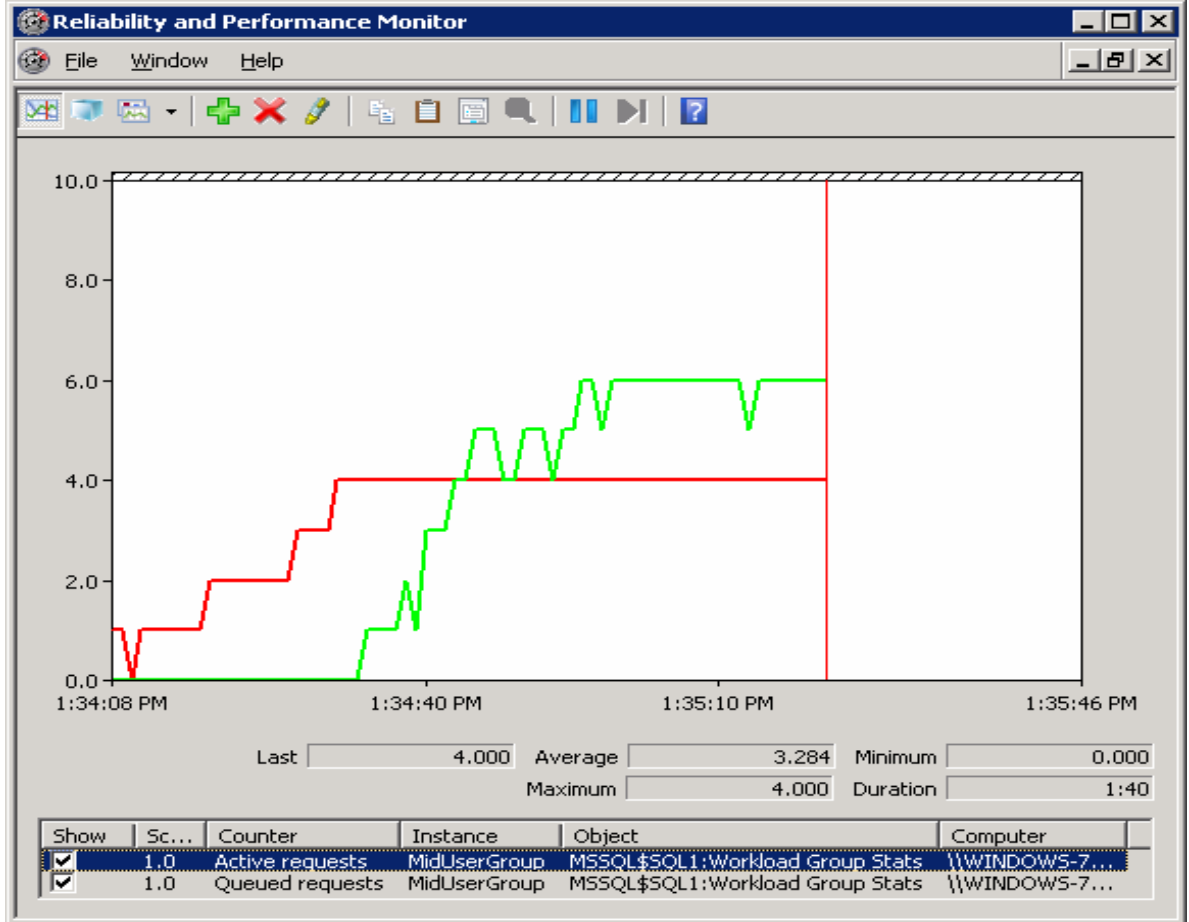


Figure 21 shows how Resource Governor controls the number of concurrent requests. We monitored the active requests, limited to four, and the queued requests.

Figure 21. Number of concurrent requests controlled by Resource Governor



To evaluate the Resource Governor tool, we tried the following exercise on a four-way server. We used the policy defined in Table 2.

Workload	Degree of parallelism	Concurrent requests
PowerUser	4	2
MidUser	2	4
LowUser	1	8

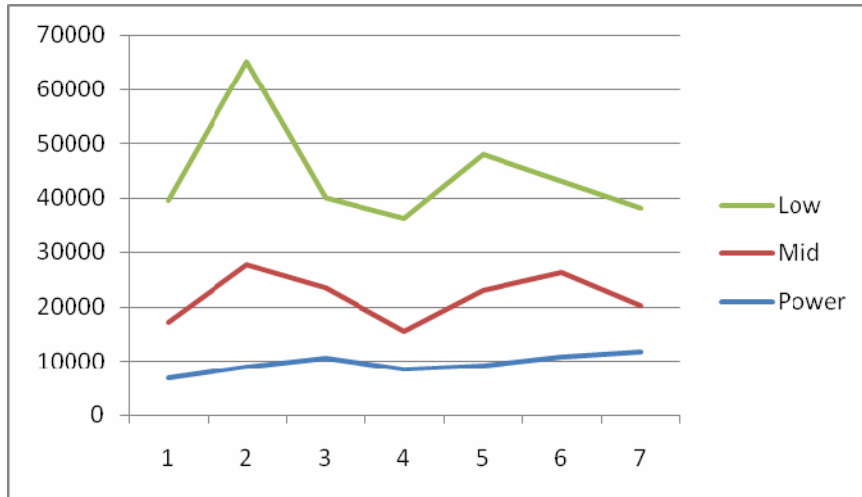
Table 2. Resource Governor settings

We then executed the same requests (T-SQL statements) using the three different users connected to the same table:

```
Sqlcmd -S%COMPUTERNAME%\SQL1 -Umiduser -Ppassword -ddatabase
-Q"select count(*) from dbo.telco_fact32"
```

The number of concurrent requests corresponds to the maximum concurrent requests defined in Resource Governor: for example, two concurrent requests of PowerUser were running in parallel. We had enough resources available on the server to load the entire table into memory so no physical disk I/Os were involved. We can note in Figure 22 the significant impacts to the response times.

Figure 22. Average response time (in ms) of a query executed by the three Resource Governor-defined workloads



The different Resource Governor policies can be controlled using the Microsoft performance monitor tool. Some dedicated counters are available in the Workload Group Stats counters group.

Figure 23. Performance monitor counters for Resource Governor

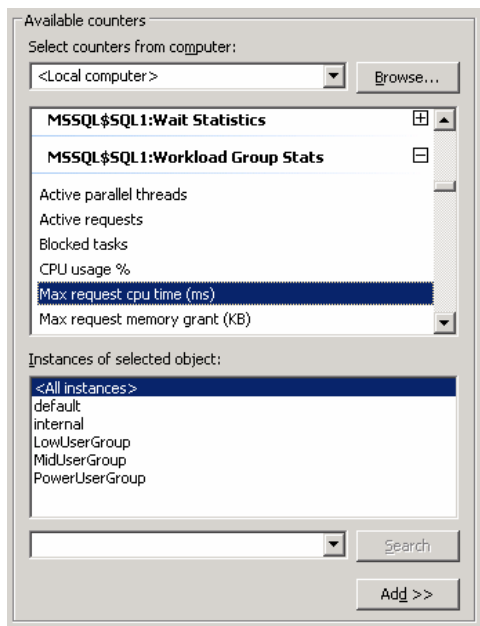


Figure 24 shows the impact of the maximum CPU% settings. We have defined two resource pools with different CPU settings. Three workloads are allocated to the Demo pool and one workload to the Other pool (see Figure 25). The query executed in each of the workloads is the same; the only difference is the username we used to connect to SQL Server. The workload generated by the user

“sa” (on the right in Figure 24) is obviously using more CPU resources (CPU usage %) than the three workloads assigned to the Demo pool. Figure 26 shows the CPU utilization while running the different workloads.

