

# Integration Services: Performance Tuning Techniques

SQL Server Technical Article

Writers: Elizabeth Vitt, Intellimentum and Hitachi Corporation

Contributors: Donald Farmer, Microsoft Corporation; Ashvini Sharma, Microsoft Corporation; Stacia Misner, Hitachi Consulting

Published: November 2005

Applies To: SQL Server 2005

Summary: This white paper describes common performance-tuning techniques that you can apply to your SQL Server Integration Services (SSIS) data integration solutions.

## Copyright

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. **MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT**.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

 2005 Microsoft Corporation. All rights reserved.

Microsoft and Windowsare either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

## Table of Contents

[Introduction 1](#_Toc118614856)

[SSIS Engine Overview 1](#_Toc118614857)

[The run-time engine 1](#_Toc118614858)

[The data flow engine 1](#_Toc118614859)

[Buffer Usage 2](#_Toc118614860)

[Execution Trees 3](#_Toc118614861)

[Evaluating Design Alternatives 5](#_Toc118614862)

[Buffer Sizing 6](#_Toc118614863)

[Influencing buffer sizing 6](#_Toc118614864)

[Buffer guidelines 8](#_Toc118614865)

[Parallelism 9](#_Toc118614866)

[Configurable settings 9](#_Toc118614867)

[Design approaches 9](#_Toc118614868)

[Measuring Performance 12](#_Toc118614869)

[Troubleshooting 12](#_Toc118614870)

[Gaining Visibility 15](#_Toc118614871)

[Conclusion 16](#_Toc118614872)

[About the Authors 16](#_Toc118614873)

[Elizabeth Vitt, Intellimentum 16](#_Toc118614874)

[Hitachi Consulting 16](#_Toc118614875)

#### Introduction

When you architect data integration solutions, your design decisions not only determine how successfully your solution meets functional requirements, but also how well your solution meets performance requirements. To make the right performance design decisions, you need to understand the performance architecture of your data integration tool and, just as importantly, the techniques that enable you to maximize the tool’s utilization of system resources such as memory and CPU.

Microsoft® SQL Server™ 2005 Integration Services (SSIS) provides full-featured data integration and workflow engines coupled with a rich development environment for building high-performance data integration solutions. SSIS provides a variety of optimization opportunities to help you maximize resource utilization while successfully meeting the needs of your specific data integration scenario.

#### SSIS Engine Overview

Before you can take advantage of specific tuning techniques in SSIS packages, it is important to familiarize yourself with the SSIS architecture. This architecture consists of two major components: the run-time engine and the data flow engine.

##### The run-time engine

The run-time engine is a highly parallel control flow engine that coordinates the execution of tasks or units of work within SSIS and manages the engine threads that carry out those tasks. For the most part, the performance of the run-time engine is most heavily influenced by conditions external to SSIS, such as network bandwidth, and interaction with external systems such as database servers, FTP servers, or email servers. When SSIS runs an Execute SQL Task, for example, it sends a call to the target database and then waits for a response from the database server before it continues. In this scenario, the performance of the Execute SQL Task is more dependent on the performance of the query execution than on the SSIS run-time engine.

##### The data flow engine

When you use SSIS for data integration, in addition to the run-time engine, you use the data flow engine that manages the data pipeline. The data flow engine is invoked by a special task in SSIS called the Data Flow task. When the Data Flow task executes, the SSIS data flow engine extracts data from one or more data sources, performs any necessary transformations on the extracted data, and then delivers that data to one or more destinations.

With data integration solutions, you will likely spend a large part of your performance tuning time optimizing the data flow engine. Like the run-time engine, the data flow engine is influenced by external conditions; however, within SSIS, there are a variety of settings that you can manipulate to tune the data flow engine’s performance based on the requirements of the data integration operations.

#### Buffer Usage

Behind the scenes, the data flow engine uses a buffer-oriented architecture to efficiently load and manipulate datasets in memory. The benefit of this in-memory processing is that you do not need to physically copy and stage data at each step of the data integration. Rather, the data flow engine manipulates data as it is transferred from source to destination.

As this data flows through the pipeline, SSIS attempts to reuse data from prior buffers as much as possible when additional operations are performed. How buffers are used and reused depend on the type of transformations that you use in a pipeline.

