IBM® Smart Analytics System

Best practices

Ingesting data into an IBM Smart Analytics System

Austin Clifford
IBM DB2 Data Warehouse QA Specialist

Garrett Fitzsimons
IBM Data Warehouse Best Practices Specialist

Richard Lubell
IBM Smart Analytics Systems Information Developer

Issued June 2011

Executive summary

This paper is targeted at people involved in the design and development of data ingest applications based on DB2® Database for Linux, UNIX, and Windows database software. In particular, this paper focuses on ingesting data into the IBM® Smart Analytics System environment with configurations based on System x® and Power Systems™ servers.

The key design goals of ingest application design are to balance the required rate of ingest with the availability of processing resources, to ingest data without affecting data availability and to maintain flexibility in the volumes of data ingested. Identifying service level objectives for the speed and volume of data ingested into the data warehouse will determine the architecture, design, and development of the data ingest application.

The design of a data warehouse database is focused on query performance. Employing staging tables to load and transform data ahead of ingest minimizes interference with query workloads. Further benefits are gained through managing database locking and logging and system resource usage with the software tools available with DB2 and IBM Optim™ software available with the IBM Smart Analytics System.

You can minimize the resources needed to perform the movement of data from the staging tables to the production tables by using table partitioning to subdivide data during transformation, creating parallel ingests and reducing overall workload.

The data ingest application design should accommodate an active warehousing environment where all workloads are intended to run online and concurrently. This allows data to be processed as soon as it is made available and anticipates the growth of the data warehouse.

A data ingest application should contain individual components that can process data as promptly and completely as possible after it arrives. This helps reduce the time needed for the ingest application development process, isolate errors through targeted testing, and reduce the volumes of data that require reprocessing when errors occur.

The IBM Smart Analytics System product incorporates best practices guidelines in building a stable and scalable data warehouse environment. Using the recommendations in this paper will help you develop a data ingest application that will be scalable, balanced, and flexible enough to meet future as well existing needs.

Introduction

This paper describes best practices for designing and implementing a data ingest application for an IBM Smart Analytics System data warehouse. The paper covers how to ingest large volumes of extracted data into the data warehouse in preparation for analytical queries. This paper assumes that you have a working knowledge of DB2 software including partitioned databases. DB2 Database Version 9.7 fix pack 3 was used in the test system for this paper.

Database administration should not be part of the data ingest workload, though data ingest application designers and developers should work with the database administrator to help ensure that all workloads are balanced. The data presented in this paper is based on testing and experience with customers needs.

A well-designed data ingest application helps to achieve business requirements for the availability of data. Defining your service level objectives dictates the characteristics of your data ingest applications. Design the solution to work concurrently with all other workloads while also taking advantage of features of the partitioned database. Parallelism, co-location and control of the ingest rate can maximize the efficiency of loading data into staging tables and subsequently inserting data into production tables. Statistics and monitoring are used to define a baseline and enable tuning for optimum performance.

This paper builds on and complements a series of best practices papers that examine all aspects of data warehouse design and implementation for a DB2 data warehouse. Refer to these papers, referenced in the "Further reading" section, for further information.

Planning for data ingest

When planning to build your data ingest application, consider the overall environment where the application operates, the data warehouse design and competing workloads. Business requirements for the availability of data, together with constraints imposed by resources, determine your data ingest application design. These requirements are expressed as service level objectives (SLOs). Use these objectives to develop a data ingest application that meets achievable targets and operates in concert with all other workloads.

The following factors can affect your data ingest design. Integrate these recommendations into your planning phase:

- For most operations, use the tools available within the core IBM Smart Analytics System modules. For complex or high volume Extract, Transform, and Load (ETL) operations, use IBM InfoSphere® DataStage® software as the ETL tool for building a data ingest application.
- Avoid locking data that is required by the query workload or saturating the DB2 network with data ingest traffic during peak query times. For allocation of system resources, the query workload has the highest priority among data workloads.
- Identify and plan for peak data volumes rather than the average data volume expected and allow for system outages.
- Incorporate the population of all objects that are affected by the new data into the ingest process.

Know when data is scheduled for ingest, the volume of data to be presented and the expectation for data availability.

Calculating service level objectives

Before designing and developing a data ingest application, identify your service level objectives. These objectives consist of the volume of data to be loaded and the time and resources available to complete the load.

To calculate SLOs, establish the ingest rate per data module and the ingest schedule. Use the processing window and data volumes to calculate the rate of ingest required for each of the data sources. Express the target data ingest rate per data module as megabytes per second (MB/s).

The **processing window** is the period when data is ingested into the data warehouse tables. **Data volume** refers to the quantity of raw data presented for ingest during each processing window. Establish both of these figures for each ingest cycle as a starting point for further calculations.

Ingest rate

Calculate ingest rate as follows:

1. Estimate the volume of data in megabytes (MB) to be ingested per ingest cycle.
2. State the time in seconds allowed for the ingest process to complete.
3. Divide the volume by the time and then divide by the number data modules receiving data.

