Paper 102-31

Advanced Warehousing with SAS®9

Gary Mehler, SAS Institute Inc., Cary, NC

ABSTRACT

The world of data warehousing, data marts, and data integration continues to progress as more complex Business Intelligence environments become more widespread. This paper discusses the new and improved capabilities of SAS Data Integration Server (formerly known as SAS ETL Server) and SAS®9 in general, to meet these challenges. These capabilities are discussed in the context of meeting specific business challenges and include the areas of administration of complex systems, metadata synchronization, ways to work with parallel execution of processes, and working with parallel storage. Monitoring during the operational phase of a warehouse is also suggested, and updated suggestions are given for multi-developer ETL environments.

Armed with knowledge about these areas, warehouse developers can keep their productivity high as they address existing warehouse challenges and still have time to focus on new integration challenges as they arise.

INTRODUCTION

A recent research report by the Data Warehousing Institute surveyed organizations about their use of data integration technologies (White 2005). This report documents that traditional batch ETL (Extraction, Transformation, and Load) is important in over 75% of organizations. Batch ETL needs are seen as remaining at roughly the same level of use and importance over the next two years. However, more incremental methods of data management are seen to be growing at a fast clip during this same period. In contrast with the processing of all data in nightly batch ETL jobs, the importance of Changed Data Capture to facilitate operation on smaller sets of updated data will grow significantly in the next two years. This growth is largely due to increasing data volumes and shrinking batch windows.

Another area of significant growth in data integration involves what the report calls *Online ETL*. This type of data integration is sometimes called real-time data processing and reflects the need for near-real-time access to data. Warehousing teams need to be prepared for this growth in online ETL. The key factor here is one of low data latency; as more important decisions are being made, stagnant data becomes a risk factor and fresher data can lead to better analytic decisions.

The key to preparing for new data integration needs is to ensure that current ETL-focused warehousing environments are working optimally. This is important both in terms of the development and management environment as well as in the operational use of existing warehouse processes. As more personnel and hardware focus is directed to near-real-time integration in the future, we need to ensure that current processes and commitments aren't going to interfere with future needs. We'll look at back-end optimizations that can help workloads complete in the minimum possible time, and we'll also look at ways for data warehouse developers to get their work done efficiently. Both of these are important so that IT organizations can be prepared for new integration challenges in the future.

This paper describes how SAS Data Integration Server and other features of SAS®9 can be used to perform these activities. If you aren't familiar with SAS Data Integration Server, it is the next generation data management package from SAS that supersedes SAS Warehouse Administrator.

PARAMETERS AND PROCESS RE-USE

Basic data management or ETL processes are usually described with a process flow diagram as shown in Figure 1. The diagram describes the steps through which data flows from source through transformations to its target storage. In a typical process flow, explicit names are used for the source and target tables. In Figure 1, the Retail Source Data table is completely specific and references a specific data table on which to operate.

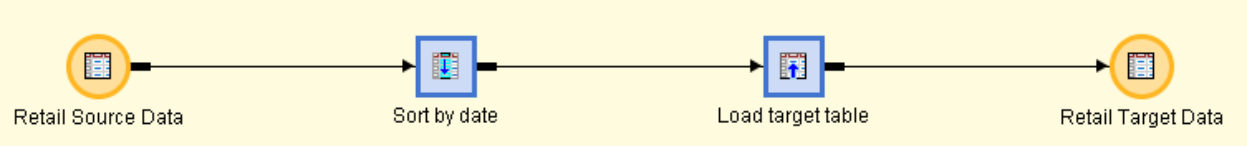

Figure 1. A Basic Process Flow – Data flows from Retail Source Data to Retail Target Data and is sorted along the way.

If you wanted to perform a similar operation to the one shown in Figure 1 but on different source and target data, you would copy an existing job and paste the new copy of it. In the new copy, you would make the changes that were needed to process a different set of data. While this strategy works fine for a small number of data combinations, in larger organizations, it can lead to a proliferation of process flows that need to be separately executed and maintained.

To aid with the re-use of process flows, the concept of parameters has been introduced. A *parameter* is really a SAS macro variable. SAS Data Integration Studio makes it easy to manage parameters through user interface support that defines and sets these variables. In the simplest case, if you wanted to process a different set of retail data, instead of specifying the explicit physical location of the data, you would use a parameter to specify all or part of the location. In a scenario in which the source data needs to be stored and processed for separate geographic areas, you might use a parameterized name like `RetailData&StateParm` instead of a basic name like `RetailData`. If you are familiar with SAS macro variables, this syntax will seem very familiar. Depending on the value of the parameter, various types of data can be processed by the same basic process flow (including variants like those shown here) depending on the manner in which the data needs to be stored and processed:

- `RetailDataCA`
- `RetailDataNC`
- `RetailDataUSA`
- `RetailDataWest`

For the preceding variants, the `StateParm` macro variable is set to: CA, NC, USA, or West, respectively.

Parameter definitions in SAS Data Integration Studio include the ability to specify a default value. If set, this value will be defined and used when processing or viewing the data to enable an initial process flow to be designed and tested. So, you might use the default value CA (the standard abbreviation for the state name of California) for the parameter `StateParm` in the current example—if you have test data from California to work with when designing the process flow. Various settings for the parameters can be controlled in user-written transformations by lookup tables or by pre-process code that is set when the process flow is used. To set a value yourself, set it in a generated transformation or in pre-process code that is executed before your process flow is run.

Many types of values can be parameterized and then used in process flows. For example,

- physical table names
- column names
- filesystem paths
- libraries
- database schemas
- user names or passwords
- values to be used in transformational logic

ITERATION

In addition to re-using process flows in the basic sense described earlier, it might be desirable to execute a large number of the same process flow at one time if a range of data needs to be processed at the same time. If you are processing `RetailData`, you might want to execute the same process for each piece of source data. If you have an orderly collection of data, it might include names like these:

- `RetailData`**`CA`**
- `RetailData`**`NC`**
- `RetailData`**`OR`**
- `RetailData`**`WA`**
- `RetailData`**`AZ`**
- `RetailData`**`NM`**

If all this data is ready to process, you can run it all in a sequence. This is a very common situation if data is received from multiple providers, or when data volumes are so large that it would be better for performance purposes to execute smaller units.

When the same process flow is executed for a set of similar inputs, this is called *iteration*. This means that the same process flow is run iteratively for each input in sequence. The scope of the iteration is called a *loop*, and SAS Data

Integration Studio includes transformations to delimit the iterated code. Looping is a common programming concept that can now be applied to process flows. The diagram in Figure 2 shows that looping is desired.

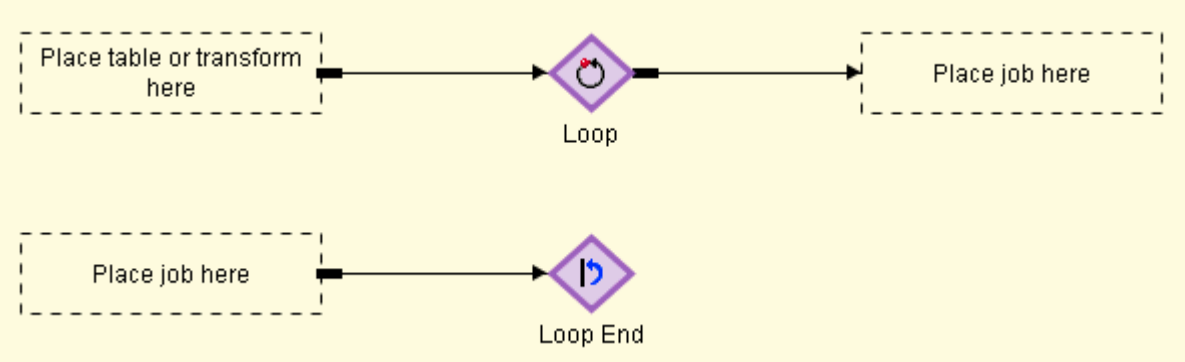

Figure 2. Loop and Loop End Transformations

The Loop and Loop End transforms enable you to drop a basic process flow onto another process flow and execute the basic process iteratively. Any process flow can act as the inner flow, and it will be executed some number of times, as controlled by the outer process flow. This capability enables process flows to control and re-use other process flows, which results in more efficient use of process flows in repetitive situations.

This is especially helpful when the basic process flow uses parameterized values so that it can be used over a collection of varying sources, targets, or transformational logic. Instead of setting the parameter values in the basic process flow, you want to control the values for each iteration of the loop. For our retail example, this might include setting a parameter such as `StateParm` to each value of California, Washington, Oregon, North Carolina, and so on. This would allow each data element to be processed, in turn.