* **Row Transformations** - Row transformations either manipulate data or create new fields using the data that is available in that row. Examples of SSIS components that perform row transformations include Derived Column, Data Conversion, Multicast, and Lookup. While these components might create new columns, row transformations do not create any additional records. Because each output row has a 1:1 relationship with an input row, row transformations are also known as *synchronous transformations*. Row transformations have the advantage of reusing existing buffers and do not require data to be copied to a new buffer to complete the transformation.
* **Partially blocking transformations** - Partially blocking transformations are often used to combine datasets. They tend to have multiple data inputs. As a result, their output may have the same, greater, or fewer records than the total number of input records. Since the number of input records will likely not match the number of output records, these transformations are also called *asynchronous transformations*. Examples of partially blocking transformation components available in SSIS include Merge, Merge Join, and Union All. With partially blocking transformations, the output of the transformation is copied into a new buffer and a new thread may be introduced into the data flow.
* **Blocking transformations** - Blocking transformations must read and process all input records before creating any output records. Of all of the transformation types, these transformations perform the most work and can have the greatest impact on available resources. Example components in SSIS include Aggregate and Sort. Like partially blocking transformations, blocking transformations are also considered to be asynchronous. Similarly, when a blocking transformation is encountered in the data flow, a new buffer is created for its output and a new thread is introduced into the data flow.

Transformations are not the only components that can be categorized as synchronous or asynchronous. Sources are a special type of asynchronous component. For example, an RDBMS source component creates two types of buffers: one for the Success output and one for the Error output. By contrast, destinations are a special type of synchronous component. You will see the interactions of source and destinations components when you examine the Execution Trees of a package.

#### Execution Trees

Execution trees demonstrate how your package uses buffers and threads. At run time, the data flow engine breaks down Data Flow task operations into execution trees. These execution trees specify how buffers and threads are allocated in the package. Each tree creates a new buffer and may execute on a different thread. When a new buffer is created such as when a partially blocking or blocking transformation is added to the pipeline, additional memory is required to handle the data transformation; however, it is important to note that each new tree may also give you an additional worker thread.

Examine the execution trees in the example depicted in Figure 1 and Table 1 where two Employee datasets are combined together and then aggregated to load into a common destination table.

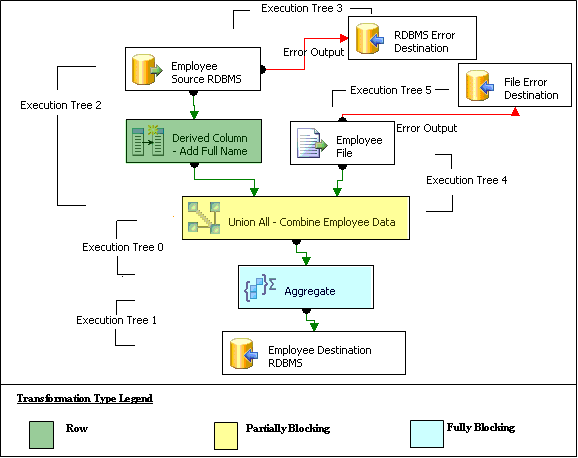


Figure 1: Example package

**Note:**Execution trees are listed in the table in the order that they execute.

Table 1: Execution trees defined

|  |  |
| --- | --- |
| Execution Tree Enumeration | Description |
| begin **execution tree 2**  output "OLE DB Source Output" (27)  input "Derived Column Input" (172)  output "Derived Column Output" (173)  input "Union All Input 1" (411)  output "Derived Column Error Output" (174)  end execution tree 2 | In **Execution Tree 2**, SSIS reads data from the Employee OLE DB Source into the pipeline, a Derived Column transformation adds another column, and SSIS passes data to the Union All transformation. All of the operations in this execution tree use the same buffer; data is not copied again once it is read into the OLE DB Source Output. |
| begin **execution tree 3**  output "OLE DB Source Error Output" (28)  input "OLE DB Destination Input" (2603)  output "OLE DB Destination Error Output" (2604)  end execution tree 3 | In **Execution Tree 3**, SSIS creates a buffer to hold error records from the asynchronous Employee OLE DB Source before loading them into a destination error table. |
| begin **execution tree 4**  output "Flat File Source Output" (2363)  input "Union All Input 3" (2098)  end execution tree 4 | In **Execution Tree 4**, SSIS reads data from the Employee Flat File Source and passes it to the Union All. These two operations use the same buffer. |
| begin **execution tree 5**  output "Flat File Source Error Output" (2364)  input "OLE DB Destination Input" (3818)  output "OLE DB Destination Error Output" (3819)  end execution tree 5 | In **Execution Tree 5**, a buffer is created to hold errors from the asynchronous Employee Flat File Source before loading them into a destination error table. |
| begin **execution tree 0**  output "Union All Output 1" (412)  input "Aggregate Input 1" (2472)  end execution tree 0 | In **Execution Tree 0**, the Partially Blocking Union All transformation is executed and a new buffer is created to store the combined data and the aggregate is calculated. |
| begin **execution tree 1**  output "Aggregate Output 1" (2473)  input "OLE DB Destination Input" (150)  output "OLE DB Destination Error Output" (151)  end execution tree 1 | In **Execution Tree 1**, after the Fully Blocking Aggregate transformation is completed, the output from the Aggregate operation is copied into a new buffer and data is loaded into the OLE DB Destination. |

