

**Resource Governor in SQL Server 2012**

SQL Server Technical Article

**Writer:** Guy Bowerman

**Technical Reviewers:** Jim van de Erve, Lindsey Allen (ZHU), Madhan Arumugam Ramakrishnan, Xin Jin

**Published:** July 2012

**Applies to:** SQL Server 2012

**Summary:** SQL Server 2012 delivers enhancements to the Resource Governor that reflect a growing need for centrally managed database services to provide multitenancy to customers who require isolated workloads. This document describes the enhancements and why they were added. It includes a self-contained walk through that you can try in order to gain familiarity with the new features.

Copyright

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 Microsoft. All rights reserved.

Contents

[Introduction 4](#_Toc330804956)

[New Resource Governor Features in SQL Server 2012 5](#_Toc330804957)

[Why Change Resource Governor? 5](#_Toc330804958)

[New Feature Summary 6](#_Toc330804959)

[A Sample Workload Isolation Use Case 6](#_Toc330804960)

[Part 1 – Initial Setup – Create Users and Workloads 7](#_Toc330804961)

[Part 2 - Set Up Resource Governor 7](#_Toc330804962)

[Part 3 – Run the Workload 8](#_Toc330804963)

[Part 4 – Monitoring 9](#_Toc330804964)

[Part 5 –Workloads That Compete 10](#_Toc330804965)

[Best Practices 16](#_Toc330804966)

[Troubleshooting 17](#_Toc330804967)

[Conclusion 18](#_Toc330804968)

[References 18](#_Toc330804969)

# Introduction

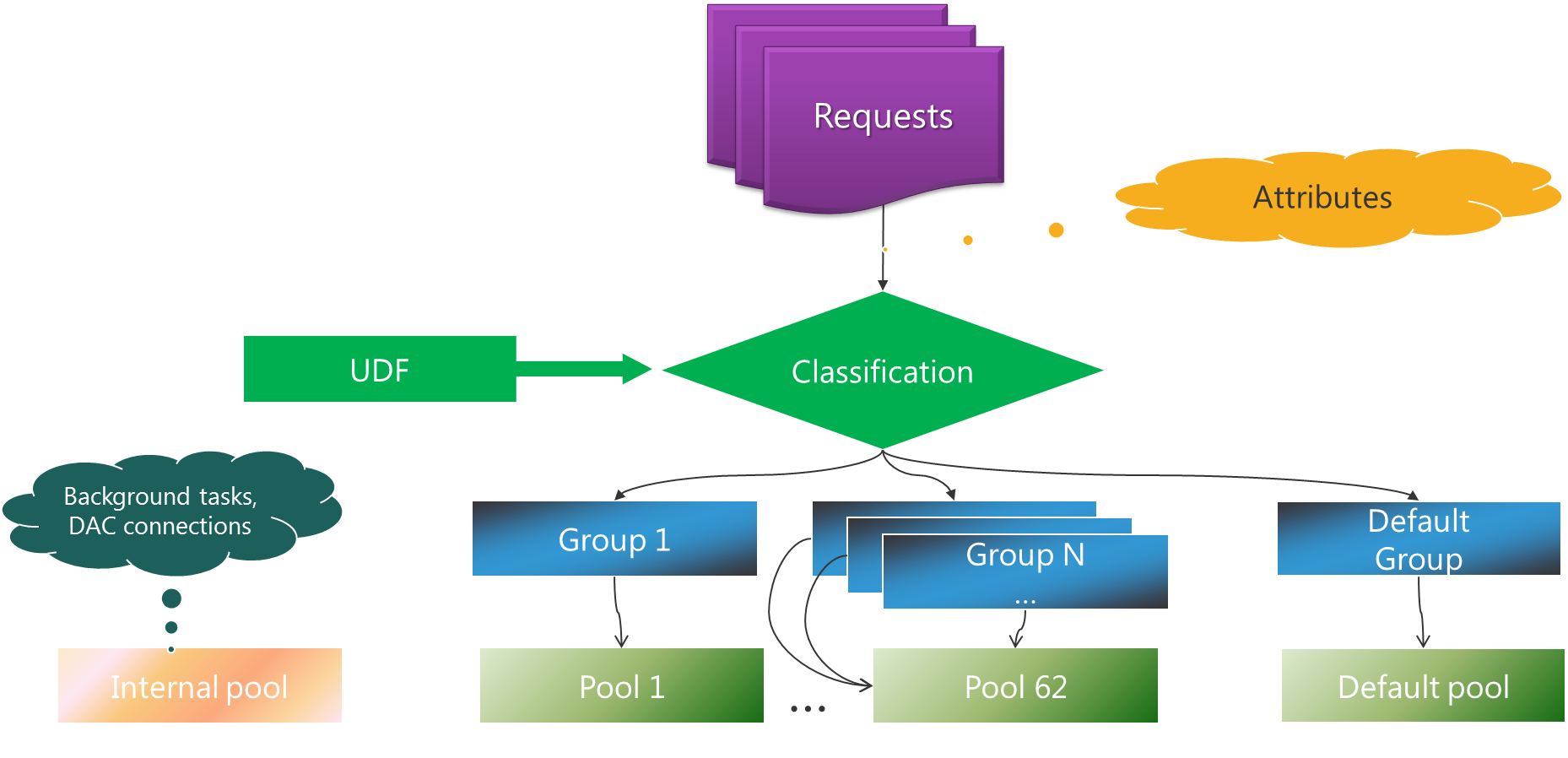
[SQL Server Resource Governor](http://msdn.microsoft.com/en-us/library/bb933866.aspx) was introduced in Microsoft SQL Server 2008 Enterprise as a way to provide multitenancy and resource isolation for single SQL Server instances serving multiple client workloads. The Resource Governor feature allows you to set limits on the amount of memory and CPU resources incoming requests can use, and it provides a way to isolate and limit runaway queries, adding fine-grained resource tracking for chargeback and delivering predictable performance. For more information about this feature, see [Resource Governor](http://msdn.microsoft.com/library/bb933866.aspx) (http://msdn.microsoft.com/library/bb933866.aspx) in SQL Server Books Online.