The implementation of indexes, materialized query tables (MQTs), and replicated tables increases the number of transactions generated in the database. Take into account the refresh schedule for summary tables and MQTs when calculating the processing window. Establish where these objects exist and when and how they are refreshed.

Ingest schedule and SLO values

Business needs dictate how quickly data must be available for query. Data ingest schedules are broadly categorized as daily batch window, intraday, or continuous feed. Your ingest schedule incorporates one or more of these categories. This information is integrated into the ingest schedule, which influences your processing window as well as workload balancing.

Understand the service level objectives for other workloads in your environment and how they affect the data ingest workload.

Define your service level objectives by documenting the values you have identified for schedule, processing window, volume, modules, and rate. Use these metrics to derive capacity requirements. Consider this resource usage in the context of time, data sources, and other workloads when designing your data ingest application.

Architecture and design principles for data ingest

Meeting your service level objectives requires a database design and data ingest application architecture that provide the required data ingest rate. Participate in the database design phase, if possible, to promote a data model that helps ensure an efficient ingest architecture.

Best practices for data ingest application architecture

The architecture of your data ingest application should facilitate the recommended data ingest process, which is to first load source data into staging tables where it can be validated and transformed, then move the data into the production tables as logged transactions. Using a staging area to load and transform data minimizes activity on the production tables which reduces locking and increases data availability.

Use the DB2 load utility to load data into staging tables, and then use the INSERT statement to move data from staging into production tables to minimize the effect on the query workload.

When designing your data ingest application, employ the following practices:

- To isolate errors and provide flexibility, design individual independent application components to process data in logical blocks rather than in a single sequence from source file to production table.
- To balance resource usage, design application components that process discrete phases of the ingest cycle when data arrives rather than implementing an end-to-end design that requires all data to be available before starting.
- To enforce data availability upon completion of the ingest operation, incorporate the refresh of MQTs into the data ingest application architecture. To minimize resource usage, avoid refreshing MQTs when data does not need to be available.

Design components that process data in discrete steps as soon as it becomes available.

The DB2 load utility uses the distribution map for the target table to split the data into data sets specific to each database partition. The data sets are moved directly to each database partition and loaded directly into the partitioned staging table. The data is then transformed in the staging tables and moved into the production tables by using the INSERT subselect method which takes advantage of a SELECT subquery in an INSERT statement. The following diagram shows the architecture of the DB2 load utility within an IBM Smart Analytics System:

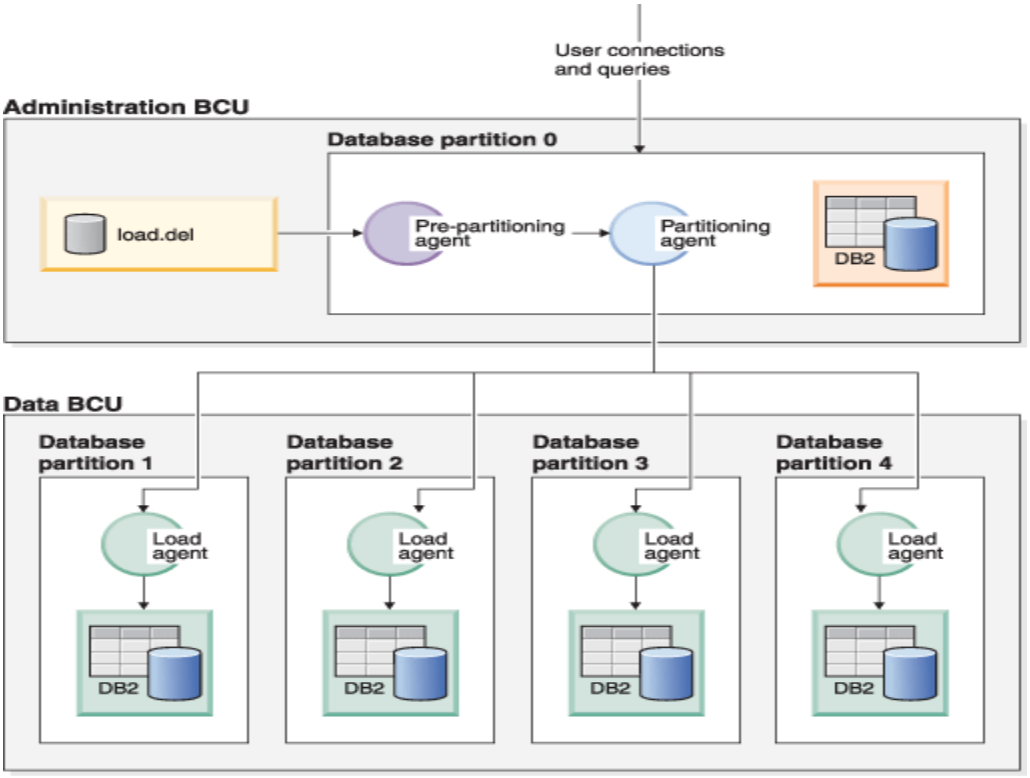

Figure 1 DB2 Load architecture

Use the same distribution key on source staging and target production tables to enable collocation when moving data from staging to production tables. This enables a SELECT statement in the INSERT subselect to execute as a collocated query, which does not generate inter-partition traffic. Additional network traffic is generated when a query on one database partition can be completed only by requesting data from another database partition.

