TATA CONSULTANCY SERVICES
Experience certainty.

Performance Tuning in DataStage

02/07/2015

Target Corp LIM Upgrade FP Off

Target Relationship

Divya Sangili

DataStage – Performance Tuning

sangili.divya@tcs.com

knowmax
Our Intelligence Hub

Confidentiality Statement

Code of Conduct-Statement

We, in our dealings, are self-regulated by a Code of Conduct as enshrined in the Tata Code of Conduct. We request your support in helping us adhere to the Code in letter and spirit.
We request that any violation or potential violation of the Code by any person be promptly brought to the notice of the Local Ethics Counsellor or the Principal Ethics Counsellor or the CEO of TCS. All communication received in this regard will be treated and kept as confidential.

Overview

This document provides the overview of performance tuning methods in DataStage .

Objective

- About Performance Tuning
- Methods of Monitoring
- Points to consider in Jobs for Performance tuning
- Job design best practices
- Conclusions

Table of Content

Overview 3
Objective 3
1. Introduction 5
2. Performance Monitoring Methods 5
2.1 Usage of Job Monitor 5
2.2 Usage of Score Dump 5
2.3 Usage of Resource Estimation 6
2.4 Usage of Performance Analysis 8
3. Most common points for datastage jobs in performance tuning 9
4. General Job Design Best Practices 12
4.1 Columns and type conversions 12
4.2 Transformer stages 12
4.3 Sorting 13
4.4 Sequential files 13
4.5 Runtime Column Propagation 15
4.6 Join, Lookup or Merge 15
4.7 Databases 15
4.8 Partitioning Considerations 16
5. Conclusion 17
6. Reference 18

1. Introduction

Data integration processes are very time and resource consuming. The amount of data and the size of datasets are constantly growing but data and information are still expected to be delivered on-time. Performance is therefore a key element in the success of a Business Intelligence & Data Warehousing project. To guarantee the agreed level of service, management of data warehouse performance and performance tuning have to take a full role during the data warehouse and ETL development process. However tuning is not straightforward always. A chain is only as strong as its weakest link. In this context, there are five crucial domains that require attention when tuning an IBM Infosphere DataStage environment:

- System Infrastructure
- Network
- Database
- IBM DataStage Installation & Configuration
- IBM DataStage Jobs

It goes without saying that without a well performing infrastructure the tuning of IBM DataStage Jobs will not make much difference. As the first three domains are usually outside the control of the ETL development team, this article will only briefly touch upon these subjects and will mainly focus on the topics related to the developments done within the IBM InfoSphere DataStage layer. There are also major differences between the underlying architecture of the DataStage Server Edition and the DataStage Parallel Edition. This article will only cover performance tuning for the IBM InfoSphere DataStage Enterprise Edition v 8.x.

One of the initial steps of performance tuning, is monitoring the current performance of the DataStage jobs. It is very important to understand what step in the job is consuming the most time and resources. To do this analysis several tools and functionalities of IBM Infosphere DataStage can be used.

2. Performance Monitoring Methods

2.1 Usage of Job Monitor

The IBM InfoSphere DataStage **job monitor** can be accessed through the IBM InfoSphere **DataStage Director**. The job monitor provides a useful snapshot of a job's performance at a certain moment of its execution, but does not provide thorough performance metrics. Due to buffering and some of the job semantics, a snapshot image of the flow might not be a representative sample of the performance over the course of the entire job. The CPU summary information provided by the job monitor is useful as a first approximation of where time is being spent in the flow. That is why a job monitor snapshot should not be used in place of a full run of the job, or a run with a sample set of data as it does not include information on sorts or similar components that might be inserted automatically by the engine in a parallel job. For these components, the score dump can be of assistance.

2.2 Usage of Score Dump

In order to resolve any performance issues it is essential to have an understanding of the data flow within the jobs. To help understand a job flow, a **score dump** should be taken. This can be done by setting the APT_DUMP_SCORE environment variable to "true" prior to running the job.

When enabled, the score dump produces a report which shows the operators, processes and data sets in the job and contains information about:

- Where and how data was repartitioned.
- Whether IBM InfoSphere DataStage has inserted extra operators in the flow.
- The degree of parallelism each operator has run with, and on which nodes.
- Where data was buffered.

