

# SQL Server 2008数据仓库可伸缩性相关功能介绍

SQL Server 技术文档

作者: Eric N. Hanson, Kevin Farlee, Stefano Stefani, Shu Scott, Gopal Ashok, Torsten Grabs, Sara Tahir, Sunil Agarwal, T.K. Anand, Richard Tkachuk, Catherine Chang, 以及 Edward Melomed, Microsoft Corp.

技术检查员： Eric N. Hanson, Microsoft Corp.

发布日期: 2007年12月

适用产品: SQL Server 2008

摘要: SQL Server 2008在数据仓库的可伸缩性方面有了巨大的飞跃，从来没有一款产品能够像SQL Server 2008这样轻松的满足大型企业对于数据仓库的需求。SQL Server 2008中集成了一系列的产品来帮助您构建数据仓库并对其中的数据进行查询和分析。这其中包括SQL Server 的关系型数据库系统，Analysis Services, Integration Services, 以及Reporting Services. 此文档将介绍上述组件所带来的高性能以及新的管理功能，它们都能够提高数据仓库的可伸缩性。

### 

### 目录

[SQL Server 2008数据仓库可伸缩性相关功能介绍 1](#_Toc197280818)

[目录 3](#_Toc197280819)

[简介 1](#_Toc197280820)

[数据仓库新功能 1](#_Toc197280821)

[SQL Server 关系型数据库管理系统针对数据仓库的改进 1](#_Toc197280822)

[星型关联 2](#_Toc197280823)

[分区表并行 3](#_Toc197280824)

[**对齐分区索引视图** 4](#_Toc197280825)

[GROUPING SETS 4](#_Toc197280826)

[MERGE 5](#_Toc197280827)

[变更数据捕捉 6](#_Toc197280828)

[**最低限度日志记录INSERT** 7](#_Toc197280829)

[**数据压缩** 8](#_Toc197280830)

[**备份压缩** 8](#_Toc197280831)

[Resource Governor 8](#_Toc197280832)

[集成服务的改进 10](#_Toc197280833)

[Lookup 性能 10](#_Toc197280834)

[**管道性能** 11](#_Toc197280835)

[分析服务的改进 12](#_Toc197280836)

[MDX查询性能：块计算 12](#_Toc197280837)

[**查询和回写性能** 14](#_Toc197280838)

[**分析服务针对备份的增强** 14](#_Toc197280839)

[**用于分析服务的可扩展共享数据** 15](#_Toc197280840)

[**报表服务的改进之处** 16](#_Toc197280841)

[**报表可伸缩性** 16](#_Toc197280842)

[**服务器可伸缩性** 16](#_Toc197280843)

[总结 16](#_Toc197280844)

[参考文献 17](#_Toc197280845)

#### 简介

Microsoft® SQL Server™ 2008 提供了一套全面的数据仓库平台，它可以通过集成统一的产品套装来帮助你构建、管理数据仓库，并实现对用户的洞察力，另外还可以让最终用户和IT人员更具竞争力，以满足大型企业的需求。

SQL Server 2008中首屈一指的改进就是在整个产品套装中提升了可伸缩性，以充分满足大型企业的需求。在这里我们将介绍那些新功能和改进能够提升数据仓库方面的体验。构建、管理、呈现，这在SQL Server 2008中都可以轻松实现。

#### 数据仓库新功能

下表显示了SQL Server 2008中可扩展性方面的新功能，以及它们在那些方面可以为数据仓库相关的操作提供帮助。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 构建 | 管理 | 提供洞察力 |
| SQL Server 关系型数据库管理系统 | MERGE 语句  变更数据捕捉（CDC）  最低限度日志记录INSERT | 备份时采用压缩技术 | 星型模型的性能  在分区表中更快速的实现并行查询  GROUPING SETS |
| Resource governor | |
| 数据压缩  对齐分区索引视图 | | |
| 集成服务 | Lookup 性能  管道性能 |  |  |
| 分析服务 |  | 备份 | MDX 查询性能: 块计算查询以及回写性能 |
| 可伸缩的共享数据库 | | |
| 报表服务 |  | 报表的可伸缩性  服务器的可伸缩性 | |

表 1: SQL Server 2008 的改进

这篇白皮书概述了SQL Server 2008中每一个组件在数据仓库方面的改进，以及它们如何帮助你从数据仓库中获得更多的好处。如需完整详细的了解如何使用这些功能，请参考SQL Server 2008 联机丛书（BOL）

#### SQL Server 关系型数据库管理系统针对数据仓库的改进

SQL Server 2008 的关系型数据库管理系统同之前的版本相比有了巨大改进，因此在创建、管理以及查询大型数据仓库时的性能会更好。下面这部分内容将详细说明表1中所列出的关系型数据库管理系统在数据仓库方面的改进。

##### 星型关联

如果采用维度化对数据仓库进行模型化，那么你的主要工作将集中在星型关联查询中。这些查询都遵循一个通用的模型，即事实数据表（fact table）同一个或多个维度表（dimension table）进行联接。另外星型查询通常依靠维度表的非键列来实现筛选条件并在事实数据表的某一列(称之为*measure*列)中执行聚合(例如SUM)。对于行的碎片处理，很多种星型查询的性能在SQL Server 2008中都得到了显著提升。

新技术依靠bitmap filter功能，即*Bloom filters* (参考Wikipedia 2007 ,*Bloom filter,* <http://en.wikipedia.org/wiki/Bloom_filter>)，它可以让SQL Server 在查询赋值过程中，针对后续处理及早的排除不符合条件的行。同其它产品所使用的查询处理技术相比，这样可以大量节约CPU 使用时间。据我们观察，在使用新的星型查询时，整个关系型数据仓库的性能会提高15-25%。一些个别查询速度甚至能够比以前快7倍以上。