This example demonstrates how execution trees can help you understand buffer usage in a common SSIS package. This example also highlights how Partially Blocking transformations like Union All and Fully Blocking transformations like Aggregate create new buffers and threads whereas Row Transformations like Derived Column do not.

Execution trees are enormously valuable in understanding buffer usage. You can display execution trees for your own packages by turning on package logging, enabling logging for the Data Flow task, and then selecting the Pipeline Execution Tree event. Note that you will not see the execution trees until you execute the package. When you do execute the package, the execution trees appear in the Log Events window in Business Intelligence (BI) Development Studio.

#### Evaluating Design Alternatives

Once you master how different transformation types influence package execution, you can make better performance design choices. In Figure 2, consider how the following two alternate designs for the same data integration scenario impact performance.

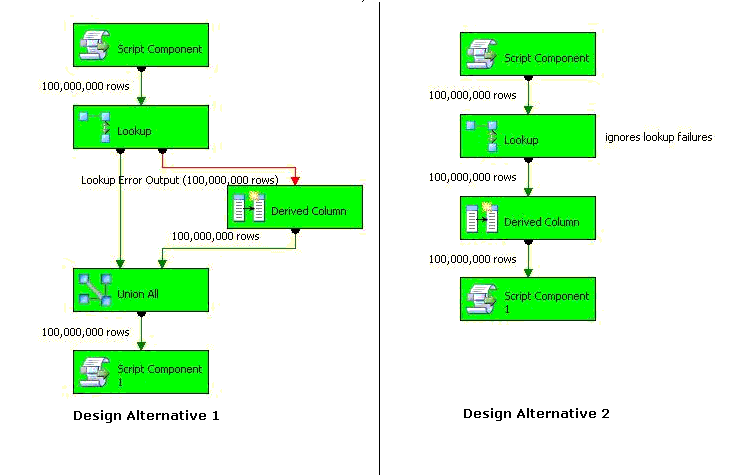


Figure 2: Design alternatives

Table 2: Design alternatives evaluation

|  |  |
| --- | --- |
| Design Description | |
| In this design, a Script Component generates 100,000,000 rows that first pass through a lookup. If the lookup fails because the source value is not found, then an error record is sent to the Derived Column transformation where a default value is assigned to the error record. After the error processing is complete, the error rows are combined with the original data set before loading all rows into the destination. | Like Design Alternative 1, this design uses the same Script Component to generate 100,000,000 rows that pass through a lookup.  Instead of handling lookup failures as error records, all lookup failures are ignored. Rather, a Derived Column transformation is used to assign values to the columns that have NULL values for the looked up column. |
| Performance Impact | |
| With two execution trees in this scenario, the biggest performance bottleneck is related to the extra copy of the data in memory created for the Partially Blocking Union All transformation. | The performance of this solution is approximately 21% faster than Design Alternative 1. With one execution tree in this scenario, the operations are consolidated and the overheard of copying data into a new buffer is avoided. |

#### Buffer Sizing

In addition to using row transformations where possible to limit the number of buffers that are created and used, within SSIS you have the ability to influence buffer sizing; that is, the number of records that are read into a buffer. Your overall goal is to pass as many records as possible through a single buffer while efficiently utilizing memory.

##### Influencing buffer sizing

At execution time before the data is read from the sources, SSIS automatically tunes buffer sizes to achieve maximum memory utilization based on a series of input parameters. To help SSIS do the best job it can when sizing buffers, you need to be aware of the following input parameters.