Because you are using a data integration tool that processes data in relational tables, it makes sense to use tables to specify the set of values to be used as input to an iterative loop. Using tables gives you an easy way to set and use these parameter values because they can be read from a table or generated dynamically and used in their tabular form.

In Figure 3, you see a simple use of iteration. In this example, you are reading a sequence of values such as CA from the table called Geographic Areas. One row at a time, each line of this file is passed as input into the loop transform and filled in for the parameter `StateParm` that is referenced in the Load Target Table inner process flow. With this parameter set, the process flow in Load Target Table is executed. If you need to run a process for each of the 50 states, you can use those as rows in the Geographic Areas table.

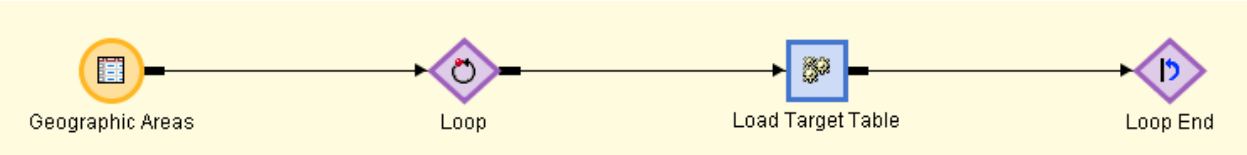

Figure 3. A Basic Iterative Process Flow

If the input table to the loop has multiple columns, how do you know which column will get passed to the loop and assigned to your parameter? Because you are using a data integration tool that processes relational data using column-level mappings, you can define a mapping between the desired column and the corresponding parameter. As seen in Figure 4, each line that is read from the Control input file is mapped in sequence to the `StateParm` parameter. If you know that you always process data that is partitioned into separate data tables for each state in the U. S., then there will be 50 lines in the Control file to be processed. Setting up a loop in this way is much easier to manage and maintain than having 50 separate process flows that need to be individually developed, maintained, and run.

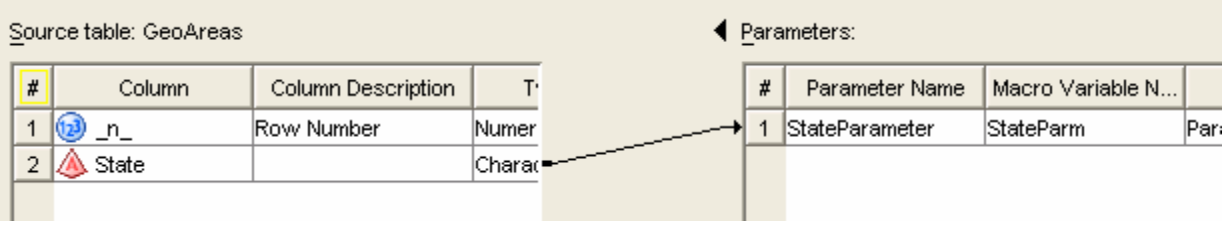

Figure 4. Mapping Occurs in the Loop Transform to Set Parameter Values for Each Incoming Line of Source Data

As indicated in Figure 4, the Loop transform also automatically generates a counter value for each loop iteration that is called _n_. This can also be mapped to a parameter. Although this example maps and uses only a single parameter value, any number of values can be used in a loop. Perhaps you want to use the counter value to keep track of an output file name or for some other counting purpose. If the Geographic Areas table has other columns, each of them can also be used. For example, if you want to use the two-letter state abbreviation in table names, but you also want to have the full text available for other annotations, both can be mapped to different parameters that are used to fill in these values in a synchronized way for each row of data input to the loop.

If you would rather determine dynamically the set of data to iterate, you can use a transform that dynamically determines a set of values such as members of a library. Then, you can use that list as input to the loop. In Figure 5, you see how iteration can be controlled by dynamically looking at the members of a library. Each table that's found in the library will be set as an individual input to the loop. This is especially useful if the set of inputs can vary or if processing is required on a dynamically changing set of inputs. The Library Contents transform determines when the process flow has executed the set of items to be passed through the loop into the Basic ETL job.

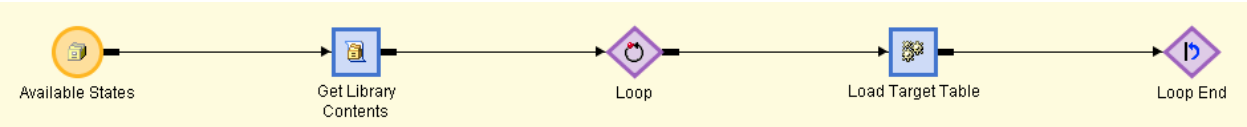

Figure 5. Dynamic Generation of the Set of Iteration Values Using the Library Contents Transform

If a large set of data needs to be processed, loops can be stacked in what is called a Nested set of process flows. Nesting enables a job that calls a basic job iteratively to itself to be called iteratively over a collection of different values. Figure 6 shows an example of this. The first process flow (Load for Time Periods) calls the second process flow (Load Geographic Areas), which in turn calls the third process flow (Load Target Table). The first job can iterate over a collection of libraries, while the second job iterates over each table that is found in the current library.

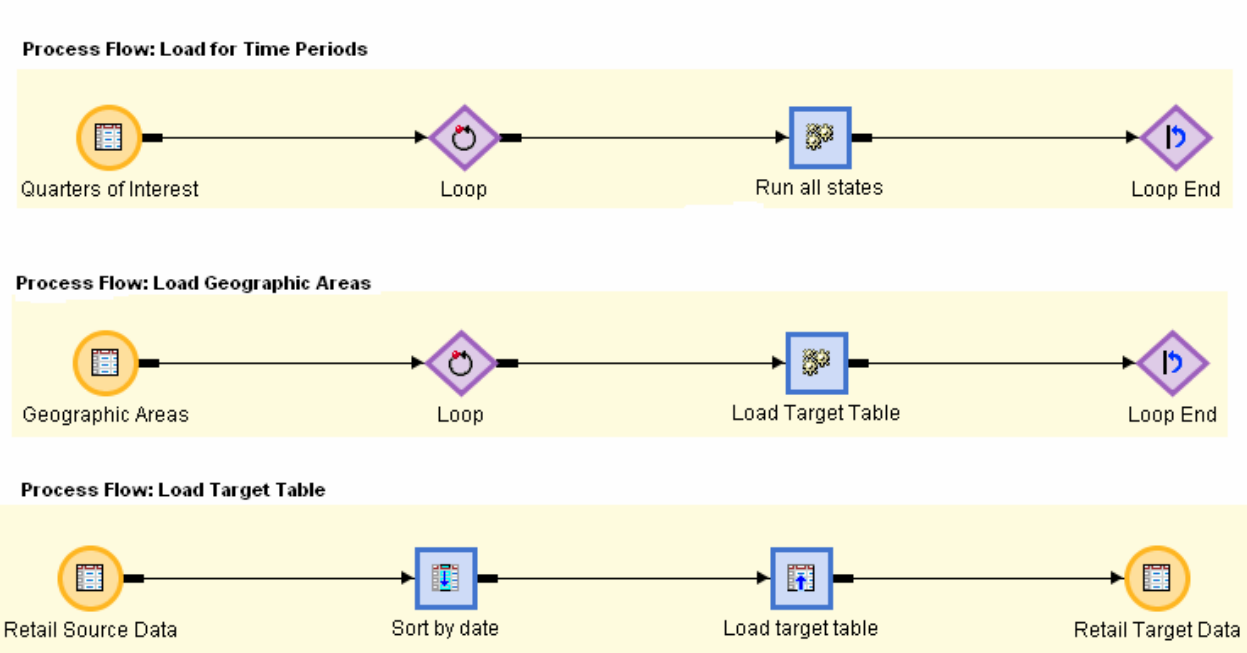

Figure 6. Processing Larger Collections of Data Using Nested Iteration

This is useful when multiple dimensions are being processed. In Figure 6, data is processed from the 50 states in the U.S., but for a range of time-based values. This becomes especially useful when data volumes are so large that data sets are partitioned in multiple ways, as you see here. To be most useful in this case though, the innermost names will be doubly parameterized. So, instead of using names like `RetailData&StateParm`, you would use names like `RetailData&StateParm&TimePeriod` and end up with data table names like `RetailDataCA2Q2006`. If you have a lot of data to process, such as quarterly data over 10 years (or 40 quarter years) for the 50 states in the U.S., you can execute 2000 successive process flows with the three actual process flows that you see in Figure 6. You would need two control tables for the time periods and state names, or you could use the Library Contents transform if you know that the data for each state is stored in a separate data library.