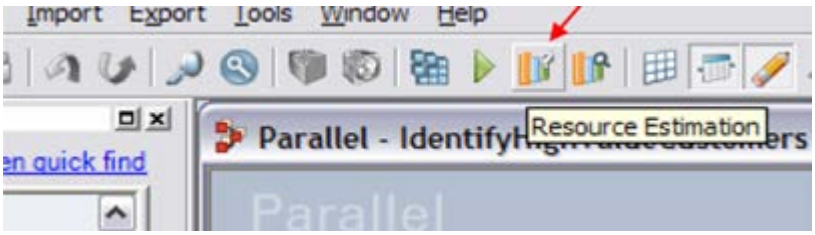
The score dump information is included in the job log when a job is run and is particularly useful in showing where IBM InfoSphere DataStage is inserting additional components/actions in the job flow, in particular extra data partitioning and sorting operators as they can both be detrimental to performance. A score dump will help to detect superfluous operators and amend the job design to remove them.

2.3 Usage of Resource Estimation

Predicting hardware resources needed to run DataStage jobs in order to meet processing time requirements can sometimes be more of an art than a science.

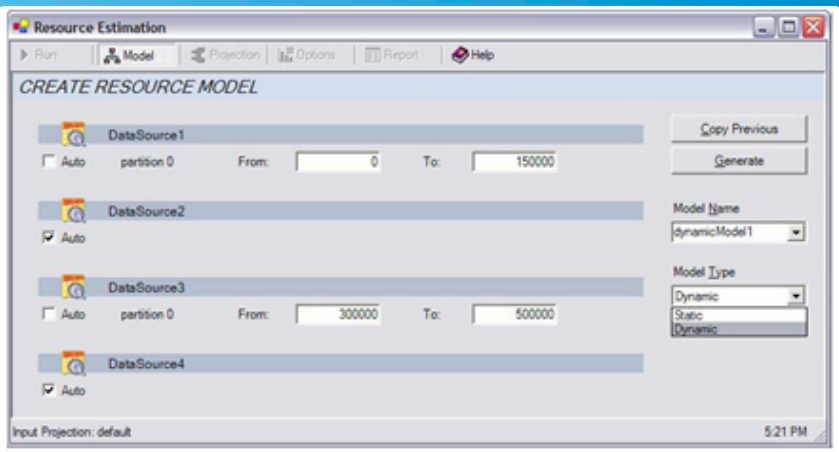
With new sophisticated analytical information and deep understanding of the parallel framework, IBM has added **Resource Estimation** to DataStage (and QualityStage) 8.x. This can be used to determine the needed system requirements or to analyze if the current infrastructure can support the jobs that have been created.

Within a job design, a new toolbar option is available called Resource Estimation.

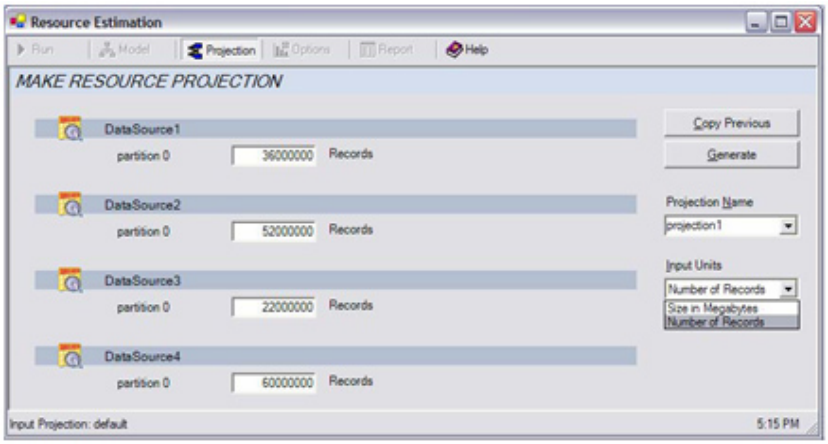


This option opens a dialog called Resource Estimation. The Resource Estimation is based on the job models. There are two types of models that can be created:

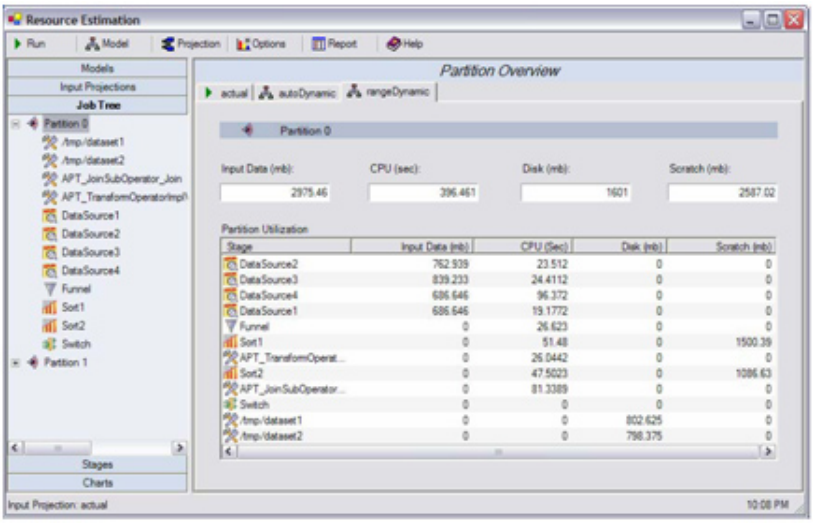
- *Static*. The static model does not actually run the job to create the model. CPU utilization cannot be estimated, but disk space can. The record size is always fixed. The "best case" scenario is considered when the input data is propagated. The "worst case" scenario is considered when computing record size.
- *Dynamic*. The Resource Estimation tool actually runs the job with a sample of the data. Both CPU and disk space are estimated. This is a more predictable way to produce estimates.