* **Estimated Row Size –** Estimated Row Size is not a specific SSIS setting. Rather, it is something that SSIS calculates based on the metadata that it collects about your source data at design time. You can shrink the row size by identifying the smallest possible data types for all of your columns as early in the data flow as possible. This is especially important for flat file data sources because each column is automatically read into SSIS as a string data type unless you configure the column’s data type explicitly.
* **DefaultMaxBufferRows** – **DefaultMaxBufferRows** is a configurable setting of the SSIS Data Flow task that is automatically set at 10,000 records. SSIS multiplies the Estimated Row Size by the **DefaultMaxBufferRows** to get a rough sense of your dataset size per 10,000 records. You should not configure this setting without understanding how it relates to **DefaultMaxBufferSize**.
* **DefaultMaxBufferSize** – **DefaultMaxBufferSize** is another configurable setting of the SSIS Data Flow task. The **DefaultMaxBufferSize** is automatically set to 10 MB by default. As you configure this setting, keep in mind that its upper bound is constrained by an internal SSIS parameter called **MaxBufferSize** which is set to 100 MB and can not be changed.
* **MinBufferSize** – While **MinBufferSize** is not configurable, it is still important to know about it because SSIS uses this internal parameter to gauge whether you have set the **DefaultMaxBufferSize** too low. **MinBufferSize** is defined by the granularity of your operating system’s virtual memory allocation functionality. Typically, this is set to 65,536 bytes, but it differs from machine to machine.

Depending on your configured values for these input parameters, SSIS tunes buffer sizes at execution time using one of the following scenarios.

* **Scenario 1 –** When Estimated Row Size \* **DefaultMaxBufferRows** exceeds **MaxBufferSize**, SSIS reduces the number of rows that will be stored in a given buffer to manage the memory footprint.

For example, if SSIS calculates an Estimated Row Size of 15,000 bytes per record, then the anticipated buffer size would be calculated as 15,000 bytes/record \* 10,000 records, which is approximately 143 MB and about 1.5 times greater than the **DefaultMaxBufferSize** of 100 MB. Because the anticipated size exceeds the **DefaultMaxBufferSize**, SSIS reduces the number of records per buffer approximately by a factor of 1.5 to get below the 100 MB threshold. In this scenario, each buffer is sized to hold approximately 6,600 records. Keep in mind that when this adjustment takes place, SSIS does not know how many records you have in your source file. When the data is actually processed, SSIS creates as many instances of the buffer as necessary to hold the source data set. Continuing with the example, if you have 200,000 source records, SSIS creates approximately 30 buffer instances of that buffer type. A *buffer type* just refers to a buffer’s columnar structure. For example, a buffer with columns A, B, C will be known to SSIS as a buffer with type ABC. In practice, SSIS actually goes one step further and assigns each buffer type a numeric value.

* **Scenario 2 -** When Estimated Row Size \* **DefaultMaxBufferRows** is less than **MinBufferSize**, SSIS increases the number of rows that will be stored in a given buffer to maximize memory utilization.

For example, if your Estimated Row Size is 5 bytes per record (much smaller than the previous example), 5 bytes/record\* 10,000 records, is approximately 48 KB which is less than a **MinBufferSize** of 64 KB. Because the anticipated size is less than the **MinBufferSize**, SSIS slightly increases the records to reach the 64 KB threshold.

* **Scenario 3 –**If the Estimated Row Size \* **DefaultMaxBufferRows** is somewhere in between **MinBufferSize** and **DefaultMaxBufferSize**, then SSIS attempts to size the buffer as closely possible to result of Estimated Row Size \* **DefaultMaxBufferRows** using a multiple of the **MinBufferSize** to increase memory utilization.

For example, if your Estimated Row Size is 500 bytes per record, 500 bytes/record\* 10,000 records, is approximately 4.8 MB which is less than **DefaultMaxBufferSize** of 10 MB, but greater than a **MinBufferSize** of 64 KB. In this scenario, SSIS will make adjustments to size the buffer as closely as possible to 4.8 MB.

##### Buffer guidelines

In practice, you must test these settings based on your own environment, but you can start with the following general guidelines.

* Reduce your Estimated Row Size as much as possible by removing any unnecessary columns and configuring data types correctly. Any opportunity that you have to reduce the size of the source data set before operations begin saves memory resources.
* Start with the SSIS default settings for **DefaultMaxBufferRows** and **DefaultMaxBufferSize**. Turn on package logging with the **BufferSizeTuning** property enabled. This property adds information to the log that shows you where SSIS has adjusted the buffer size. You will see entries like the following
* If your data integration is similar to Scenario 1, you will see: *Rows in buffer type 0 would cause a buffer size greater than the configured maximum. There will be only 383 rows in buffers of this type.*
* If your data integration is similar to Scenario 2, you will see: *Rows in buffer type 3 would cause a buffer size less than allocation minimum, which is 65536 bytes. There will be 1365 rows in buffers of this type.* Note that 65536 bytes is the MinBufferSize of the machine on which this package executes.
* Tweak the values for **DefaultMaxBufferRows** and **DefaultMaxBufferSize** to get as many records into a buffer as possible. Setting these values too low causes SSIS to create many small buffers instead of fewer but larger buffers, which is a great scenario if you have enough memory.
* As you tweak the **DefaultMaxBufferRows** and **DefaultMaxBufferSize**, realize that once the **MaxBufferSize** is exceeded, the setting for **MaxNumberofRows** no longer matters because SSIS always scales down the number of records per buffer to maximize memory utilization.
* Note that the **DefaultMaxBufferRows** and **DefaultMaxBufferSize** are configured separately for each Data Flow task. When you integrate data from multiple data sources using a single Data Flow task, these two settings will only impact the data source components and transformations in that task. Also note that the determination of how many rows per buffer is done per buffer type.