PARALLEL EXECUTION

Because we've been discussing potentially large numbers of sequential iterations of the basic process flow, naturally the question of throughput and performance comes to mind. Even if you can process each basic process flow in just a few minutes, the overall time that's required to execute 2000 of them can be substantial. For this reason, you might want to execute these iterations in parallel. Depending on the hardware that you have available, you can choose to execute all iterations in parallel (although probably not 2000 of them!) or some smaller number, concurrently. The smaller number might need to be fixed in cases in which you know you always can run, at most, two iterations at a time. If you have larger hardware available, this can be a larger number, or the number can be determined when the process flow is executed so it can adapt to the hardware on which it is run. This is controlled by settings on the Loop transform.

By default, iterations in SAS Data Integration Studio are run in sequence. That is, without any parallelization, so that only one iteration runs at a time. However, as shown in Figure 7, you can change that to allow iterations of the process flow to run in parallel. If you want parallel execution, you should have a multi-processor (SMP) computer or a set of computers on which you can run the process flows. After all, you are trading a longer sequential time on a single processor to a shorter elapsed time on more processors.

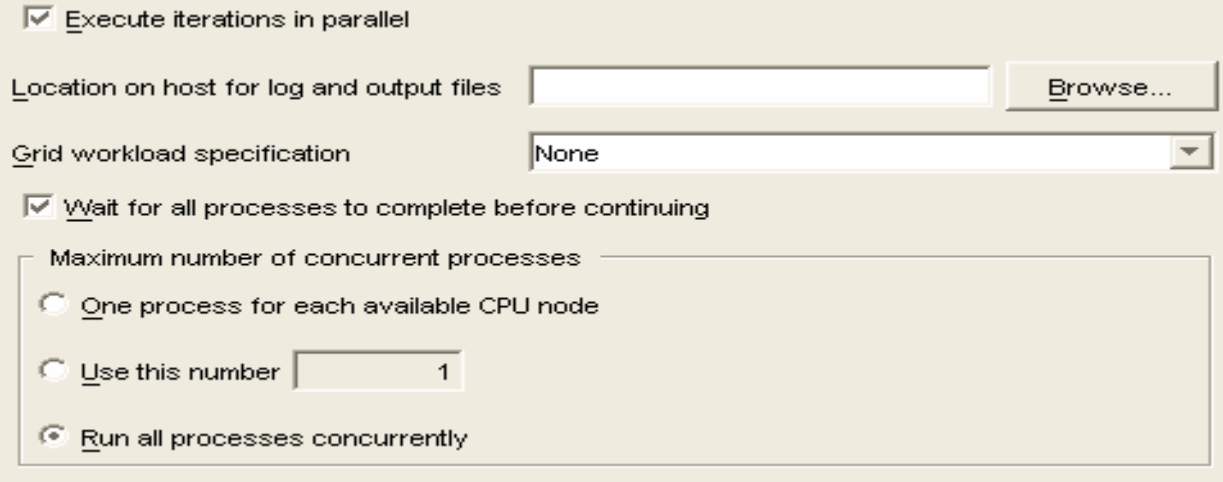

Figure 7. Parallelization Options for Iteration

Parallel execution requires that SAS/CONNECT be configured or SAS Grid Manager licensed. The difference between the two is that MP CONNECT in SAS/CONNECT lets the process flows exploit the multiple processors that you might have in a multi-processing or multi-core computer, while SAS Grid Manager harnesses the power of a collection of separate systems to run the process flows in parallel. Clearly, there are a wide range of parallelization choices available if you have adequate hardware to run many things at the same time. Grid computing can enable you to harness the power of a collection of relatively smaller, lower-priced computers (grid nodes), and it can provide an economical alternative to a large SMP computer.

RUNNING ON A GRID

SAS Grid Manager provides capabilities that are used in SAS Data Integration Studio to enable parallel execution of process flows using iteration. This allows the segments (or loops) of a large workload to be distributed across a grid of computing nodes, as needed, until the workload is complete. SAS Grid Manager dynamically determines node availability and monitors grid nodes to determine which node is the best candidate to receive the next workload segment. This determination can be based on many factors, but it often considers the current load under which all grid nodes are running at any given time. The node that has the lowest CPU load becomes the best candidate on which to run the next workload segment.

To illustrate how this works, let's use a more complex workload than that described earlier. Let's study a standard workload based on 1990 U.S. census data. This workload uses a collection of text files that are processed to build a star schema in preparation for analytic processing. Because the data is for each of the 50 states and the District of Columbia, 51 text input files will be processed and 51 star schemas will be generated. Separate star schemas work well for this example because the data is geographically segmented, and each set of data is disjoint from the others.

Figure 8 shows a diagram of how the star schema is built. First, a text file that contains more than 100 columns of personal and household values is processed, based on a standard file format definition. For this example, a total of over 60 gigabytes of text input data needs to be processed in order to include all the files.

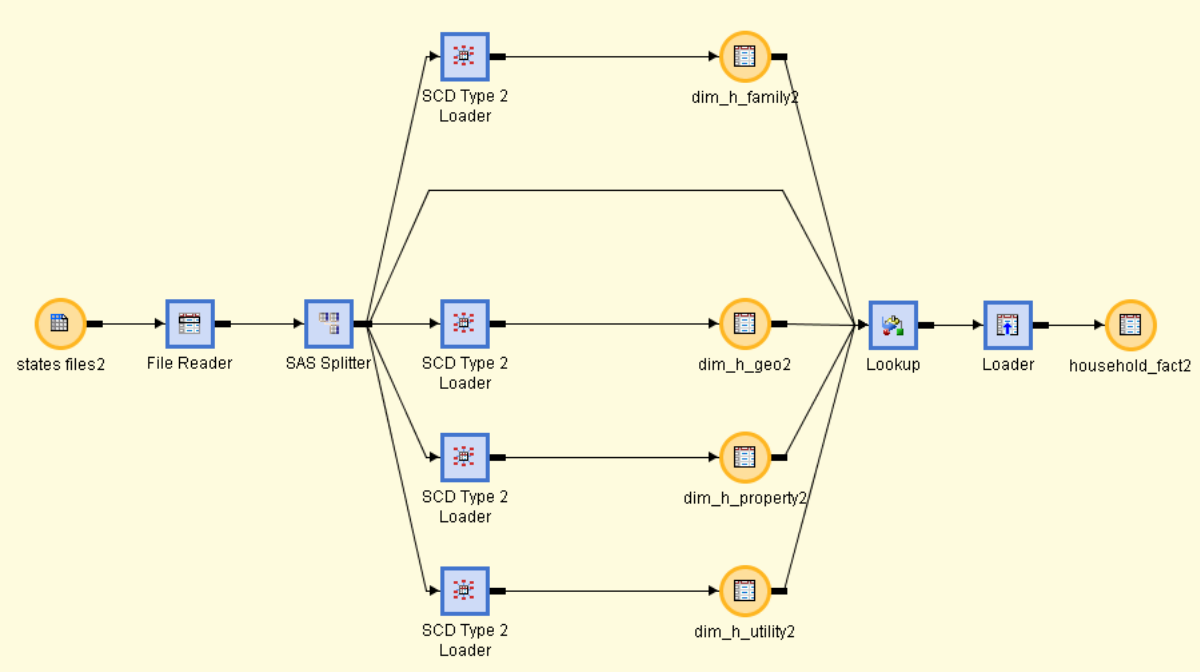

Figure 8. A Basic Process Flow That Builds a Star Schema of Four Dimensions and a Fact Table

This data is then processed into four dimensions: geography, family data, property data, and utilities data. Each dimension is loaded into a Slowly Changing Dimension (SCD) Type II storage target. This is the standard way to store data for analysis in order to retain history information, but it is computationally complex due to the amount of processing that is needed to create the structure of the tables required. Afterward, a single fact table is constructed. This table uses the Lookup transform to find text values in the dimension tables that are stored as numeric keys in the final fact table. The set of resulting fact tables is more than 15 gigabytes in size.

Each file can be processed individually, and a simple way to process the 51 files is to execute each job in sequence, one after the other. In this case, each segment of processing can consume a single computer. However, if the segments are run concurrently, the workload can be distributed over multiple computers (like grid nodes) to run in parallel.

Figure 9 shows a process flow that runs a basic workload (Build Star Schema, as seen in Figure 8) in parallel. As shown earlier, the Loop Begin transform can be configured to run each element sequentially or in parallel, with various levels of elements to run at a time. The following sections describe a few configurations and evaluate how this works.