Figure 24. Impact of Resource Governor policies on the CPU usage

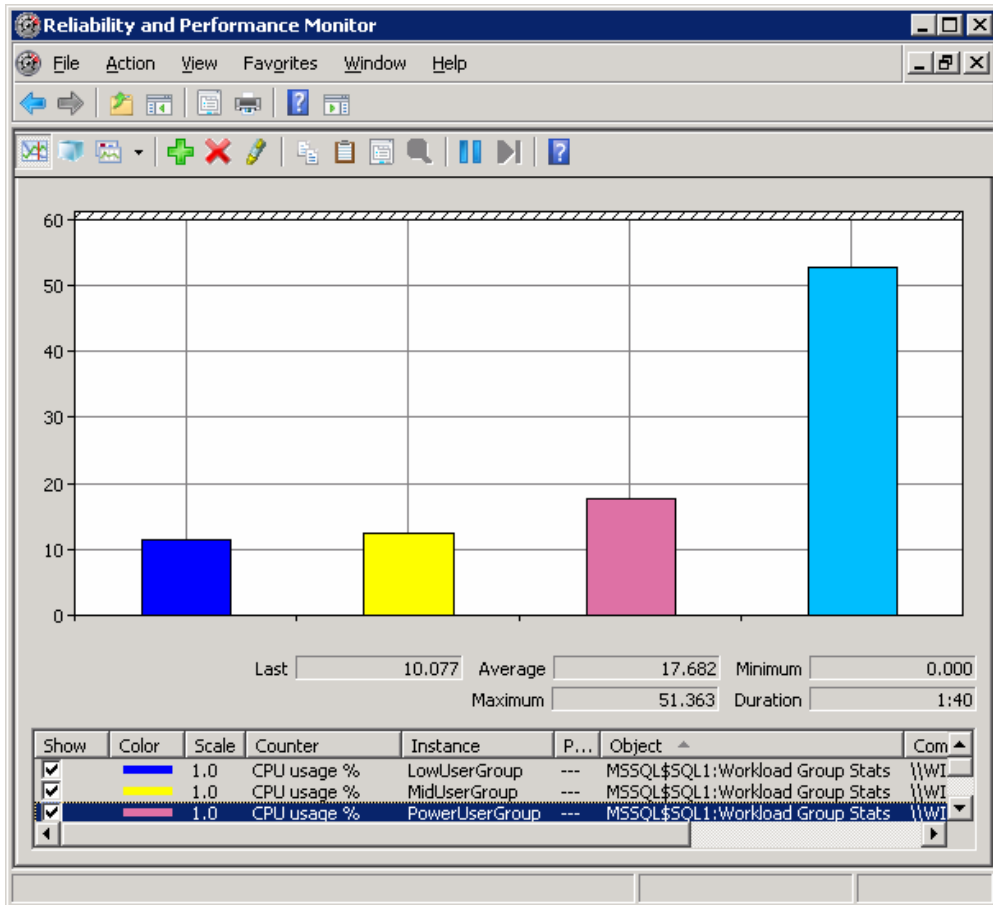


Figure 25. Resource Governor settings: Two customized pools, Demo and Other

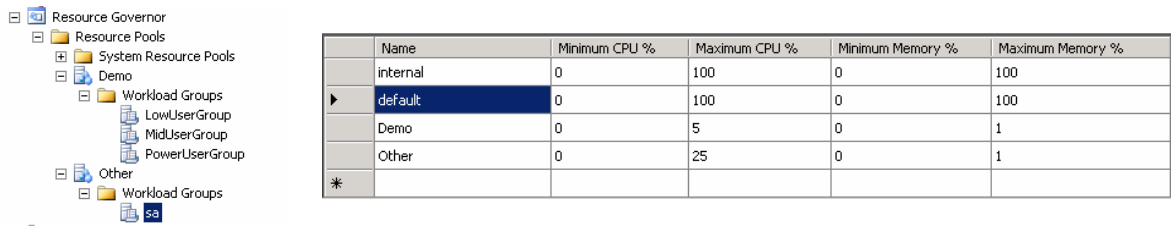
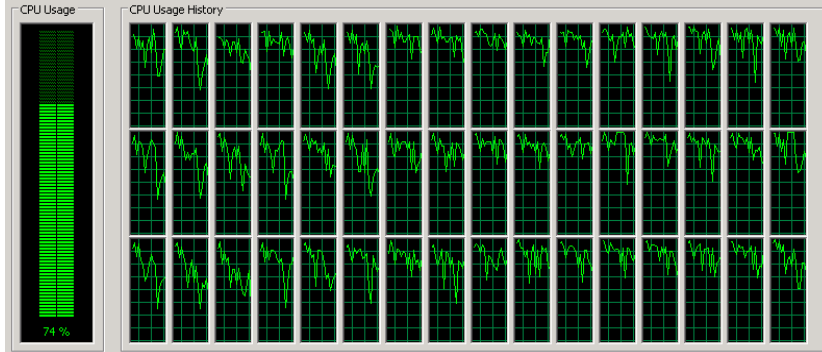


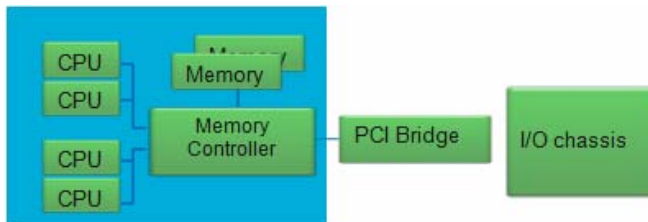
Figure 26. Task manager under load during Resource Governor tests



NUMA architecture

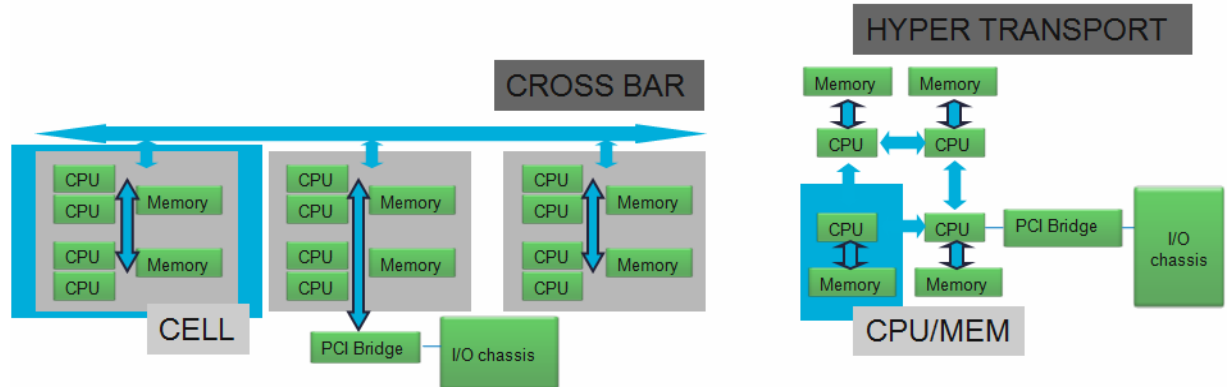
The Non-Uniform Memory Access (NUMA) architecture is often implemented on medium- to high-end Itanium and ProLiant servers. Among other things, it enables you to set up servers with more than eight processors. Using the Symmetric Multi-Processor (SMP) architecture usually limits the number of sockets to four. Figure 27 shows the SMP architecture.