#### Parallelism

Parallelism is a great technique to improve the performance of your data integration operations. SSIS natively supports the parallel execution of packages, tasks, and transformations. The trick to successful parallelism is to configure operations within the constraints of your system resources.

##### Configurable settings

Within SSIS, the control flow for each package is controlled by a setting called **MaxConcurrentExecutables**, which specifies the maximum number of SSIS threads that can execute in parallel per package. By default, this is set to -1, which translates to the number of logical machine processors plus 2.

If SSIS runs on a dedicated server and you have a lot of operations that run in parallel, you will likely want to increase this setting if some of the operations do a lot of waiting for external systems to reply. On the other hand, if you do not have a dedicated SSIS machine and your data integration application runs alongside several other applications, you may need to reduce this setting to avoid resource conflicts.

##### Design approaches

As you design packages for parallelism, you need to decide whether to run some or all of the operations in the package in parallel. As with buffer sizing decisions, decisions about parallelism are best made when you take into account available system resources.

Consider the tradeoffs of different design approaches that apply parallelism to a package that reads data from a source database, aggregates the data four different ways, and then loads each aggregated data set into a different destination table.

**Note:**This information was gathered during a 64-bit performance study at the Unisys Center of Excellence using Unisys ES7000 servers. For more detailed information regarding this study, please see the ETL Performance white paper at http://www.unisys.com/eprise/main/admin/corporate/doc/ELTSQL.pdf!href(http://www.unisys.com/eprise/main/admin/corporate/doc/ELTSQL.pdf). A similar whitepaper is under development with AMD on 64 bit multi core boxes. Please check the AMD site for updates http://www.amd.com/us-en/!href(http://www.amd.com/us-en/).

* **Parallelize** **Destination** **Operations** - In the design approach pictured in Figure 3, data is read from a source, passed through one aggregation transformation with four different sets of group by operations, and then loaded into four destination tables. The only parallel operation in this design is the loading of the four destination tables from the aggregate output. The parsing of the source file and the aggregate calculation are not parallel operations.

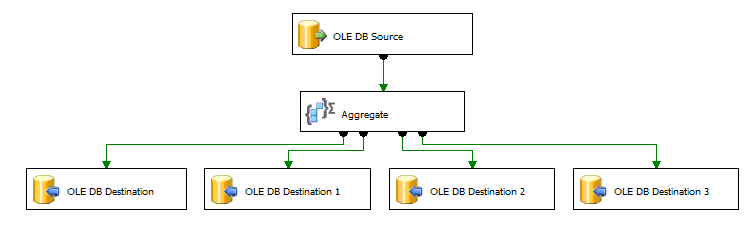


Figure 3: Parallelize destination operations

If you have a machine that has multiple CPUs, then this approach is not going to enable you to leverage them effectively. This design approach is best when you have a memory constrained machine and if you have multiple aggregates that can be derived from each other. The aggregate transform will create the aggregate with the lowest granularity and then derive all related aggregates from that. For example, if you have two aggregates: (1) sales data aggregated by year and (2) sales aggregated by year by geography, the aggregate transform automatically creates the sales aggregated by year and geography and then derives the yearly summary from that.

* **Partially Parallelize** **Operations** - In the design approach pictured in Figure 4, data is read from a source, passed through a multicast transformation that creates four identical result sets that are passed to four different aggregation transformations, and then loaded into four destination tables.

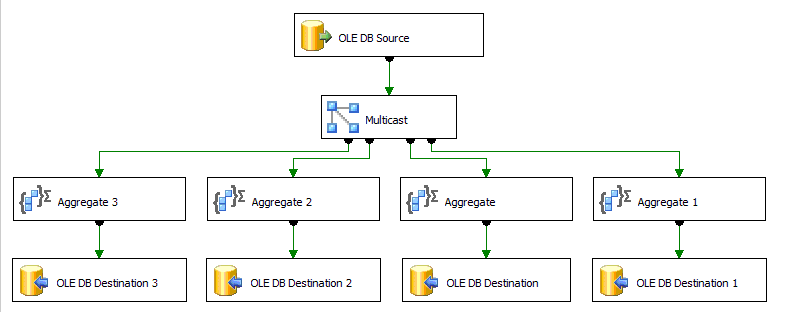


Figure 4: Partially parallelize operations

In spite of what you might think, the only parallelism in this scenario is the loading of the destination tables. Reading the data source, performing the multicast, and aggregating data each operate in the same execution tree and therefore share the same memory and threading.