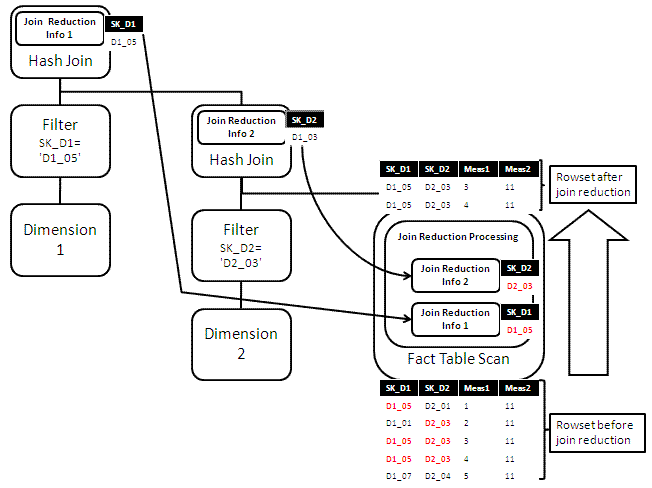


图 1: 对于高效数据仓库，星型查询可以减少联接处理

新的星型联接使用一系列的哈希联接，为所涉及的每一个维度表构建一个哈希表。构建哈希表会产生一些额外信息，名叫*bitmap filter*, Bitmap filter 即图1中标记为“Join Reduction Info.”的方框，对于那些在联接过程中将被排除的行，这些筛选器会在扫描事实数据表时先将它们几乎全部排除掉。如此一来CPU 就不需要花时间来复制将会被排除掉的行，也不需要在哈希表中去检测它们，从而节约了时间。上图中显示了在对事实数据表扫描时的筛选效果。SQL Server 2008的查询执行器同样会在执行过程中对bitmap 进行重新排序，将最有可能被选中的放在首位，从而进一步节省了CPU 的时间，这是因为一旦bitmap对事实数据表的行检查失败，那么该行将被跳过。

新的星型查询优化技术只在Microsoft SQL Server 2008 企业版中提供。当可以降低查询消耗时，SQL Server 会自动按照星型联接模型对查询进行优化，你不需要对应用程序做任何修改就可以体验到性能的显著提升。

##### 分区表并行

你希望从你的硬件设备中获得最大的性能吗？SQL Server 2008中的*分区表并行（partitioned table parallelism,* PTP）功能可以帮助你实现这个愿望。数据仓库应用程序经常需要从事实数据表中收集大量的历史数据，这些数据通常是按照日期来进行分区的。在SQL Server 2005中，对不同分区中的查询使用不同的线程，这样有时会影响包含分区表的查询的性能，尤其是在配置了多核处理器的并行共享存储多处理机（shared memory multiprocessor, SMP）计算机中。分区表并行可以更好的利用现有硬件设备的处理能力，无论一个查询涉及到多少个分区，都可以提升针对分区表的并行查询性能。 该功能默认执行，不需要手动调整或配置。下图显示了在一个典型数据仓库场景中分区表并行的影响。

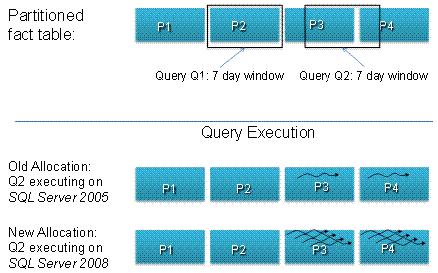


图 2: 分区表并行

假设一个事实数据表中的销售数据按照销售日期位于四个分区中，每个分区都包含七天的数据，如图表的上部分所显示。查询Q 是过去七天的销售总和。这个查询根据其执行的时间不同，可能会作用于不同的分区。如查询Q1所示，它涉及一个单独的分区P2，而Q2则涉及两个分区，因为相关的数据在执行时跨过了P3和P4。

在SQL Server 2005中执行Q1和Q2可能会产生一些意外的状态，这是因为针对单独分区的查询，系统可以分配所有的线程，因此Q1查询将由所有可用线程围绕P2进行并行处理（执行没有在图中显示）。但是在Q2情况下，即使底层硬件还有额外的可用线程，执行器也只为分区P3和P4各自分配一个单独的线程。因此在8路计算机上，Q2只利用了CPU可用资源的2/8(25%)，而且很可能比Q1执行得要慢很多。

在SQL Server 2008中执行Q1和Q2会更好地利用可用硬件，因此具有更好的性能和可预测性更强的动作。在Q1情况下，执行器依然可以将所有可用的线程用于处理P2中的数据（没有显示）。而在Q2情况下，则会生成一个并行计划，执行器按计划轮流指派所有可用线程到P3和P4，它产生的效果如图中New Allocation所示。这时CPU仍然被完全利用，而Q1和Q2的性能则相差无几。在这种全新的线程轮流指派方式下，处理器的核心越多，分区表并行所能提升的性能就越明显。当一个查询所访问的数据全部都在主内存缓冲池中——对于最近使用的分区来说这是种典型情况，我们在对涉及两个分区的查询所进行的内部测试中，速度提升幅度超过了16倍。注意，实际结果将取决于查询、数据组织和硬件配置。

有关线程分配策略以及分区表机制的管理特性，请参考SQL Server 2008联机丛书。

##### **对齐分区索引视图**

对齐分区索引视图使你能够更有效地创建并管理关系型数据仓库中的聚合，并在一些之前不能有效使用它们的场合中使用它们，从而提升了查询性能。通常情况下，你有一个按日期分区的事实数据表，在此之上定义索引视图(聚合)，以加快查询。当你转到一个新的表分区时，定义在分区表上的对齐分区索引视图所匹配分区也会自动转过去。

这与SQL Server 2005相比是个显著的提高，在SQL Server 2005中你必须在使用ALTER TABLE SWITCH转入或转出一个分区之前，删除所有定义在分区表上的索引视图。SQL Server 2008中的对齐分区索引视图功能对于大型分区表上的索引视图非常实用，同时还可以节省整个分区表上重建聚合的开销，自动维护聚合，以及实现索引视图匹配（重写自动查询以利用聚合解决只涉及基础表而不涉及聚合的查询）。关于索引视图的详细信息，请参考Microsoft TechNet 中的文章： [Improving Performance with SQL Server 2005 Indexed Views](http://www.microsoft.com/technet/prodtechnol/sql/2005/impprfiv.mspx).