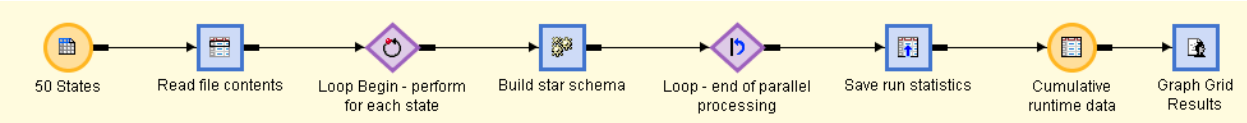

Figure 9. An ETL Configuration to Run a Loop That Executes the Desired Processing for Each Input Data File

The diagram in Figure 9 includes a transform called “Save run statistics”, which shows that you can access run-time data from loop execution and store it for later analysis. In this case, you will generate a graph that indicates how the workload was distributed across the grid nodes.

When run in parallel, SAS Grid Manager determines which resource can best run the next portion of the workload, and that information is used by SAS Data Integration Server to submit and monitor that portion on the best computing resource. Parallel execution can lead to greatly reduced total run times because of the amount of processing that can be run at the same time on different computing resources.

GRID HARDWARE

To illustrate the operation of SAS Grid Manager and SAS Data Integration Server, let's look at how this can be run in a blade server environment. Processing blades are a cost-effective way to provide large numbers of grid nodes for parallel processing. Although a blade is a complete computer, it can share system components with other blades in a chassis, and it's economical in terms of electrical power use and space required and generates less heat and noise.

In Figure 10, the hardware configuration that's used is a blade server that contains ten individual blades (thin computer modules). Think of each blade as having the equivalent in power to a workstation-class personal computer. A central filesystem is used for all data access. This allows any grid node (blade) to operate on any data element that it needs to operate on. All processing and control activities are executed on the blades in this server.

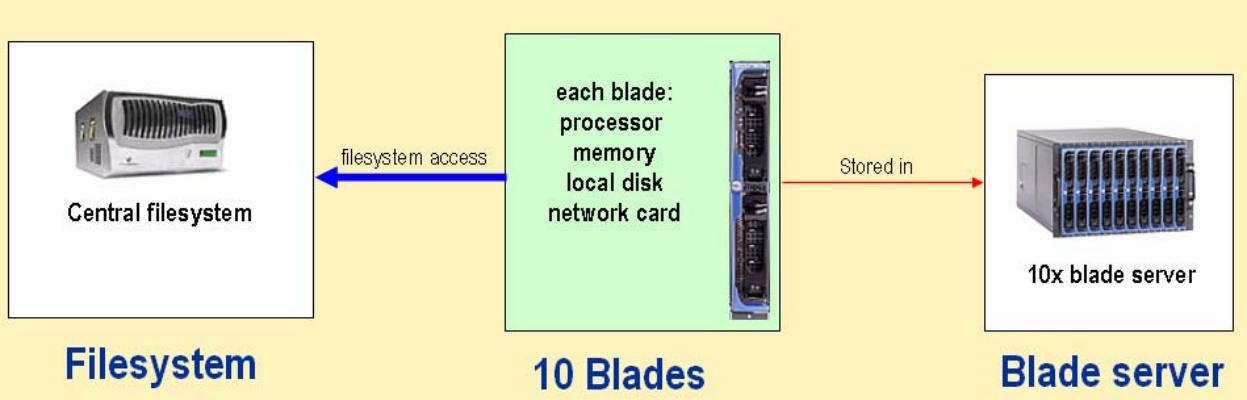

Figure 10. An Example Grid Configuration

Each blade is used as a controller machine. It hosts the load distribution software, a metadata server that is used for central control of the processing, and the main execution application server. The load distribution software is Platform LSF Suite, which is a component of SAS Grid Manager and provides grid-management infrastructure. The main execution application server runs the main SAS process that's shown in Figure 9. This controlling session uses the SAS Grid Manager to launch and manage execution of the workload elements that are shown in Figure 8. The controlling process also keeps track of run-time statistics and saves that information for later analysis.

MEASURING BASELINE PERFORMANCE

Whenever you are using parallel computing, the first step is to establish a baseline for basic, un-improved performance for the desired workload. In this example, the 51 census workloads are run in sequence; each job finishing before the next one begins. This is a typical execution pattern if a single computer is used to complete a set of processing tasks. For baselining purposes here, the jobs are run across six nodes of the blade server, but control parameters in SAS Data Integration Server allow only one workload to run at a time.

Figure 11 shows the workload run across six nodes (A through F) and the varying amounts of CPU load that occurred over a 588-minute interval. The differing colors show that SAS Grid Manager distributed the workload across various nodes in the grid, but that only one node was active at any given time. The rotation of colors suggests that SAS Grid Manager did a good job of determining a lightly loaded node on which to run each successive workload element, because the use of all six grid nodes is relatively even. This shows that nearly 10 hours of processing is required to complete the 51 workloads that comprise the census data processing. The average CPU load of about 0.25 further verifies that no more than one process was running at a time across the six nodes. Later, this will be compared with an average CPU load that is greater than 1.0 (see Figure 13).

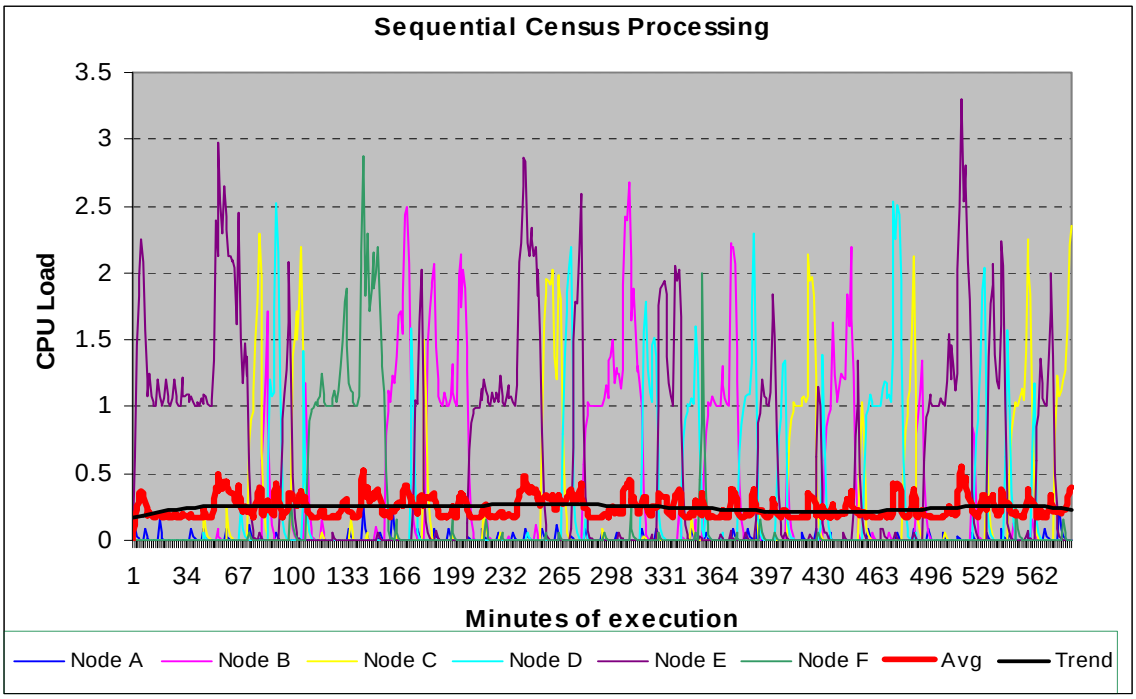

Figure 11. Elapsed Run Time Required for Sequential Execution of the Census Workload

This sequential baseline run enables you to see how quickly these workloads can be completed on the given hardware. Is this enough baseline data to have before beginning your quest toward highly scalable performance? Not exactly. Although elapsed time is a good indicator, you'll also want to understand how much time each workload element requires individually. This is important because you expect elapsed time to decrease in parallel execution, but you don't necessarily know whether you're wasting CPU cycles due to bottlenecks unless you compare actual run times for individual workload elements.

The detection and analysis of bottlenecks is of critical importance when searching for an optimal parallel configuration. As more processes run in parallel, the demand for performance from hardware elements such as disk drives and networks can eliminate some time savings. Therefore, it's important to understand the point past which running additional parallel processes won't deliver a useful decrease in elapsed time.