If you would like to run the aggregates in parallel, in this scenario, you can introduce a Union All transformation after the Multicast to create a new execution tree. Remember that Union All is a partially blocking transformation and always creates a new execution tree. When you introduce a Union All, data will be copied into additional buffer, but you will also gain additional threads to perform the aggregate in parallel.

* **Parallelize All Operations** - In the design approach pictured in Figure 5, there are four independent sets of operations that each read data from the source, aggregate data, and load data into a distinct destination.

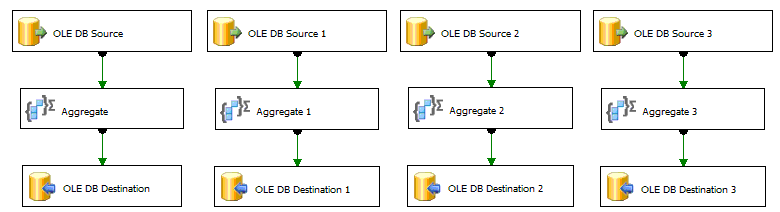


Figure 5 – Parallelize All Operations

In this scenario, all operations are performed in parallel: data source extraction, aggregate, and insertion to the database. If your server is not constrained by memory and has multiple CPUs, this approach provides a high-performance solution; however, you may be wasting resources by treating every operation uniformly and by reading the same data set four different times.

* **Optimize the Slowest -** As an alternative to uniform parallelism, you may want to consider the targeted design approach displayed in Figure 6 where you focus on the slowest component in your package that is most likely to benefit from parallelism. In this design approach, one of the four aggregates has been identified as the slowest aggregate operation and has been broken out into a separate data stream.

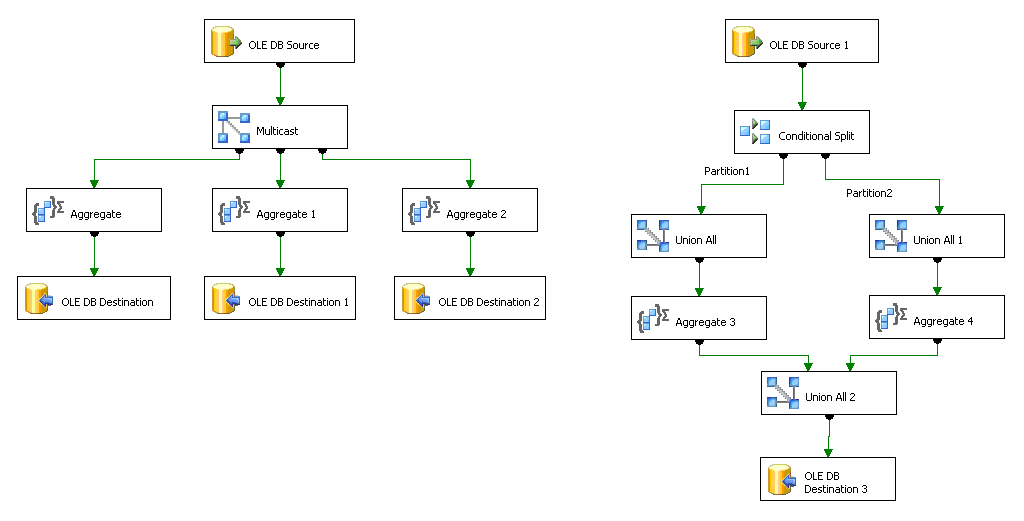


Figure 6: Optimize the slowest

To speed up performance in the new data stream, the aggregate has been broken down into two “partitions” by using a conditional split that segregates the source data based on key ranges. After each aggregate is calculated, the data sets are combined with a Union All transformation. The resulting data is loaded into the destination.

This hybrid design approach demonstrates how you can apply parallelism to the specific operations that can benefit the most from the performance gain without potentially wasting machine resources.

#### Measuring Performance

One of the most important aspects to successful performance tuning is the ability to measure and monitor performance over time. As you make performance changes to your SSIS packages, you need to have a mechanism to measure whether or not those changes made a positive difference.