下图显示了在一个分区里切换时聚合怎样跟随基础表分区移动。

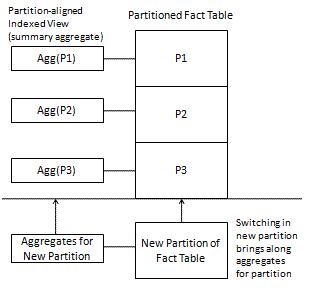


图 : 对其分区索引视图

##### GROUPING SETS

GROUPING SETS 使你能够编写一个生成多个组并返回单独结果集的查询。这个结果集等同于对不同的分组记录进行UNION ALL。使用GROUPING SETS，你可以关注于你的业务所需要的不同级别信息（分组），而不是如何组合多个查询结果。GROUPING SETS通过改进查询性能，使你可以很简单地编写出具有多个分组的报表。

下面的示例虽然简单，但是很具代表性，使用AdventureWorksDW样例数据库，你可能会在特定的报表阶段想看看下面的聚合：

* 按季度和国家统计的总销售量
* 所有国家按季节统计的总销售量
* 总销售量

如果没有GROUPING SETS ，那么你要获得这个结果就必须运行多个查询，或者使用UNION ALL 组合这些查询，以便获得一个结果集。而有了GROUPING SETS ，你的查询可以使用如下表达式：

SELECT D.CalendarYear, D.CalendarQuarter, T.SalesTerritoryCountry

, SUM(F.SalesAmount) AS SalesAmount

FROM dbo.FactResellerSales F

INNER JOIN dbo.DimTime D ON F.OrderDateKey = D.TimeKey

INNER JOIN dbo.DimSalesTerritory T ON

F.SalesTerritoryKey = T.SalesTerritoryKey

WHERE D.CalendarYear IN (2003,2004)

GROUP BY GROUPING SETS (

(CalendarYear, CalendarQuarter, SalesTerritoryCountry)

, (CalendarYear, CalendarQuarter)

, () )

ORDER BY D.CalendarYear, D.CalendarQuarter, T.SalesTerritoryCountry

通常情况下，该查询结果将显示为如下数据透视表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | Country | | | Period Totals |
| Canada | Mexico | USA |
| Year, Quarter | 2007 | Q1 | 1000 | 1000 | 5000 | 7000 |
| 2007 | Q2 | 1000 | 1000 | 6000 | 8000 |
| 2007 | Q3 | 1200 | 1300 | 7000 | 9500 |
| 2007 | Q4 | 1300 | 1400 | 8000 | 10700 |
| Grand Total: | | | | | | 35200 |

表 2: 以数据透视表格式显示的GROUPING SETS查询的输出

GROUPING SETS 所提供的简洁性和性能优势随着可分组数目的增加，越发明显。

##### MERGE

MERGE 语句允许你在一个Transact-SQL 语句中对一个表或视图执行多个数据库操纵语言（DML）进行操作（INSERT、UPDATE和DELETE）。目标数据表或视图与一个数据源联接起来，这些DML操作将在该联接的结果之上执行。MERGE 语句有三个WHEN 条件子句，使你可以对结果集中的一个给定记录执行一个专门的DML动作：

* 对于同时存在于目标表和源表中的每一条记录，WHEN MATCHED 条件子句允许你对目标表中的给定记录执行UPDATE或DELETE操作。
* 对于存在于源表中而不存在于目标表中的每一条记录，WHEN [TARGET] NOT MATCHED 条件子句允许你将一条记录INSERT到目标表中。
* 对于存在于目标表中而不存在于源表中的每一条记录，WHEN SOURCE NOT MATCHED 条件子句允许你UPDATE 或DELETE 目标表中的给定记录。

你还可以对每一个WHEN条件子句指定一个搜索条件来选择要对记录执行哪种类型的DML操作。MERGE语句的OUTPUT条件子句包括一个新的虚拟字段，叫做$action，你可以使用它来标识对每一条记录所执行的DML操作。

在数据仓库环境中，MERGE 语句用来执行对缓慢变化维度(Slowly Changing Dimensions, SCD)有效的插入和删除操作以及在很多普通场景中维护事实数据表。比其运行单独的插入、更新和删除语句，MERGE 语句具有更高的性能，因为它只要求将数据传递过来。

SQL Server 2008还推出了一个针对插入语句的扩展功能，它允许插入语句使用嵌套INSERT、UPDATE、DELETE或MERGE 语句的OUTPUT 条件子句返回的记录。

假设你有一个**DimBook**表(**ISBN、Price、IsCurrent**)，它用于跟踪一个书店中每一本书的历史价格和当前的价格。更改价格和添加新书是按周进行的，每星期会生成一个源表**WeeklyChanges** (**ISBN、Price**)，这些更改会应用于**DimBook** 表。每一本新书都会插入一条记录。在这一周中价格有变化的现有书籍会以IsCurrent=0进行更新，并且会插入一条新记录以反映新价格。下面这个Transact-SQL 语句将使用新的MERGE和INSERT功能执行这些操作。

INSERT INTO DimBook(ISBN, Price, IsCurrent)

SELECT ISBN, Price, 1

FROM

(

MERGE DimBook as book

USING WeeklyChanges AS src

ON (book.ISBN = src.ISBN and book.IsCurrent = 1)

WHEN MATCHED THEN

UPDATE SET book.IsCurrent = 0

WHEN NOT MATCHED THEN

INSERT VALUES (src.ISBN, src.Price, 1)

OUTPUT $action, src.ISBN, src.Price

) AS Changes(action, ISBN, Price)

WHERE action = 'UPDATE';