With SQL Server 2012 you can provide more complete isolation of CPU resources for workloads and put caps on CPU usage for a higher level of predictability, and you can control a greater proportion of SQL Server memory allocations.

**Resource Pools, Workload Groups, and Classification**SQL Server Resource Governor introduced the concept of *resource pools* as a fundamental implementation of resource isolation within a SQL Server instance. Resource pools are controlled with Transact-SQL and other management interfaces such as [SMO](http://msdn.microsoft.com/en-us/library/ms162169.aspx), and they can be assigned minimum and maximum CPU and memory resources through the CREATE RESOURCE POOL and ALTER RESOURCE POOL statements.

SQL Server 2012 supports up to 62 user-definable pools, increased from 18 in SQL Server 2008. There are also two built-in resource pools: one called **internal** which is reserved for system tasks and is not configurable, and a user-configurable pool called **default** whereworkloads will run by default.

Each user resource pool can be associated with one or more *workload groups*, which are logical entities that represent one or more client workloads.Incoming sessions can be associated with these workload groups by way of a user definable **Classifier** function which runs after login and can call system functions to evaluate various login properties such as user name, workstation name, database name. The following diagram shows how these components fit together to classify incoming connections into resource pools.



**Figure 1**: Resource Governor session classification

For more information, including a definitive and in-depth introduction to the Resource Governor features in SQL Server 2008 and SQL Server 2008 R2, see the white paper [Using the Resource Governor](http://msdn.microsoft.com/library/ee151608.aspx) (http://msdn.microsoft.com/library/ee151608.aspx) by Aaron Bertrand and Boris Baryshnikov.

This article examines the new features introduced in SQL Server 2012, discusses why they were added, and describes how to use them.

# New Resource Governor Features in SQL Server 2012

## Why Change Resource Governor?

As the number of CPU cores per socket increases and the cost of large machines decreases, we have heard comments like the following from SQL Server hosting partners who use the Resource Governor to isolate SQL resources for their customers: “*With newer commodity servers having 10 and 12 cores per CPU, we are stranding CPU resources on servers since we are limited by the maximum number of pools.*“ Hosters who want to provide their clients with cost-efficient database as a service offerings based on SQL Server need more flexibility to divide up the resources available to large machines between workloads, and to minimize the problems that occur when workloads compete with one other for resources and one workload has a negative impact on the resources available to another workload—the so-called “noisy neighbor” problem.

With the increasing power of machines and virtualization software, the number and size of SQL Server instances being virtualized and managed in private clouds increases. As a result it is becoming more important to provide users of shared resources with more effective isolation. Workloads need to run on partitioned resources within a SQL Server instance without suffering noisy-neighbor problems.

Another trend we have seen is an increase in the use of chargeback for resource usage. Tracking and charging for the use of internal resources within enterprises and by public hosting providers is becoming more commonplace, with a consequent need to provide predictable billing to the consumers of these resources.

## New Feature Summary

To address these needs, the following features were added in SQL Server 2012:

* Increasing the maximum number of resource pools from 20 to 64, enabling partitioning of larger machines between more workloads.
* Adding a CAP\_CPU\_PERCENT option to resource pools to set a hard-cap limit on CPU resource usage, providing greater predictability.
* Adding an AFFINITY option to resource pools, allowing individual resource pools to be affinitized to one or more schedulers and NUMA nodes, providing greater isolation of CPU resources.
* Governance of multiple page allocations by the SQL Server Memory Manager in addition to single page allocations. This is a result of a major redesign in SQL Server 2012, with the Memory Manager becoming the central operating system interface for page allocation. For more information about Memory Manager changes, see the [SQLOS & Cloud Infrastructure Team blog](file:///C:\Users\guybo\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\N4K2BAZ1\SQLOS%20&%20Cloud%20Infrastructure%20Team%20blog) (http://blogs.msdn.com/b/sqlosteam/).

To show how the new resource governance features work and how to use them, here is a simple demonstration that shows workload isolation in action.

# A Sample Workload Isolation Use Case

In this scenario, you are hosting a private cloud and maintaining a centralized database service that supports two departments, Sales and Marketing. These departments share the same database between two CPU-intensive workloads. To keep track of costs, you track the resources each department uses. Then you charge the departments individually for the CPU resources they use.

Both departments require predictable performance for their queries. The Sales department wants its queries to always have the resources they need, and ideally it wants an isolated experience of 100 percent of a CPU core available to them at all times. Yet at the same time its users must share the database instance with other departments. Meanwhile, the Marketing department has lower-priority queries. They still need predictable performance, but within a fixed resource budget.

Your job is to provide workload isolation, predictable performance, and predictable billing in this environment.

**Note**: To show the new features clearly, this walkthrough focuses on CPU resources. It does not cover governing memory, which can also be controlled by using SQL Server Resource Governor.

The following walkthrough can be stepped through as a tutorial. To try it yourself, start with a machine or a virtual machine with SQL Server 2012 Enterprise, Evaluation, or Developer installed, configured with two CPU cores.

## Part 1 – Initial Setup – Create Users and Workloads

First, create the user logins for the Sales and Marketing workloads. (This assumes that SQL Server mixed mode authentication is enabled for this instance as an instance property.)

CREATE LOGIN Sales WITH PASSWORD = 'UserPwd', CHECK\_POLICY = OFF

CREATE LOGIN Marketing WITH PASSWORD = 'UserPwd', CHECK\_POLICY = OFF

CREATE USER Sales FOR LOGIN Sales;

CREATE USER Marketing FOR LOGIN Marketing;

GO

The first part of the demo shows workloads competing against one another for CPU time, so before you start using Resource Governor, affinitize SQL Server to a single CPU so that all workloads must share a single CPU.

sp\_configure 'show advanced', 1

GO

RECONFIGURE

GO

sp\_configure 'affinity mask', 1

GO

RECONFIGURE

GO

## Part 2 - Set Up Resource Governor

SQL Server Resource Governor will be used to associate the Sales and Marketing workloads with separate resource pools. This way you can track their resource usage separately with Performance Monitor. However, initially no further Resource Governor configuration will take place; that is, no CPU resources will be allocated to the resource pools yet, so users will not see any difference between using Resource Governor and not using it.