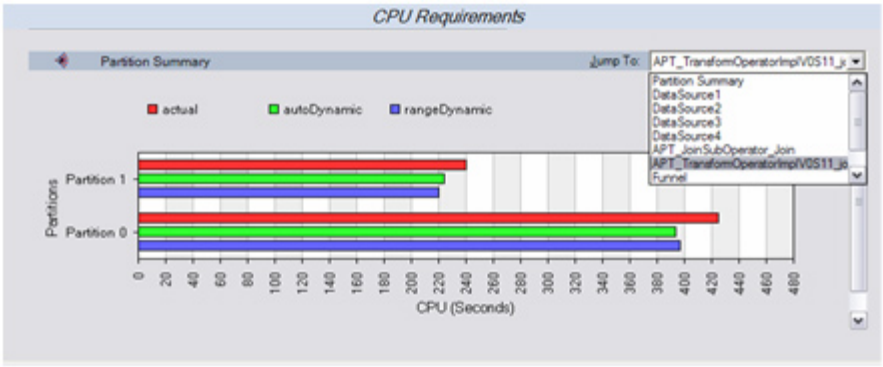
Resource Estimation is used to project the resources required to execute the job based on varying data volumes for each input data source.



A projection is then executed using the model selected. The results show the total CPU needed, disk space requirements, scratch space requirements, and other relevant information.



Different projections can be run with different data volumes and each can be saved. Graphical charts are also available for analysis, which allow the user to drill into each stage and each partition. A report can be generated or printed with the estimations.



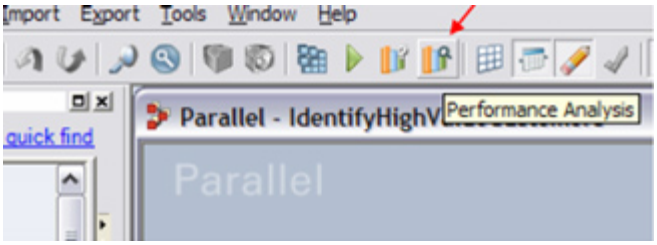
This feature will greatly assist developers in estimating the time and machine resources needed for job execution. This kind of analysis can help when analyzing the performance of a job, but IBM DataStage also offers another possibility to analyze job performance.

2.4 Usage of Performance Analysis

Isolating job performance bottlenecks during a job execution or even seeing what else was being performed on the machine during the job run can be extremely difficult. IBM Infosphere DataStage 8.x adds a new capability called **Performance Analysis**.

It is enabled through a job property on the execution tab which collects data at job execution time. (Note: by default, this option is disabled). Once enabled and with a job open, a new toolbar option, called Performance Analysis, is made available.

This option opens a new dialog called Performance Analysis. The first screen asks the user which job instance to perform the analysis on.

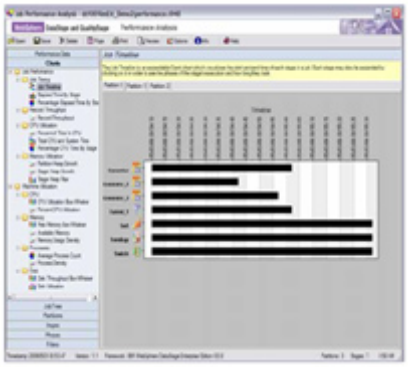


Detailed charts available for are then that specific job run including:

- Job timeline
- Record Throughput
- CPU Utilization
- Job Timing
- Job Memory Utilization

- Physical Machine Utilization (shows what else is happening overall on the machine, not just the DataStage activity).

Each partition's information is available in different tabs.



A report can be generated for each chart.

Using the information in these charts, a developer can for instance pinpoint performance bottlenecks and re-design the job to improve performance.

In addition to **instance performance**, overall machine statistics are available. When a job is running, information about the machine is also collected and is available in the Performance Analysis tool including:

- Overall CPU Utilization
- Memory Utilization
- Disk Utilization

Developers can also correlate statistics between the machine information and the job performance. Filtering capabilities exist to only display specific stages.