Figure 27. SMP architecture









The NUMA architecture introduces the concept of the node. A node consists of processors, physical memory, and input/output port(s). Each node communicates with the other nodes of the same server through a bus. A node can be considered an SMP block.

Figure 28. NUMA architecture: Cell-based architecture (HP Integrity) and Opteron-based servers (HP ProLiant)



Within the HP Server portfolio, two types of servers use the NUMA architecture: the HP ProLiant AMD Opteron-based servers and the mid-range and high-end HP Integrity Servers. See Figure 28 and Figure 29.

Figure 29. SMP and NUMA servers in the HP portfolio

<i>HP SMP Servers</i>	<i>HP NUMA servers</i>
 <p>HP ProLiant DL580 G5 Server series</p>	 <p>HP ProLiant DL585 G2 Server series</p>
 <p>HP ProLiant BL680c G5</p>	 <p>HP ProLiant BL685c</p>
 <p>HP Integrity RX6600</p>	 <p>HP Integrity RX8640</p>

Windows and NUMA

To take advantage of the NUMA architecture, Windows Server 2008 polls the hardware to get important architecture information: number of nodes, number of processors, and memory capacity by node. In fact, a running process on one of the processors should use in-priority memory that is local to the node rather than remote memory. The memory access latency differs if the memory is addressed locally (within the same node) or remotely (through the crossbar switch). Application programmer interfaces (APIs) included in Windows Server 2008 are at the developers' disposal to determine the server configuration.

The NUMA behavior of Windows is automatically enabled when informed by the hardware of the NUMA topology of the server. There is no simple way to check that the NUMA mode is enabled. The only method consists of using the NUMA API to poll the server. Some of the functions of the API are listed below:

- `GetNumaHighestNodeNumber` - returns the number of nodes.
- `GetNumaProcessorNode` - returns the node number for a given processor.
- `GetNumaNodeProcessorMask` - returns a binary mask of processors available in a node.
- `GetNumaAvailableMemoryNode` - returns the available memory for a specific node.

Microsoft provides a detailed description of the NUMA API at <http://msdn2.microsoft.com/en-us/library/aa363804.aspx>

Important: On most of the NUMA servers, you can hide the NUMA architecture so that the OS will see the server as an SMP server. Studies on AMD-based servers with different applications give performance figures with or without the NUMA layout exposed. (See AMD Opteron processor scalability and performance in the ProLiant DL585 <http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00302781/c00302781.pdf>). For high-end configurations using HP Integrity servers, HP recommends exposing the NUMA architecture because access to remote memory can have a significant impact (latency). If you hide the NUMA layout on a four-node server, 75% of the memory access is remote.

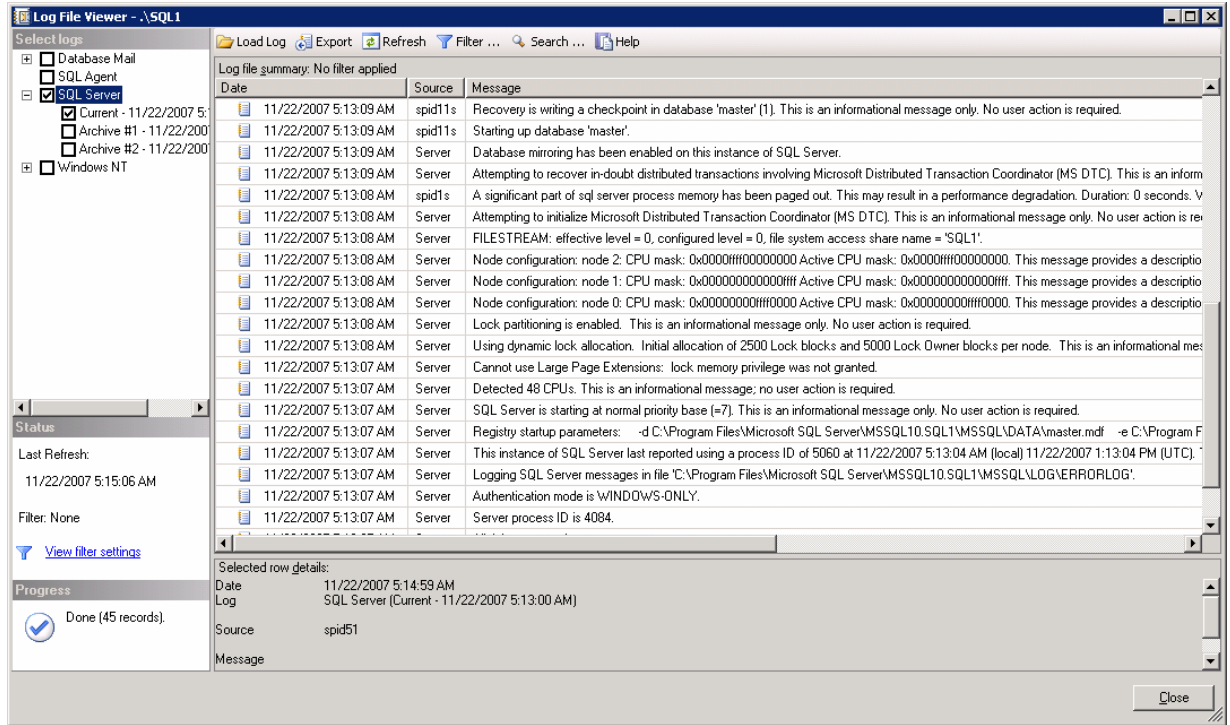
This setting is controlled either with the BIOS for an HP ProLiant AMD device or at the partition level on an HP Integrity cell-based server. Note that the HP ProLiant DL585 has, by default, NUMA memory configuration as opposed to the partition on an HP Integrity, which uses interleaved memory by default.

SQL Server 2008 and NUMA

Having the OS aware of the NUMA layout is essential to leverage the architecture. The application can also be designed to take advantage of this architecture: SQL Server is a good example of a NUMA-aware application.

Each time it starts, SQL Server 2008 queries the Windows NUMA APIs. If SQL Server detects that it is running on a NUMA server, the information will be reported in the SQL Server ERRORLOG (Figure 30). SQL Server will then create one buffer cache per node.