##### Troubleshooting

A great technique to troubleshoot poor performance is to isolate the execution of various parts of your package to find the slowest operations. As you isolate the execution of various operations, you can establish a baseline that you can track over time. To isolate SSIS operations, consider the following steps.

1. **Establish the Overall Package Execution Speed** - When you execute your package, the overall execution speed is the total amount of time it takes from start to finish.

Overall Execution Speed = Source Speed + Transformation Speed + Destination Speed

For example, if you execute a package and it takes five minutes to execute from beginning to finish, the Overall Execution Speed is five minutes.

1. **Isolate the Source and Transformation Speed** – You can find out how fast SSIS is reading from a source and performing transformations by using the following steps.
2. Create a copy of your original package.
3. Remove the destination associated with the source. If you have more than one destination, isolate one at a time.
4. Replace the destination with a RowCount transformation.
5. Measure the execution performance from Source through Transformations to RowCount.

The result of this measurement is the Source and Transformation Speed. If this is too slow and you have more resources (such as CPU) available, consider applying some of the techniques described above (such as partitioning or introducing new execution trees) to increase the throughput.

As you troubleshoot, you will use RowCount transformations at various points in your package to isolate performance bottlenecks. A RowCount transformation simply counts the number of records being passed through the pipeline. It does not add any measurable amount of performance overhead.

For example, by performing these steps you may find out that it takes four minutes to read data from the source and perform the transformations. When compared to the Overall Execution Speed of five minutes, 80% of the execution time is spent reading and transforming data. With some simple math, you can calculate the destination speed.

1. **Calculate the Destination Speed –** Once you know both the Overall Speed and the Cumulative Source and Transformation Speed, you can calculate how long it takes to load into the destination as follows:

Destination Speed = Overall Speed – (Cumulative Source and Transformation Speed)

For example, with the measurements you have made in steps 1 and 2, you can calculate Destination Speed as follows:

Destination Speed = 5 minutes – (4 minutes) = 1 minute

1. **Isolate the Source Speed**– In step 2 you measured the cumulative source and transformation speed. Since this is a cumulative number, you do not know how much time is spent reading data and how much time is spent transforming data. To find out how fast SSIS is reading from a data source, perform the following steps.
2. Create a copy of your original package.
3. Remove all transformation(s) and destination(s) associated with a source.
4. Replace them with a single RowCount transformation.
5. Measure the execution performance from the Source to RowCount.

This result of this measurement is the Source Speed. This is theoretically the fastest your data flow can work. If this is too slow, you’ll need to focus your attention on optimizing the source adapter so that it returns rows faster. For example, through this process you may find out that it takes 15 seconds to read data from a file data source. When compared to the Overall Execution Speed of five minutes, you realize that 6 percent of the execution time is spent reading data. Now you can do some simple math to determine the actual amount of time spent transforming the data.

1. **Calculate the Transformation Speed** - With the isolated Source Speed and the Cumulative Source and Transformation Speed, you can calculate the Total Transformation Speed as follows:

Transformation Speed = (Cumulative Source and Transformation Speed) - Source Speed

For example, with the measurements that you have made in steps 1 – 3, you can calculate Transformation Speed as follows:

Transformation Speed = 4 minutes – (15 seconds) = 3.75 minutes

You have now identified the major speed parameters summarized in Table 2.

Table 2 – Performance summary

|  |  |
| --- | --- |
|  | Execution Time |
| Source Speed | 15 seconds |
| + Transformation Speed | 3.75 minutes |
| **= Cumulative Source and Transformation Speed** | **4** **minutes** |
| + Destination Speed | 1 minute |
| **= Overall Package Execution Speed** | **5** **minutes** |

1. **Isolate Individual Transformations** –If you need to identify the performance of a specific transformation for further troubleshooting, you can use the following steps.
2. Createa copy of the package that you used in step 4 to isolate the Source and Transformation Speed.
3. Remove one transformation.
4. With all other transformations and destinations intact, replace the removed transformation with a RowCount transformation.
5. Measure the execution performance.
6. Compare execution performance with the original Source and Transformation speed. The difference tells you the performance of that transformation.

For example, given that more than 60% of the time is spent on transformations, you focus on finding which transformation is taking the most time. You remove a lookup transformation and find out that the Cumulative Source And Transformation Speed drops from 4 minutes to 2.5 minutes. Comparing the two scenarios, you conclude that your lookup transformation is taking 1.50 minutes. See Table 3 for the performance summary of the isolated lookup.