As shown in Figure 12, the run times per workload element vary greatly. This is because the amount of data being processed for each state varies greatly. Some workloads (such as California, New York, and Texas) take much longer to complete than others (such as Wyoming or Alaska). This variability adds another level of complexity to the use of parallel computing, and highlights the importance of understanding the performance baseline for the workload.

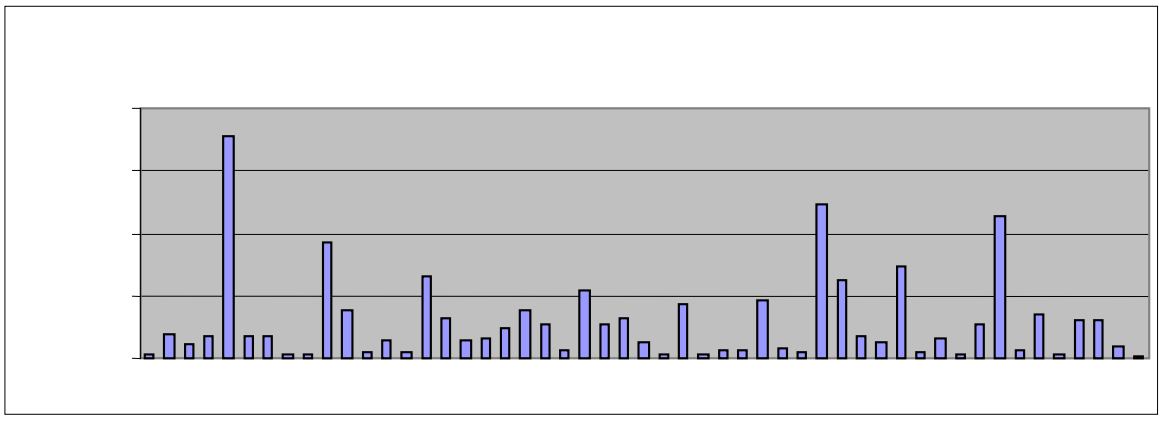

Figure 12. Run Times for Each Workload Element During Sequential Execution

PARALLEL EXECUTION PERFORMANCE

As described earlier, running in parallel is accomplished by setting a different Loop transform value in SAS Data Integration Studio to allow more than one workload element to execute at the same time. For the grid hardware described earlier, the execution number six was chosen. This allowed one job to run at a time on each grid node (or blade). It also meant that the heavy processing requirement for each workload element had full access to the hardware of the blade, including its local disk storage and memory. Because of this, the only resource in contention across the workload elements was access to the central storage location for source and target access. By choosing to use six of the ten available grid nodes, this example simulates a case in which other grid nodes might be in use for other processing requirements and would therefore be unavailable for the census process flows.

As shown in Figure 13, the 51 workload elements were completed in 108 minutes. The figure also shows that all blade nodes were used during the bulk of the execution run. This suggests that the distribution of workload elements was done efficiently and with good effect. These results further suggest that major bottlenecks were avoided in order to achieve these results. Looking at the average CPU load, you can see that the numbers range from 1.0 to 2.0. The lower number (1.0) indicates that the grid nodes remained productive; the higher number (2.0) shows that some of the workload processing ran in parallel due to multi-threading in the SAS Server that executed the process flows.

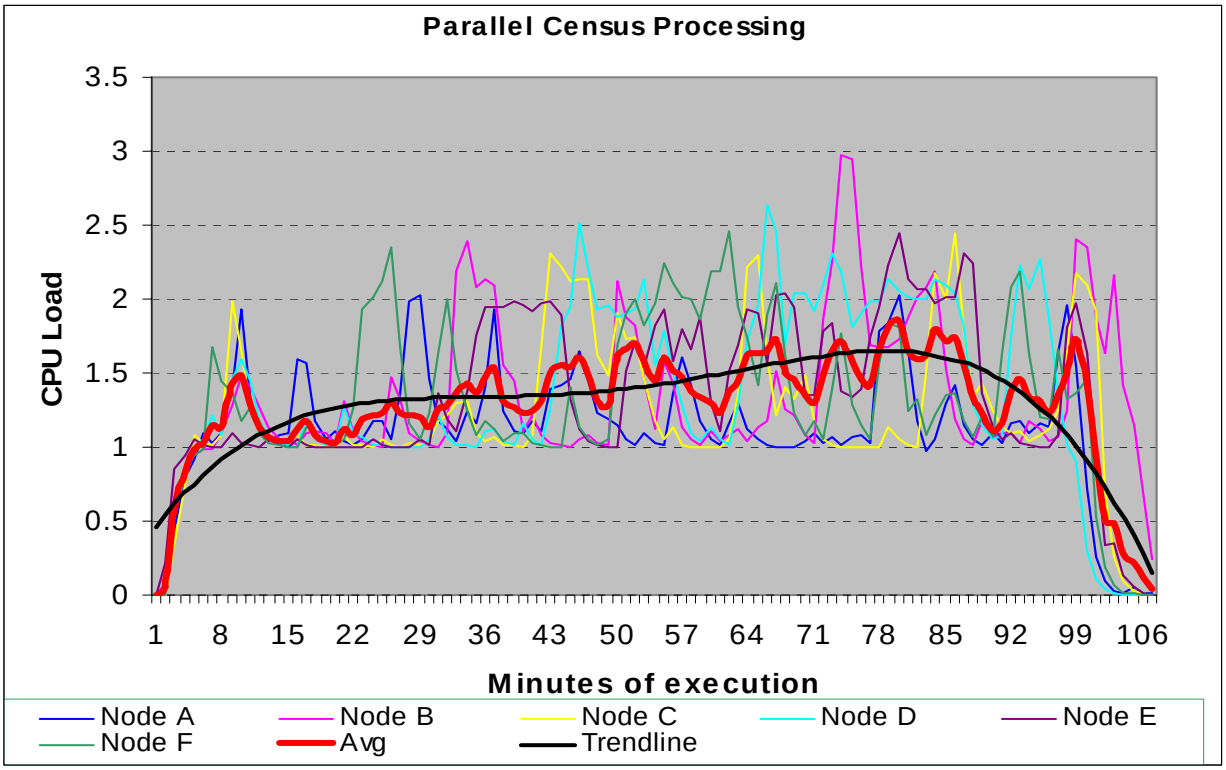

Figure 13. Execution in Parallel across Six Blade Server Nodes

Analysis shows that the 588-minute workload now took only 108 minutes to complete. The same amount of work was finished in 18.4% of the time. To compare this with linear or "perfect" scalability, in which the six processing nodes allow the work to complete in 1/6th of the time, "perfect" scalability suggests that the workload can be completed in 16.7% of the original time. However, because of overhead such as network and central file-system contention, "perfect" scalability is rarely achieved. However, for the data shown in Figure 13, scalability is only 1.7% less than "perfect".

It's useful at this point to look at the efficiency of execution of each workload element. In this case, you are interested in whether the workload elements required a longer period of time to run in parallel execution mode than when they were run sequentially. If these time periods are substantially different, you are losing efficiency on each workload element due to some bottlenecked resource.

As shown in Figure 14, the time that's required for each workload element is displayed as is shown earlier in Figure 12. Looking at longer running elements such as California (ca), you see that this workload took slightly longer to complete, and a detailed analysis shows that there was less than 10% overhead that resulted from running six jobs at-a-time in parallel. This suggests that excellent levels of scalability would have been achieved if the workload elements had been more uniform.