Figure 30. NUMA detection in the ERRORLOG



Each node will also have a lazy writer thread to manage the buffer cache as shown in Table 3.

spid	Ecid	status	loginame	hostname	blk	dbname	cmd	reqid
6	0	background	sa		0	NULL	LAZY WRITER	0
7	0	background	sa		0	NULL	LAZY WRITER	0
11	0	background	sa		0	NULL	LAZY WRITER	0

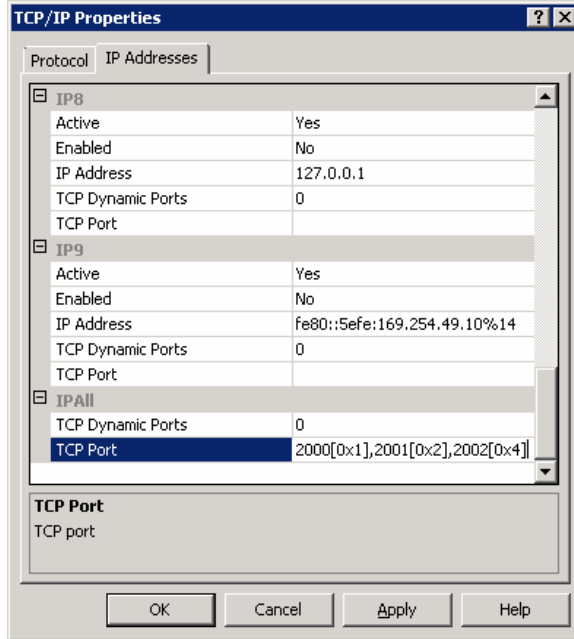
Table 3. Lazy writer threads on a SQL Server configured with three nodes

After the nodes are detected, the administrator can create TCP/IP listeners who can be assigned to these nodes. Figure 31 shows how to use the SQL configuration manager to create these listeners. The syntax is the following: <port number>[<node number>], <port number>[<node number>], If these ports are specified in the queries, they will be automatically routed to the CPUs belonging to the node. The following command shows how to specify a port in the connection string:

```
Sqlcmd -Stcp:%COMPUTERNAME%\SQL1,2000 -Usa -Ppassword -
ddatabase -Q"select count(*) from dbo.telco_fact32"
```

Note that to optimize connection affinity, the degree of parallelism should be set to 1, so that the initial routed connection remains on the same CPU. This is true only for serial plan execution. This fits perfectly with OLTP workloads but not for parallel query execution.

Figure 31. SQL configuration manager: TCP/IP properties for hard/soft NUMA settings



Thread affinity to a node is the ability of SQL Server to maintain a thread within the same node. The built-in SQL Server load balancer can move a thread from one CPU to another. With thread affinity, the thread remains in the node. For example: SQL Server is running on a two-node server and each node is based on eight CPUs. For a given query, if the plan determines that 8 is the ideal number for the degree of parallelism (or if DOP is set to 1), SQL will use eight CPUs that belong to the same node.

In another scenario, a 16-processor server is based on two physical nodes (cell). If a query needs 8 CPUs in order to be processed, the 8 CPUs chosen by the SQL engine will belong to the same node.

These responses to queries can be tracked and monitored: Performance monitor counters are available in SQL Server to monitor the buffer caches initiated by SQL Server for each node. See Figure 32.

Figure 32. SQL Performance monitor counters for NUMA servers

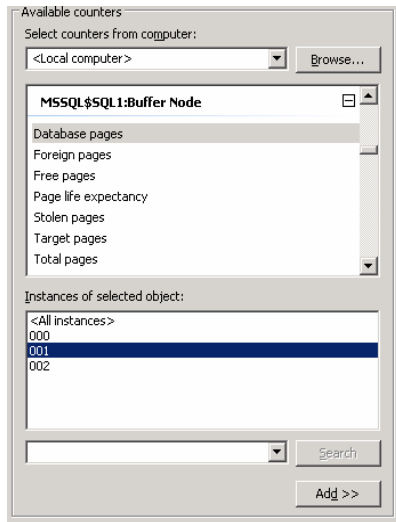


Figure 33 shows the CPU utilization on an rx8640 server when we targeted a TCP/IP port assign to cell 1 (node 1). Only the CPUs in that cell are in use. Figure 34 shows that the data is loaded on the local buffer cache (MSSQL\$SQL1:Buffer Node:001).

Figure 33. SQL Queries targeting a node (a cell for an HP Integrity server)

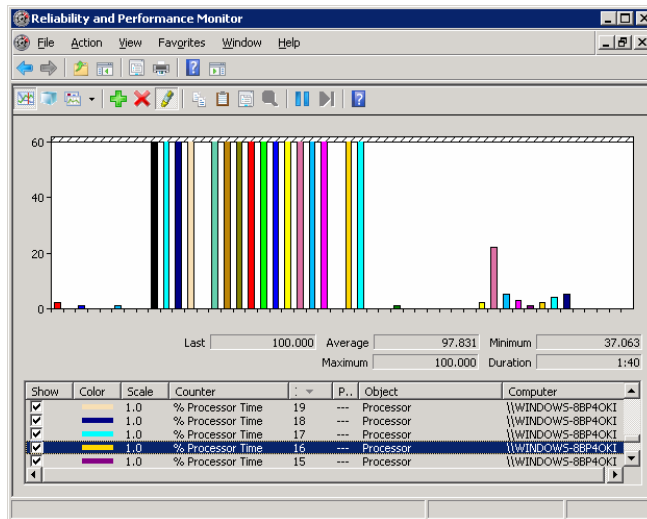
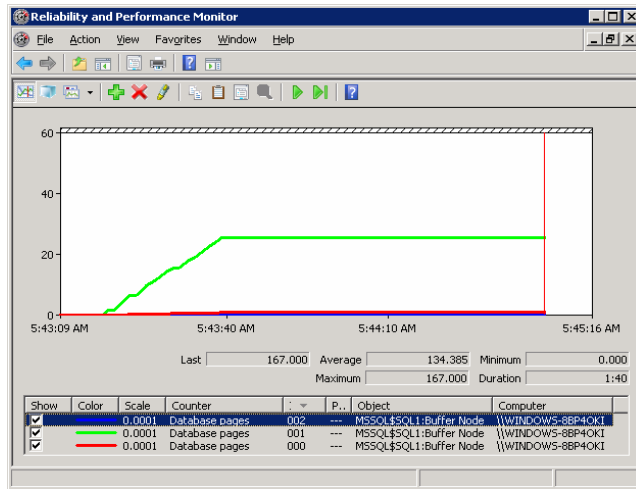
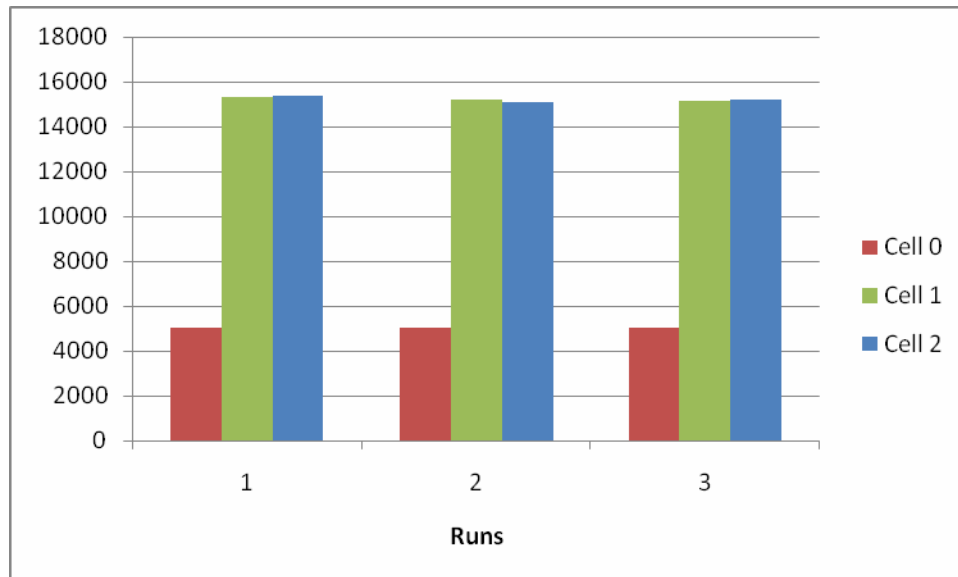