Enable collocated queries to increase INSERT with subselect statement performance by aligning distribution keys on partitioned staging and production tables.

The load utility is more efficient that the import utility in moving large amounts of data into the database and is recommended in data warehouse environments.

Best practices for database design

A poorly designed database can have a negative effect on both ingest and query performance through increased locking, increased use of resources, and an inability to recover from errors or run concurrent operations.

Design tables and table spaces to minimize input and output operations and to maximize workload concurrency.

Table space design

Design table spaces to maximize data availability and maintainability by following these practices:

- Separate tables that are not logically related into different table spaces so that all database operations can be targeted at only the affected tables.
- Place production, staging, and metadata tables in separate table spaces and schemas. Avoid creating production tables that have a dependency on staging tables or table spaces.
- Deploy an average table space size of approximately 75 GB per database partition. This table space size facilitates efficient management and maintenance of the database and allows for multiple operations to be executing in multiple table spaces in parallel.

Table design

Design staging tables to minimize input and output and increase data availability by following these practices:

- Implement indexes on staging tables to improve INSERT with subselect performance when using range-based inserts.
- Implement unique indexes on staging tables in conjunction with the FOR EXCEPTION parameter of the LOAD command only when you wish to identify and reject duplicate records.
- Avoid implementing multi-dimensional clustering (MDC) tables in your staging area because the maintenance of block indexes increase input and output. Implement MDC tables in your production database design instead.
- Avoid retaining ingested data in staging tables. Retention leads to slower ingest performance and increased table maintenance.
- Compress staging tables to reduce input and output to disk and improve INSERT subselect statement performance through more efficient buffer pool usage.
- Avoid range partitioning in staging tables because retaining staged data degrades performance.

Design production tables to minimize input and output and increase data availability by following these practices:

- Use MDC tables with a dimension of “day” for large production tables. This setting reduces index maintenance and locking during INSERT with subselect statements.
- Specify a larger value for the **`CACHE`** parameter when creating a **`SEQUENCE`** object to generate surrogate keys in high volume loads.
- Avoid replicating tables for enhancing the ETL process because the cost in additional processing and administration tasks can offset the benefits gained during ingest.

Loading data into the staging area

The DB2 load utility is designed for loading large volumes of data into the database and is optimized for partitioned databases. The load operation is efficient at processing large batches of data, though larger batches do use more resources. The load utility is the fastest method of ingesting data into a partitioned database because the load utility reads the distribution map to presplit and sort data into data sets specific to each database partition.

Configuring the database memory parameters for load

The DB2 software installed on the IBM Smart Analytics System is configured following best practice recommendations. For large data volumes or multiple parallel load operations, you might need to tune configuration parameters to determine the optimum values.

The load operation uses memory as configured through the **`util_heap_sz`** database configuration parameter. The default setting in an IBM Smart Analytics System for the **`util_heap_sz`** database configuration parameter is 65,536 4 K pages.

Increase the **`util_heap_sz`** database configuration parameter when you run multiple load operations in parallel or you plan to run the **`LOAD`** command concurrently with other utilities such as the **`BACKUP`** and **`REORG`** commands.

Use the `DATA BUFFER` parameter of the `LOAD` command to control the amount of memory used when multiple load operations are submitted in parallel.

Where multiple load operations are executed in parallel and the **`DATA BUFFER`** parameter is not specified, each subsequent load operation acquires 25% of remaining memory, causing diminishing returns. Determine the parameter value by noting the amount of memory used by each operation and calculating how much memory is needed during peak activity. You should also monitor DB2 network activity to determine if there is a negative effect on query workload.

Setting `LOAD` command parameters

There are several parameter options available when using the **`LOAD`** command. Refer to the "Further reading" section for references to complete descriptions of each parameter and its usage. The following parameter values are recommended:

- **`DATA BUFFER`**

  Use this parameter of the **`LOAD`** command to determine the amount of memory available in the **`util_heap_sz`** buffer assigned to the load operation.

- **`REPLACE`**

  Use this parameter to force the staging table to be cleared before data is loaded, avoiding the need to manually delete data or drop and create tables.

- **`NONRECOVERABLE`** (**`COPY YES`**)

Use this parameter to disable transaction logging and exclude the staging table space from any rollforward operations. Using the **COPY YES** parameters incurs additional processing of loaded data.

- **MODIFIED BY ANYORDER**

  Use this parameter to increase the number of database partitions that the load utility uses to sort and presplit the data during the load operation.

- **STATISTICS USE PROFILE**

  Use this parameter to have the **LOAD** utility collect statistics as per the profile for the table into which data is being loaded. Using this parameter is recommended where the INSERT with subselect statement uses joins for key lookups.