##### 变更数据捕捉

变更数据捕捉(Change Data Capture, CDC)是SQL Server 2008中推出的一个新的数据跟踪功能，它主要是为数据仓库场景设计的，可以有效的跟踪和获取对用户表所做的数据改动，并使你能够以一种简单的方式来访问变更数据。一般情况下，在一个操作数据库中使用CDC来捕捉变更，以便用于之后转移到数据仓库中。在SQL Server中使用CDC 就不再需要使用插入相关的方法，例如用户触发器、时间戳字段、以及高昂的查询来确定[操作系统](http://blog.csdn.net/onlyzhangqin/category/312865.aspx)中什么发生了改变。

与变化数据一起获得的辅助信息使得CDC可以提供许多问题的答案。例如，下面列出了一些CDC可以有效提供答案的问题：

* 我想获得在12:00 A.M.和12:00 P.M 之间发生改变的所有记录。
* 我想要知道这个改变是插入、更新、还是删除。
* 对于一条更新记录，我想知道哪个(些)字段发生了改变。

提取、转换和加载(ETL)场景下最能发挥CDC 的作用。随着数据量的增加以及全局操作所导致的维护窗口的缩减，使得优化ETL处理变得尤为重要。变更数据捕捉为你提供了一个非常有用的方法，在增量基础上提取变更，从而降低整个ETL处理时间。

下图提供了对变更数据捕捉的组件概述。

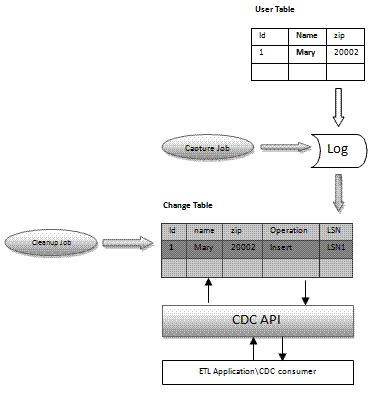


图4: 变更数据捕捉

CDC使用一个捕捉作业从SQL Server事务日志中提取变更信息，生成变更表。CDC API使你可以编写一个应用程序，从变更表中获得信息，也可以在ETL包中使用它。CDC清除作业删除了变更表中不再需要的信息。

##### **最低限度日志记录INSERT**

一般情况下，当你往一个数据库中写数据时，必须将对磁盘执行两次写入操作：一次是写到日志，一次是写到数据库中。这是因为数据库系统使用一个undo/redo 日志，所以在需要的情况下可以回滚或重做事务。但在某些重要的情况(涉及插入数据到现有的表中，从而加速你的ETL处理速度的情况)下，可以将数据只写入到磁盘一次。这就是SQL Server 2008中的最低限度日志记录INSERT特性。

最低限度日志记录只记录回滚所不支持的实时恢复的事务所需的信息，它只在批量日志记录和简单恢复模型情况下可用。当一个最低限度日志记录事务提交时，会发布一个检查点用以将脏数据页面发送到磁盘并截断日志。最低限度日志记录通过提高性能和降低所需日志空间大小，从而极大地改进了大规模INSERT操作。特别是，你必须对目标表使用表锁(TABLOCK)。

在SQL 2005中可以采用最低限度日志记录的操作包括批量导入操作、SELECT INTO 、以及索引创建和重建。SQL 2008将之扩展到INSERT INTO…SELECT FROM T-SQL 操作，它在满足任意以下条件的情况下，将大量记录插入到一个已有的表中：

* 将记录插入到一个具有集群索引而没有非集群索引的空表
* 插入到一个没有索引但是可以是非空的堆里面

使用最低限度日志记录INSERT的主要场景是：在特定的文件组上创建一个空表，这样就可以控制数据存放的物理位置。然后使用INSERT INTO…SELECT FROM以一种最低限度日志记录的形式来组装它。这样数据将放置在你所希望的地方，并且只将它写到磁盘一次。

##### **数据压缩**

在SQL Server 2008中新的数据压缩特性通过以可变长度存储的形式存储固定长度的数据，并减少冗余数据，从而降低了表、索引或其分区的子集大小。能够节省的空间大小取决于架构和数据的分布。我们使用大量数据仓库数据库进行了测试，得到的统计情况是真实的用户数据库的大小最多可以减少87%（7比1的压缩比），但是更多情况下只能降低50-70%（压缩比大约在2比1到3比1之间）。

SQL Server提供了两种压缩类型，如下所示：

* **行（ROW）**压缩能够以可变长度存储格式来存储固定长度类型。所以举例来说，如果你有一个字段数据类型为BIGINT，其固定格式将占据8个字节的存储空间，压缩之后它将使用可变的字节数——从0到8的字节数。由于字段值是以可变长度来存储的，所以在一个记录里每个字段会存储一个额外的4位长度代码。此外，0和NULL值除了这个4位代码之外不占任何存储空间。
* **页面（PAGE）**压缩是建立于行压缩的基础上的。它存储页面上普遍使用的字节格式，然后将这些值引用给各自的字段，通过这种方法将冗余数据的存储降低到最小。字节格式标识是不受类型约束的。在页面压缩中，SQL Server使用两种技术优化页面使用的空间。

第一项技术叫做*列前缀（column prefix）*。在这种情景下，系统寻找一个通用字节样式作为页面上特定列中所有数值的一个前缀。表或索引的所有列都重复这个过程。计算得来的这个列前缀值作为一个标记存储，数据或索引将这个标记作为公共前缀参考，如果可能的话，每一个字段都这么做。