The information collected and shown in the Performance Analysis tool can easily be analysed to identify possible bottlenecks. These bottlenecks are usually situated in the general job design, which will be described in the following chapter.

3. Most common points for datastage jobs in performance tuning

- Select suitable configurations file (nodes depending on data volume).
- Select buffer memory correctly.
- Select proper partition.
- Turn off Runtime Column propagation wherever it's not required.
- Taking care about sorting of the data.
- Handling null values (use modify instead of transformer).
- Try to decrease the use of transformer. (Use copy, filter, modify).
- Use dataset instead of sequential file in the middle of the vast jobs.

- Take maximum 20 stages for a job for best performance.
- Select Join or Lookup or Merge (depending on data volume).
- Stop propagation of unnecessary metadata between the stages.

Points we need to consider:

- Staged the data coming from ODBC/OCI/DB2UDB stages or any database on the server using Hash/Sequential files for optimum performance.
- Tuned the OCI stage for 'Array Size' and 'Rows per Transaction' numerical values for faster inserts, updates and selects.
- Tuned the 'Project Tunable' in Administrator for better performance.
- Used sorted data for Aggregator.
- Sorted the data as much as possible in DB and reduced the use of DS-Sort for better performance of jobs.
- Removed the data and columns not used from the source as early as possible in the job.
- Worked with DB-admin to create appropriate Indexes on tables for better performance of DS queries.
- Converted some of the complex joins/business in DS to Stored Procedures on DS for faster execution of the jobs.
- If an input file has an excessive number of rows and can be split-up then use standard logic to run jobs in parallel.
- Before writing a routine or a transform, make sure that there is not the functionality required in one of the standard routines supplied in the sdk or ds utilities categories. Constraints are generally CPU intensive and take a significant amount of time to process. This may be the case if the constraint calls routines or external macros but if it is inline code then the overhead will be minimal.
- Try to have the constraints in the 'Selection' criteria of the jobs itself. This will eliminate the unnecessary records even getting in before joins are made.
- Tuning should occur on a job-by-job basis.
- Use the power of DBMS.
- Try not to use a sort stage when you can use an ORDER BY clause in the database.
- Using a constraint to filter a record set is much slower than performing a SELECT … WHERE….
- Make every attempt to use the bulk loader for your particular database. Bulk loaders are generally faster than using ODBC or OLE.

- Minimize the usage of Transformer (Instead of this use Copy modify Filter Row Generator.
- Use SQL Code while extracting the data.
- Handle the nulls properly by using modify stage.
- Minimize the warnings.
- Reduce the number of lookups in a job design.
- Try not to use more than 20 stages in a job if expected data volume is too high.
- Use IPC stage between two passive stages Reduces processing time.
- Drop indexes before data loading and recreate after loading data into tables.
- Check the write cache of Hash file. If the same hash file is used for Look up and as well as target disable this Option.
- If the hash file is used only for lookup then enable Preload to memory. This will improve the performance. Also check the order of execution of the routines.
- Don't use more than 7 lookups in the same transformer; introduce new transformers if it exceeds 7 lookups.

- Use Preload to memory option in the hash file output.
- Use Write to cache in the hash file input.
- Write into the error tables only after all the transformer stages.
- Reduce the width of the input record - remove the columns that you would not use.
- Cache the hash files you are reading from and writing into. Make sure your cache is big enough to hold the hash files.
- Use ANALYZE.FILE or HASH.HELP to determine the optimal settings for your hash files.
- Ideally, if the amount of data to be processed is small, configuration files with less number of nodes should be used while if data volume is more, configuration files with larger number of nodes should be used.
- Partitioning should be set in such a way so as to have balanced data flow i.e. nearly equal partitioning of data should occur and data skew should be minimized.
- In DataStage Jobs where high volume of data is processed, virtual memory settings for the job should be optimized. Jobs often abort in cases where a single lookup has multiple reference links. This happens due to low temp memory space. In such jobs $APT_BUFFER_MAXIMUM_MEMORY, $APT_MONITOR_SIZE and $APT_MONITOR_TIME should be set to sufficiently large values.
- Sequential files should be used in following conditions. When we are reading a flat file (fixed width or delimited) from UNIX environment which is FTP'ed from some external system
- When some UNIX operations have to be done on the file use of sequential file for intermediate storage between jobs is not allowed. It causes performance overhead, as it needs to do data conversion before writing and reading from a UNIX file.
- In order to have faster reading from the Stage the number of readers per node can be increased (default value is one).
- Usage of Dataset results in a good performance in a set of linked jobs. They help in achieving end-to-end parallelism by writing data in partitioned form and maintaining the sort order.
- Look up Stage is faster when the data volume is less. If the reference data volume is more, usage of Lookup Stage should be avoided as all reference data is pulled in to local memory.
- Sparse lookup type should be chosen only if primary input data volume is small.
- Join should be used when the data volume is high. It is a good alternative to the lookup stage and should be used when handling huge volumes of data.
- Even though data can be sorted on a link, Sort Stage is used when the data to be sorted is huge. When we sort data on link (sort / unique option) once the data size is beyond the fixed memory limit , I/O to disk takes place, which incurs an overhead. Therefore, if the volume of data is large explicit sort stage should be used instead of sort on link. Sort Stage gives an option on increasing the buffer memory used for sorting this would mean lower I/O and better performance.