- **FOR EXCEPTION**

  Use this parameter to specify an exception table that receives copied rows that have violated unique key or primary key index rules.

- **MESSAGES**

  Use this parameter to specify the destination for warning and error messages that occur during the load operation. If the complete path to the file is not specified, the **LOAD** utility uses the current directory and the default drive as the destination. If the name of a file that exists is specified, the utility appends the information to that file.

- **MODIFIED BY DUMPFILE**

  Use this parameter to specify a file to record the content of rows rejected during load. The DUMPFILE parameter must be a full path to the dump file. One dumpfile is created per database partition.

An example of using the LOAD command

This example of the **LOAD** command uses `customer.del` as the source file with `|` as the pipe delimiter. `stg3.customer` represents the staging schema and table in which data is to be loaded.

```
LOAD FROM customer.del OF DEL MODIFIED BY ANYORDER
DUMPFILE=/db2home/bcuaix/rejects/st3.customer_rejects.txt coldel|
MESSAGES stg3.customer_messages.txt REPLACE INTO stg3.customer FOR
EXCEPTION stg3.customer_excepts STATISTICS USE PROFILE NONRECOVERABLE
DATA BUFFER 4000 PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (1, 5,
9, 13, 17, 21, 25, 29, 33, 37, 41, 45)
```

The **MODIFIED BY ANYORDER** and **PARTITIONING_DBPARTNUMS** parameters specify the database partitions used in splitting and sorting data, and the **DATA BUFFER** parameter specify the amount of memory used. Messages are captured for processing when failures such as data type errors occur. These messages are then written to an output file for each database partition. Unique index violations are rejected and inserted into an exceptions table. Statistics are collected as specified by the profile for the table.

Use indexes on staging tables only to improve INSERT with subselect performance or when needed to move data to the next phase of the ingest process

Exceptions tables must exist in the same database partition group as the table targeted for data loading. Refer to the "Further reading" section for resources providing a more complete description of exception processing with the **`LOAD`** utility.

The load operation exits with a return code that you should capture in metadata tables for each database partition to determine success, failure, and trends in error rates. When processing errors occur, minimize the volume of valid data that needs to be reprocessed as a result of being in the same batch as invalid data.

- Reject data in rows rather than batches. This division avoids reprocessing valid data multiple times
- Redirect rejected data to error tables. Data can then be corrected and reintroduced into the ingest processing cycle.

Executing multiple load operations in high-performance environments

Establish the amount of network traffic generated by the load operation through testing and then balance this with your service level objectives for ingest rate and for resource usage. Take into account other utilities and workloads that might be running in production to determine the optimum number of load operations to run in parallel. When ingesting large volumes of data, monitor performance to establish the optimum number of load operations; execute no more than six load operations in parallel per database partition.

Parallelism can help increase your ingest rate when larger volumes of data need to be ingested. Implement the following practices:

- Create multiple sets of staging tables for a data source in the staging area to enable multiple data load operations to be submitted in parallel.
- Incorporate range-based processing into insert and transform components to enable more than one range of data to be processed in parallel. Range-based processing also allows greater control over locking and logging.

Multiple load operations can process sets of data into different staging tables in parallel. Implement the following practices:

- Control the ingest rate by increasing or decreasing the number of load operations being submitted in parallel.
- Use control tables in the metadata area to manage the status of a file and the number of retries.