第二项技术叫做*页面级字典（page level dictionary）*。这个字典存储行和列的数值，并存储在一个字典中。然后通过修改列来引用改字典。

压缩伴随着额外的CPU消耗。这是在你对压缩数据进行查询或执行DML操作时产生的。行压缩所消耗的CPU要低于页面压缩，但是页面压缩可以提供更好的压缩比。因为有很多种工作负载和数据格式，所以SQL Server 将压缩粒度确定在分区级别。你可以选择压缩整个表或索引或分区子集。例如，在一个数据仓库中，如果CPU负载是你所关注的重点，另外你还想节省一些磁盘空间，那么就可以在不常被访问到的分区上使用页面压缩，而不压缩经常被访问和操纵的分区。这样既降低了CPU总负载，又节约了磁盘空间。如果I/O性能是你所关注的重点，或者你需要提升可用磁盘空间，那么使用页面压缩来压缩所有数据或许是最好的选择。如果频繁涉及的页面的工作集缓存放在主要内存缓冲池中，那么压缩可以将速度提升好几倍，而如果放在内存中则不可以。对一个用来测试SQL Server 2008的大型内部数据仓库查询性能基准的初步测试结果显示，压缩节省了58%的磁盘空间，平均降低了15%的查询时间，另外CPU的平均消耗提高了20%，一些查询的速度甚至提高了七倍。真实结果将取决于你的工作负载、数据库和硬件。

##### **备份压缩**

备份压缩帮助你以多种方式节省空间。

降低SQL备份的大小可以节省很多磁盘空间。所有的压缩结果依赖于被压缩的数据本身，压缩到50%是很常见的，甚至可能压缩到更小。这样你可以使用较少的存储以保持备份处于联机状态，或者使用相同的存储保持更多的备份版本处于联机状态。

备份压缩还帮助你节省了时间。传统的SQL备份几乎完全受I/O性能的限制。通过降低备份过程的I/O负载，我们实际上加快了备份和恢复的速度。

当然天下没有免费的午餐，备份在空间和时间上的节省是以耗费CPU为代价的。好的一面是I/O时间的节省弥补了CPU时间的增加，而且你可以利用Resource Governor 控制备份过程所消耗的CPU比例。

##### Resource Governor

SQL Server 2008中新的Resource Governor可以控制分配给关系型数据库中不同部分的工作负载的CPU和内存资源的数量。它可以用来防止失控查询（阻止资源分配给其它工作负载的查询）以及为你的工作负载的重要部分预留资源。SQL Server 2005资源分配策略平等对待所有的工作负载，并按需分配共享资源(例如CPU带宽、内存)。这有时会引起资源分配不按比例，从而导致性能不均衡或意料外的速度降低。

Resource Governor 的主要目的如下：

1. **监控：**监控每组请求的资源消耗（工作负载分组）。
2. **可预测性：**能够对存在资源竞争的环境中预测工作负载的执行。这是通过明确制定工作负载间的资源边界来完成的（通过资源池控制）。资源边界的使用还能防止或降低查询失控的可能性。Resource Governor所提供的监控能力更容易发现失控查询。
3. **优先级：**可以设置工作负载优先级。

要理解资源监控器，有三个新的概念是很重要的：工作负载分组、资源池、分类（以及用户定义的分类器函数）。

* **组：**一个*工作负载组（workload group）*，或*组（group）*，是一个用户指定的请求分类，它与应用于每一个请求的分类规则类似。组的值存在于资源消耗聚合监控和一个用于组内所有请求的统一策略中。组为其成员定义了策略。
* **池：**一个*资源池（resource pool）*，或*池（pool）*，它显示了服务器一部分物理资源。根据它的设置，池可以有固定大小（其最大资源使用和最小资源使用设置是相等的）或者在多个池之间共享一部分（它的最小值小于它可用的最大值 ）。在这种情形下，“共享”只是意味着资源提供给最先请求资源的池。在默认配置下，所有的资源都是共享的，从而保证了对SQL Server 2005的兼容性。
* **分类：***分类*（*Classification*）是一组用户编写的规则，使得Resource Governor可以将请求分类到前面所介绍的组里面。它是通过一个分级的Transact-SQL用户定义函数(UDF)来执行的，UDF旨在作为Resource Governor的一个“分类器UDF”。

下图对这些概念进行了描述。

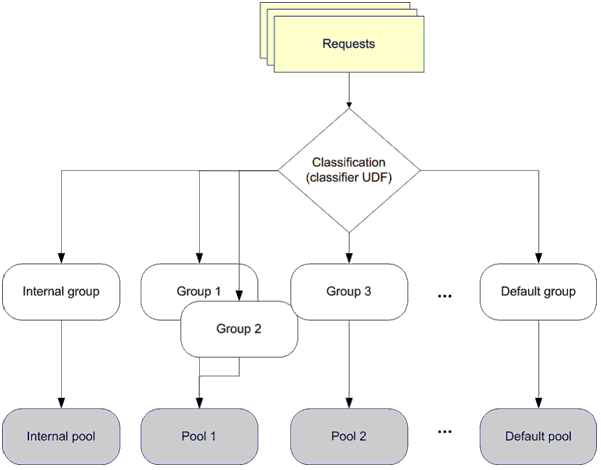


图 : Resource Governor 示例：请求，分类，组，以及池

使用Resource Governor 不一定需要更改应用程序。

#### 集成服务的改进

采用ETL将数据从操作系统中移到数据仓库中是一个很耗时的工作。为了使这个过程更快，SQL Server 2008 Integration Services (SSIS) 推出了两个重要的可扩展功能：改进的Lookup性能和改进的转换管道性能。

##### Lookup 性能

在SSIS中的Lookup 组件运行得更快，并且比在SQL Server 2005中更容易编程。lookup可以测试在记录流里每一行记录是否在另一个数据集中也有一个相匹配的记录。一个lookup就像一个数据库联接操作。一般情况下在整合过程中会使用lookup，例如从源系统中生成数据仓库的ETL层。

Lookup将建立用于保存从探测数据集获得记录的缓存。在SQL Server 2005中，Lookup组件只能从特定的OleDb连接里获得数据，而且缓存内容只能使用一个SQL查询来获得。在SQL Server 2008中，新版本的Lookup可以使用在同一个包或不同包里的单独管道来生成缓存的内容。你可以从几乎任何地方使用源数据。

SQL Server 2005在每次使用缓存的时候将它重新加载。例如，如果在同一个包里面有两个管道，每一个都要求相同的参考数据集，那么每个Lookup组件将缓存自己的副本。在SQL Server 2008中，你可以将这个缓存保存到虚拟内存或文件中存储。这意味着在相同的包里面，多个Lookup组件可以共享相同的缓存。你可以将这个缓存保存到一个文件并与其它包共享。这个缓存文件格式为了加快速度进行了优化，并且对它的访问比从原始关系型数据源重新加载这个参考数据集要快几个数量级。

在SQL Server 2008中，Lookup组件推出了不匹配缓存(miss-cache)功能。当这个组件配置为直接对数据库进行查找时，不匹配缓存功能通过有选择性的将参考数据集中的不匹配入口键值加载进缓存从而节省时间。例如，如果这个组件从进来的管道得到值123，但是Lookup组件已经知道在参考数据集里没有匹配入口，这个组件将不会再在参考数据集里查找123。这样就避免了在数据库中进行一次多余而又耗资源的查询。这种不匹配缓存特性在某些场合下可以将性能提高40%。

其它对Lookup组件的改进之处包括：

* 优化I/O路径使得缓存加载和查找操作更快速。
* 更直接的用户界面，简化了Lookup组件的配置，特别是缓存选项的配置。
* 至少同参考数据集中的一个入口不匹配的记录会被发送到不匹配输出。错误输出只处理错误，例如截断。
* 在查找转换中的查询语句可以在运行时做更改，使得编程转换更加灵活。
* 改进信息和错误消息来帮助故障排除和性能分析。

下图展示了一个使用这个新Lookup的场景。

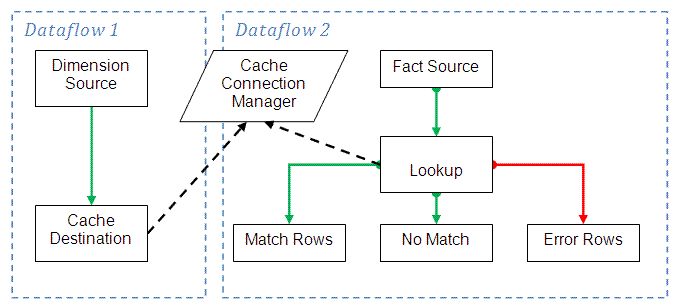


图 : Lookup 场景

数据流1从一个定制源组装了一个缓存连接管理器（Cache Connection Manager，CCM），然后数据流2使用相同的CCM来组装lookup的缓存。这个图片还显示了Lookup组件3个输出的各自用途。

##### **管道性能**

在SQL Server 2005 SSIS中要求一个单独线程自己进行的工作，在SQL Server 2008 SSIS 中可以用几个线程一起协作进行。这使你的ETL性能可以提高几倍。

在SQL Server 2005 SSIS 中，管道并行是非常粗糙的。当用户有一个简单的包，其中具有一个或两个执行树时，只会使用一个或两个处理器，并且这个包可能不会获益于具有多个处理器的计算机。即便是用户使用多点传送将数据流逻辑上分割，一个多点传送的所有输出路径页也属于同一个执行树，并且它们由SQL Server 2005 SSIS数据流任务连续执行。

为了获得高级别的并行，在SQL Server 2008 SSIS 中的管道允许更多的并行处理，这意味着使用多处理器计算机可以获得更高的性能。

通过使用一个共享线程池，多点传送的多个输出可以同时执行。简而言之，这个多点传送可以在每一个输出上具有一个可用缓冲，并且不只有一个用于传递给每一个输出的缓冲（还有一个可用线程。你不需要使用“Union All”来实现更多的并行。

例如，假设你有一个包含具有四个输出的多点传送数据流。每一个输出都流入一个聚合里。在SQL Server 2005 SSIS 中，同一时间只处理一个聚合。在SQL Server 2008 SSIS 中，这四个聚合可以并行处理。

下图显示了增强的SQL Server 2008管道并行是怎样工作的。

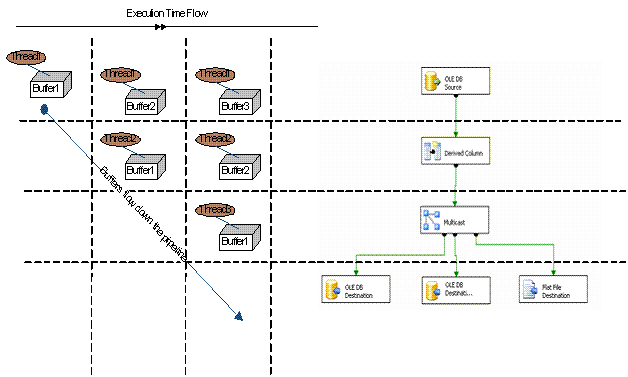


图 7: 集成服务中改进的管道并行

#### 分析服务的改进

SQL Server 2008 Analysis Services (SSAS) 采用新的块计算、回写和可扩展的共享数据库执行功能显著提高了查询速度。管理方面还提升了备份更大规模数据库的能力。

##### MDX查询性能：块计算

在SQL Server 2008 SSAS中改进的块计算可以加快MDX 查询处理，其方法是通过只处理多维数据集空间的非null值，因为它并没有在null 单元浪费时间。子空间计算的主要思想是与一个计算的逐个单元评估进行比较。假设一个RollingSum 用于计算去年和今年的销售总和，而一个查询是查找RollingSum 2005年所有产品的总和。