1. Create the resource pools.

CREATE RESOURCE POOL SalesPool

CREATE RESOURCE POOL MarketingPool

1. Create the workload groups.

CREATE WORKLOAD GROUP SalesGroup

USING SalesPool

CREATE WORKLOAD GROUP MarketingGroup

USING MarketingPool

GO

1. Create a classifier function that is triggered when a user connects. The function associates the connection with a specific workload group based on the incoming user’s login (that is, whether the login comes from Sales or Marketing).

CREATE FUNCTION CLASSIFIER\_V1()

RETURNS SYSNAME WITH SCHEMABINDING

BEGIN

DECLARE @val varchar(32)

SET @val = 'default';

IF 'Sales' = SUSER\_SNAME()

SET @val = 'SalesGroup';

ELSE IF 'Marketing' = SUSER\_SNAME()

SET @val = 'MarketingGroup';

RETURN @val;

END

GO

ALTER RESOURCE GOVERNOR

WITH (CLASSIFIER\_FUNCTION = dbo.CLASSIFIER\_V1)

GO

1. Now activate Resource Governor.

ALTER RESOURCE GOVERNOR RECONFIGURE

GO

## Part 3 – Run the Workload

The following simple loop demonstrates a CPU-intensive workload to illustrate clearly the effects of using the Resource Governor. That is, it just makes the CPU work without doing anything else.

SET NOCOUNT ON

DECLARE @i INT

DECLARE @s VARCHAR(100)

SET @i = 100000000

WHILE @i > 0

BEGIN

SELECT @s = @@version;

SET @i = @i - 1;

END

If you save this loop as a file with the name workload.sql, you can start the Sales workload at a command prompt as follows.

start sqlcmd -S localhost -U Sales -P UserPwd -i "workload.sql"

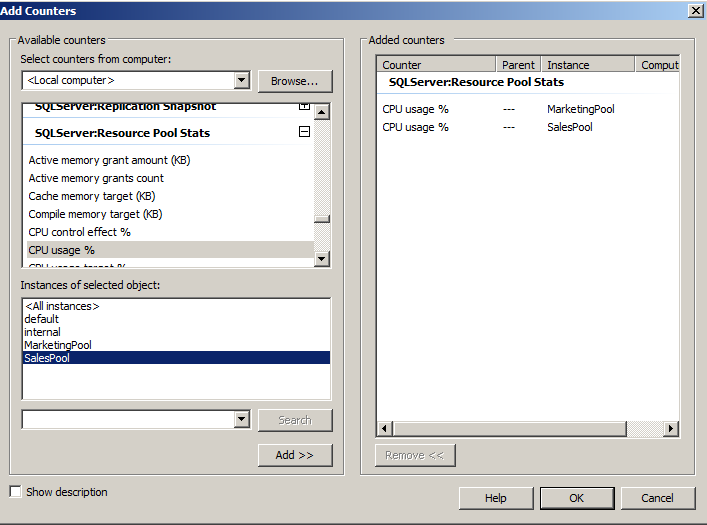
With this workload running, the CPU usage for SQL Server goes straight to 100 percent of the single core that SQL Server is affinitized to.

## Part 4 – Monitoring

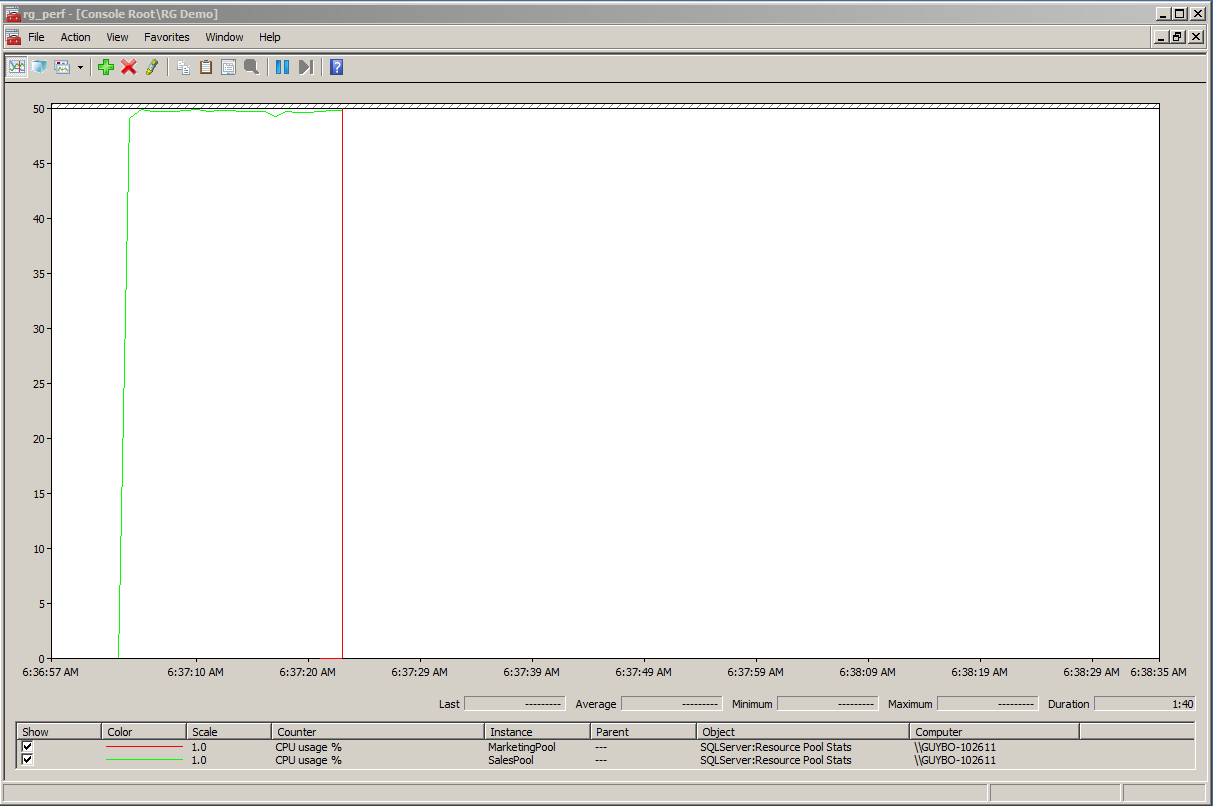
You can track the resource usage by pool using Performance Monitor. The relevant performance counter is:

**SQLServer:Resource Pool Stats->CPU usage**

You can add a counter to track for each pool by using the **Add Counters** dialog box.