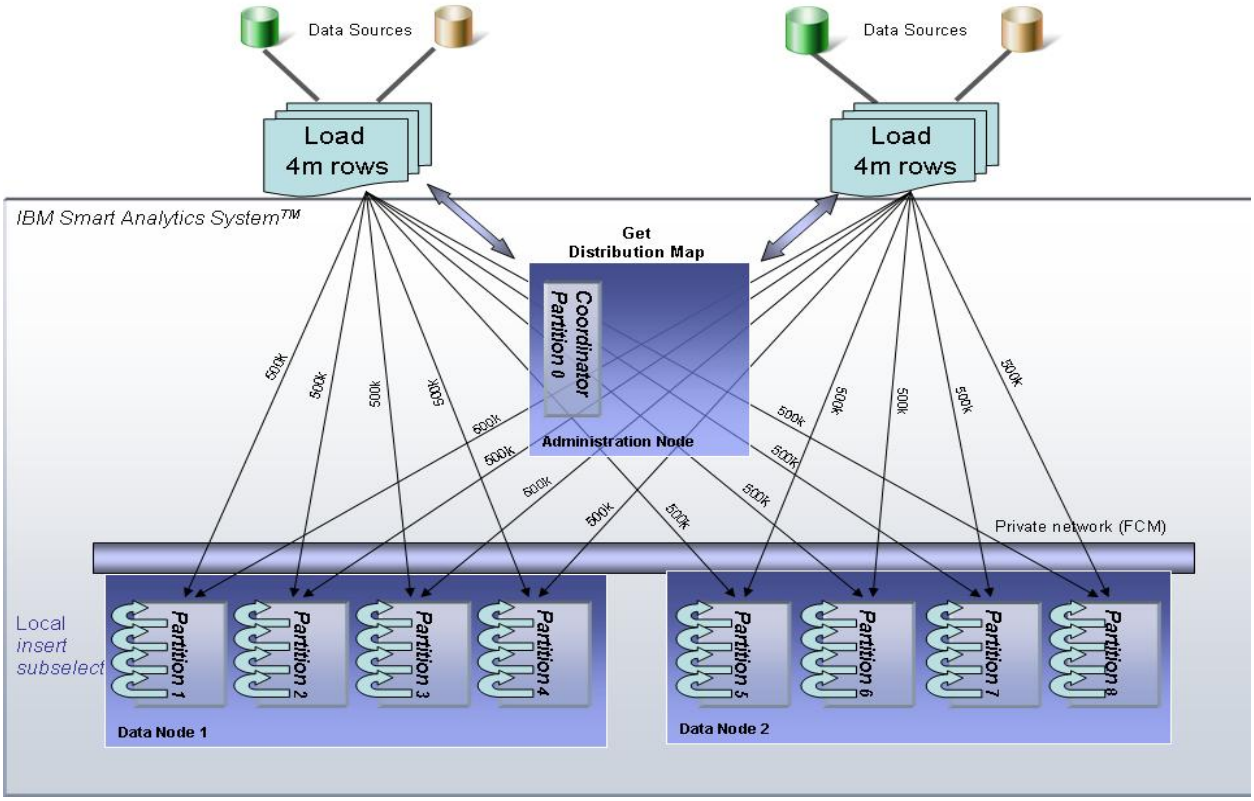

Figure 2. Parallel processing of load and INSERT subselect operations

Figure 2 depicts two load operations, each loading four million rows in parallel into a partitioned staging table on database partitions 1 to 8. The partitioned staging table uses the same distribution key as the target production table. The load utility first queries the distribution map on the administration module to determine how to presplit data before sending each data set directly to the correct database partition. The INSERT subselect then takes place locally on each database partition.

For high-performance environments, the following parameter values are recommended:

- **`MAX_NUM_PART_AGENTS`**

  Use with the **`MODIFIED BY ANYORDER`** parameter to increase the number of partitioning agents used in sorting and pre-splitting data when the total number of database partitions is greater than the default value of 25.

- **`PARTITIONING_DBPARTNUMS`**

  When there are more than 25 database partitions, the `LOAD` command will use the first 25 database partitions, possibly causing skewed performance. Use this parameter to distribute sorting and pre-splitting processing evenly across large environments with more than 25 database partitions by specifying the database partitions to be used. This parameter is used with the **`MAX_NUM_PART_AGENTS`** and **`MODIFIED BY ANYORDER`** parameters.

Inserting data into the production tables

Move data from the staging tables to the production tables by using a SELECT statement within an INSERT statement to select the data from the staging table and insert it into the production table.

The implementation of collocated staging tables as recommended in the "Loading data into the staging area" section helps to ensure that data in the staging tables is already located on the home database partition into which it is inserted.

When data has been loaded into the database, implement any changes to the data through the use of SQL commands. Extracting data a second time adds unnecessary input and output to the ingest process.

Using the INSERT with subselect statement

Using the INSERT with subselect statement allows the query workload to continue while data is being ingested into the data warehouse.

The INSERT statement, unlike the load utility, is a logged transaction. The database remains recoverable and the target table and table space remain in a normal state during the INSERT operation. Replication and disaster recovery methods that use log-shipping are therefore supported.

The following recommendations apply when using the INSERT with subselect statement:

- Parameterize the number of rows to be inserted per INSERT with subselect statement to govern the commit size and control the effect on locking and logging.
- Control the ingest rate to manage resource availability by increasing or decreasing the number of INSERT with subselect statements submitted in parallel.
- Avoid executing insert and online backup operations on the same table space at the same time. Executing insert and online backup operations on the same table space at the same time increases the amount of transaction logs needed during recovery and reduces flexibility in choosing recovery methods.

Commit size influences both ingest rate and resource usage. For normal processing windows, 10,000 rows per database partition is a good reference point. Use a larger commit size when more resources are available to the data ingest application, during off-peak windows or where concurrency with other workloads is not required. Further increase the ingest rate by issuing multiple INSERT with subselect commands in parallel.

Managing lock escalation

Lock escalation, when a lock is escalated from a row to a table level exclusive lock, is triggered when there is not enough lock space for other concurrent applications. Excessive locking uses additional CPU resources and reduces concurrency due to lock conflicts.

Locking might occur during the ingest process where multiple INSERT with subselect statements are executed in parallel against the same table causing a deadlock. Where locking occurs, use a smaller commit size to minimize locking at the expense of your ingest rate or consider tuning the configuration parameters as referenced in the "Setting parameters for parallel inserts" section.

Monitor and prevent lock escalation by controlling the commit size in conjunction with the `LOCKLIST` and `MAXLOCKS` database configuration parameters.

Monitor and prevent the escalation of row level locks to table exclusive locks by setting appropriate values for the **`LOCKLIST`** and **`MAXLOCKS`** database configuration parameters.