- It is also advisable to reduce the number of transformers in a Job by combining the logic into a single transformer rather than having multiple transformers.
- Presence of a Funnel Stage reduces the performance of a job. It would increase the time taken by job by 30% (observations). When a Funnel Stage is to be used in a large job it is better to isolate itself to one job. Write the output to Datasets and funnel them in new job.
- Funnel Stage should be run in "continuous" mode, without hindrance.
- A single job should not be overloaded with Stages. Each extra Stage put in a Job corresponds to lesser number of resources available for every Stage, which directly affects the Jobs Performance. If possible, big jobs having large number of Stages should be logically split into smaller units.
- Unnecessary column propagation should not be done. As far as possible, RCP (Runtime Column Propagation) should be disabled in the jobs.
- Most often neglected option is "don't sort if previously sorted" in sort Stage, set this option to "true". This improves the Sort Stage performance a great deal.

- In Transformer Stage "Preserve Sort Order" can be used to maintain sort order of the data and reduce sorting in the job.
- Reduce the number of Stage variables used.
- The Copy stage should be used instead of a Transformer for simple operations.
- The "upsert" works well if the data is sorted on the primary key column of the table which is being loaded.
- Don't read from a Sequential File using SAME partitioning.
- By using hashfile stage we can improve the performance. In case of hashfile stage we can define the read cache size & write cache size but the default size is 128MB.
- By using active-to-active link performance also we can improve the performance. Here we can improve the performance by enabling the row buffer, the default row buffer size is 128 KB.

4. General Job Design Best Practices

The ability to process large volumes of data in a short period of time depends on all aspects of the flow and the environment being optimized for maximum throughput and performance. Performance tuning and optimization are iterative processes that begin with job design and unit tests, proceed through integration and volume testing, and continue throughout the production life cycle of the application. Here are some performance pointers:

4.1 Columns and type conversions

Remove unneeded columns as early as possible within the job flow. Every additional unused column requires additional buffer memory, which can impact performance and make each row transfer from one stage to the next more expensive. If possible, when reading from databases, use a select list to read only the columns required, rather than the entire table. Avoid propagation of unnecessary metadata between the stages. Use the Modify stage and drop the metadata. The Modify stage will drop the metadata only when explicitly specified using the DROP clause.

So only columns that are really needed in the job should be used and the columns should be dropped from the moment they are not needed anymore. The OSH_PRINT_SCHEMAS environment variable can be set to verify that runtime schemas match the job design column definitions. When using stage variables on a Transformer stage, ensure that their data types match the expected result types. Avoid that DataStage needs to perform unnecessary type conversions as it will use time and resources for these conversions.

4.2 Transformer stages

It is best practice to avoid having multiple stages where the functionality could be incorporated into a single stage, and use other stage types to perform simple transformation operations. Try to balance load on Transformers by sharing the transformations across existing Transformers. This will ensure a smooth flow of data.

When type casting, renaming of columns or addition of new columns is required, use Copy or Modify Stages to achieve this. The Copy stage, for example, should be used instead of a Transformer for simple operations including:

- Job Design placeholder between stages
- Renaming Columns
- Dropping Columns
- Implicit (default) Type Conversions

A developer should try to minimize the stage variables in a Transformer stage because the performance of a job decreases as stage variables are added in a Transformer stage. The number of stage variables should be limited as much as possible.

Also if a particular stage has been identified as one that takes a lot of time in a job, like a Transformer stage having complex functionality with a lot of stage variables and transformations, then the design of jobs could be done in such a way that this stage is put in a separate job all together (more resources for the Transformer Stage).

While designing IBM DataStage Jobs, care should be taken that a single job is not overloaded with stages. Each extra stage put into a job corresponds to less resources being available for every stage, which directly affects the job performance. If possible, complex jobs having a large number of stages should be logically split into smaller units.