Note that on a two-CPU machine, if SQL Server is affinitized to a single CPU, the CPU usage of the Sales pool will go to a maximum of 50 percent on the graph (that is, 100 percent of one CPU) like this.



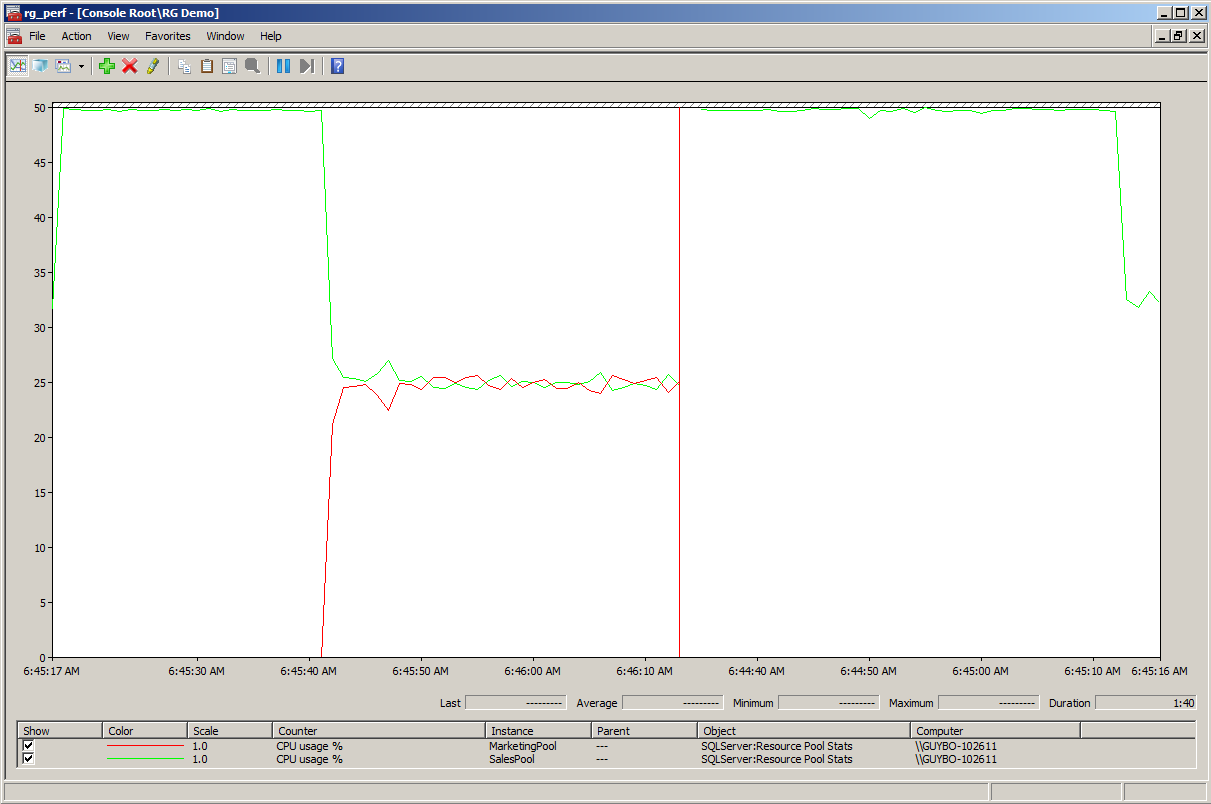
So any time you see the chart counters in this walkthrough, double them to get the CPU usage percentage of a core.

## Part 5 –Workloads That Compete

Add another workload to represent Marketing.

start sqlcmd -S localhost -U Marketing -P UserPwd -i "workload.sql"

Now the two workloads are competing and the CPU resources are split evenly between the Sales (green) and Marketing (red) resource pools.



Because the Sales queries take priority, you can make sure they are assigned a greater proportion of the CPU resources by adjusting the Sales resource pool and setting a minimum CPU value. Set this value to 70 percent for the Sales pool.