- The **`LOCKLIST`** parameter indicates the storage allocated to the lock list. The lock list comprises locks held by all applications connected to the database. Increasing this value will increase the amount of memory reserved for the lock list.
- The **`MAXLOCKS`** parameter defines the percentage of the lock list that must be filled before the database manager performs lock escalation. Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list

The default value for the LOCKLIST parameter in an IBM Smart Analytics System is 16,384 4K pages; the default value for the MAXLOCKS is 10.

To select the optimal values of **`MAXLOCKS`** and **`LOCKSLIST`** for your system, use the tools such as the DB2 GET SNAPSHOT command, the db2top monitoring utility, DB2 Performance Expert, or IBM Optim Performance Manager to monitor the lock list that displays memory in use and locks held during data ingest.

Using range-based inserts in high-performance environments

The number of rows inserted through the INSERT with subselect statement affects locking, transaction log archiving, and ingest rate. Submitting multiple INSERT with subselect statements in parallel can help ensure a sufficient ingest rate without using the same resources as a single larger insert.

Issue multiple INSERT with subselect statements in parallel against the same staging table by using the WHERE clause to select ranges of data from the staging table. The number of records returned to the INSERT statement can be increased or decreased by adjusting this range.

Consider increasing the value of the MAXLOCKS and LOCKLIST parameters when using parallel inserts against the same table. Introduce indexing on the staging tables where the range of data being moved represents less than 10% of the total number of rows in the staging table to improve performance of the INSERT with subselect statement.

In the following example, when inserting rows into a table called LINEITEM, the PARTKEY column is used to split the data to be inserted into 5 sets of data. Each dataset is then submitted in parallel by using the INSERT with subselect statement. The number of rows between *x* and *y* determines the commit size:

```
insert into dw.lineitem select * from stage.stg1_lineitem where partkey
between x and y
```

where:

- x is the start number for the range of rows to be selected from the staging table and inserted into the production table.
- y is the end number for the range of rows to be selected from the staging table and inserted into the production table.

- x and y are determined by the minimum PARTKEY value, maximum PARTKEY value, total number of rows to be inserted, and the number of parallel INSERT with subselect statements to be executed.

Setting parameters for parallel inserts

To implement a more precise grouping method, retrieve the exact number of rows to be inserted on the primary or distribution key and divide by the commit size needed, as shown in the following example:

- The LOCKLIST requirement for ingest can be expressed by the following formula where 256K represents the maximum memory requirement for each lock:

  commit size * 256K * number parallel INSERT statement / 4096

- A commit size of 1000 rows per database partition where five INSERT with subselect statements are issued against the same staging table in parallel would result in:

  1000 * 256 * 5 / 4096 = 156 pages

- Where the commit size increases to 100,000 rows per database partition the resulting number of pages increases greatly:

  100,000 * 256 * 5 / 4096 = 31,250 pages

For the second scenario, the value for LOCKLIST could be increased to 31,250 and the MAXLOCKS value increased to 25% to allow each of the five parallel INSERT statements to lock 20% of the table.

Making data available for query

The ingest cycle is completed when all query-facing objects have been modified in line with your service level objectives. Tasks to accomplish this include:

- Replicating tables
- Refreshing materialized query tables
- Maintaining statistics

Replicating tables in high-performance environments

Standard practice is to create the base table to be replicated on the administration module and then replicate that table across all data modules.

For high volumes of data or high numbers of database partitions, improve performance by balancing the network traffic evenly across the data modules using these steps:

1. Create the base table or MQT that you want to replicate as a partitioned table across all the database partitions.
2. Create the replicated table in a table space across all partitions using the REPLICATED clause of the CREATE TABLE statement.
3. Issue SET INTEGRITY.
4. REFRESH the replicated table.
5. Issue REFRESH and RUNSTATS for the replicated table.

The following example is a sample of the CREATE statement needed to create the replicated table based on the partitioned base table:

```
CREATE TABLE schema.replicatedtable AS ( SELECT * FROM
schema.dimensiontable ) DATA INITIALLY DEFERRED REFRESH DEFERRED
MAINTAINED BY SYSTEM COMPRESS YES NOT LOGGED INITIALLY REPLICATED IN
tablespace INDEX IN index_tablespace
```

Refreshing materialized query tables

Identify which materialized query tables are related to the tables targeted for inserting data. MQTs need to be refreshed to integrate the ingested data and make it available for query. Establish when data in the MQTs needs to be available to the business and incorporate the time needed to refresh the MQTs into your ingest rate and SLOs.

Incorporate MQTs, summary and replicated tables into the ingest cycle to control when the data is refreshed.