Table 3 – Isolated lookup performance summary

|  |  |  |  |
| --- | --- | --- | --- |
|  | Original Execution Time With Lookup | Troubleshooting Execution Time No Lookup | Time Difference Lookup Perf |
| Source Speed | 15 seconds | 15 seconds | 0 (no change) |
| + Transformation Speed | 3.75 minutes | 2.25 minutes | 1.5 minutes |
| **= Cumulative Source and Transformation Speed** | **4** **minutes** | **2.5** **minutes** | **1.5** **minutes** |

Based on this information you tune the cache of the lookup transformation and improve the performance from 1.5 minutes to 30 seconds. By doing this, you improve execution time by 20% as demonstrated in Table 4.

Table 4 – Optimized Performance Summary

|  |  |  |
| --- | --- | --- |
|  | Original Execution Time | Optimized Design Execution Time |
| Source Speed | 15 seconds | 15 seconds |
| + Transformation Speed | 3.75 minutes | 2.75 minute |
| Tuned Lookup Transformation | 1.5 minutes | 30 seconds |
| Other Transformations | 2.25 minutes | 2.25 minutes |
| + Destination Speed | 1 minute | 1 minute |
| **= Overall Package Execution Speed** | **5** **minutes** | **4** **minutes** |

While this example demonstrates how to isolate the package operations in a simple scenario, you can apply these troubleshooting principles to more complex packages to quickly identify the components that require tuning.

##### Gaining Visibility

As you change and enhance your design, you will also want to take advantage of the ability to monitor and log metadata about package executions.

* **SSIS Logging** – SSIS allows you to log both tasks and packages to various logging providers such as XML, SQL Server, or text files. Logging allows you to view the performance of a package over time and to track performance as machine resources change and data volumes increase. Be aware that SSIS provides rich logging support which can be expensive to turn on completely. Review the logging events available and only log those events necessary.
* **SSIS Performance Counters** – SSIS provides several performance counters that you can use to gain visibility into how resources are utilized during package execution. For example, you can view the number of rows read and the number of buffers in use as the package executes. One specific performance counter that is particularly useful is **Buffers Spooled**. If Microsoft Windows® runs out of physical memory during package execution or the process executing the package runs out of virtual memory, SSIS begins to spool buffers to files. Once this occurs, performance will degrade significantly, so it is a good idea to monitor this setting and make sure that you have enough memory for your operations.
* **SQL Server Profiler** – When you extract data from or load data into SQL Server, you can use SQL Server Profiler to review the operations and query plans that are happening behind the scenes in SQL Server. As you monitor database activity, you may find tuning opportunities in the SQL Server RDBMS, such as the need to modify your indexing scheme.

#### Conclusion

Performance tuning is a continuous balancing act between design decisions and resource availability. To help you manage this balancing act, SSIS provides a flexible data architecture that you can use to build high performance data integration solutions. By understanding how SSIS performance architecture leverages memory and CPU, you can make more informed design decisions that take advantage of performance opportunities to maximize your system resources. You can then more accurately understand scaling requirements as your solutions grow in the future.

For more information:

<http://www.microsoft.com/technet/prodtechnol/sql/default.mspx>

#### About the Authors

##### Elizabeth Vitt, Intellimentum

Elizabeth Vitt has ten years of business development, project management, consulting, and training experience in business intelligence. Her industry experience includes BI implementations in retail, manufacturing, and financial services. She has specialized experience as an educator in data warehousing, ETL, and OLAP design and implementation. Ms. Vitt is an author of Microsoft Official Curricula courses for Microsoft Business Intelligence product offerings as well as the MSPress, *Business Intelligence: Making Better Decisions Faster*. In anticipation of the launch of SQL Server 2005, Ms. Vitt has successfully implemented SQL Server 2005 for early adopter customers.

##### Hitachi Consulting

Hitachi Consulting, the global consulting company of Hitachi, Ltd. (NYSE: HIT), is a recognized leader in delivering proven business and IT solutions to Global 2000 companies across many industries. Hitachi Consulting leverages decades of business process, vertical industry, and leading-edge technology experience to understand each company’s unique business needs. From business strategy development through application deployment, Hitachi consultants are committed to helping clients quickly realize measurable business value and achieve sustainable ROI.

Hitachi Consulting is a Microsoft Certified Gold Partner for Business Intelligence and an exclusive provider of curriculum and instructors for the Microsoft SQL Server 2005 Business Intelligence Ascend training program. Hitachi Consulting is also an experienced systems integrator and has successful SQL Server 2005 BI implementations at companies participating in the Microsoft Technology Adoption Program (TAP).

Hitachi Consulting offers a client-focused, collaborative approach and transfers knowledge throughout each engagement. For more information, visit www.hitachiconsulting.com!href(http://www.hitachiconsulting.com). Hitachi Consulting – Inspiring your next success®