      RollingSum = (Year.PrevMember, Sales) + Sales

      SELECT 2005 on columns, Product.Members on rows WHERE RollingSum

该计算的逐个单元评估过程如下图所示。

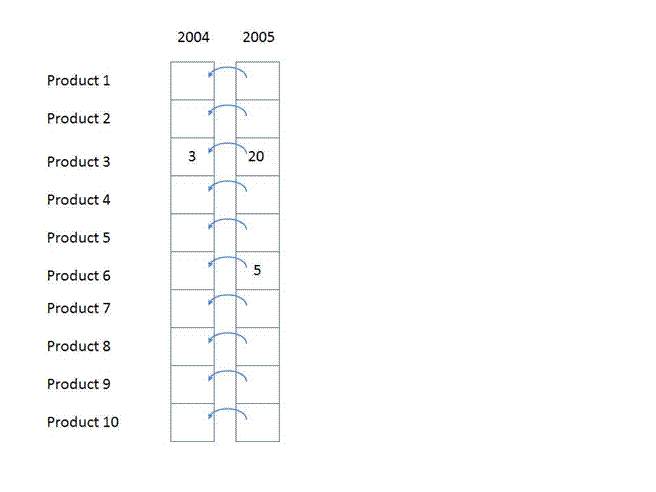


图 8: 逐个单元评估示例

[2005, all products] 的10个单元轮流评估。对于每一个单元，我们回到上一年，取得销售值，并将它添加到今年的销售里。该方法有两个明显的性能问题。

首先，如果数据是稀疏的，那么即使是会返回一个null值的单元也会被计算。在这个例子里，计算除了Product3和Product6以外的任何一个单元都是种浪费。这个影响可能极大——在一个稀疏立方体种，被评估的单元数目可能会相差好几个数量级。

其次，即使数据总的来说是密集的——意味着每一个单元都有一个值并且没有浪费时间访问空单元，也还是有重复的工作。每一个产品都重复做了相同的工作（例如获得上一年成员、为上一年单元建立新的上下文、检查递归）。将这个工作从评估每一个单元的内部循环中删除将会使得更为高效。

现在假设使用一个子空间计算方法来执行相同的例子。首先，我们以自己的方式建立一个执行树，确定应该填写哪块空间。假设我们需要为下面的查询计算空间：

[Product.\*, 2005, RollingSum]

假设有这个计算，这意味着我们必须先计算空间：

[Product.\*, 2004, Sales]

接着这个空间：

[Product.\*, 2005, Sales]

然后对这两个空间应用‘+’操作符。

销售是一个基本测量，所以我们简单获得存储引擎数据将这两个空间填写在叶子节点，然后生成这个树，应用这个操作符填写根节点的空间。因此获得了这个记录(Product3，2004，3)以及这两个记录{ (Product3，2005，20)，(Product6，2005，5)}，并对它们应用了+操作符来生成结果。

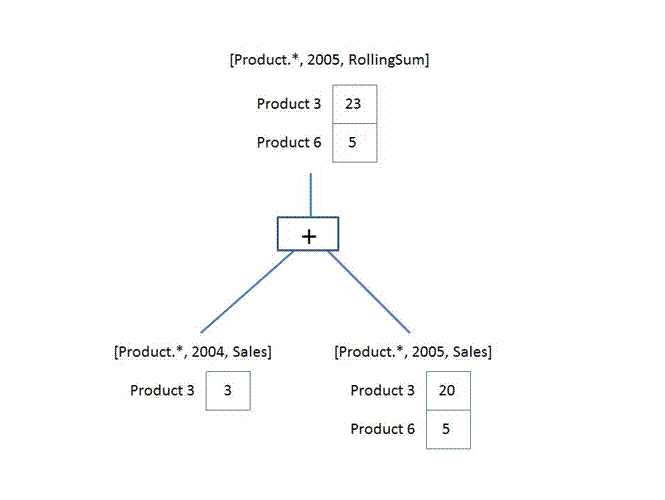


图 9: 避免对NULL单元进行操作的块计算例子

+操作符操作于空间，不是简单的数量值。它结合两个空间以生成一个包含每个空间中产品的空间，它的值是它们的总和。

我们只对可用于结果的数据进行操作，不打算对整个空间执行计算。

##### **查询和回写性能**

回写操作的性能，以及对回写数据的查询，在SQL Server 2008分析服务中获得了提高。在分析服务中的单元回写是提供给终端用户在叶子级或聚合级更新单元值的能力。单元回写为每一个测量组使用一个特别的回写分区，它存储了最新的单元值和原始值之间的不同(delta)。当一个MDX查询请求这个测量组的单元数据时，存储引擎访问所有分区，包括回写分区，并将结果聚合以生成正确的单元值。

在SQL Server 2005和更早的版本中，分析服务要求回写分区具有ROLAP存储。这通常是单元回写中发生性能问题的原因，因为ROLAP分区按需查询关系型数据源以获得它们的数据。在SQL Server 2008中，我们允许回写分区使用MOLAP存储。从压缩MOLAP格式获得回写数据比查询关系型数据源要快得多。因此，MOLAP回写分区具有比ROLAP更好的查询性能。这个性能改进的多少是很大不同的，并且取决于一些因素，包括回写数据的大小和查询本身。

MOLAP回写分区还应该提高了单元回写性能，因为服务器从内部发送查询来计算回写delta，而这些查询很可能访问回写分区。注意，回写事务提交可能会慢一些，因为服务器不只要更新回写表，还必须更新MOLAP分区数据，但是这与获得的其它性能相比就无关紧要了。

##### **分析服务针对备份的增强**

在SQL Server 2008服务中你会发现其中的一个性能改进是新的备份存储子系统。现在的备份存储子系统已经重写了，它使得可以得到更好的性能和可扩展性。这个改变对于你的应用程序来说是透明的——使用它不必改动代码。

新的备份存储子系统为分析服务备份文件推出了一个新的格式。这个文件名称扩展名没有改变。但是，内部的格式不同了，所以备份可以很好的升级到可以处理TB规模的数据库。

SQL Server 2008分析服务备份完全向后兼容SQL Server 2005分析服务。它使得你可以恢复在SQL Server 2005分析服务中备份的数据库。SQL Server 2008分析服务不具有以SQL Server 2005分析服务中所使用的旧格式来存储备份的能力。

新的高性能备份存储子系统允许客户执行新的备份场景。而在以前你需要依靠不成熟的文件系统拷贝工具来备份大型数据库，现在你可以使用与事务型系统集成在一起的内置备份子系统，并且可以与其它操作并行运行备份。