Employ the following practices using the REFRESH command to update the MQT:

- Set MQTs to REFRESH DEFERRED rather than REFRESH IMMEDIATE in a data warehouse environment. This setting avoids unnecessary refresh activity and allows control over refresh timing.
- Develop a separate application component to submit the REFRESH command when the data ingest processes has completed. Submit the minimum possible number of REFRESH processes.
- For high-performance environments, staging tables are recommended for large data volumes because they facilitate an incremental refresh of MQTs, applying only those changes made since the previous REFRESH. Issue the following SQL statements to create an incrementally refreshed MQT:

  1. CREATE TABLE <*mymqt*> AS (<*select stmt for MQT*>) DATA INITIALLY DEFERRED REFRESH DEFERRED
  2. CREATE TABLE <*mymq_staging*> FOR <*mymqt*> PROPAGATE IMMEDIATE
  3. SET INTEGRITY FOR <*mymqt*> MATERIALIZED QUERY IMMEDIATE UNCHECKED
  4. SET INTEGRITY FOR <*mymq_staging*> STAGING IMMEDIATE UNCHECKED
  5. REFRESH TABLE <*mymqt*>

Maintaining statistics

The DB2 optimizer uses statistics and metadata about tables to generate an access plan that results in fast, predictable query performance. The IBM Smart Analytics System is configured to gather statistics on tables by default through the **`AUTO_RUNSTATS`** database configuration parameter. Using automatic statistics collection performs the following functions:

- Collects statistics on catalog tables.
- Provides real-time statistics for declared global temporary tables (DGTT) and created global temporary tables (CGTT).
- Ensures that no tables are missed.
- Skips a table where the statistics on that table are up to date.

Follow these recommendations for the data ingest application where statistics are maintained manually:

- Define a statistics collection profile for each table. Use that profile when invoking the RUNSTATS command as part of the ingest process.
- Ensure that the manual statistics collection process is not scheduled to be executed during the ingest process.
- Collect statistics on a production table when a significant percentage of rows, for example 20%, have changed since the last statistics collection.
- Collect statistics on MQTs after they are refreshed, not before.

Monitoring, managing, and controlling the data ingest application

Continuous monitoring of the data ingest application helps ensure that data is being ingested in line with service level objectives. Capture and review metadata for each component of the data ingest application to see the elapsed time, errors, and execution retries for each ingest cycle. Use this information to determine trends in performance and identify components that might need to be improved.

Use the methods listed in this section to manage, control, and monitor the data ingest workload.

Monitoring workload performance

Monitor the following items:

- Load operations, number of rows processed, start and end time and success or failure.
- Insert operations, number of rows processed, start and end time and success or failure.
- Number of retries per file where a lock or other database error has occurred.

Optim Performance Manager (Optim PM) provides a graphical user interface to monitor, diagnose, and solve database performance problems and is available as part of the IBM Smart Analytics System solution stack. Optim PM is integrated with the DB2 workload manager (WLM) and can be used to monitor performance through the WLM configuration.

The following article discusses how to deploy IBM Optim Performance Manager in large-scale environment: http://www.ibm.com/developerworks/data/bestpractices/opmlargescale/index.html

Managing workload concurrency

Use DB2 Workload Manager (WLM) to manage the number of data ingests processes that can be submitted concurrently, so that if the number of allowed operations is exceeded, then they are queued in WLM.

The resources used by the INSERT statement itself cannot be throttled or managed, but the action can be managed through the associated user ID by implementing DB2 WLM. For AIX accounts, DB2 WLM can be aligned to the AIX WLM implementation.

Refer to the following paper for further information on DB2 Workload Management: http://www.ibm.com/developerworks/data/bestpractices/workloadmanagement/

Monitoring locking behavior

Monitor locking behavior by using the **`db2pd`** command, snapshots, and also IBM Optim Performance Manager when testing your data ingest application. Use the data output to determine if changes are required to the database configuration parameters.

Refer to “Further reading” for details about monitoring performance and optimizing queries in an IBM Smart Analytics System.

Controlling the data ingest application

Metadata refers to information captured by the data ingest application about the performance of each component.

Establish a performance baseline for each data ingest application component to determine how long each process should take to complete and if degradation in performance is occurring over a period or at certain times of the day, week, or month. Use this information to identify peak and off-peak windows on an ongoing basis. Compare baseline and actual performance metrics against SLOs to establish success or failure.

Engine-based ETL tools such as InfoSphere DataStage software have embedded metadata capabilities. These capabilities can be maintained and augmented on the application module.

See Appendix A for information on metadata capture in the test environments used for this paper.

Conclusion

Design your data ingest application to process data as soon as it is made available. The rate at which data is ingested should be controlled depending on the resources available and the ingest rate required.

Planning for the design and development of the ingest application should occur as early in the data warehouse implementation cycle as possible and take into account the other applications and workloads that will operate in the environment.

Consider the main recommendations in this paper when planning, designing, developing and operating your data ingest application in an IBM Smart Analytics System.