4.3 Sorting

A sort done on a database is usually a lot faster than a sort done in DataStage. So – if possible – try to already do the sorting when reading data from the database instead of using a Sort stage or sorting on the input link. This could also mean a big performance gain in the job, although it is not always possible to avoid needing a Sort stage in jobs.

Careful job design can improve the performance of sort operations, both in standalone Sort stages and in on-link sorts specified in other stage types, when not being able to make use of the database sorting power.

If data has already been partitioned and sorted on a set of key columns, specify the "don't sort, previously sorted" option for the key columns in the Sort stage. This reduces the cost of sorting and takes more advantage of pipeline parallelism. When writing to parallel data sets, sort order and partitioning are preserved. When reading from these data sets, try to maintain this sorting if possible by using the *Same* partitioning method.

The stable sort option is much more expensive than non-stable sorts, and should only be used if there is a need to maintain row order other than as needed to perform the sort.

The performance of individual sorts can be improved by increasing the memory usage per partition using the **Restrict Memory Usage** (MB) option of the Sort stage. The default setting is 20 MB per partition. Note that sort memory usage can only be specified for standalone Sort stages, it cannot be changed for inline (on a link) sorts.

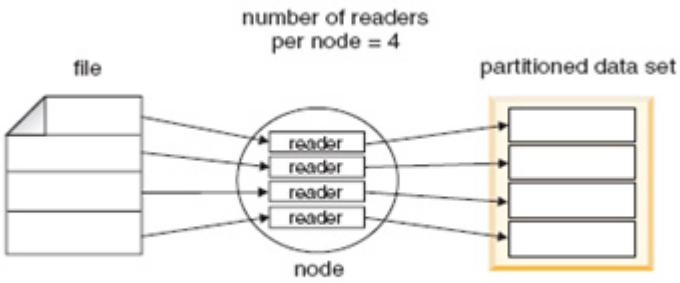
4.4 Sequential files

While handling huge volumes of data, the **Sequential File stage** can itself become one of the major bottlenecks as reading and writing from this stage is slow. Certainly do not use sequential files for intermediate storage between jobs. It causes performance overhead, as it needs to do data conversion before writing and reading from a file. Rather Dataset stages should be used for intermediate storage between different jobs.

Datasets are key to good performance in a set of linked jobs. They help in achieving end-to-end parallelism by writing data in partitioned form and maintaining the sort order. No repartitioning or import/export conversions are needed.

In order to have faster reading from the Sequential File stage the number of readers per node can be increased (default value is one). This means, for example, that a single file can be partitioned as it is read (even though the stage is constrained to running sequentially on the conductor mode).

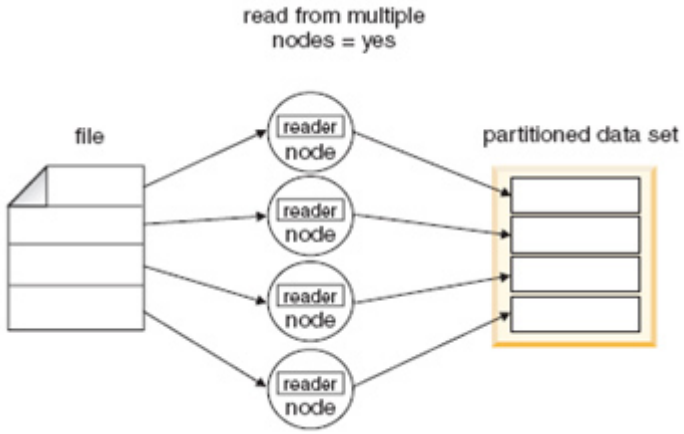
This is an optional property and only applies to files containing fixed-length records. But this provides a way of partitioning data contained in a single file. Each node reads a single file, but the file can be divided according to the number of readers per node, and written to separate partitions. This method can result in better I/O performance on an SMP (Symmetric Multi-Processing) system.



Read from multiple nodes

It can also be specified that single files can be read by multiple nodes. This is also an optional property and only applies to files containing fixed-length records. Set this option to "Yes" to allow individual files to be read by several nodes. This can improve performance on cluster systems.

IBM DataStage knows the number of nodes available, and using the fixed length record size, and the actual size of the file to be read, allocates to the reader on each node a separate region within the file to process. The regions will be of roughly equal size.



The options "Read From Multiple Nodes" and "Number of Readers Per Node" are mutually exclusive.

4.5 Runtime Column Propagation

Also while designing jobs, care must be taken that unnecessary column propagation is not done. Columns, which are not needed in the job flow, should not be propagated from one stage to another and from one job to the next. As much as possible, RCP (**Runtime Column Propagation**) should be disabled in the jobs.

4.6 Join, Lookup or Merge

One of the most important mistakes that developers make is to not have volumetric analyses done before deciding to use Join, Lookup or Merge stages.