Figure 34. Database pages loaded on the buffer cache of the target node



To demonstrate the impact of the latency on NUMA architecture, we loaded a table into the buffer cache created by SQL Server on node 0 (cell 0). Then we ran a query targeting simultaneously node 0, node 1, and node 2 using TCP/IP affinity. The results are presented in Figure 35. Note that if we use the CPUs that belong to cell 0, the response time is lower because we are using local memory. On the other hand, if we use CPUs from other nodes (1 or 2), the response time is higher.

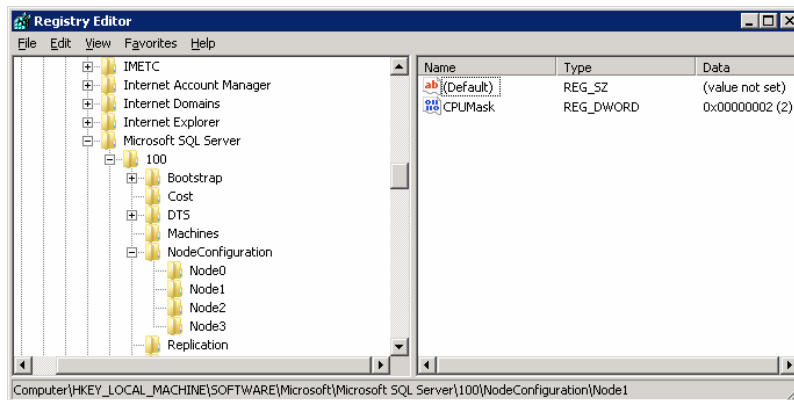
Figure 35. Impact of remote memory access on a query response time (ms)



NUMA software

The NUMA architecture is linked to the hardware configuration: hence the so-called NUMA hardware. To extend the granularity of resource management, SQL Server has introduced the concept of Soft NUMA. One physical node can be divided into several logical nodes. As a consequence, in a scenario with a one node based server (SMP), this node can be divided into multiple logical nodes. The configuration differs slightly from the NUMA hardware: first you define all logical nodes and all the processors assigned to them. To each node, you assign a CPU mask. This setting should be done in the Windows registries. A given CPU cannot belong to multiple Soft NUMA nodes. See Figure 36.

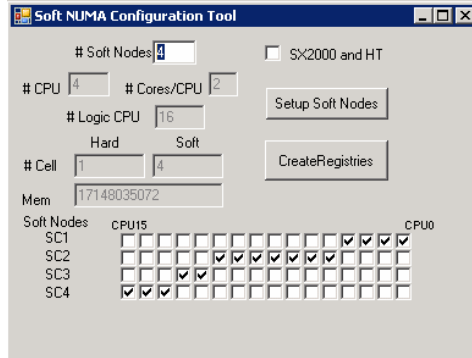
Figure 36. Registry editor for Soft NUMA configuration



By using the same logic, it is possible to define a TCP port to be assigned to one or more logical nodes. A TCP connection established through this port will use exclusively the processors assigned to the same port.

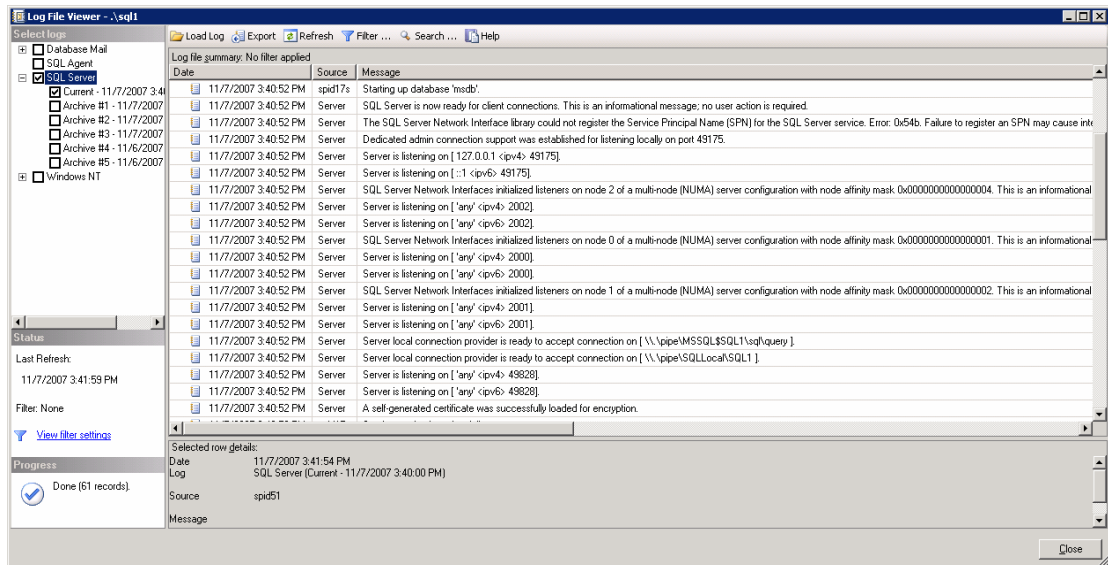
In order to avoid dealing with the registries, we have created a tool that allows you to graphically design your Soft NUMA nodes. It also has the capability to detect NUMA hardware in case your server architecture is NUMA based. See Figure 37.

Figure 37. Soft NUMA configuration tool



In Figure 37, we show a server with 16 logical CPU (4 sockets, 2 cores per socket with hyperthreading on). We have defined four Soft NUMA nodes with 4, 7, 2, and 3 threads. Note that a restart of the SQL Server instance is required after any change to the Soft NUMA settings. Figure 38 shows the ERRORLOG reported by SQL with Soft NUMA activated.