ALTER RESOURCE POOL SalesPool

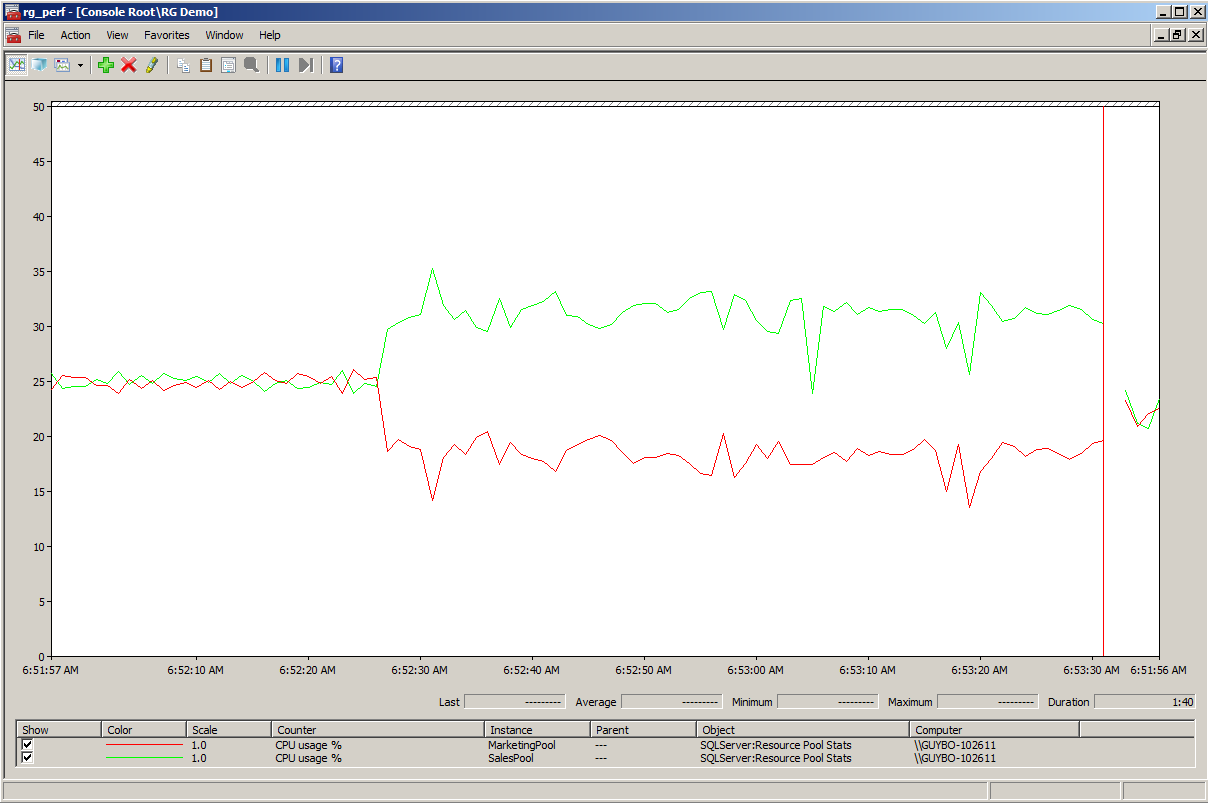
WITH (MIN\_CPU\_PERCENT = 70)

GO

ALTER RESOURCE GOVERNOR RECONFIGURE

GO

Now the Sales workload is clearly getting more of the CPU resources, and close to 70 percent of a single CPU, while the Marketing workload is getting closer to 30 percent.



When both workloads are maximizing a single CPU like this, it is still difficult for the SQL Server scheduler to divide the work into proportional quanta and give the Sales workload exactly 70 percent or more of the CPU. Later this paper discusses how to use a new feature, scheduler affinity, to isolate workloads to individual or groups of schedulers. However, in order to show how you can use Resource Governor to split up workloads running on single or shared CPUs, assume that the number of CPUs is limited and these workloads have to share a single scheduler running on a single CPU. You can make the Sales workload closer to 70 percent by setting a maximum value of 30 percent for the Marketing pool. The MAX\_CPU\_PERCENT value tells the scheduler how to allocate resources when there is contention between workloads.

ALTER RESOURCE POOL MarketingPool

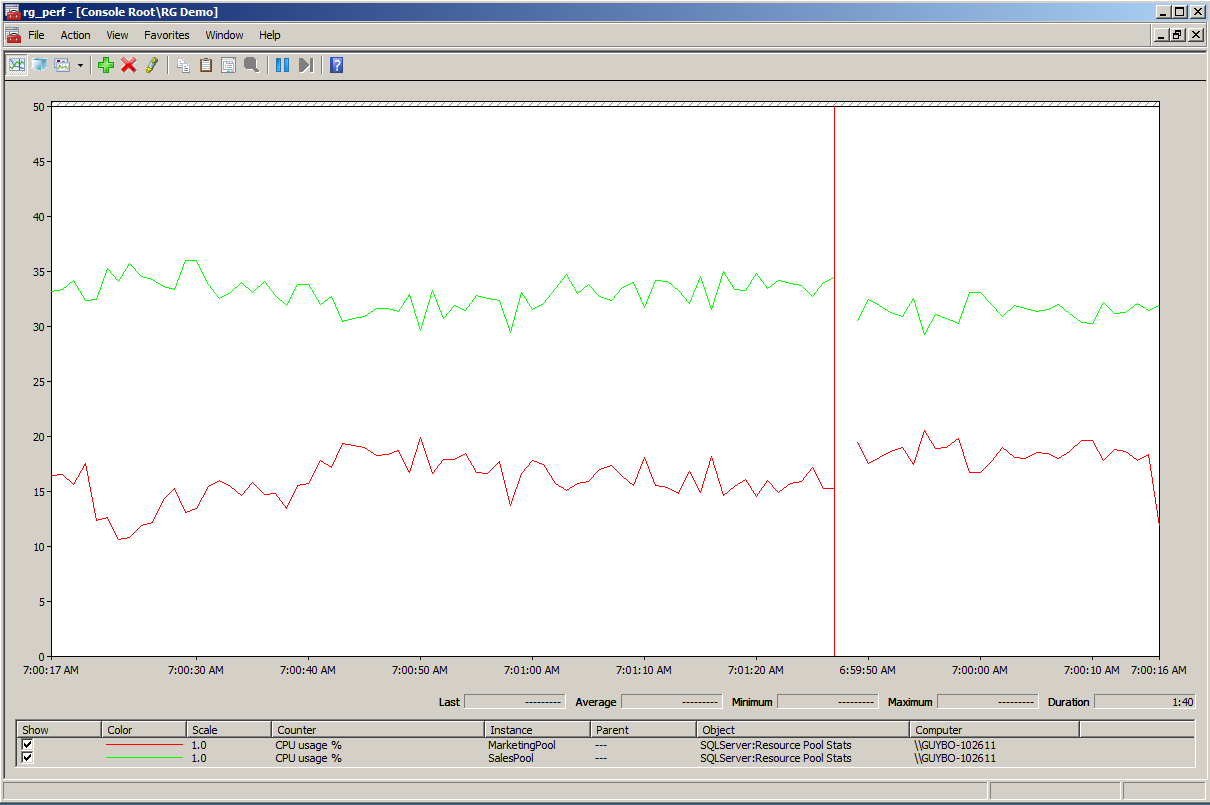
WITH (MAX\_CPU\_PERCENT = 30)

GO

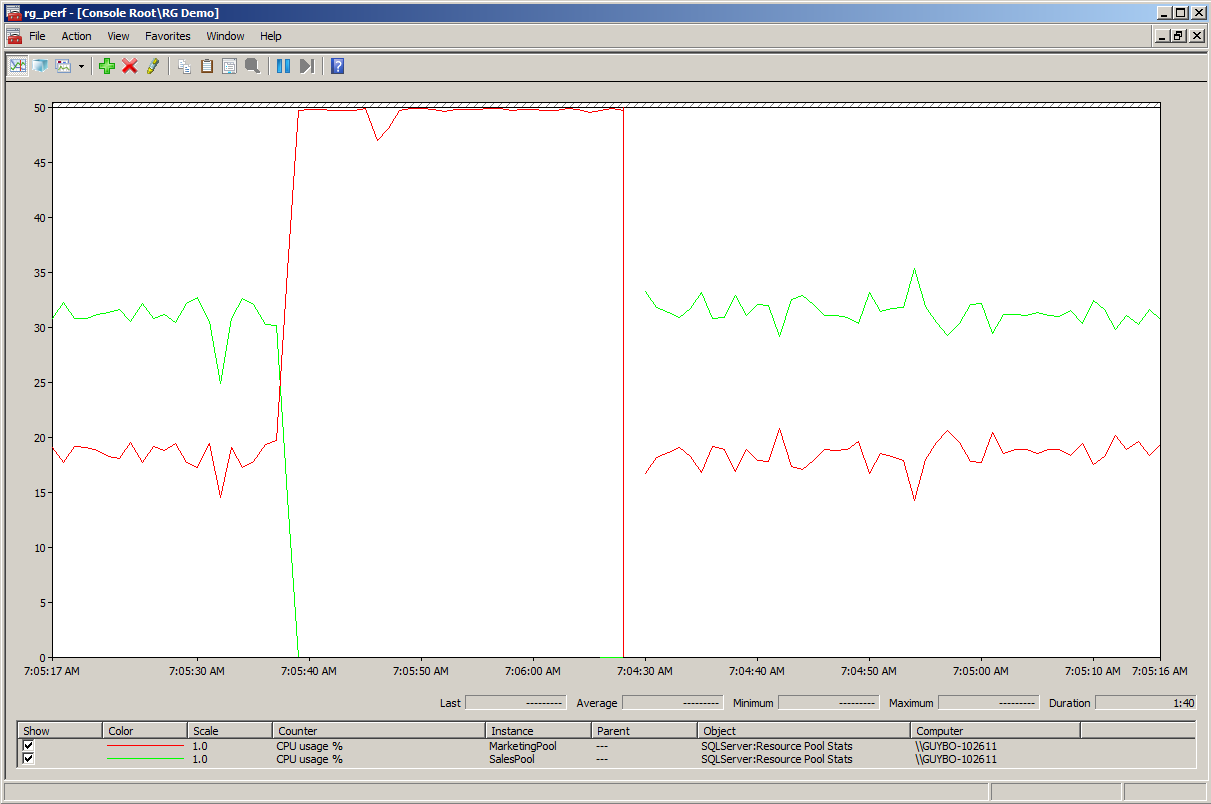
ALTER RESOURCE GOVERNOR RECONFIGURE

GO

Now the CPU resource split is much nearer the desired 70/30 ratio.



But what would happen to the Marketing workload if the Sales workload went down or stopped altogether? Would it stay at the maximum value of 30 percent? The following figure shows the results of such a change.



No. The MAX\_CPU\_PERCENT setting is an *opportunistic* maximum. If there is available CPU capacity, the workload uses it up to 100 percent. The “maximum” value only applies when there is contention for CPU, so if the Sales workload is switched off, the Marketing starts using 100 percent of the CPU.

In many circumstances this is exactly what is needed, because it is usually undesirable for the CPU to remain idle. But because in this case Marketing is being charged for their resource usage and they want predictable billing, they don’t want more than 30 percent of a core. Suppose you want to reserve resources for other workloads and always keep the Marketing workload at 30 percent or lower. You can do this by using a new Resource Governor feature in SQL Server 2012 called the **CAP\_CPU\_PERCENT** resource pool setting, which sets a hard cap on the CPU usage. Set a hard cap of 30 percent on the Marketing pool.