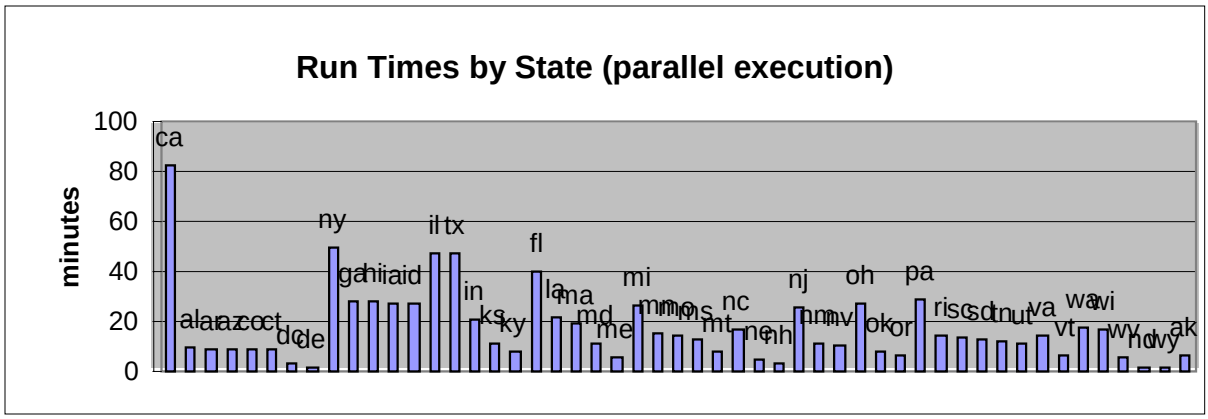

Figure 14. Execution Profile for Each Workload Element

You might also note that the order of states in Figure 14 is different from that in Figure 12. Initially, the execution order was alphabetical, but running elements of varying size proved to be a problem here. Longer -running elements such as those for Texas and New York were started later in the process and could be left running by themselves, after all the shorter elements that were launched earlier had completed. In this case, the diagram of run times (like those shown in Figure 13) indicated that most grid nodes were idle in a large final portion of the overall run. As a result, the input list to process was re-ordered based on run times. It would have been straightforward to use the run-time statistics that were automatically generated from an initial run to re-order the inputs for a subsequent run, but this was also problematic. In this case, if the data for all the largest files was read at the same time, more network and central file-server contention resulted, and run times for those files increased. The approach used here was to randomly scatter the longest 25% of run times across the initial 50% of the files in the input list to distribute this load more evenly over time.

Although this concept is not presented here, it is possible to over-drive the grid nodes. If 20 processes at a time are selected to run on the grid nodes, it results in overworking the network and central fileserver. Although the overall elapsed time looks good in this case, you would see that the elapsed time per file gets much larger as a result of bottlenecks on the fileserver and network.

GRID PERFORMANCE ANALYSIS

Executing workloads such as the processing of census data on a grid can have significant performance benefits. As shown earlier, high degrees of parallelization result in this configuration. Other workloads that can be distributed across a grid should see similar speed-ups in performance. Of course, your "mileage" might be different. A few caveats are given here.

Although this workload is I/O intensive, it has only a beginning and a final phase that require large amounts of data to be transmitted across the network to the central storage location. Most intermediate storage requirements can be met by local, temporary disk use on each blade server. This turns out to be a good usage profile for general data processing, and takes good advantage of the separate computing resources on each blade server. In a blade environment, local disk and memory use is isolated from other workload elements, so the impact of one element on another can be minimized. This might not be as easy to accomplish on a single, large, multi-CPU computer that's attempting to run the same parallel workload.

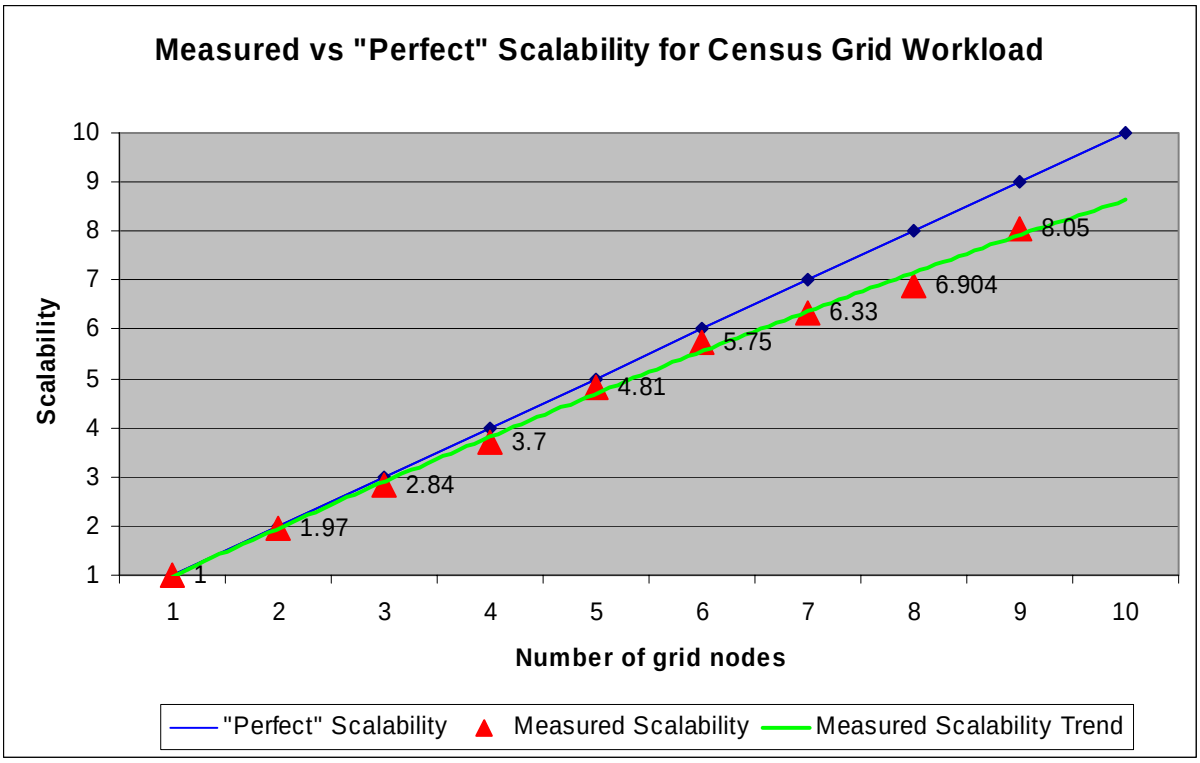

Figure 15. Measured Scalability for ThisTest Versus "Perfect" Scalability

SAS Grid Manager did a very good job of distributing the workload elements across the nodes of the grid. The ability in SAS Data Integration Studio to control the level of parallelization enables performance-aware users to take full advantage of their computing hardware for large-scale data processing. Figure 15 shows that results very close to the scalability ideal were achieved for this workload.

SHARING THE GRID

When running large workloads on a grid, it is not only important to maximize the performance of your own workload, but to be sensitive to other users and workloads that might be sharing the same grid. Therefore, setting the parallelization limit in SAS Data Integration Studio to a number that is smaller than the total set of available hardware can be desirable. For example, in addition to data integration flows, there might be analytic users of the grid performing large sets of scoring algorithms.

Figure 16 shows that the 51 census workload elements are executed across six active grid nodes. In this case, the number of concurrent processes was also set to six. The result is that SAS Grid Manager keeps the six nodes equally busy during the run of the workload. Although the distribution of workload elements on each grid node is somewhat random, this actual result shows an instance of near-perfect distribution of the workloads. You know this because very little time is left at the end of the workload during which any node is idle, waiting for others to complete their workload elements.

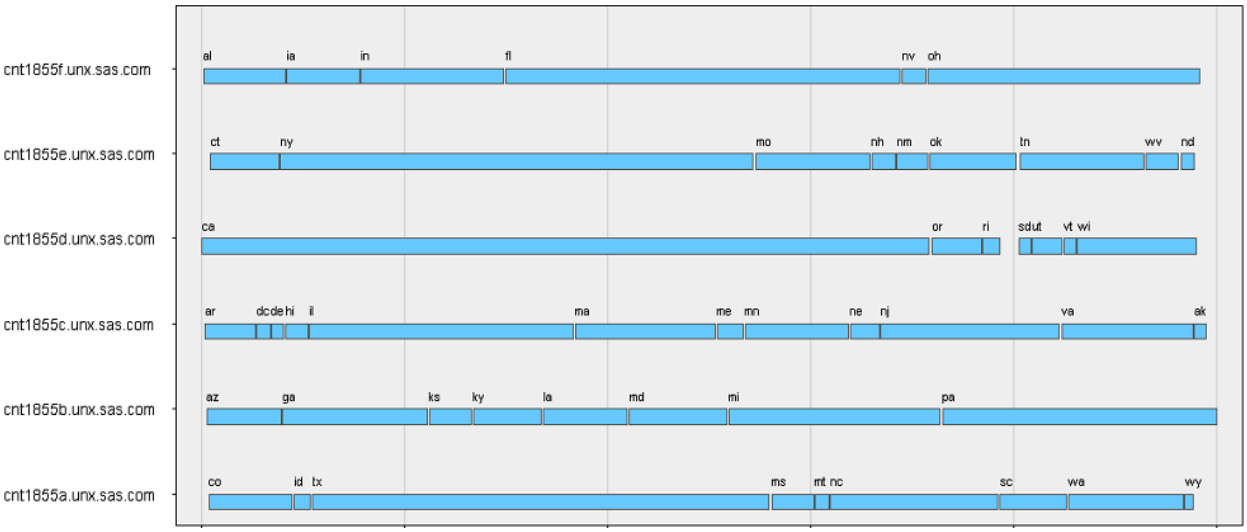

Figure 16. Census Workload Executed on Six Grid Nodes. SAS Data Integration Studio was configured to allow six jobs to run at a time. Grid nodes are along the Y axis, and elapsed time across the X axis.