Figure 38 ERRORLOG showing the Soft NUMA configuration and the corresponding TCP/IP ports

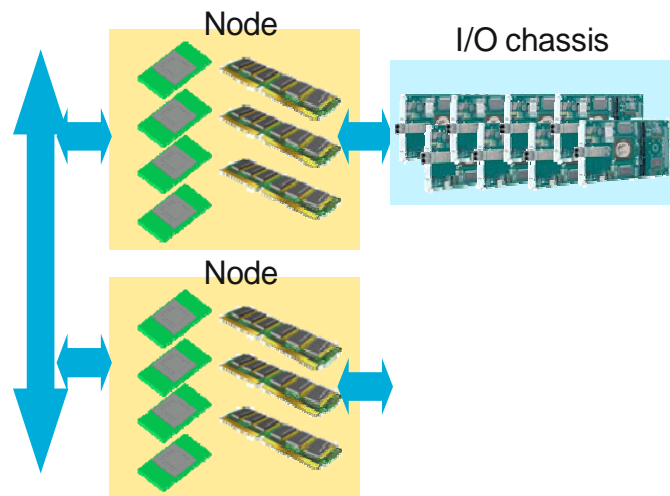


Note that if you decide to use Soft NUMA capabilities on a cluster, the settings should be replicated manually across all the nodes. In case the nodes do not have the same number of CPUs, the Soft NUMA settings can differ in order to adjust the resources if there is a failover to a server with fewer resources. The TCP/IP instance properties (stored in the registries), as opposed to the node configuration, are replicated among all the cluster nodes. This allows one Soft NUMA node to have two physical CPUs assigned on one cluster node and four on another one.

Storage and NUMA architecture

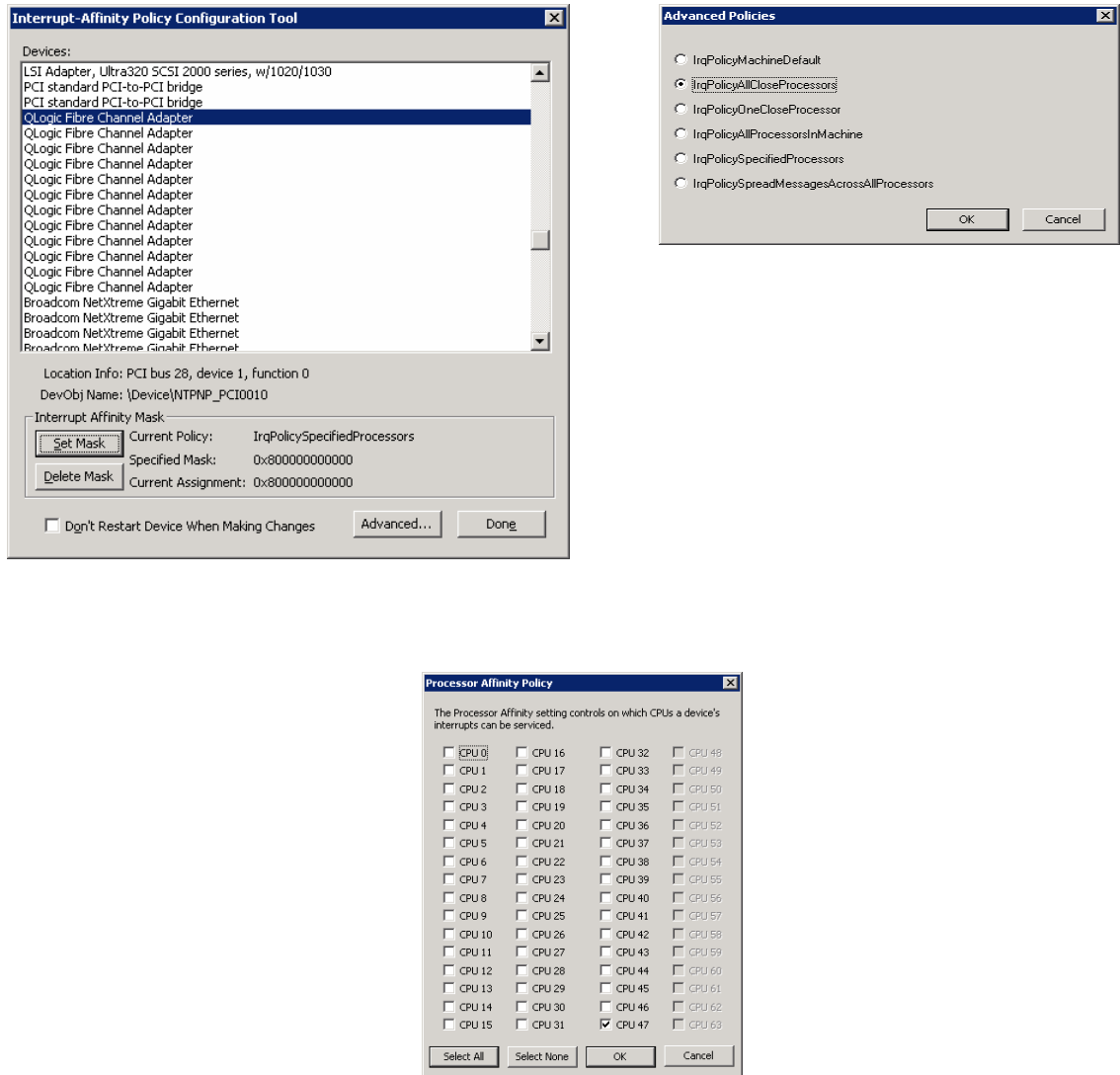
Storage is obviously one of the key components for a database configuration. Since a NUMA node can potentially have direct access to disk I/O, the OS manages the interrupt generated by the local devices using the local CPUs.

Figure 39. I/O chassis with HBAs connected directly to a NUMA node



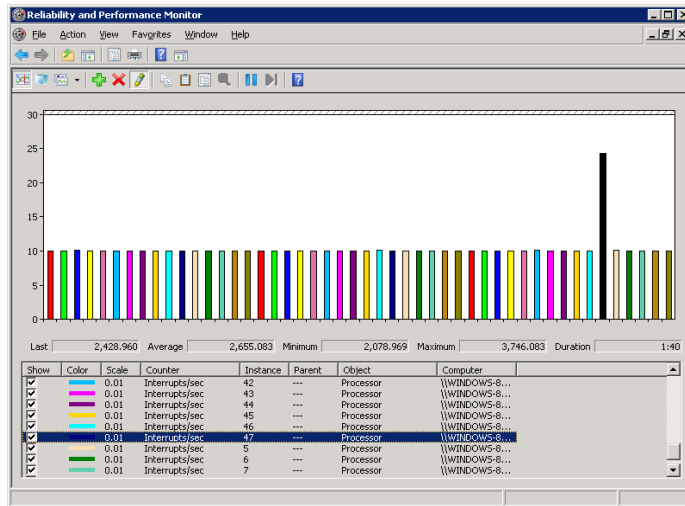
With the introduction of Windows Server 2003, the operating system had the ability to get the NUMA layout. It was able to detect the number of nodes and the resources within each node (CPUs and memory). Windows Server 2008 now has the ability to get the I/O layout as well, meaning the relationship between a host-based adapter (HBA) and a node (see Figure 39). This will optimize the CPU utilization. With the Interrupt-Affinity Policy Configuration Tool (<http://www.microsoft.com/whdc/system/sysperf/IntPolicy.mspx>), you can select one of the advanced policies for a given device or define a CPU mask.