IBM DataStage does not know how large the data set is, so it cannot make an informed choice whether to combine data using a Join stage or a Lookup stage. Here is how to decide which one to use …

There are two data sets being combined. One is the primary or driving data set, sometimes called the left of the join. The other dataset are the reference data set or the right of the join.

In all cases, the size of the reference data sets is a concern. If these take up a large amount of memory relative to the physical RAM memory size of the computer DataStage is running on, then a Lookup stage might crash because the reference datasets might not fit in RAM along with everything else that has to be in RAM. This would result in a very slow performance since each lookup operation can, and typically will, cause a page fault and an I/O operation.

So, if the reference datasets are big enough to cause trouble, use a join. A join does a high-speed sort on the driving and reference datasets. This can involve I/O if the data is big enough, but the I/O is all highly optimized and sequential. After the sort is over, the join processing is very fast and never involves paging or other I/O.

4.7 Databases

The best choice is to use Connector stages if available for the database. The next best choice is the Enterprise database stages as these give maximum parallel performance and features when compared to 'plug-in' stages. The Enterprise stages are,

- DB2/UDB Enterprise
- Informix® Enterprise
- Oracle Enterprise
- Teradata Enterprise
- SQLServer Enterprise
- Sybase Enterprise
- ODBC Enterprise
- iWay Enterprise
- Netezza Enterprise

Avoid generating target tables in the database from the IBM DataStage job (that is, using the Create write mode on the database stage) unless they are intended for temporary storage only. This is because this method does not allow, for example, specifying target table space, and inadvertently data-management policies on the database can be violated.

When there is a need to create a table on a target database from within a job, use the Open command property on the database stage to explicitly create the table and allocate table space, or any other options required. The Open command property allows to specify a command (for example some SQL) that will be executed by the database

before it processes any data from the stage. There is also a Close property that allows specifying a command to execute after the data from the stage has been processed. (Note that, when using user-defined Open and Close commands, locks should be specified where appropriate).

Tune the database stages for 'Array Size' and 'Rows per Transaction' numerical values for faster inserts, updates and selects. Experiment in changing these values to see what the best performance is for the DataStage job. The default value used is low and not optimal in terms of performance.

Finally, try to **work closely with the database administrators** so they can examine the SQL-statements used in DataStage jobs. Appropriate indexes on tables can deliver a better performance of DataStage queries.

Also try to examine if the job is faster when the indexes are dropped before data loading and recreated after loading data into the tables. Recreation of the indexes also takes some time, so test if this has a performance gain or a performance loss on the total process chain.

4.8 Partitioning Considerations

Choose a partition method which makes sure that the number of rows per partition is close to equal. This will minimize the processing work load and there by improves the overall run time. Any stage that process a group of related records must be partitioned using a keyed partition technique. (Exampls: in the case of Aggregator stage, Remove duplicate, Change capture, Change apply, Join, Merge stages etc, as well as for transformers that process group of related records)

- Minimize repartitioning as it decreases the performance unless the partition distribution is highly skewed. Repartitioning results in overhead of network transport as well as even distribution of data among partitions is also gets disturbed.

- Specify hash partitioning for stages that require processing of group of related records. Partitioning keys should include only those key columns that are necessary for proper grouping If the grouping is on a single integer key column, go for Modulus partition on the same key column If the data is highly skewed and the key column values and distribution will not change significantly over time, use the Range partitioning technique.
- Use Round robin partition to distribute data evenly across all partitions. (If grouping is not needed).This is very much suggested when the input data is in sequential mode or it is very much skewed Same partitioning requires minimum resources and can be used for optimization of job and to eliminate repartitioning of the already partitioned data.
- When the input data set is sorted in parallel, we need to use Sort merge collector, which will produce a single sorted stream of rows. When the input data set is sorted in parallel and range partitioned, the ordered collector method is more preferred for collection.
- For round robin partitioned input data set use round robin collector to reconstruct rows in input order, as long as the data set has not been re partitioned or reduced.
- Minimize the use of sorts in a job.