The gaps that are shown in each grid node's execution indicate times during which SAS Grid Manager was launching new workload elements after preceding elements completed. There is an interval allowed for launching a new session to enable machine loads to reach a steady state so that load balancing can be executed effectively.

As seen more clearly in Figure 17, the grid deployment software can make good use of computing resources when more resources are available than are needed. In this case, there is essentially one "free" node at any given time because fewer workload elements are allowed to run at a time than there are available grid nodes. This suggests that data integration workloads can co-exist well with other workloads on a larger set of grid nodes. For example, gaps in the data integration workload requirements can be filled by analytic workloads. Conversely, other workloads can co-exist with workloads like the ETL census workload when computing hardware is available. The lesson here is to realize that a collection of computers that comprise a grid can be an expensive budget item that can be shared with others as needed.

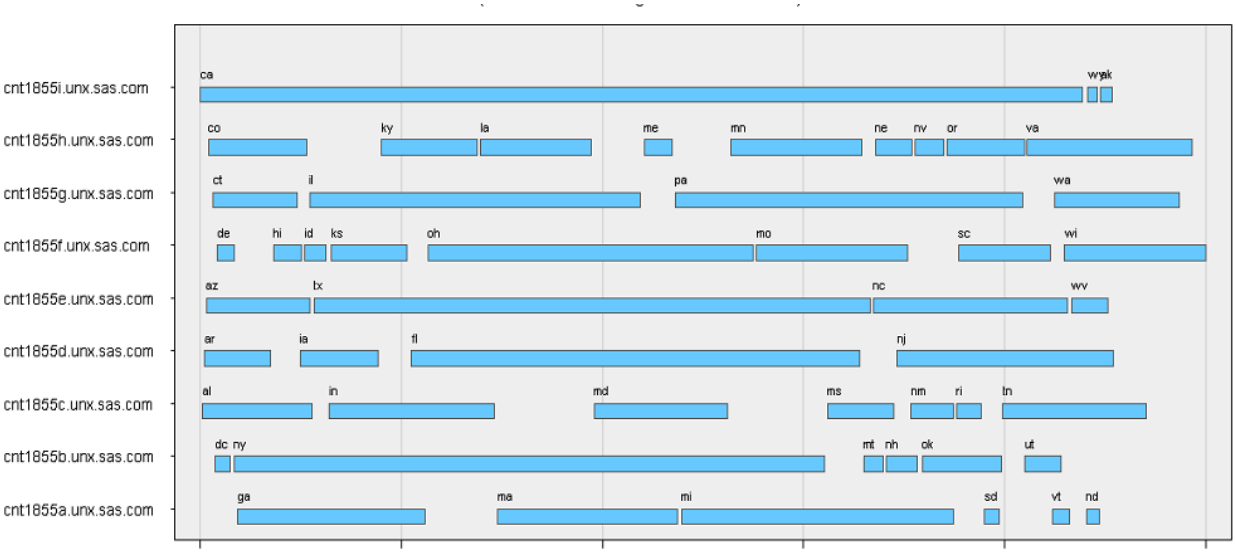

Figure 17. Census Workload Running across Nine Grid Nodes. SAS Data Integration Studio allowed eight workload elements to run at a time.

CHANGE ANALYSIS – HANDLING ORGANIZATIONAL CHANGE

As business needs evolve, one aspect in which this change is reflected is in data model evolution. In large organizations, data models are managed centrally and modeling tools such as ERwin from Computer Associates are used.

As an example, let's look at how a data model might evolve in an organization. Looking at the upper-left table in the diagram in Figure 18 (the Car_Dim table), you can see that the data modeler has defined a set of columns in that table to be able to store enough information to meet the business needs for that area. The Car_Dim table might look just fine initially, but it is not unusual for the needs to change over time and require changes in the table structure.

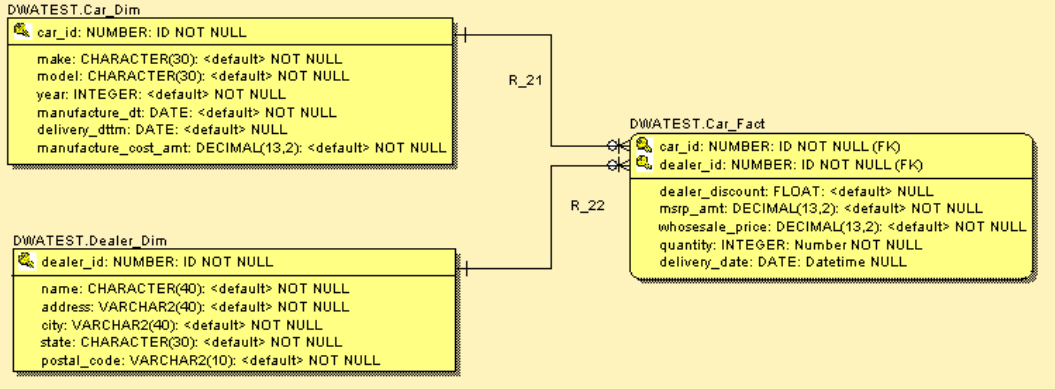

Figure 18. A Small Data Model. This diagram shows how three tables are related.

In the updated version of the Car_Dim table that is shown in Figure 19, several of the columns have changed. This can occur as deletions if data is found that is not being used, as additions if new data needs to be managed, or as modifications if attributes of some data element need to change. In the updated version (Figure 19), notice that some columns were removed and some columns were added, and that an attribute in one column (delivery_dttm) has changed to reflect a constraint change. In addition, Car_Dim now reflects a changed natural order of columns.

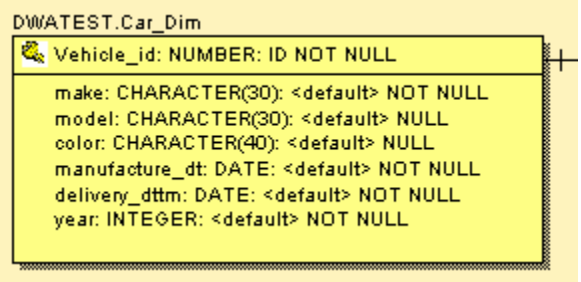

Figure 19. An Updated Version of the Car_Dim Table Structure. Various aspects of the table have been modified.

All these changes need to be communicated to you and the rest of the data management group, perhaps in the form of a written memo or maybe just in terms of a different data model diagram or description file. This can make it difficult for the data management team to update any data flows that are already developed in order to accommodate these changes, and change they must. It's likely that the model update will trigger the operational system administrators to change their table structures. If the process flows that consume this data don't also change, run-time errors will occur and bad data can be propagated through the warehousing system.

What you and the rest of the data management team need is a systematic way to see and integrate these changes. It would be helpful if a view inside their data management tool could help you see the set of differences and their impact, and then help you apply the changes in "your world". This is a specialized task, and a workbench-like user interface would be helpful to streamline this task. The next section takes you through such a workbench and describes how this operation can be performed.

A new feature in SAS Data Integration Studio is called Change Analysis. Change Analysis is performed when a data model is updated by using the Metadata Importer. In the past, it was possible only to import a new data model. If an updated data model was encountered, the Metadata Importer registered a new set of metadata for the relational tables that were described in the metadata import information. An update/comparison mode has been added that enables you to specify that you want to process only the changed elements of the metadata import information, as you'll see below.

In update mode, a comparison is performed between the newly registered metadata and existing metadata in a SAS repository. This is a new metadata import mode. In this mode, there are two ways to perform comparisons. By default, only tables that are included in the new metadata import are compared to their previous versions. This is helpful if only a subset of a larger data model is being updated. The second way is a complete comparison of tables in a library. In this instance, tables that previously existed in the SAS repository but aren't included in the import are candidates for deletion from the SAS repository. Because this is a potentially dangerous choice, this second approach should be used with care, but it's necessary in some cases to flag tables that should be deleted.

When the comparison is complete, you are shown a set of comparison results. These results are stored in a SAS library so that they can be used multiple times, and so that metadata updates can be performed either as a team effort or in multiple steps. In either case, additional comparisons can be performed so that the process of applying the updates can be managed, tracked, and verified as complete when all work is done.

