IBM® PureData® System for Operational Analytics®

Best practices

Building a data migration strategy with IBM InfoSphere Optim High Performance Unload

Konrad Emanowicz
DB2 Data Warehouse QA Specialist
IBM Ireland Lab

Garrett Fitzsimons
Best Practices Specialist for Warehouses
IBM Ireland Lab

Richard Lubell
DB2 Information Development
IBM Ireland Lab

Issued: December 2012

Executive Summary

This paper is targeted at persons that are involved in planning, configuring, designing, implementing, or administering a data warehouse that is based on DB2® Database for Linux®, UNIX, and Windows® software. The examples in this paper apply generally but are focused on the IBM ® PureData System for Operational Analytics.

IBM InfoSphere Optim High Performance Unload for DB2 for Linux, UNIX, and Windows V4.02 is a high-speed tool for unloading, extracting, and migrating data in DB2 for Linux, UNIX, and Windows databases. High Performance Unload (HPU) is designed to extract and migrate data from DB2 table space containers. A data migration strategy that uses HPU minimizes storage needs and automates many manual tasks.

HPU uses named-pipes and parallel LOAD operations to stream data from the source to the target database, minimizing the need to stage the data on disk. You can direct HPU to determine where different partition maps, software levels, and key constraints are used and automatically handle these events during data migration

HPU can also unload subsets of data from target database without the need to access the DB2 software layer. With this functionality you can migrate data online from larger production systems to smaller pre-production or development environments.

HPU has other uses in data extraction from DB2 backup images which are covered in the companion paper “Best Practices Using IBM Optim High Performance Unload as part of a Recovery Strategy in an IBM Smart Analytics System”

Introduction

This paper describes best practices for incorporating the use of HPU into your migration strategy and implementing HPU for an IBM PureData System for Operational Analytics data warehouse. This paper covers how to migrate different sets of data between two databases.

To use this paper, you should have a basic knowledge of HPU software as well as DB2 software as implemented in a partitioned data warehouse environment. The further reading section in this paper contains links to product documentation and papers that are referenced in the paper.

The first and second sections outline the benefits of using HPU and the possible considerations of integrating it into your system.

The third section reviews the HPU Migration Process, HPU control files, HPU parameters and recommends best practices for creating control files.

The fourth section provides details on how to install and configure HPU for migration in an IBM PureData System for Operational Analytics.

The fifth section offers specific best practice recommendations for different scenarios.

The appendix section outlines the configuration of the test system that was used in developing this paper.

This paper builds on and complements a series of best practices papers that discuss aspects of a DB2 data warehouse. Refer to these papers in "Further Reading" for more information.

Incorporating HPU into your data migration strategy

HPU is a high-speed stand-alone utility for migrating data between DB2 databases on Linux, UNIX, and Windows systems. HPU unloads large volumes of data quickly by reading data directly from table space containers or backup images rather than through the DB2 software layer. HPU accelerates the migration of data through parallel processing between all source and target servers without the need to stage data to disk.

The key benefits that you gain by implementing a data migration strategy that uses HPU are the abilities to:

- Migrate all or a subset of a database from source to target database

  For example, use a `WHERE` clause to select a range of data to migrate or explicitly specify a data partition, table space or database partition.

- Migrate data between databases with different topologies, software levels, and partition maps

  For example, use HPU control file parameters to control the migration of a subset of data from a large production environment to a smaller development with fewer database partitions, a different partition map and a different DB2 software level.

- Eliminate the need to stage the source data on storage before the data is loaded to the target database

  For example, named pipes are used to stream data from the source to target environment, then the LOAD command is automatically invoked to load data into the target database.

- Control effects on system resources on the source database

  For example, you can configure HPU to restrict the processor and memory resources that are used on the source system. The LOAD utility can be optimized or throttled on the target database. You can also choose to migrate data offline from a backup image or online from table space containers.

- Automate the migration process so that it can be scheduled to run unattended

  For example, because HPU is command line-driven and uses control files, you can schedule data subsets to be migrated automatically during off-peak periods or after ETL or backup operations are complete.

Avoid contention with ETL processes by scheduling automated data migration outside of ETL, backup and other data operations.

Existing migration strategies involve laborious procedures such as backup and restore operations, data redistribution, export and load commands, and moving source data over the network to the target systems. Using HPU helps simplify the migration process by avoiding manual steps, skipping the staging phase and speeding migration through direct transfer to target tables by using named pipes.

Implementing HPU

When you use HPU to migrate data from source to target database there are installation and configuration considerations that need to be addressed. An implementation of HPU should minimize the number of configuration points and avoid unintentional use of resources. The following sections cover how to install, configure and set up your environment for HPU.

Installing HPU

HPU must be installed on each host of the source database where you intend to unload data and on each host of the target database where the data is loaded. The same version of HPU must be used across all source and target data nodes. Use the `db2hpu --version` command to determine the version installed on each data node.

Make the HPU installation package available to each individual host by placing it on the shared `/db2home` file system. This action avoids the task of copying the installation package to each host.

Migrating a consistent transaction set

Where you migrate data from an online database or from an online backup image, the result set migrated could contain an inconsistent record of data, because HPU does not access transaction logs. This might be acceptable for your needs in a test or development environment.

Where you need to migrate a consistent data transaction set:

- Use the HPU control file parameters to quiesce the table from which you want to migrate data.

  This action flushes all data for the table space in the buffer pool to the table space container on disk and locks the table during the unloading operation.

- Schedule data migration activities outside of ETL windows to ensure a consistent result set.

  Avoid data migration that requires a read-access lock on the table space when your ETL tasks or online backup operations are scheduled to run. In this case, read-access workloads would be supported but no changes to the table space would be allowed. ETL processes are unable to write data while the HPU data migration is performed.

Unload data from offline backup images or online backup images created outside of ETL operations to help ensure data consistency.

Allocating resources for HPU

HPU uses all available processor resources on the source system. For example, on an IBM PureData System for Operational Analytics there are sixteen cores. Unthrottled, HPU starts a thread for each core and unloads data in parallel using sixteen threads.

You can throttle resources on each data node by dedicating a number of processor cores to the HPU migration process. You can also throttle the resources that are used on the target system by throttling the DB2 load utility responsible for loading the migrated data.

When HPU migration is performed by using named pipes, there are some cases when staging storage can be required. When XML or Large Object (LOB) columns are used, the entire table must be staged to flat files before they are ingested into the target system. When migrating data from database backup files you must have sufficient storage capacity in the staging area for data unloaded from the backup files.

Understanding HPU

The HPU migration consists of unloading, repartitioning, and loading data between a source and target database. By understanding how HPU operates you can best configure your environment and implement your data migration strategy. Figure 1 shows the data migration process between a source and target database with a different topology:

- The source database contains data nodes DataNode 1, DataNode 2, and DataNode 3 and has a total of 28 database partitions plus a coordinator node.
- The target database contains data nodes DataNode 1 and DataNode 2 and has a total of eight database partitions plus a coordinator node.

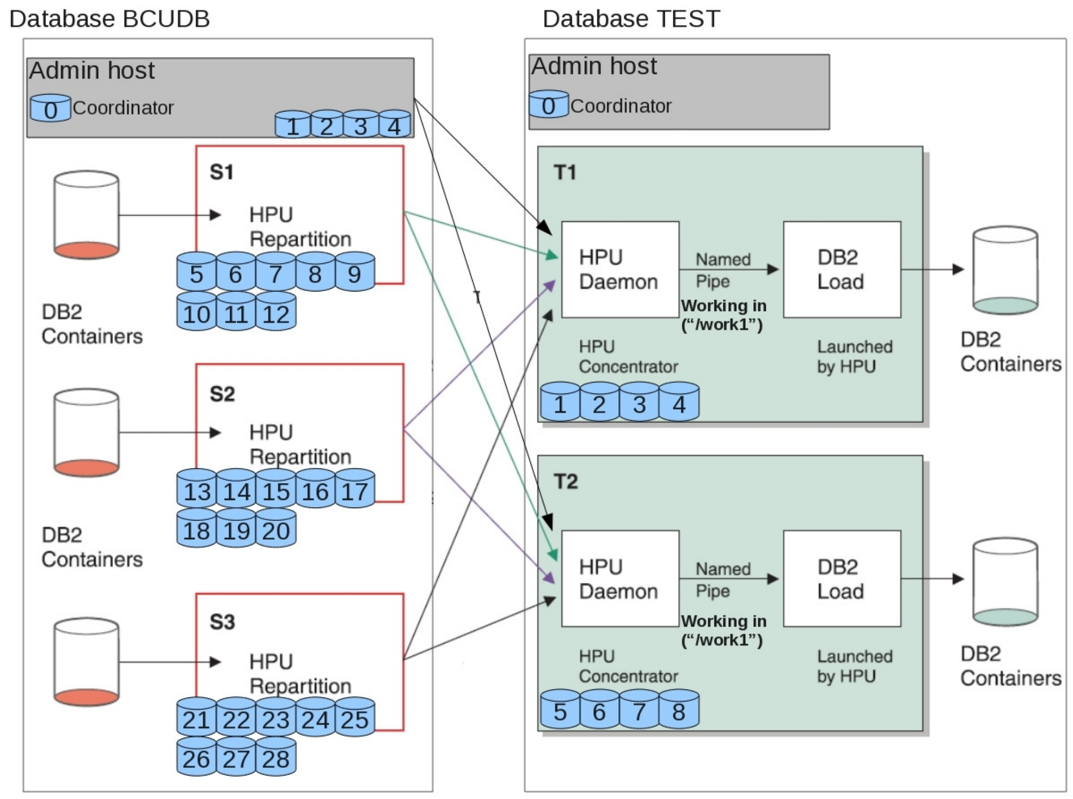

Figure 1 Data migration process using HPU

The HPU process represented by figure 1 consists of the following tasks:

1. HPU unloads data on each database partition in parallel on the source database.
2. HPU repartitions data according to the DB2 distribution map created by HPU based on the target table details that are provided in the control file.

3. HPU sends the output streams from the source database across the network to the HPU daemon on the target database (DataNode 1 and DataNode 2 in figure 1).

   The target daemon creates a single named pipe for each database partition on the target system, in figure 1 it is represented as the `/work1` directory, and initiates the DB2 LOAD command.

4. The HPU daemon consolidates the multiple streams from each of the source database partitions into a single stream, which HPU sends to each named pipes associated with a target partition.

5. The HPU daemon calls the LOAD command to load data into the target database.

The process for migrating from a backup file is slightly different than presented in figure 1. Data must first be staged from database backup files on the source system before it is migrated through named pipes to the target system. Dedicate the same storage path on each source data node for the staging process. Allocate storage capacity for staging equivalent to the table space size of the table that is being migrated.

Using the HPU control file

HPU migration is controlled and operated through a control file. The structure of the control file consists of two control blocks: the GLOBAL block and MIGRATE block. By understanding the control file syntax you can help ensure the correct migration sequence.

GLOBAL Block

The GLOBAL control block contains configuration data that is common to all migrate blocks in the HPU control file. There is only one GLOBAL block per control file and the GLOBAL block must be the first block in the control file. Any parameters in this block override its equivalent configuration file defaults.

The GLOBAL block designates the control settings that are used by default for each MIGRATE block unless an option is overridden at a lower level. For example, the QUIESCE and LOCK options control table space locking and buffer pool state but you can override these options in each MIGRATE block.

The following example shows the GLOBAL block syntax:

```
-- Global Block
GLOBAL CONNECT TO BCUDB
DB2 NO QUIESCE NO LOCK NO;
```

MIGRATE Blocks

The MIGRATE block specifies the table space, tables, and SELECT statement for the data that is being migrated. You can specify multiple MIGRATE blocks in a single control file to determine a sequence of migration tasks.

The key syntax elements that are used in a MIGRATE block are presented in table 1 below.

Parameter	Description	Usage
PART	Specifies database partitions for source tables.	Use this option to migrate tables from a subset of source partitions.
FAST_SELECT	Specifies details for the source table	Use this block to specify a select statement for the source table and a specific range for ranged partitioned table.
TARGET ENVIRONMENT	Specifies the target system details	Use this option to specify the target instance name, target database name, and the server where the database is cataloged and where the DB2 LOAD command is called by HPU.
LOCK OPTION	Specifies locking on the source database	Use this option to indicate whether a read-only lock is to be held during unloading on the table space of the source table.
QUIESCE OPTION	Enables quiescing table spaces on the source database	Use the default option (YES) to make the related buffer pool pages flush to disk before the start of unload.
TARGET KEYS	Specifies details for the target table	Use this clause to specify the partitioning key details and the sequence of database partition numbers on the target table.
WORKING IN	Specifies the HPU processing path	Use this option to specify the location for pipes/files on the target data nodes.
XML/LOB IN	Specifies staging details on the target system for the LOB/XML columns data of the table that is being migrated	Use this parameter to specify the staging paths and file names for the LOB/XML columns data on the target system.
FORMAT	Specifies the output details	Use this clause to select the output format of the data and specify the target table name.

Table 1. Key HPU data migration parameters when using the MIGRATE block

Use the SELECT clause with the PART clause to migrate a subset of data when the target environment has less storage available.

Target keys

The TARGET KEYS clause is the primary syntax element that is employed during data migration. Information that is provided by the clause is used by HPU to generate the DB2 distribution map that is used to distribute and load data into the target table.

The TARGET KEYS clause consists of two sections:

Database partitions

This section specifies the sequence of the database partition numbers. For example, PARTS(1:10) specifies that data is migrated for database partitions 1 and 10.

Partitioning key details

This section specifies the columns details for DB2 partitioning key of the target table. There are three ways to specify the partitioning key information:

- The CURRENT keyword keeps the current definition of the partitioning key. It sets the partitioning key for the target table to the same value or values as identified in the FAST_SELECT block in the source table. For example:

```
TARGET KEYS(CURRENT PARTS(1:M))
```

- The DEFAULT keyword indicates that the first valid column in the FAST_SELECT block is used in the partitioning key. For example:

```
TARGET KEYS(DEFAULT PARTS(1:M))
```

- The explicit column list provides either column names or column numbers. For example:

```
TARGET KEYS((3,4) parts(1:10))
TARGET KEYS((col1,col3) parts(1:10))
```

Use the HPU options for target database partitions and distribution key to help ensure that data is re-partitioned correctly when you are migrating data to a database with a different partition map.

Range partitioned tables

When migrating data from range partitioned tables, HPU can process multiple ranges in parallel based on the number of processor cores and database partitions available on the data node. To determine how many ranges can be processed in parallel, use the following calculation:

`INTEGER`(number of processor cores/ number of partitions).

Resource usage, especially memory, can be high for parallel migration of many ranges and can consume more resources than intended on the source database system.
To control the resources available to HPU do not migrate whole partitioned table in one step. Use the `nbcpu` HPU configuration parameter to limit the number of processor cores available to HPU.

Migrate range partitioned tables as a sequence of ranges within a single control file; choose the most recent range first.

Using the recommended control file parameters

When you create a control file, it is recommended to:

- Use FORMAT DELIMITED INTO clause when the source and target table name are different.
- Use QUIESCE YES and LOCK YES options to help ensure a safe and consistent unload where no modifications to the table are allowed until the unload process is complete. The QUIESCE YES option flushes all pages of the source table from the DB2 buffer pool. The LOCK YES places a share lock on the table and to prevent the table from being modified.
- Use the CURRENT keywords in the TARGET KEYS clause when the partitioning key for the target and source tables are the same.
- Use CURRENT PARTS(ALL) only when target tables exist on all database partitions of the target instance.
- To exclude specific partitions from the full list of partitions of the target instance use EXCEPT PARTS() clause in TARGET ENVIRONMENT clause together with CURRENT PARTS(ALL).
- Use a location other than the default `/tmp` path in the WORKING IN and XML/LOB IN clauses to ensure that appropriate disk space is available when migrating tables with XML or LOB columns. Tables with XML or LOB columns have to be staged to files temporarily and cannot be processed through pipes.
- Use separate TARGET KEYS clause for each target table in its FAST_SELECT block to migrate tables in different database partition groups.

The following example shows a control file migrate block that is used to migrate data from source_tabschema.source_tabname table on database partitions 1:N to target_tabschema.target_tabname table on partitions 1:M. The target database name is target_db_name and exists on target_instance_name instance. The database is cataloged on target_hostname.

```
-- Migrate block migrating from nodes 1:N to 1:M
MIGRATE TABLESPACE
PART(1:N)
QUIESCE YES LOCK YES DB2 NO
-- Select statement and target environment details
SELECT * FROM source_tabschema.source_tabname;
TARGET     ENVIRONMENT(INSTANCE     "target_instance_name"     on
"target_hostname" IN target_db_name)
TARGET KEYS(CURRENT PARTS(1:M))
WORKING IN("/path")
FORMAT DELIMITED INTO  target_tabschema.target_tabname;
```

Controlling resources with HPU

HPU is designed to use all available resources to unload and migrate data at high speeds. Consider the limitations on storage, processor, and memory resources you established during the planning phase.

Storage Capacity

When you migrate data from backup images, storage capacity is needed on the source data nodes for staging data from the backup files. The storage capacity that is allocated per data node for a migrated table must be the total size of the table space on the data node.

For tables with LOB or XML columns, named pipes cannot be used and the data must be staged to disk first before the load phase. Ensure that there is also sufficient storage capacity on the target data nodes by determining the size of the table, including the LOB and XML files.

When you are unloading a table from table space containers to files, the total storage capacity that is required equals the table size.

Processor resources

HPU uses all processor cores and each partition is always processed by at least one core.

- When there are fewer cores than partitions, the number of partitions that are processed in parallel equals the number of cores.
- When there are more cores than partitions, all partitions are processed in parallel by more than one core.
- The number of cores that are processing each thread is calculated as `INTEGER(`Number of cores / Number of Partitions).

Use the `nbcpu` configuration parameter on the source system to set the maximum number of processor cores used by HPU when migrating data.

In a scenario that involves a database outage, it is recommended to allow HPU to use all available processor cores to unload data as quickly as possible. In an environment where queries are running during migration, configure HPU to restrict the resources that are consumed, reducing the amount of processor capacity that is used.

For example, for a data node on an IBM PureData System for Operational Analytics with sixteen cores and eight database partitions, and with 50% of processor cores dedicated to HPU by setting `nbcpu`=8, all eight database partitions are processed in parallel with each processed by one core as INTEGER(Number of cores/Number of Partitions)=INTEGER(4/4)=INTEGER(1)=1. With the default settings all sixteen cores would be used with each partition processed by two cores.

For data node with 4 partitions and 18 processor cores and with `nbcpu` set to 10, each partition is processed by two cores as INTEGER(Number of cores/Number of Partitions)=INTEGER(10/4)=INTEGER(2.5)=2.

Memory resources

Use the `bufsize` parameter to define the HPU buffer. It is recommended to use the default HPU buffer pool size in most cases. The minimum accepted value is 262144 (256 kilobytes), the maximum and default accepted value is 4 MB. Use the minimum value when you are migrating from many source nodes.

Influence HPU usage of processor resources on the source system by specifying the number of cores to be used on each data node.

User process resource limits

For migration scenarios that involve many source database partitions, change the default settings for user process resource limits to unlimited. The key limits on the target system on each data node are: `data seg size` and `stack size`. You can check and control these settings with the `ulimit` command in the user session. These are called soft settings because you can configure them up to the maximum defined by the hard limits as set by the root user.

The following example shows a command that is used to check the current soft settings on AIX system:

```
ulimit -a
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) soft
file size               (blocks, -f) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 2000
pipe size            (512 bytes, -p) 64
stack size              (kbytes, -s) 65536
cpu time               (seconds, -t) unlimited
max user processes              (-u) 4096
virtual memory          (kbytes, -v) unlimited
```

To change the `data seg size` settings to unlimited, use the following command:

```
ulimit -d unlimited
```

Monitoring HPU migration

HPU can consume all available memory and processor resources to achieve maximum throughput of data, which can sometimes cause contention and paging. Contention with concurrent processes that have higher priority for processor capacity can slow or even stop the HPU migration process. Migration can also terminate abnormally when all paging space is fully utilized. It is recommended that you observe resource usage to help ensure successful migration.

Monitor system resources to:

- Determine the resources used by HPU to prevent contention

Determine whether processor resources available to HPU are limited because of contention with other processes by analyzing processor usage in the system.

Use the `TOP` command on Linux and AIX or TOPAS on AIX to check how much processor and physical memory is consumed. The amount of processor and memory consumed is expressed as a percentage of total resources available on the server.

1. Identify the HPU process using the "db2hpu" keyword in the COMMAND column for the `TOP` command and in the Name column for the `TOPAS` command.

2. Verify that the processor resources used by HPU are expected for the HPU `nbcpu` parameter settings.

For example, when the `nbcpu` parameter is set to half the number of cores available on the server, processor usage for a "db2hpu" process should not exceed 50%. The following sample TOPAS output shows contention between db2hpum6 process and java process:

```
    Tue Dec 18 16:16:50 2012   Interval:  2        Cswitch    5984  Readch     4864
                                                   Syscall   15776  Writech   34280
    Kernel   63.1   |##################        |  Reads         8  Rawin         0
    User     36.8   |##########                |  Writes     2469  Ttyout        0
    Wait      0.0   |                          |  Forks         0  Igets         0
    Idle      0.0   |                          |  Execs         0  Namei         4
                                                   Runqueue   11.5  Dirblk        0
    Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue   0.0
    lo0     213.9   2154.2  2153.7   107.0   106.9
    tr0      34.7     16.9    34.4     0.9    33.8  PAGING           MEMORY
                                                   Faults     3862  Real,MB    1023
    Disk    Busy%     KBPS     TPS KB-Read KB-Writ  Steals     1580  % Comp     27.0
    hdisk0    0.0      0.0     0.0     0.0     0.0  PgspIn        0  % Noncomp  73.9
                                                   PgspOut       0  % Client    0.5
  Name          PID    CPU% PgSp Owner              PageIn        0
 db2hpum6      16684   83.6 13.1 root               PageOut       0   PAGING SPACE
    java       12192   12.7 12.2 root               Sios          0   Size,MB     512
    lrud        1032   2.7  0.0 root                                  % Used      1.2
```

```
      aixterm    19502  0.5  0.7 root                NFS (calls/sec)  % Free     98.7
      topas       6908  0.5  0.8 root                ServerV2       0
      ksh        18148  0.0  0.7 root                ClientV2       0   Press:
      gil         1806  0.0  0.0 root                ServerV3       0   "h" for help
```

The db2hpum6 and java processes are together consuming 100% of available processor resources, causing contention between them. It is recommended to stop the java process or limit the processor resources available by HPU by adjusting the HPU `nbcpu` configuration parameter to lower the number of processor cores available to HPU.

Where you need the capability to manipulate HPU during processing, you should use OS/WLM capabilities and LOAD parameters within DB2 on the target database.

- Check for paging

A shortage of physical memory can cause paging on the source system that can in some cases trigger HPU migration failure. Use the `vmstat` command to view the `pi` and `po` (page in and page out) columns on AIX, and the `si` and `so` (swap in and swap out) columns on Linux. Non-zero values indicate that paging is occurring. The following example shows sample output for the `vmstat` command:

```
kthr     memory             page              faults        cpu        time
  ----- ----------- ------------------------ ------------ ----------- --------
   r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa hr mi se
   0  0 45483   221   0   0   0   0    1   0 224  326 362 24  7 69  0 15:10:22
   0  0 45483   220   0   0   0   0    0   0 159   83  53  1  1 98  0 15:10:23
   2  0 45483   220   0   0   0   0    0   0 145  115  46  0  9 90  1 15:10:24
```

TO check whether the HPU process "db2hpum6" is triggering the paging process, refer to the "Paging" Column in the `TOP` command output or "PgSp" Column in the `TOPAS` command output.

- Track the progress of the migration operation

Use the `db2 list utilities show detail` command on the target system to check the progress of the migration. The command shows the progress for the current `DB2 LOAD` command that was started by HPU.The following example shows sample output for the `db2 utilities` command:

```
ID                                 = 186
Type                               = LOAD
Database Name                      = TEST
Member Number                      = 0
Description                              = [LOADID: 1891.2012-12-18-
11.48.27.128866.0    (65530;32768)]    [*LOCAL.bcuaix.121218114833]
OFFLINE   LOAD   DEL   AUTOMATIC   INDEXING   INSERT   NON-RECOVERABLE
TEST.TB_SALES_FACT
Start Time                         = 12/18/2012 11:48:27.158263
State                              = Executing
Invocation Type                    = User
Progress Monitoring:
   Phase Number                    = 1
      Description                  = SETUP
      Total Work                   = 0 bytes
      Completed Work               = 0 bytes
      Start Time                   = 12/18/2012 11:48:27.158270

   Phase Number [Current]          = 2
      Description                  = LOAD
      Total Work                   = 100000 rows
      Completed Work               = 220 rows
      Start Time                   = 12/18/2012 11:48:29.168612

   Phase Number                    = 3
      Description                  = BUILD
      Total Work                   = 12 indexes
      Completed Work               = 0 indexes
      Start  Time                                      =  Not  Started
```

Configuring HPU in an IBM PureData System for Operational Analytics

You must create a directory structure for migration to ensure the required storage capacity is available. Additional configuration on the target system is required whenever the target instance name is different from the source instance name. Use a shared HPU configuration file to minimize the administration of HPU configuration files.

Creating a directory structure for data migration

HPU requires a directory structure where named pipe files and staging data can be located. This directory structure accommodates:

- When backup images are used and data needs to be unloaded and staged on the source database system
- LOB and XML data that needs to be staged on the target database system before it is loaded

HPU uses a single absolute directory path for each data host. On the target database system, data can be loaded in parallel from multiple file system paths across each database partition.

On an IBM PureData System for Operational Analytics, use the `/bkpfs` (backup and cold storage) file system for HPU staging areas and for creating a working directory for named pipes and LOB and XML flat files.

On IBM PureData System for Operational Analytics, create a directory for HPU in the `/bkpfs` file system on each user host and each data host. For example, for pipes, LOB and XML files or backup staging, create `/work1` directory link on each host. For example, the following commands show how to create the required link:

```
mkdir /bkpfs/bcuaix/NODE0001/HPU
ln /bkpfs/bcuaix/NODE0001/HPU /work1
```

For backup files staging, the `stagedir` HPU configuration parameter in the `db2hpu.cfg` file should be set to your directory.

Use a directory structure on storage that does not conflict with storage used for table space containers when you are staging or streaming data.

Creating and sharing a single configuration file across all nodes

Minimize administration of configuration files by creating a shared configuration file on the administration node and modifying the local configuration file on each node to reference it. To create a single configuration file that is shared across all nodes on which HPU is installed:

1. Make a copy of the default configuration file, customize it as required and save it on a file system that is accessible across all nodes; `/db2home` is recommended.
2. Add the `dir_cfg` parameter to the default HPU configuration file on each data node to reference the shared HPU configuration file.
3. Make further customizations to the referenced configuration file only.

Target system settings

When the user name used for the target instance is different than the user name used on the source instance, create the source instance user on each of the target data nodes. You must also grant the user the appropriate DB2 privileges, including LOAD and INSERT privileges on all target tables. For example, the relevant privileges for user bcuaix for table BI_SCHEMA.TB_SALES_FACT would be as follows:

```
db2 grant load on database to bcuaix
db2 grant insert on table  BI_SCHEMA.TB_SALES_FACT to bcuaix
```

Ensure a user with the name used by source instance exists on the target system and has the appropriate database privileges to load and insert into the target tables.

Data migration scenarios

Each of the following migration scenarios was performed by the authors to validate the accompanying recommendations. Refer to Appendix A for details of the test systems that were used.

Migrating data between databases with different topologies

HPU can migrate data between two databases with different topologies by interpreting the source and target partition maps and re-partitioning data on the target database when necessary. Figure 2 shows migration between a BCUDB database and a TEST database with different database topologies.

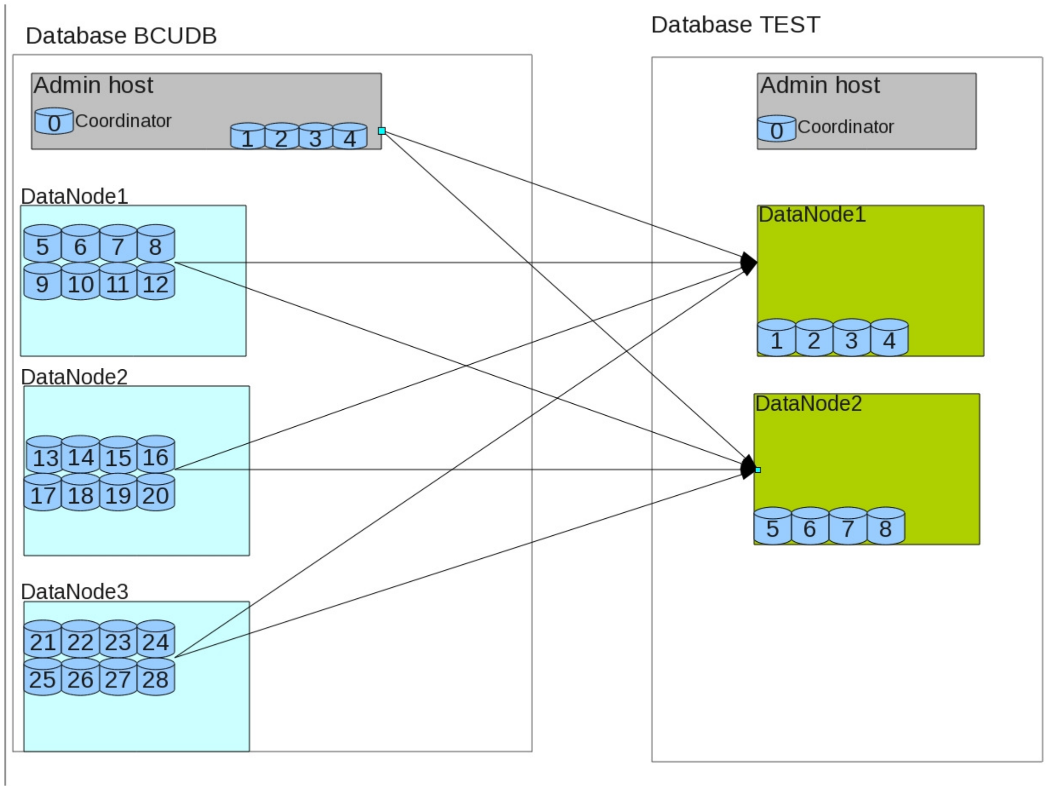

Figure 2 Data migration process between different database topologies

The database topologies are different since each of them has a different number of data nodes and database partitions. The source database has three data nodes and 29 database partitions and the target database has two data nodes and nine database partitions.

The key considerations for data migration between databases with different topologies are:

- Partitioning key details for each target table

  The sequence of database partitions must be explicitly specified in each MIGRATE block unless the target table exists in IBMDEFAULTGROUP database partition group, the default pattern that is used by HPU. If the target tables exist in different database partition groups, you must use either a separate TARGET KEYS clause in the FAST_SELECT block for each table or two separate migrate blocks.

- Table space migration

  Whole table space migration can be performed only when all the target tables use the same DB2 distribution map (target tables exist in the same table space or in the same database partition group) and target table names are the same as source table names. For other cases process each table in a separate migrate block to prevent HPU migration failure.

Perform table space migration only when all target tables exist in the same DB2 database partition group and all target tables names match the source table names.

- Different HPU migration options

  Use a separate migrate block per table when different LOCK and QUIESCE options are required for different tables.

Use a separate migrate block for each table when different LOCK and QUIESCE options are required for the tables.

For example, to migrate data for three tables BI_SCHEMA.TB_SALES_FACT fact table and BI_SCHEMA.TB_STORE_DIM, BI_SCHEMA.TB_CUSTOMER_DI dimension tables from a source system with three data nodes each with eight database partitions to another database with two data nodes each with four database partitions the following control file was used:

```
GLOBAL CONNECT TO BCUDB;
-- Migrate Block for Fact table
MIGRATE TABLESPACE
PART(1:28)
DB2 NO LOCK YES QUIESCE YES
TARGET ENVIRONMENT(INSTANCE "bcuaix" on "bluejay06" IN
TEST )
WORKING IN("/work1")
SELECT * FROM BI_SCHEMA.TB_SALES_FACT;
TARGET KEYS((3,4) parts(1:8))
FORMAT DELIMITED INTO TEST.TB_SALES_FACT
;
-- Migrate whole TBS_DIM table space
MIGRATE TABLESPACE TBS_DIM
PART(ALL)
DB2 NO  LOCK YES QUIESCE YES
```

```
TARGET ENVIRONMENT(INSTANCE "bcuaix" on "bluejay06" IN
TEST )
TARGET KEYS(current  parts(0))
WORKING IN("/work1")
FORMAT DELIMITED
;
```

The PART(ALL) parameter prompts HPU to unload data from all source partitions for each table. Because dimension tables are processed as a whole, a separate MIGRATE block must be used for fact and dimension tables. The partitioning key for the target fact table is different (consists of different columns) than the key on the source table therefore the TARGET KEYS specifies the appropriate column numbers, columns 3 and 4. This information can be specified only in the FAST_SELECT block and not in the TARGET ENVIRONMENT block.

The two source dimension tables exist in the same table space. Their target table names also exist in one table space and have the same names as the source tables. If all conditions are met, migrate both dimension tables in one MIGRATE TABLESPACE block without specifying the SELECT statement for each of the tables. The TBS_DIM table space name must be specified explicitly in the MIGRATE block. The dimension tables do not have to process first since the foreign key from the fact table is not enforced and the sequence is not important.

The LOCK YES QUIESCE YES for the tables ensures that the related buffer pool pages are flushed to disk before the start of unload, and no modifications to the table are allowed until unloading is complete.

Migrating a data subset

HPU migration can migrate different subsets of data from the source database. For example, a data subset might consist of fact table rows for a specific date range and a group of commonly used dimension tables.

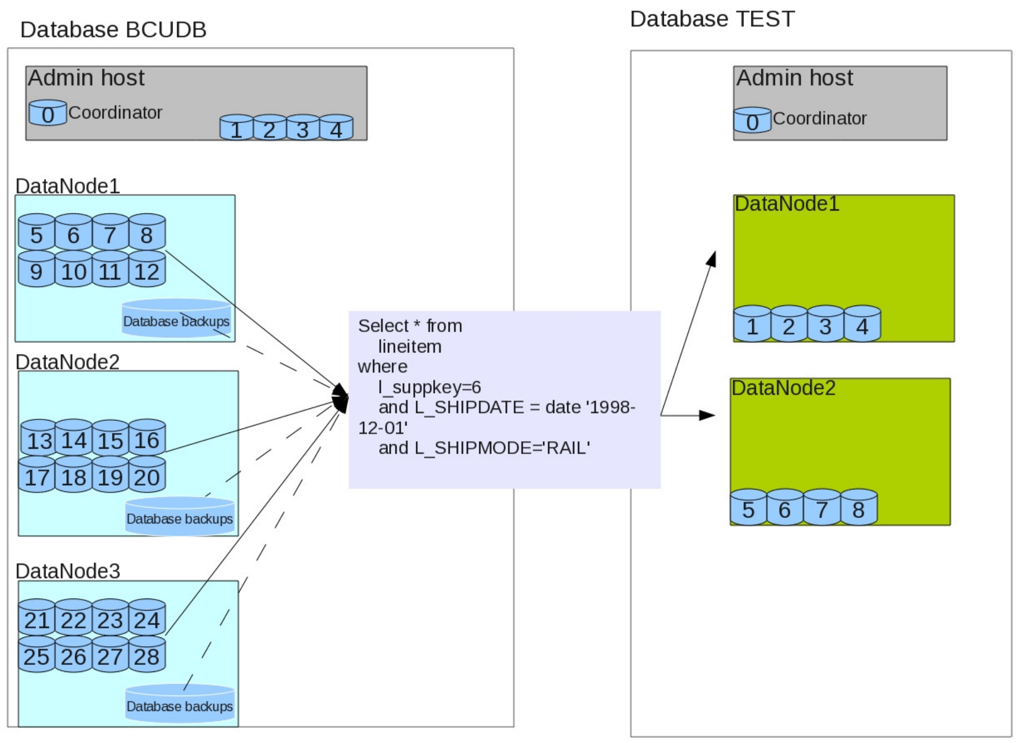

Figure 3 Data migration of data subsets

Figure 3 shows a subset of table data that can be migrated with HPU from either table space containers or backup images. When using backup images, data is extracted directly from the backup files; access to the source database is not required. The backup files can also exist on a different system than the production database to completely minimize the effect on the production.

HPU can facilitate a data lifecycle management strategy. Where data has been detached from the database, HPU can be used to retrieve data from a backup image associated with archived data. This process allows data to be retrieved for auditing or data governance reasons.

Use HPU migration to migrate different subsets of data from backup files when past data is required and the access to the production database is not allowed or restricted, for example, during ETL and online periods.

HPU migration offers several ways to specify the subset of data for migration:

- Restrict use of the WHERE clause to filter the data on different columns in the Select clause for each table
- Specify a range for range partitioned tables with DATAPARTITION ID (NAME) keys
- Use DB2 temporal tables to access data from a point in time in the past
- Use the PART clause to specify specific source database partitions

For example, to migrate data extracts from range partitioned fact table the following control file could be used:

```
GLOBAL CONNECT TO BCUDB;
-- Migrate block for the fact table
MIGRATE TABLESPACE
PART(1:28)
-- Buffer pool is flushed and write access to the table prevented
during the Migration

DB2 NO LOCK YES QUIESCE YES
TARGET ENVIRONMENT(INSTANCE "bcuaix" on "bluejay06" IN
TEST )
WORKING IN("/work1")
-- Select statement and target keys
SELECT * FROM  BI_SCHEMA.TB_SALES_FACT where STORE_ID between 101
and 250
;
DATAPARTITION ID (2)
TARGET KEYS(current parts(1:8))
FORMAT DELIMITED INTO TEST.TB_SALES_FACT
;
```

This scenario demonstrates migrating a subset of fact table BI_SCHEMA.TB_SALES_FACT. The requested data set has rows where value for STORE_ID column is 101 - 250 and which exist only in data partition 2.

The following example demonstrates how to migrate the temporal dimension table BI_SCHEMA.TB_PRODUCT_DIM data from the past time between '2012-06-18' and '2012-06-19':

```
GLOBAL CONNECT TO BCUDB;
-- Migrate Block for Dimension table
MIGRATE TABLESPACE
PART(0)
DB2 NO LOCK YES QUIESCE YES
TARGET ENVIRONMENT(INSTANCE "bcuaix" on "bluejay06" IN
TEST )
WORKING IN("/work1")
-- Select statement and target keys
```

```
select * from BI_SCHEMA.TB_PRODUCT_DIM FOR SYSTEM_TIME between
'2012-06-18' and '2012-06-19' where PRODUCT_DESCRIPTION like
'%fzc%';
TARGET KEYS(current parts(0))
FORMAT DELIMITED INTO TEST.TB_PRODUCT_DIM MODIFIED BY
identityoverride
;
```

In the following example the rows from BI_SCHEMA.TB_PRODUCT_DIM table with specific production description and with values from the past period between '2012-06-18' and '2012-06-19' were migrated. The history table for the BI_SCHEMA.TB_PRODUCT_DIM table was used to retrieve the data from the past.

Instruct HPU to use the "MODIFIED BY IDENTITYOVERRIDE"option for the target dimension table TEST.TB_PRODUCT_DIM. This modifier is used since an identity column defined as GENERATED ALWAYS is present on the target table. Accept explicit, non-NULL data for such a column to prevent the DB2 engine from generating new values.

Preserve the values from the source database for target tables with identity columns defined.

The commands which were used to enable the system-period temporal table feature for table BI_SCHEMA.TB_PRODUCT_DIM were as follows:

```
ALTER TABLE BI_SCHEMA.TB_PRODUCT_DIM ADD COLUMN sys_start
TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN;
ALTER TABLE BI_SCHEMA.TB_PRODUCT_DIM ADD COLUMN sys_end
TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END;

ALTER TABLE BI_SCHEMA.TB_PRODUCT_DIM ADD COLUMN ts_id
TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID;
ALTER TABLE BI_SCHEMA.TB_PRODUCT_DIM ADD PERIOD
SYSTEM_TIME(sys_start, sys_end);

CREATE TABLE BI_SCHEMA.TB_PRODUCT_DIM_hist LIKE
BI_SCHEMA.TB_PRODUCT_DIM IN TS_DIMENSIONS;

ALTER TABLE BI_SCHEMA.TB_PRODUCT_DIM ADD VERSIONING USE HISTORY
TABLE BI_SCHEMA.TB_PRODUCT_DIM_hist;
```

Migrating data between databases with different versions of DB2 and different distribution maps

Differences in distribution maps can necessitate "single stream" migration. Figure 4 shows how "single stream" migration works.

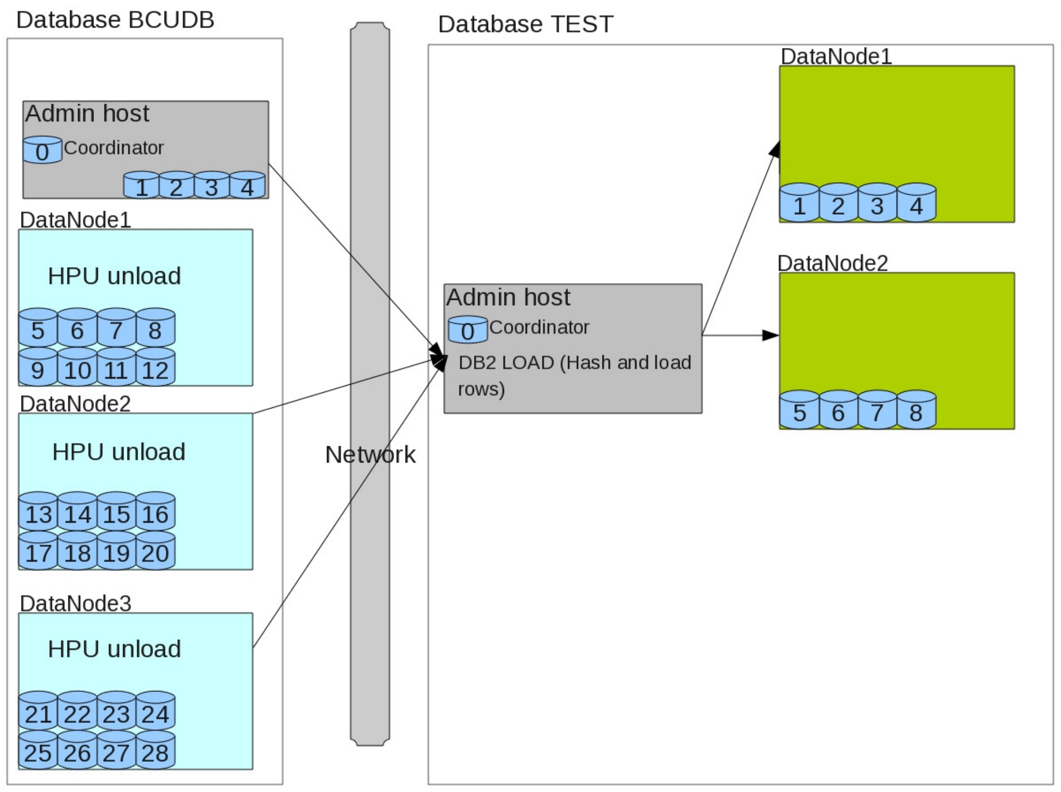

Figure 4 Migrating data when partition maps are different

During single stream HPU migration data is unloaded but not repartitioned (hashed) by HPU on the source database. The unloaded data is sent through single pipe on the target coordinator for the DB2 LOAD command to repartition and load the rows.

There are two situations when "single stream" migration is recommended:

- Source and target database use different distribution map size

  DB2 supports two distribution map sizes: 4K (4 096 entries) and 32K (32 768 entries). A typical example is when one database is V9.5 using a 4K distribution map and the other database is V9.7 with a 32K map. In this case single stream migration is the only available option.

- Source and target databases have distribution maps that are the same size but the target map is not in standard round robin order and the map does not have a repeating list of target partitions

  The target distribution map is created by HPU by repeating the sequence of target partitions specified in the PARTS() clause in the TARKET KEYS block until the DB2 distribution map array is filled. Some changes of the map can leave no repeating pattern (list) of target partitions in the target distribution map. The map can be changed deliberately, for example, to eliminate data skew issues on the database or a redistribute command is run. After the redistribute command the map might have no repeating pattern for target partitions. Customization of a map can also leave the map without a pattern.

  In this case, either run single stream migration or provide the transformed version of the full map text in the control file, in the PARTS() clause, and use the standard migration. Perform the following steps to use the transformed map:

  1. On the target database save the map into a file

     ```
     db2gpmap -d target_db_name -m map_file.out -t tabschema.tabname
     ```

  2. Transform the map by replacing spaces and new line characters with a comma with unix shell:

     ```
     cat map_file.out | sed 's/ /,/g' | tr '\n' ',' | sed '$s/.$//' > map_file_transformed.out
     ```

  3. In the control file provide the text of the transformed map from the map_file_transformed.out file in the PARTS() clause within the TARGET KEYS block

  4. Execute HPU migration

The key considerations and recommendations for “single stream migration” are:

- Performance degradation

  The standard migration with row hashing performed by HPU and DB2 parallel load offers better throughput than single stream migration especially on larger partitioned database environments. In a system with several data nodes, use default partition-to-partition migration to avoid the network bottleneck caused by the single pipe processing on the coordinator node.

- Distribution map structure, including changes, on the target system

Use the `db2gpmap` command on the database to extract the map into a file. When no repeating pattern of partitions exists and the migration performance is crucial, it provides the transformed version of the full map text in the control file to use the standard migration rather than single stream migration

Enforce the standard migration for non-standard database partitioning architecture when performance is critical.

- Verification of distribution map size

  Extract the map with the `db2pgmap` command and check the number of entries. When it has 4096 entries, it is a 4K map and when it has 32768 it is a 32K map. Verify the map for both the source and target database partition groups.

- The TARGET KEYS clause

  This clause is ignored for single stream migration as no HPU partitioning takes place

The following example shows a control file that is enforcing the single stream migration for BI_SCHEMA.TB_SALES_FACT table:

```
GLOBAL CONNECT TO BCUDB;
-- Migrate Block
MIGRATE TABLESPACE
PART(1:28)
DB2 NO LOCK YES QUIESCE YES
TARGET ENVIRONMENT(INSTANCE "bcuaix" on "bluejay06" IN
TEST REPART NO)
WORKING IN("/work1")
SELECT * FROM BI_SCHEMA.TB_SALES_FACT where PRODUCT_ID=10;
FORMAT DELIMITED INTO TEST.TB_SALES_FACT
;
```

The target database partition group in which the target table exists was redistributed recently and is no longer in round robin order. There is no repeating pattern of target logical nodes in the target distribution map. Since the source table is not large specify the “REPART NO” in the TARGET ENVIRONMENT clause option to migrate the data in a single stream mode. The TARGET KEYS clause is not used as repartitioning is not performed by HPU.

Conclusion

Use IBM InfoSphere Optim High Performance Unload as a tool to streamline specified database migration scenarios:

- Avoid contention with ETL processes by scheduling automated data migration outside of ETL, backup and other data operations.
- Unload data from offline backup images or online backup images created outside of ETL operations to help ensure data consistency.
- Use the SELECT clause in conjunction with the PART clause to migrate a subset of data when the target environment has less storage available.
- Use the HPU options for target database partitions and distribution key to help ensure that data is re-partitioned correctly when you are migrating data to a database with a different partition map.
- Migrate range partitioned tables as a sequence of ranges within a single control file; choose the most recent range first.
- Influence HPU usage of processor resources on the source system by specifying the number of cores to be used on each data node.
- Use a directory structure on storage that does not conflict with storage used for table space containers when you are staging or streaming data.
- Ensure a user with the name used by source instance exists on the target system and has the appropriate database privileges to load and insert into the target tables.
- Perform table space migration only when all target tables exist in the same DB2 database partition group and all target tables names match the source table names.
- Use a separate migrate block for each table when different LOCK and QUIESCE options are required for the tables.
- Use HPU migration to migrate different subsets of data from backup files when past data is required and the access to the production database is not allowed or restricted, for example, during ETL and online periods.
- Preserve the values from the source database for target tables with identity columns defined.
- Enforce the standard migration for non-standard database partitioning architecture when performance is critical.

Appendix A. Configuration of test systems used

Two test systems were used for this paper.

1. The source system was an IBM Smart Analytics System E7700 that consisted of four servers; a foundation module and three data modules. The foundation module had a single database partition that supported the catalog function, the coordinator function and non-partitioned metadata, staging and data warehouse tables and four database partitions. Each data module had eight database partitions that contained partitioned data warehouse tables.

2. The target system was an IBM Smart Analytics System E7100 that consisted of three servers; an administration module and two data modules. The administration module had a single database partition that supported the catalog function, the coordinator function and non-partitioned metadata, staging and data warehouse tables. Each data module had four database partitions that contained partitioned data warehouse tables.

The software versions installed were:

- Optim High Performance Unload for DB2 for Linux, UNIX, and Windows version 32 bits 04.02.100
- DB2 version: DB2 v10.1.0.2 FP2
- AIX version: 6.1.0.0

Staging area on the target system

The directory `/work1` was used for the HPU named pipes. Symbolic links were used to ensure that this directory was available on each data node so that data was processed successfully on each data node. The staging area is a temporary area and HPU removes temporary pipes and files once the migration operation is finished for a particular table. The following example shows the commands used to create the symbolic links on the first and second target data nodes. The first data node already contained the directory /work1. The following commands were used:

```
mkdir /bkpfs/bcuaix/NODE0001/HPU
ln /bkpfs/bcuaix/NODE0001/HPU /work1
mkdir /bkpfs/bcuaix/NODE0005/HPU
ln /bkpfs/bcuaix/NODE0005/HPU /work1
```

The HPU configuration file `db2hpu.cfg` on the test system used was as follows:

```
# HPU default configuration
bufsize=4194304
db2dbdft=BCUDB
```

```
db2instance=BCUAIX
doubledelim=binary
netservice=db2hpudm412
nbcpu=8
lock=yes
quiesce=yes
stagedir=/work1
```

Further reading

- Information Management best practices:
  http://www.ibm.com/developerworks/data/bestpractices/
- DB2 for Linux, UNIX, and Windows best practices:
  http://www.ibm.com/developerworks/data/bestpractices/db2luw/
- Using IBM InfoSphere Optim High Performance Unload as part of a recovery strategy in an IBM Smart Analytics System
  https://ibm.biz/Bdx2nk
- DB2 for Linux, UNIX, and Windows Best Practices on developerWorks
  http://www.ibm.com/developerworks/data/bestpractices/db2luw
- Information Management best practices on developerWorks
  http://www.ibm.com/developerworks/data/bestpractices/
- IBM InfoSphere Optim High Performance Unload product page
  http://www-01.ibm.com/software/data/optim/high-performance-unload-db2-luw/
- Advanced Recovery Solutions for IBM DB2 for Linux, UNIX and Windows
  http://www-01.ibm.com/software/data/db2/linux-unix-windows/tools/data-recovery/

Contributors

Vincent Arrat
HPU Development Team

Jaime Botella Ordinas
IT Specialist & Accelerated Value Leader

James Cho
STSM & Chief Architect for IBM PureData System for Operational Analytics

Austin Cliford
DB2 Data Warehouse QA Specialist

Bill Minor
Information Management DB2 Tooling & Development

Notices

Trademarks