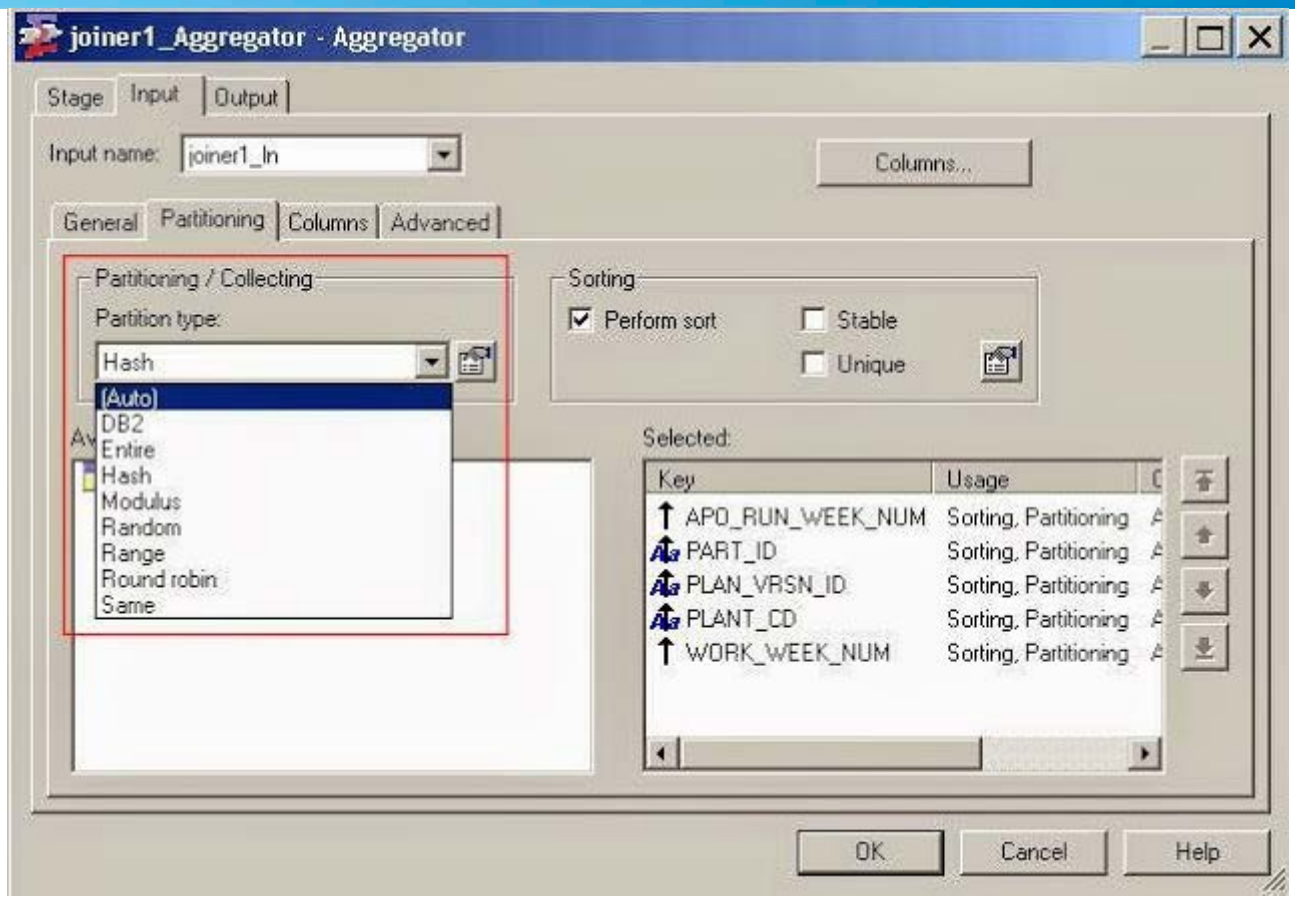


Figure: Partitioning tab in a Datastage stage properties

5. Conclusion

Performance tuning can be a labor intensive and quite costly process. That is exactly the reason why care for optimization and performance should be taken into account from the beginning of the development process. With the combination of best practices, performance guidelines and past experience, the majority of performance problems can be avoided during the development process.

If performance issues still occur even when performance guidelines have been taken into account during development, then these issues can be tackled and analyzed using the available, discussed tools such as Resource Estimation and Performance Analysis functionalities.

6. Reference

http://www.element61.be/e/resourc-detail.asp?ResourceId=188
http://datastageinfoguide.blogspot.in/2013/10/datastage-jobs-best-practices-and.html

Thank You

Contact

For more information, contact gsl.cdsfiodg@tcs.com (Email Id of ISU)

About Tata Consultancy Services (TCS)

Tata Consultancy Services is an IT services, consulting and business solutions organization that delivers real results to global business, ensuring a level of certainty no other firm can match. TCS offers a consulting-led, integrated portfolio of IT and IT-enabled infrastructure, engineering and assurance services. This is delivered through its unique Global Network Delivery ModelTM, recognized as the benchmark of excellence in software development. A part of the Tata Group, India's largest industrial conglomerate, TCS has a global footprint and is listed on the National Stock Exchange and Bombay Stock Exchange in India.

For more information, visit us at www.tcs.com.

IT Services/Business Solutions/Consulting