ALTER RESOURCE POOL MarketingPool

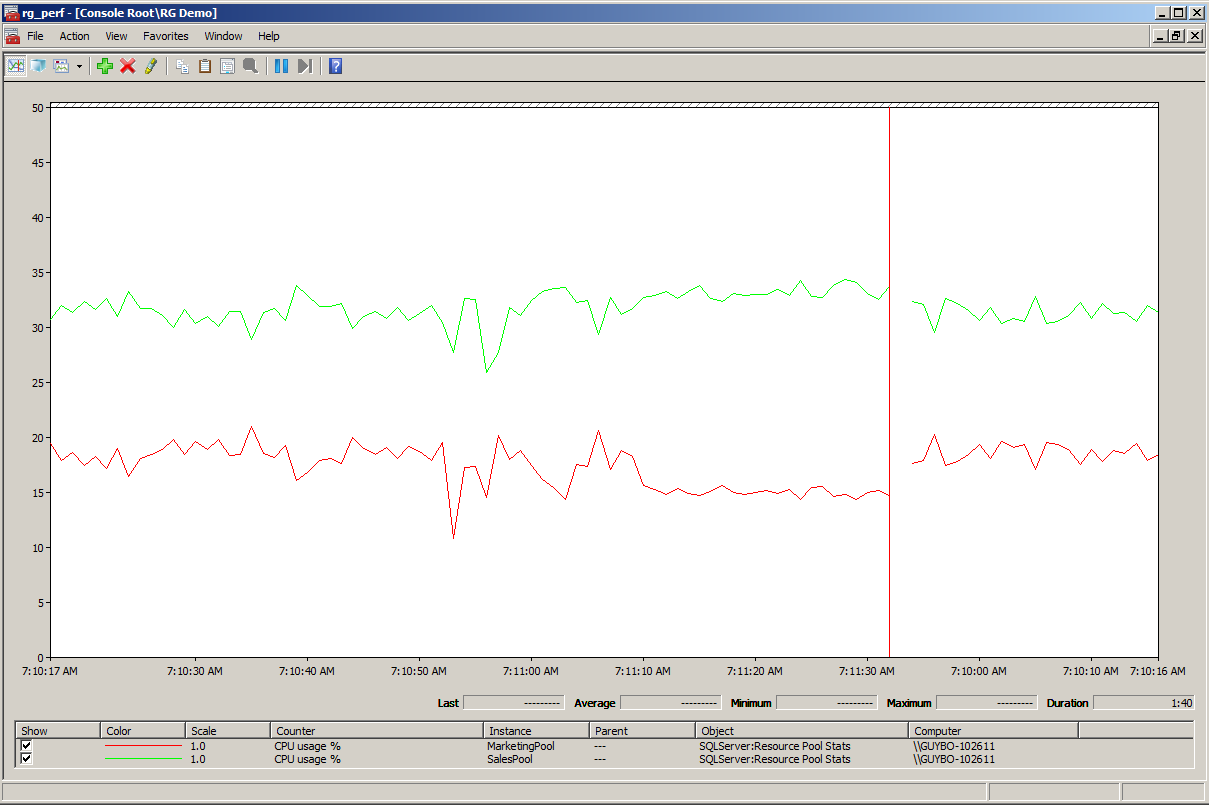
WITH (CAP\_CPU\_PERCENT=30)

GO

ALTER RESOURCE GOVERNOR RECONFIGURE

GO

Now the CPU graph for the Marketing pool flattens out and no longer goes above 30 percent. The Marketing department will get their predictable billing.



Suppose the Sales department needs a more isolated experience and wants 100 percent of a CPU core available to them at all times. They can use another new Resource Governor feature in SQL Server 2012, scheduler affinity. To show this, remove the affinity mask that you applied to SQL Server and let the SQL Server engine use both of the machine’s CPUs.

sp\_configure 'affinity mask', 0

GO

RECONFIGURE

GO

Now you can affinitize each pool to its own scheduler using the new resource pool AFFINITY option. In this case the option effectively makes a single CPU core available to each workload.

ALTER RESOURCE POOL SalesPool

WITH (AFFINITY SCHEDULER = (0))

GO

ALTER RESOURCE POOL MarketingPool

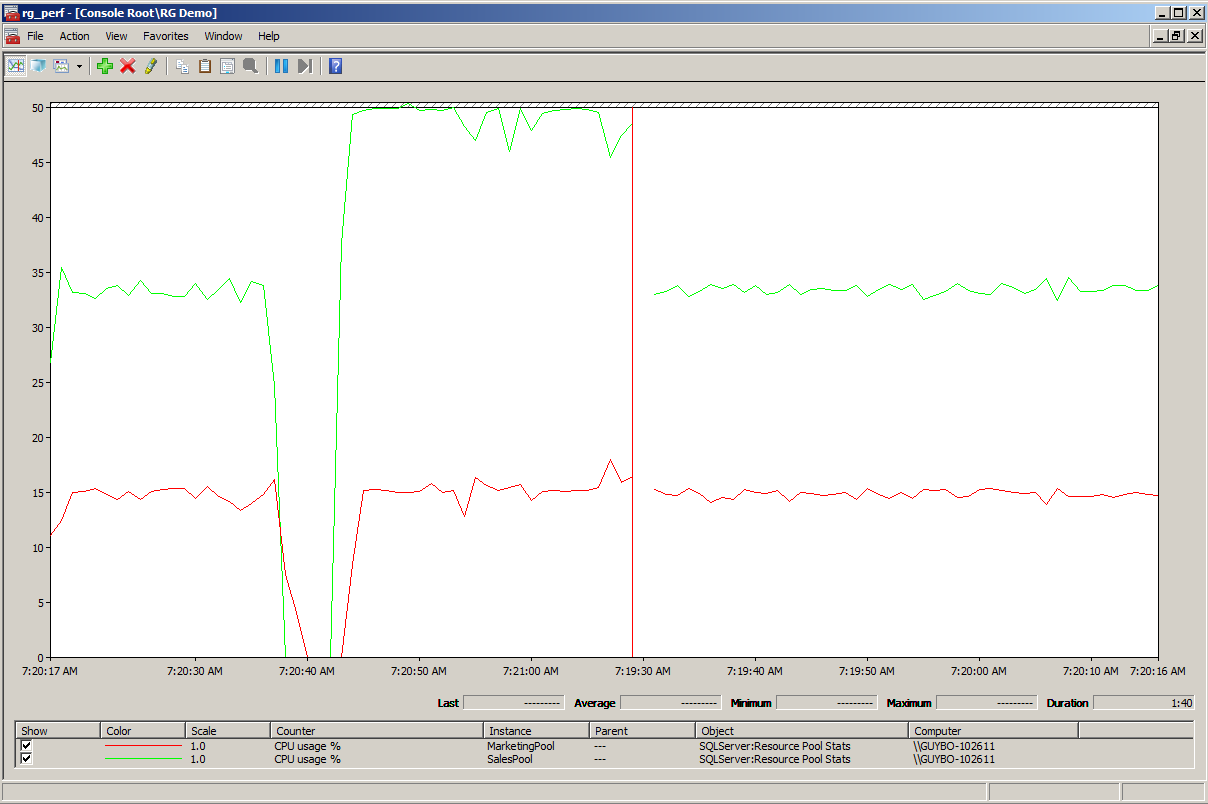
WITH (AFFINITY SCHEDULER = (1))

GO

ALTER RESOURCE GOVERNOR RECONFIGURE

GO

After you run this code, the workloads are scheduled on different CPUs. Now, with the CAP still in place, the Marketing workload continues to use 30 percent of one core, while the Sales workload uses 100 percent of the other core. With scheduler affinity in place, the CPU usage lines for each resource pool are now smooth, demonstrating complete and consistent isolation of CPU resources between these workloads.