Figure 40. Interrupt-Affinity Policy Configuration Tool



As shown in Figure 40, we forced the CPU affinity for one of the available HBAs to CPU 47. When we generated a load on one LUN available through this HBA, the number of interrupts/sec was significantly higher on the CPU specified (see Figure 41).

Figure 41. HBA interrupts assigned to processor 47



Windows Server 2008 provides a new way to manage interrupts. With Windows Server 2003, line-based interrupts were used. The interrupt was generated by the device by sending an electrical signal on a dedicated pin (interrupt line). With this technique, it was very difficult to assign a specific CPU to a given device. With Windows Server 2008, a device generates a message-based interrupt (Message-Signaled Interrupts or MSI) by writing a data value to a particular address. MSIs allow you to set a priority for the interrupt. A given CPU can now be specified to service an interrupt.

Partitioning and virtualization

Hard partitioning

Mid-range and high-end HP Integrity servers offer cell-based architectures. Partitioning makes it possible to create servers that are electrically independent within a single server (complex). To create one partition, it is necessary to specify information such as the number of cells (processors, memory) and the associated I/O controllers. Therefore, a single server can host several Windows Server partitions acting as independent systems. This provides a very high level of isolation: a workload applied to a partition will not impact the others. At any time, the different nodes available within the server can be reassigned to one partition or another. A restart of the partition is required each time you add or remove a node in the partition. This method is going to change in the near future: Windows Server 2008 will be integrating the ability to manage the hot addition of CPUs.

Pay per use

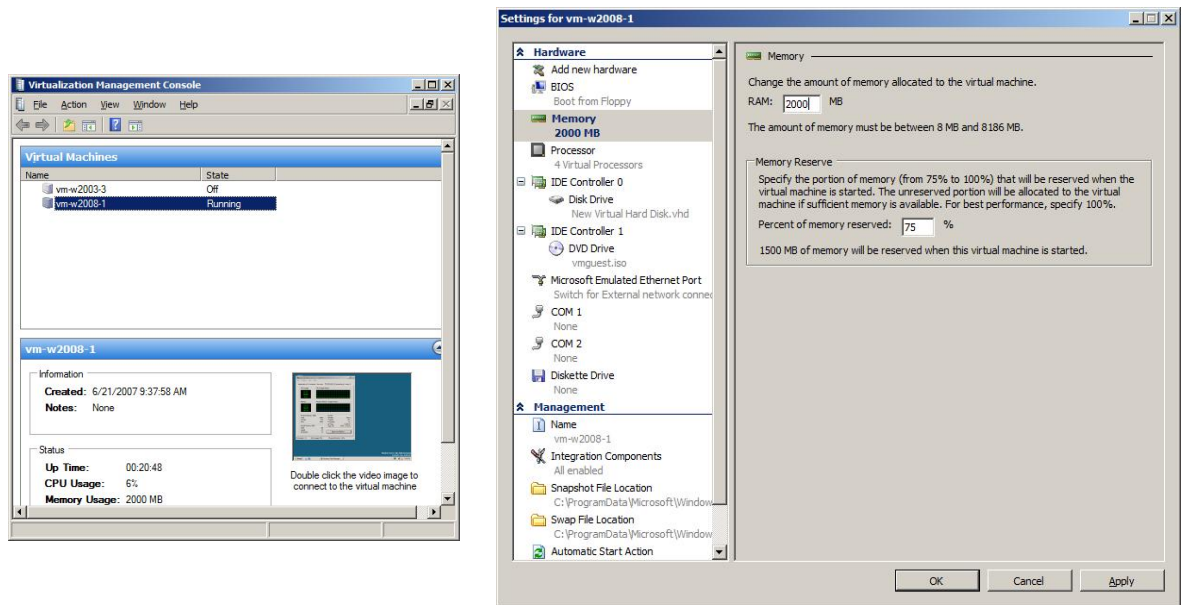
Other mechanisms enable you to optimize the use of resources on an HP Integrity Server. Some are technical solutions; others are of a financial nature. Pay per use is an example of the latter. A customer who chooses this mechanism rents an HP Integrity Server according to the usage level of processors. The calculation is based on the average occupancy level of the server for one month.

This solution provides more flexibility, especially for limited uses or to meet new business opportunities. Through an agent installed on the server, processor usage information is automatically gathered, encrypted and sent in a secure way to the HP billing center. The pay per use option is part of the HP Financial Services offerings.

Virtualization

Many virtualization engines are available to run multiple virtual machines on top of a physical server. The Windows Server virtualization layer (Hyper V) that will be provided with Windows Server 2008 will have the capability to hot add resources such as CPU, memory, networks, and storage to the Windows virtual machines with no downtime. Combined with the hot addition features of Windows Server 2008, this enables administrators to manage their hardware resources without impacting their SLA commitments. Figure 42 shows the Management Console and virtual machine properties. The amount of memory allocated can be increased while the virtual machine is running. Both Windows Server and SQL Server are able to leverage memory dynamically.

Figure 42. Windows Server virtualization



With the Virtual Server Environment (VSE) offered on HP Integrity Servers, HP provides the ability to reassign CPU resources (by percentage) to different virtual machines while they are running. Windows and SQL Server can be configured on VSE (see Figure 43).

Figure 44 shows the impact of the CPU entitlement of a Windows/SQL Server virtual machine. In our testing, the number of transactions per second decreased when moving from a CPU allocation of 80% to 20%. This entitlement can be changed dynamically.

Figure 43. Integrity Virtual Machines Manager showing the vCPU entitlement for the different virtual machines running