To begin, it is helpful to examine the complete set of differences (shown in Figure 20) between the metadata that previously existed in the SAS repository and the new metadata that is being imported. In Figure 20, you see items in the left panel that are new (indicated by a star), changed (indicated by an arrow), or deleted (indicated by an X). The Metadata Difference view enables you to see the old and new metadata side-by-side, and it can be navigated as needed. If you want to see where the table or column is used, you have access to impact analysis views that you can use to determine any additional impact a change will have.

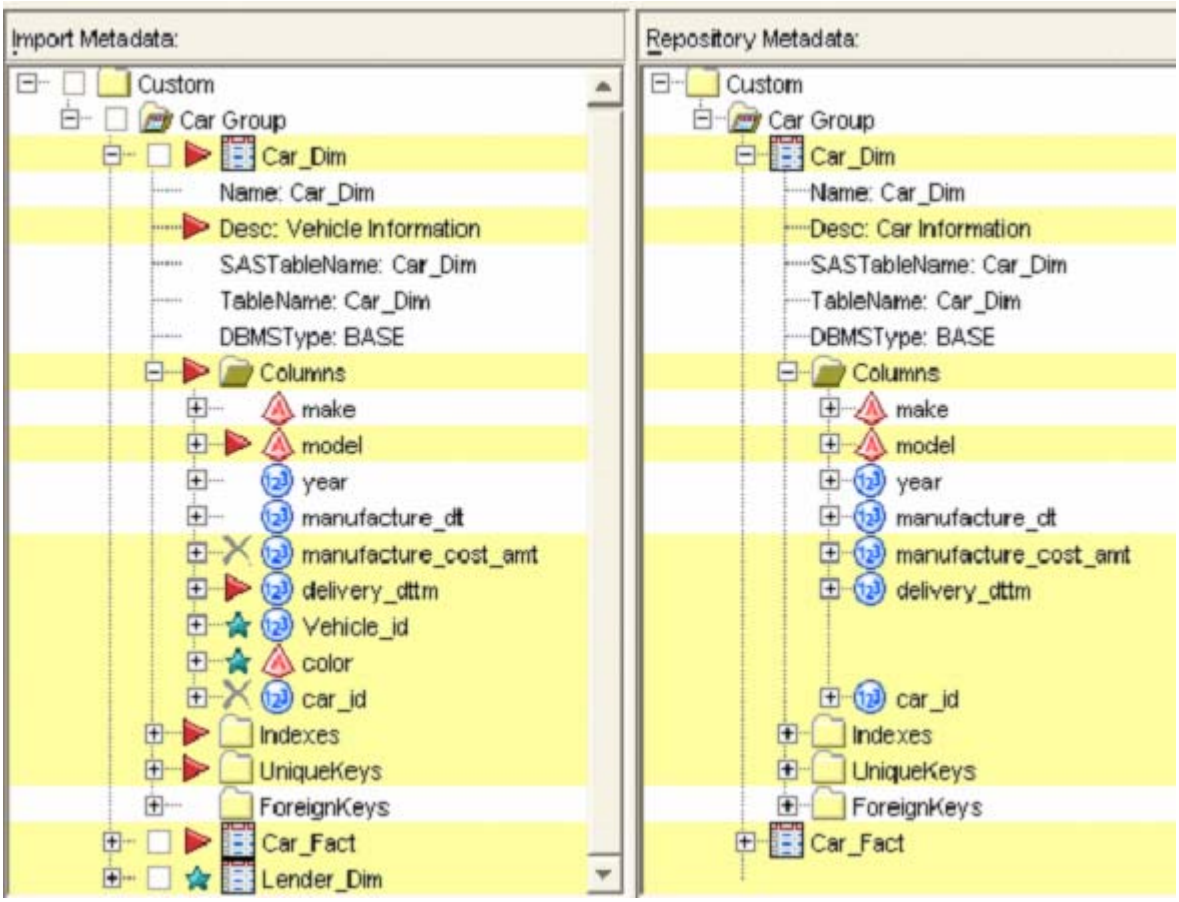

Figure 20. Metadata Difference View

Because it is recommended that these types of complex changes be made in a Change Management environment, selecting Apply to apply the changes results in the involved metadata being checked out to your project area and changes to the metadata being made for you. Later, when the changes have been applied and verified, you can check your changes in. This results in a single update operation that is change-managed and that is easy for you to perform.

This work can be done in multiple steps as well. If only a subset of changes can be made at a time, you can perform a re-compare to see that a set of differences has been resolved and that a smaller set of remaining work can be done next. In this way, large metadata differences can be processed in a manageable fashion.

Change Analysis is a major time-saver for groups in which data models do evolve, and in which the application of this evolution is tedious and error-prone. When you use a common user interface for this type of operation, the viewing, comparison, verification, and application of metadata updates can be accomplished with less overhead and risk than if you have to do it manually each time.

OTHER TOPICS

A few other advanced topics should be mentioned here. These present a set of enhancements to SAS Data Integration Server.

UPDATING METADATA BASED ON PHYSICAL TABLES CHANGES

In addition to the planned data model changes that were described in the previous section, other methods of managing data changes exist. Many organizations manage data change through their operational table structure. In this mode, changes are communicated to database administrators (DBAs) and are made in the operational system. Although it can be communicated to consumers of the data tables, it might be up to the ETL developer to determine empirically what changes have been made and to apply them to their metadata version of this information, manually. SAS Data Integration Studio has an Update Table Metadata feature that can be used for a collection of tables.

Note: It is recommended that table metadata be synchronized on a regular basis to ensure that the metadata description of your physical tables remains consistent with the physical tables themselves.

PARALLEL STORAGE

For improved throughput when working with large data volumes, parallel storage such as SAS Scalable Performance Data Server is recommended. SAS Data Integration Server has specific support for a wide range of configuration settings that can be used to optimize performance with SAS storage. A storage expert can use these optional settings and save information in the SAS metadata repository for the benefit of all data integration users.

ADVANCED LOOKUPS

A complex and time-consuming part of many data processes involves looking up values in reference tables. A recently updated feature in SAS Data Integration Server uses in-memory hash tables to speed these operations and supports many advanced types of lookups such as multi-column business keys that can be easily used. In addition, sophisticated error handling and thresholds can be set to ensure that accurate processing is completed. A good source for more detail about this topic is the paper by Nancy Rausch (see Rausch 2006 in the “References” section).

CHANGE MANAGEMENT UPDATES

When working with a team of developers, Change Management is a safe way to ensure that one person’s changes don’t overwrite changes made by someone else due to concurrent updating. Change Management is a check-out, check-in methodology that places locks on active metadata areas to prevent these types of incidents. However, previously, the level of locking was aggressively safe, which prevented large teams from using it. The previous level of locking locked all tables that were referenced by any process flow that was checked out, thereby preventing anyone else from checking out any other process flow that referenced any of the same tables. While very safe, it was a very restrictive feature. In the current release, process flows are checked out by themselves, and related tables can be checked out later. In this way, only metadata that you to change is locked, and additional developers can work on related process areas, concurrently, without preventing others from locking all the related metadata objects.

CONCLUSION

This paper discusses a variety of ways in which you can perform data warehousing tasks resulting in higher performance and higher efficiency. A number of powerful concepts are introduced and explained. Parameters facilitate re-use of process flows and allow generic items such as table definitions to be used. The ability to include process flows in other process flows enables you to manage more complex groups of operations. Iteration lets the inclusion of process flows be controlled to operate on a series of values. Parallel computing enables you to take

better advantage of the advanced computer resources that you might have or acquire. Grid computing pulls all this together into a managed environment for running high-performance, high-volume data warehousing processes. Because parallel execution of process flows is a complex but potentially rewarding process, this paper presents a detailed discussion of ways to help you understand how well you are actually parallelizing your workloads.

Change Analysis streamlines the process of updating data models in your metadata repository, while other enhancements make it possible to complete your data integration projects more efficiently. Using the new features and approaches will help you prepare for future data integration opportunities.

REFERENCES

Rausch, Nancy. 2006. "Stars and Models: How to Build and Maintain Star Schemas Using SAS Data Integration Server." *Proceedings of the Thirty-First Annual SAS Users Group International Conference,* San Francisco, CA.

White, Colin. 2005. "Data Integration: Using ETL, EAI, and EII Tools to Create an Integrated Enterprise." Data Warehousing Institute (TDWI) Research Report Series. Available www.tdwi.org/education/Webinars/.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Gary Mehler
SAS Institute Inc.
Cary, NC 27513
Work Phone: (919) 677-8000
Fax: (919) 677-4444
E-mail: gjm@sas.com
Web: www.sas.com