##### **用于分析服务的可扩展共享数据**

现在你可以就使用一个单独的数据库拷贝来升级你在许多小型服务器上的OLAP查询工作负载。SQL Server 2008分析服务通过一个叫可扩展的共享数据库（SSD）来支持这么做。

升级可以应用于很多场景和工作负载，例如处理、查询、数据和缓存管理。对于分析服务来说，最常见的升级场景是响应不断增加的并发用户数量，扩展多个服务器上的查询负载。这在过去是通过使用一个负载平衡解决方案来实现的，例如在多个服务器前面使用Microsoft Network Load Balancing （NLB）功能以及在服务器间复制数据。管理这样的环境会遇到许多挑战，而数据复制是主要的一个。可扩展的共享数据库特性使得数据库管理员可以将一个数据库标记为只读的，并将它从一个Storage Area Network（SAN）在多个服务器实例间共享，从而不再需要复制数据。这节省了磁盘空间，以及花费在拷贝数据上的时间。

下图描述了一个典型的SSD配置。

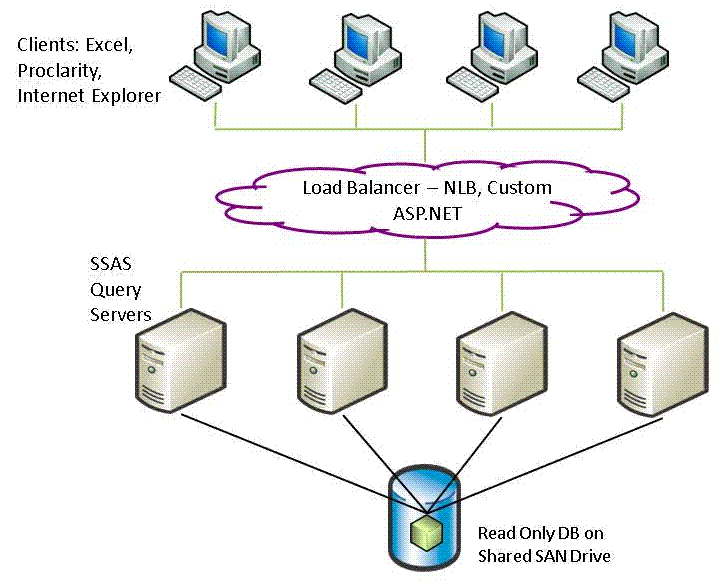


图 10: 用于分析服务的可扩展共享数据库

提高性能的一个可选解决方案是升级，用一个单独的大型服务器替代多个小型服务器。升级的好处是在一个更大型的机器上，单独的查询可以处理的更快。但是通过SSD使用升级可以为你节省硬件（假设每个处理器成本更低），并仍然满足你对许多多用户工作负载的需求。此外，SSD允许你扩展为比可以用在一个单独的大型服务器上更多的处理器。

可扩展的共享数据库特性包含三个逻辑部分：

* 只读数据库：允许将一个数据库标记为只读的
* 数据库存储位置：允许一个数据库放在服务器数据文件夹外
* 附加/分离数据库：允许从任何UNC路径附加或分离数据库

这些特性一起使得可以查询升级场景。然而，每一个特性都是独立的，并且还有查询扩展以外的用法。

用于分析服务特性的SSD与在SQL Server 2005关系型数据库中推出的SSD特性工作方式类似。

#### **报表服务的改进之处**

SQL Server 2008报表服务（SSRS）提供了性能、扩展和设计改进，使得它可以很好的满足你的企业报表需求。在这里我们着重介绍两个主要的可扩展性改进。

##### **报表可伸缩性**

SQL Server 2008报表服务报表引擎与之前版本相比具有一个较大的提高，它可以渲染比以前大得多的报表。尽管这不是数据仓库的一个重要提高（它在操作性报表中也可以使用），但是它在一些数据仓库场景中是非常有用的。如果你创建具有几百甚至上千页的报表，那么SQL Server 2008报表服务可以帮助你更快地渲染报表。而且，在相同的硬件配置下，可以渲染的最大报表规模显著地增加了。

##### **服务器可伸缩性**

SQL Server 2008报表服务不是运行在IIS内部。它可以管理它自己的内存，并具有它自己的内存限制。这使得你可以配置内存设置以便SSRS 可以更加高效地与其它服务运行在同一台机器上，例如SQL Server.

#### 总结

SQL Server提供给你在数据仓库方面所需要的所有东西。在2008版本里，它进一步扩展，比之前版本都更加可以满足最大规模企业的需求。正如这篇文章里所描述的许多数据仓库改进之处，它与之前的版本相比改进了很多。你将看到最重要的改变是用于数据仓库构建、关系型查询处理、报表和分析的可伸缩性的增强。

更多信息：

* [SQL Server Web site](http://www.microsoft.com/sql/default.mspx)
* [SQL Server TechCenter](http://technet.microsoft.com/en-us/sqlserver/default.aspx)
* [SQL Server Developer Center](http://msdn2.microsoft.com/en-us/sqlserver/default.aspx)

这篇文档对您有帮助吗？请将您的反馈提交给我们。告诉我们您的评价1(很不好)-5（很好）。您如何评价以及为何如此评价，例如：

* 您是否因为其中的优秀示例或截图、清晰的描述或其他原因打了高分？
* 您是否因为其中不恰当的示例或模糊不清的截图、不清晰的描述打了低分？

这些反馈将帮助我们提高我们今后发布的白皮书的质量。[发送反馈](mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20An%20Introduction%20to%20New%20Data%20Warehouse%20Scalability%20Features%20in%20SQL%20Server%202008)。

#### 参考文献

*Bloom filter,* Wikipedia 2007, <http://en.wikipedia.org/wiki/Bloom_filter>.

Hanson, Eric., *Improving Performance with SQL Server 2005 Indexed Views*, <http://www.microsoft.com/technet/prodtechnol/sql/2005/impprfiv.mspx>.