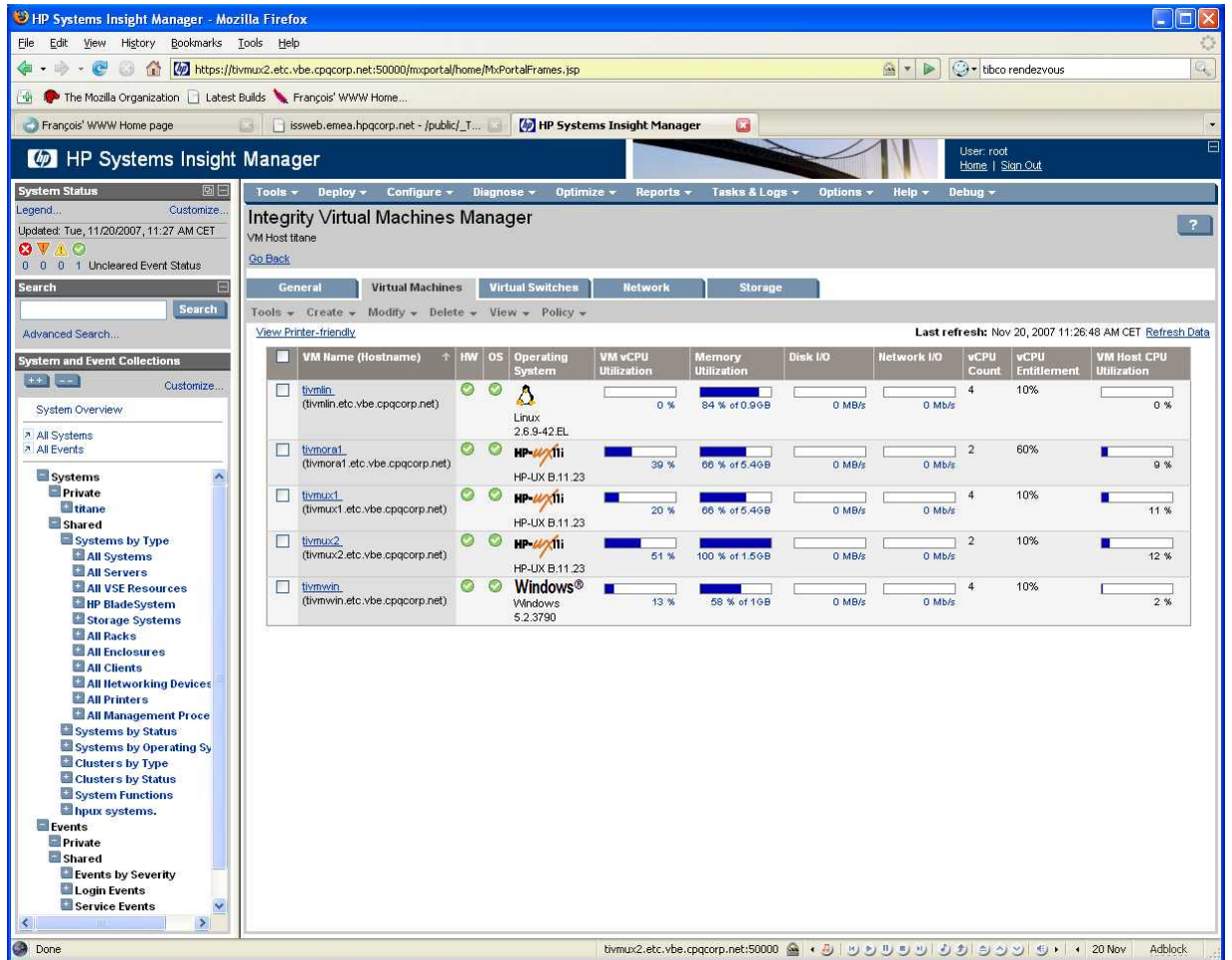
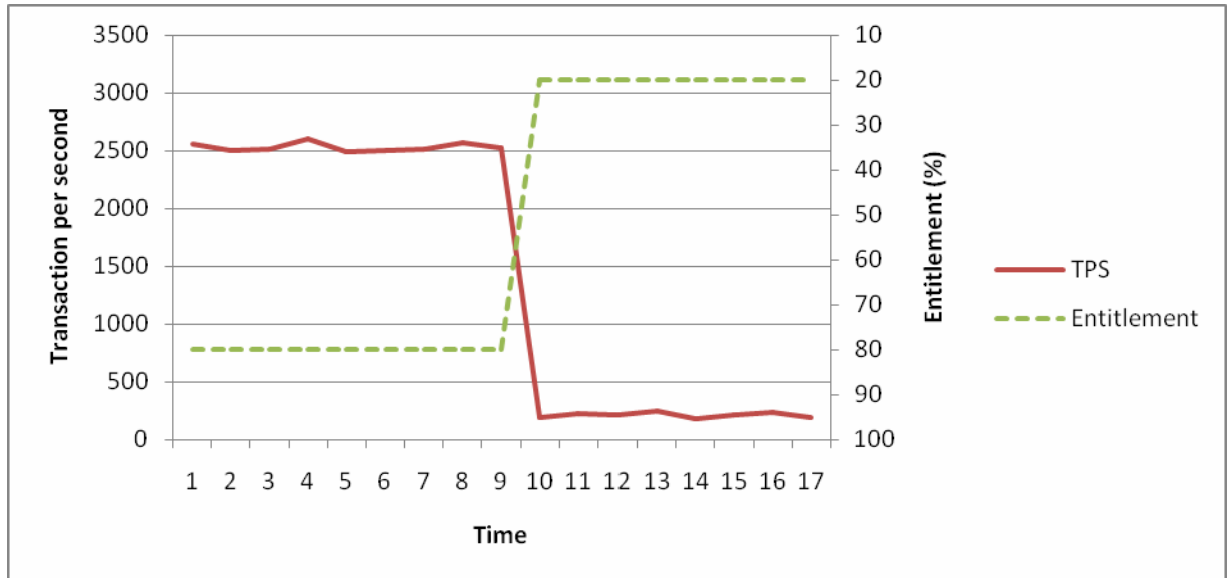


Figure 44. Impact of a vCPU entitlement (%) modification on the number of SQL transaction/sec



Conclusion

With the consolidation of corporate data, resource management is a requirement to optimize the infrastructure. In this paper we highlighted the current and future capabilities of Windows Server and SQL Server. Windows Server 2008 now provides features that allow the server to scale without downtime as well as capabilities that allow the hot addition of memory and CPUs. Windows Server can now function within the NUMA architecture used by most of the high-end servers. The operating system provides core services such as WSRM to control the resource allocation among the applications running on a server. More features have been added in the SQL Server engine. SQL can take full advantage of the operating system capabilities. For several releases, SQL Server provided features like SQL Server instances, CPU affinity, and degree of parallelism. With SQL Server 2008, Resource Governor has been added, allowing the system manager to control the resources for different workloads within the SQL instance. Implementing a database consolidation with the benefits of these technologies is a valid option for the data center.

For more information

This section lists references and their online locations.

www.hp.com

- HP Partitioning Continuum, HP, 2006
- HP ENSA extended technical overview, HP, 2006
- HP Utility Data Center Overview, HP, 2006

HP Customer Focused Testing, www.hp.com/go/hpctf

Microsoft Windows Server 2008, <http://www.microsoft.com/windowsserver2008/default.mspx>

Microsoft SQL Server 2008, <http://www.microsoft.com/sql/2008/default.mspx>

HP Industry standard OSs: Windows Server 2008

http://h18004.www1.hp.com/products/servers/software/microsoft/OS/Windowslonghorn_overview.html?jumid=reg_R1002_USEN

Implementing Microsoft Windows Server 2008 Release Candidate 0 (RC0) on HP StorageWorks Arrays white paper,

http://h18004.www1.hp.com/products/servers/software/microsoft/OS/Windowslonghorn_overview.html?jumid=reg_R1002_USEN

Windows NUMA Support, [http://msdn2.microsoft.com/en-us/library/aa363804\(VS.85\).aspx](http://msdn2.microsoft.com/en-us/library/aa363804(VS.85).aspx)

SQL Server 2008: Predictable Performance and Real-World Scalability

http://download.microsoft.com/download/c/8/4/c8470f54-d6d2-423d-8e5b-95ca4a90149a/SQLServer2008_PerfandScale_Datasheet.pdf

HP ProLiant DL585 Server Technology

<http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00180597/c00180597.pdf>

Performance Tuning Guidelines for Windows Server 2008

<http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Perf-tun-srv.docx>

HP ProLiant servers, <http://h18004.www1.hp.com/products/servers/platforms/>

Windows on HP Integrity servers, <http://h20341.www2.hp.com/integrity/cache/497701-0-0-0-121.html>

© 2008 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Itanium and Xeon are trademarks or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

4AA1-7712ENW, February 2008