Now, as far as the Sales and Marketing workloads are concerned, they are running on two isolated machines, getting the resources they need in a predictable fashion. This walkthrough showed a simple example of how some of the new SQL Server 2012 Resource Governor features can be used to provide more effective resource isolation.

# Best Practices

**Consider overall application resource consumption.**  
The demo in this paper represents an idealized workload, free of dependency on other resources such as memory and I/O. For memory, Resource Governor can provide hard caps and divide up the machine’s resources. The MAX\_MEMORY\_PERCENT resource pool setting is effectively a hard cap on memory usage rather than an opportunistic setting. One exception to memory governance is the buffer pool, which is shared to optimize for overall query performance. Resource Governor does not currently manage I/O, so other methods need to be used to minimize noisy neighbor problems with I/O, such as maximizing memory, striping, isolating I/O, and optimizing for random I/O with efficient SANs.

**Avoid sharing mixed workloads on the same CPUs.**  
When workloads compete with one another, it is easier for the scheduler to divide fairly and cap similar workloads. If you are governing resources for very variable workloads, for example, a mixture of high CPU versus high I/O, consider partitioning those workloads using scheduler affinity rather than setting CPU maximums and caps.

**Always reconfigure pool affinity after you change CPU configuration.**You affinitize a resource pool to a set of SQL Server schedulers with the expectation of isolating them to specific CPUs. What happens if the machine is shut down and CPUs are removed? For example, your SQL Server instance is running in a virtual machine and CPU usage is low, so you reconfigure the virtual machine with fewer CPUs.

At best, the scheduler affinity settings are no longer accurate, because the schedulers that a resource pool is affinitized to may now point to different CPUs. At worst they may point to nonexistent CPUs, which will result in Resource Governor failures, or affinity defaulting to Auto.

Therefore, always reconfigure scheduler affinity settings and restart the Resource Governor after you make changes to the CPU configuration.

**Follow additional best practices guidelines.**  
For more information about guidelines that cover SQL Server 2008 Resource Governor features, see the Best Practices section of the white paper [Using the Resource Governor](http://msdn.microsoft.com/library/ee151608.aspx) (http://msdn.microsoft.com/library/ee151608.aspx).

# Troubleshooting

**Are there new tools to help me with troubleshooting?**To troubleshoot issues with resource pool affinity and monitor affinity settings, SQL Server 2012 includes a new DMV:

sys.dm\_resource\_governor\_resource\_pool\_affinity

|  |  |  |
| --- | --- | --- |
| Column name | Data type | Description |
| Pool\_id | **int** | The ID of the resource pool. |
| Processor\_group | **smallint** | The ID of the Windows logical processor group. |
| Scheduler\_mask | **bigint** | A binary mask that represents the schedulers associated with the pool. |

**I am receiving scheduler affinity errors.**Suppose that when you are setting scheduler affinity for a pool to use the schedulers from 12 to 14, you see errors like this:

The SCHEDULER range that specifies SCHEDULER 12 to 14 includes at least one SCHEDULER that is not available to the current instance.  The maximum SCHEDULER number that is available to this instance is 12.

This implies that you are trying to affinitize the resource pool to schedulers that do not exist (for example, you moved your configuration to a machine that has fewer CPUs). A good way to resolve this is to query the **sys.dm\_os\_schedulers** DMV to see the exact scheduler/CPU assignments.

**I don’t see the new Resource Governor features defined in the SMO object model.**If you use Windows PowerShell or C# to configure Resource Governor using SQL Server Management Objects (SMO), you won’t be able to make use of the new features in SQL Server 2012. The new SMO classes are expected to be available in SQL Server 2012 Service Pack 1 (SP1).

**When I script out a resource pool definition in SQL Server Management Studio, it doesn’t show the new features.**  
The scripting enhancements to SQL Server Management Studio to show the new Resource Governor features like CAP\_CPU\_PERCENT and scheduler affinity have been added to SQL Server 2012 SP1.

# Conclusion

The new Resource Governor features in SQL Server 2012 make it easier to divide a machine’s CPU and memory resources to provide workload isolation and predictable performance to individual workloads in a multitenant environment. Because individual machine specifications are increasing in terms of CPU cores and memory, and therefore have the potential to serve isolated resources to a greater number of customers, the maximum number of resource pools has also increased in SQL Server 2012, from 20 to 64. In combination these features provide better multitenancy by supporting the partitioning of a machine’s compute resources into a larger number of more isolated pools.

For more information about the Resource Governor in SQL Server 2012, see [Resource Governor](http://msdn.microsoft.com/library/bb933866.aspx) [(](file:///C:\Users\guybo\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\N4K2BAZ1\()http://msdn.microsoft.com/library/bb933866.aspx) in SQL Server Books Online.

## References

[Resource Governor Overview](http://msdn.microsoft.com/en-us/library/bb933866.aspx)

[Using the Resource Governor](http://msdn.microsoft.com/en-us/library/ee151608.aspx) by Aaron Bertrand and Boris Baryshnikov

[SQL Server Management Objects (SMO) Programming Guide](http://msdn.microsoft.com/en-us/library/ms162169.aspx)

[SQL Server Cloud Infrastructure team blog](http://blogs.msdn.com/b/sqlosteam/)

**For more information:**

<http://www.microsoft.com/sqlserver/>: SQL Server Web site

<http://technet.microsoft.com/en-us/sqlserver/>: SQL Server TechCenter

<http://msdn.microsoft.com/en-us/sqlserver/>: SQL Server DevCenter

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

* Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
* Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

[Send feedback](mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20Resource%20Governor%20in%20SQL%20Server%202012).