- Know when data is scheduled for ingest, the volume of data to be presented and the expectation for data availability.
- Understand the service level objectives for other workloads in your environment and how they affect the data ingest workload.
- Use the DB2 load utility to load data into staging tables, and then use the INSERT statement to move data from staging into production tables to minimize the effect on the query workload.
- Design components that process data in discrete steps as soon as it becomes available.
- Enable collocated queries to increase INSERT with subselect statement performance by aligning distribution keys on partitioned staging and production tables.
- Design tables and table spaces to minimize input and output operations and to maximize workload concurrency.
- Use the **`DATA BUFFER`** parameter of the **`LOAD`** command to control the amount of memory used when multiple load operations are submitted in parallel.
- Use indexes on staging tables only to improve INSERT with subselect performance or when needed to move data to the next phase of the ingest process.
- Monitor and prevent lock escalation by controlling the commit size in conjunction with the **`LOCKLIST`** and **`MAXLOCKS`** database configuration parameters.
- Incorporate MQTs, summary and replicated tables into the ingest cycle to control when the data is refreshed.

Appendix A. Configuration of test system

The two test systems used for this paper were:

- An IBM Smart Analytics System 5600 with one administration module, two data modules and a total of 17 database partitions.
- An IBM Smart Analytics System 7600 with one administration module, 58 data modules and a total of 465 database partitions. This system was used for high-performance testing.

The administration module on both systems contained a single database partition that supported the catalog function, the coordinator function, and the non-partitioned metadata, staging and data warehouse tables. Each data module had eight database partitions that contained partitioned staging and data warehouse tables.

The same database design was implemented on both systems to test and understand how the recommendations in this paper performed. Production tables were range partitioned by region and month to allow data to be rolled out on a monthly basis as part of a consolidated data archiving policy. Production MDC tables were organized by day. A standard naming convention for data partitions was in place as dictated by the database administrator and was used with the data ingest application. All tables in the database were compressed.

The distribution key used for both the staging and data warehouse tables was based on the native source key. Each range partition was stored in a separate table space to allow for greater concurrency and flexibility in performing attach, detach, and backup operations. The table spaces and other database objects were created in advance by a database administrator.

Staging area

The staging area tables matched the data warehouse tables. Unique indexes were created to assist in testing range-based inserts. The large dimension and fact staging tables were partitioned and used the same distribution keys as the data warehouse tables to support collocated queries for inserts. The staging tables were not range partitioned.

Five sets of staging tables, numbered 1 - 5, were created for each data source to allow for data from all five data sources used in testing to be processed in parallel. For example the first set of staging tables were named STG1_SUPPLIER, STG1_PART, STG1_ORDERS, STG1_CUSTOMER, STG1_LINEITEM, STG1_PARTSUPP.

Metadata area

The metadata area tables held control data and data captured from the component processes in the data ingest application.

The control data table held the following parameter values:

- A commit size (number of rows) to be used by INSERT with subselect statements during peak and off-peak times.
- The number of range-based INSERT with subselect statements to issue in parallel against a staging table.

- The status of each file (loaded, failed, inserted) and the timestamp when operations started and finished.
- The status of each component for each run and the elapsed time per task.
- The number of retries, if any, for each file.
- A parameter to hold the value for the number of load operations to be submitted in parallel in on-peak and off-peak times.

Further reading

Visit the best practices page on IBM developerWorks for papers on physical database design, data lifecycle management, database backup, compression, and many other topics.

http://www.ibm.com/developerworks/data/bestpractices/

IBM InfoSphere Warehouse Version 9.7

http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.isw.release.doc/c_release_whatsnew.html

http://www-01.ibm.com/software/data/infosphere/warehouse/enterprise.html

IBM InfoSphere DataStage software

http://www-01.ibm.com/software/data/infosphere/datastage/

Partitioning, MQT and MDC features

http://www.ibm.com/developerworks/data/library/techarticle/dm-0608mcinerney/

http://www.ibm.com/developerworks/data/library/dmmag/DMMag_2010_Issue4/DistributedDBA/index.html

http://www.ibm.com/developerworks/data/library/techarticle/dm-0509melnyk/

Query Optimization in an IBM Smart Analytics System

http://www.ibm.com/developerworks/data/bestpractices/smartanalytics/queryoptimization/index.html

DB2 load utility

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.admin.dm.doc/doc/c0004616.html

DB2 load exception tables

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.admin.dm.doc/doc/c0004596.html

Contributors

John Bell

Distinguished Engineer & Data Warehouse Architect, IBM

Paul McInerney

User-Centered Design Specialist, DB2 Development, IBM

Robin Grosman

DB2 Development Manager, Buffer Pool Services & Data Movement Utilities, IBM

Katherine Kurtz

Data Warehouse Best Practices Manager, IBM

Simon Woodcock

Consultant IT Specialist, InfoSphere Warehouse, IBM

Gaurav Mehtrotra

Integration Specialist, Infosphere Warehouse, IBM

Jacques Milman

Sales & Distribution, Data Warehouse Architecture, IBM

Maksym Petrenko

Integration Specialist, DB2 Warehouse, IBM

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Without limiting the above disclaimers, IBM provides no representations or warranties regarding the accuracy, reliability or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any recommendations or techniques herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Anyone attempting to adapt these techniques to their own environment do so at their own risk.

This document and the information contained herein may be used solely in connection with the IBM products discussed in this document.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks