IBM

DB2 Optimization Techniques for SAP Database Migration and Unicode Conversion

- Advanced migration techniques
- Optimization strategies
- Best practices

Andreas Christian
Waldemar Gaida
Hans-Jürgen Moldowan
Thomas Rech

Redbooks

ibm.com/redbooks

International Technical Support Organization

DB2 Optimization Techniques for SAP Database Migration and Unicode Conversion

August 2009

SG24-7774-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (August 2009)

This edition applies to DB2 UDB Versions 9.1 and 9.5 and SAP Kernel Release 6.40 and 7.x.

Contents

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 about 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 give 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.

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.

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.

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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

Preface

SAP® migrations are a standard process nowadays. We see an increasing number of customers changing their database software to IBM® DB2® for UNIX®, Linux®, and Windows® together with SAP upgrades, Unicode conversions, hardware or operating system changes, and system consolidations.

This IBM® Redbooks® publication describes optimization strategies and best practices for migrating SAP systems towards IBM DB2 for Linux, UNIX, and Windows, as well as for performing Unicode conversions. We discuss DB2-specific recommendations and best practices for the migration process as well as for Unicode conversions.

This book is intended for experienced SAP migration consultants involved in operating system and database (OS/DB) migrations or Unicode conversions, choosing IBM DB2 as a target database platform. It addresses the advanced SAP migration techniques, considerations for database layout and tuning, and the unique DB2 capabilities, such as compressing the target database while loading data.

All techniques discussed within this book are based on extensive tests, as well as experiences collected on various migration projects. However, it is important to understand that the optimizations described in this document may have side effects, introduce risks to the overall process, or require changes to the production system. Therefore, the features discussed must be chosen wisely. They should be used only if the migration downtime window or compression requirements make it necessary to use these optimizations.

Chapter 1, “Introduction” on page 1, introduces into SAP migrations and Unicode conversions. Chapter 2, “Migration essentials” on page 7, summarizes our recommendations and findings. It can be used as a quick reference for experienced migration consultants. Readers interested in more details can find the in-depth information beginning with Chapter 3, “Tools overview” on page 17.

The detailed sections are divided into six main areas:

- Best practices and recommendations for the source system database export
- Advanced migration techniques (such as table splitting)
- Database layout and configuration
- Database import recommendations
- SAP NetWeaver® Business Warehouse migration
- Background information about Unicode

The team that wrote this book

This book was produced by a team of specialists working at IBM Germany.

Andreas Christian worked for the SAP BW/DB2 porting team at SAP in Walldorf before he joined IBM SAP DB2 Center of Excellence in 2006.

Since 1999, he has helped port SAP BW to DB2, conducted SAP BW/DB2 performance and scalability studies, and supported SAP BW/DB2 customers with the implementation and operation of SAP Business Intelligence on DB2 LUW. He is coauthor of the IBM Redbooks publication *Building and Scaling SAP Business Information Warehouse on DB2 UDB ESE*, SG24-7094. Andreas is driving IBM DB2 sales by performing DB2 proof of concepts for IBM/SAP customers. His focus areas are SAP performance optimization and tuning (especially in SAP BW environments), as well as SAP migration topics.

Waldemar Gaida is a certified SAP Technology and OS/DB Migrations consultant. Before Waldemar joined IBM, he worked as a leader of a data center for 10 years, where he worked with SAP R/2® and SAP R/3®. He joined IBM in 1998, and until 2006 was involved in customer projects helping customers covering topics related to SAP Technology. Since 2006 Waldemar has supported the team of the IBM SAP DB2 Center of Excellence. His focus areas are SAP performance monitoring and tuning, investigations on new DB2 features (for example, row compression), and SAP migration topics. Waldemar holds a degree in Geodetic Engineering from Rheinische Friedrich-Wilhelms University in Bonn, Germany.

Hans-Jürgen Moldowan is a certified SAP Technology and OS/DB Migrations Consultant, and has been working for IBM since 1997. Before joining the IBM SAP DB2 Center of Excellence in 2008, he provided SAP DB2 technical sales support within IBM to customers and partners. During this time, Hans-Jürgen was involved in numerous proof-of-concept and migration projects, as well as first-of-a-kind technology implementations. Hans-Jürgen holds degrees in Electrical Engineering from the University of Cooperative Education in Mosbach, Germany, and in Industrial Engineering from the Pforzheim University, Germany

Thomas Rech has worked in the DB2 area since 1996. Begining as a trainer for DB2, he soon moved to the SAP arena in 1997. He started in the SAP/IBM DB2 Porting team as a Developer, followed by a role as a Technical Sales Consultant. He lead the project for the internal SAP implementation of DB2 and took over the role as Technical Lead in the largest SAP/IBM proof of concept in 2006. Thomas is now an Architect and Team Lead in the IBM SAP DB2 Center of Excellence out of Research and Development in Boeblingen, Germany. Thomas holds a master's degree in Computer Science from the University of Applied Sciences in Worms.

Acknowledgements

In addition to the authors, many other people contributed to this book by providing helpful information about various aspects of the subjects that are presented in this book.

We would like to thank the following people for their support:

Christian Brockhaus
Olaf Depper
Jochen Donner
Sergiy Malikov
Thomas Matthä
Malte Schünemann
Alexander Seelert
Jens Seifert
Rainer Staib
IBM Germany

Jeremy Broughton
IBM Canada

Britta Bachert
Arndt Effern
Frank-Martin Haas
Johannes Heinrich
Michael Schnaubelt
Ralf Stauffer
Bettina Weissinger-Stelzel
Torsten Ziegler
Andreas Zimmermann
SAP Germany

Edgar Maniago
Joachim Pfefferle
SAP Canada

Emma Jacobs
Whei-Jen Chen
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbooks@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction

SAP offers software for various hardware platforms and database systems. The procedure to migrate an existing SAP system from one database system or operating system (or both) to another is known as a heterogeneous system copy or operating system and database (OS/DB) migration.

SAP has developed a set of tools that allows customers to export their source database in a database-independent format and import it into the target database. The same set of tools allows converting a non-Unicode SAP system into a Unicode one.

The process of migrating an SAP system to another platform (for example, changing the database vendor) or converting an SAP system to Unicode is basically the same procedure regarding the export and import from a technical point of view. Therefore, OS/DB migrations and Unicode conversions easily can be combined into one single project.

In this IBM Redbooks publication we describe various optimizations and best practices for converting a DB2 database to Unicode or migrating an SAP system from a non-DB2 database to DB2.

1.1 Methods outside the scope of this book

We do not discuss SAP Minimized Downtime Service (MDS) in this book. MDS is a special migration approach that allows the source system to be online while exporting the largest tables. Smaller tables are exported during an offline window. This leads into a minimized overall system downtime during a migration or Unicode conversion. The MDS option is exclusively available as a service offer from SAP and is therefore not evaluated in this book.

For changing the operating system only, you can use database-specific methods, for example, DB2 backup and restore between UNIX operating systems of the same endianess. For more information see the article *Copying Your SAP/R3 System Across Platforms Using DB2 Universal Database V8 Redirected Restore* that is available at:

http://www.ibm.com/developerworks/db2/library/techarticle/0308nesiem/0308nesiem.html

These special approaches are also not described in this book.

1.2 Heterogeneous system copy

The process of copying an SAP system while changing the operating system or the database platform is known as *heterogeneous system copy*. Briefly speaking, a heterogeneous system copy works as follows:

1. The database of the source system is exported into a database and operating-system-independent format using SAP tools.
2. A new SAP system is installed, using the export from step 1 to load the database.

Figure 1-1 illustrates this process, which is supported by SAP and has been performed with many customer systems.

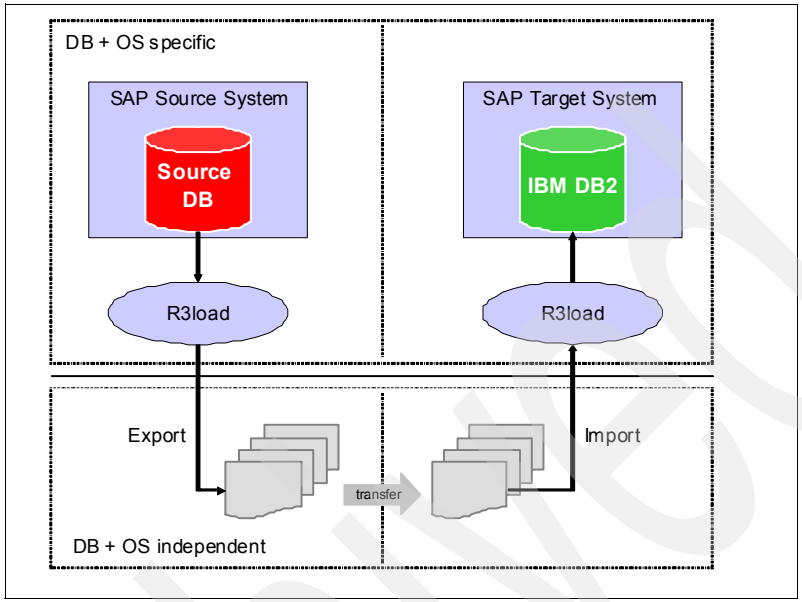


Figure 1-1 SAP heterogeneous system copy overview

While exporting or importing, the SAP system must be offline. No SAP user activity is allowed. Usually, customers allow a weekend's time frame for performing a heterogeneous system copy. If the system is large or the time frame is tight, special optimizations to the export and import process must take place.

1.3 Unicode conversion

All new SAP product releases from 2007 on are based on Unicode. The usage of multi-display multi-processing (MDMP) systems in new releases such as SAP ERP 6.0 (previously named SAP ERP 2005), which is based on SAP ECC 6.0, is no longer supported. SAP ERP 6.0 will be the release of choice for most customers. This implies that those systems must be converted to Unicode.

The technique of a Unicode conversion is very similar to that of a database migration, which consists of an export and an import phase. They are both based on the use of R3load. The Unicode conversion itself is normally executed during the export phase. It is, therefore, very easy to change the database for the target system without additional effort. Restrictions for the migration procedure due to

the Unicode conversion are mentioned and we provide you with optimization hints concerning export and import steps.

To minimize downtime for those customers who must perform an upgrade together with a Unicode conversion, SAP has developed the process *Combined Upgrade & Unicode Conversion* (CU&UC). This procedure is applicable to clients that have a 4.6C ERP system with multi-display multi-processing.

Depending on the target release (SAP NetWeaver 7.0 SR1 and later), this procedure is also applicable for the following start releases (see SAP Note 928729):

- R/3 Enterprise (4.70) with Extension set 1.10 or 2.0
- SAP ERP 2004 (SAP ECC 5.0)

Figure 1-2 shows the system landscape setup and the steps required for a combined upgrade and Unicode conversion of a source release SAP R/3 4.6C to a target release SAP ERP 2005.

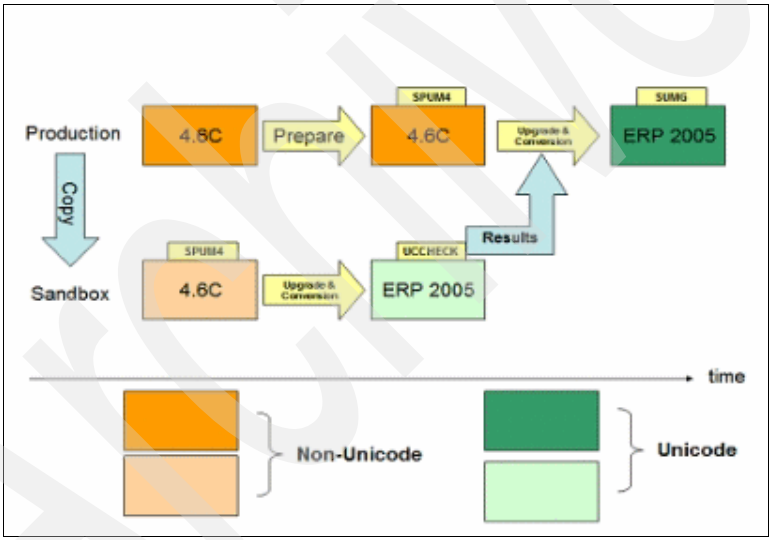


Figure 1-2 CU&UC process flow

Let us assume that the goal is to do a CU&UC to an SAP 4.6C production system. When your target is a Unicode system, you must be Unicode-compliant with your ABAP™ reports. Since the check and adjustment for this can only be done in a system that supports Unicode, you must provide such a system.

This can, for example, be done by copying your 4.6C system to a sandbox system. On this system, you perform a CU&UC. At the end you have a system

that is able to run the Unicode checks (UCCHECK). The results of these checks and also the results of the preparation steps done for the Unicode conversion (SPUM4, resulting in a vocabulary for Unicode conversion) can be moved to the production system during the CU&UC process, so you are not required to repeat these steps manually for every system.

For customers whose systems are running on older releases, SAP developed the Twin Upgrade & Unicode Conversion (TU&UC) procedure.

2

Migration essentials

In this chapter we summarize the overall findings and the essential recommendations of our study in the database migration and Unicode conversion. We explain the details in the subsequent chapters.

As a migration always means a planned downtime and is a very special phase in the life of an SAP system, ease of use and a save process are more important than reducing the downtime to the absolute minimum. The most important recommendation we would like to provide is to use only the optimizations that are required to meet the downtime target.

A prerequisite for applying any of the optimization techniques described in this book is a sound knowledge of the Heterogeneous SAP System Copy process and the corresponding SAP Technology and Migration Consultant Certification.

2.1 Database export

The recommendations provided in this section apply to the optimization of the source system database export.

2.1.1 Unsorted export

When exporting data from the source database you should choose unsorted export whenever applicable. We recommend that you:

- Use the migration monitor with the Data Definition Language (DDL) mapping file when unsorted and sorted export should be used in parallel.
- Use R3load with the latest patch level.
- Obey SAP Note 954268, which states the prerequisites for unsorted unload if you are using R3load prior to 6.x or before compile date February 10, 2006.

2.1.2 Package splitter

You can use the package splitter to split STR-files according to different criteria. This allows you to increase parallelism and ensure better granularity of the packages, which results in better resource usage during the migration. We recommend that you:

- Use the JAVA-Package splitter.
- Choose the options for top tables (`-top 50-200`), table size limit (`-tableLimit 500-1000`), and package size limit (`-packageLimit 500-2000`) for standard migrations running with SAPInst.
- Determine the tables that have the longest runtimes for more complex migrations and put their names into a file (for example, table_names.txt) that is used by the package splitter (tableFile). All these tables will be split out. You can use this option together with the options mentioned above.

2.1.3 Export server scenarios: Local or remote

If CPU resources become a bottleneck on the exporting machine, it makes sense to move the workload generated by R3load onto one or more dedicated SAP application servers. With running R3load from the application server, more CPU resources could be utilized by DB2 on the SAP database server. In our tests, this remote setup showed reduced export runtimes. Nevertheless, keep in mind that when exporting from a remote application server, data must be transferred over

the network. You should ensure that the network connection will not turn into a bottleneck.

2.2 Advanced migration techniques

Advanced migration techniques improve both the source system database export and the target system database import.

2.2.1 Socket option

If you have large tables that define the overall runtime of the migration, you can save time using the socket option. As a prerequisite, make sure that you have a fast and stable network connection when you use different servers for export and import. This is important to ensure optimal performance and to minimize the risk of failure.

You can also use this option within one server.

When you want to combine this option with other data transport techniques, you must use multiple MigMon instances due to the configuration differences in the properties file.

Be sure that you understand the setup of the socket option and its influence on other optimizations, in particular sampled compression.

You can also use the socket option for Unicode conversions if you obey the restrictions in SAP Note 971646.

2.2.2 Table splitting

If a single table or a few tables determine the export runtime of a migration, which is likely for table clusters such as CDCLS in the case of a Unicode conversion, you can use the table-splitting technique to cut these tables into pieces and export these pieces in parallel. This is done by defining WHERE clauses that select only a subset of the table. You can use the tool R3ta to determine the WHERE clauses that are stored in a single WHR file. To split the WHERE clauses into separate WHR files that contain one WHERE clause each, you can use the Where Splitter.

To speed up the export and import using table splitting you can do the following:

- Check whether you can create an index that refers to the table column that is used in the WHERE clause. Be aware that creation of additional indexes can impact production operation, as this requires additional hardware resources and may also affect the access plans of queries.
- If you have created an additional index to support table splitting, check whether you can reorganize the table to be clustered by this index. In some cases, this results in a better export performance.
- If some of the table pieces show a longer export time compared with the others, you can introduce additional WHERE conditions to further split this table part.
- Be sure to understand the advantages of using sequential DB2 LOAD versus parallel inserts and plan the migration based on your needs.
  - Although the usage of DB2 LOAD forces all R3load processes to import data sequentially into the corresponding table, performance may be in some cases as fast as with concurrent R3load processes using DB2 INSERT. However, the decision to use LOAD or INSERT varies with the infrastructure and must be tested.
  - If you have a table with many indexes, it seems to be more effective to create the indexes before the data is loaded and with more significant effect if parallel insert is used. We recommend that you validate the most efficient way of index creation as part of a migration test. If you use DB2 LOAD and an R3load process is interrupted, the load of all corresponding data packages must be restarted from scratch including the index build. Therefore, if you want to minimize the impact of an interrupted DB2 LOAD, the table should be first loaded and then the indexes should be built.
  - To force the use of DB2 LOAD for the import, set the appropriate R3load parameter or environment variable.
  - If you use DB2 LOAD for the import, enable the incremental index build. Set the environment variable `DB6LOAD_INDEXING_MODE=2`.
  - If you use the DB2 LOAD for the import, serialize the affected table using the *orderBy* entry in the import monitor properties file or use dedicated Migration Monitor instances.
  - With respect to ease of use, the parallel insert into a single table is a good starter.

2.3 DB2 layout and configuration options

Before you start the import into the DB2 target system, you should carefully plan the database layout and database configuration.

2.3.1 Table space principles and configuration

Table spaces can be created as either automatic storage table spaces managed by DB2's automatic storage management (also referred to as autostorage) or as database managed space (DMS) file table spaces in autoresize mode. The later option offers more possibilities of intervention by a database administrator and offers full control of the placement of table space containers. On the other hand, the automatic storage feature incorporates a better ease of use. We recommend that you:

- Use enough spindles for your I/O.
- Separate logging I/O from data, index, and temporary I/O.
- Try to avoid container sizes larger than 20 GB.
- Switch the file system caching off to avoid additional operating system (OS) buffering.
- Place large tables into separate table spaces and introduce new data classes.
- Avoid table spaces that are significantly larger than the others, as this may impact backup performance.

2.3.2 Logging configuration

We recommend that you:

- Switch off archival logging and use circular logging during the migration.
- Provide enough log space. A good starter is to use the planned configuration for production. More space may be required if using many parallel R3load with Inserts.

2.4 Database import

The data import phase is a long-running task. Together with the export, it influences the overall downtime phase of a heterogeneous system copy or Unicode migration. Therefore, it is important to optimize the import using SAP tools and DB2 configuration settings.

2.4.1 Import optimization techniques

There are several measures that you can take to optimize the import process:

- Default optimizations
- Optional optimizations
- Advanced optimizations

The *advanced optimizations* measures can be very complex, require significantly more testing effort, or possibly have side effects that can influence the overall migration process.

Default optimizations

Our default recommendations for database import are:

- Use the DB2 LOAD API with default settings for R3load.
- Specify enough utility heap to ensure parallelism of the DB2 LOAD API. use 200,000 as a starter.
- Configure SORTHEAP and SHEAPTHRES_SHR to accommodate the large sorts during index build. A good starting point is 50,000 pages for SORTHEAP and `2 * (SORTHEAP) * (number of R3load processes)` for SHEAPTHRES_SHR.
- Be sure that the file system caching is disabled on database level during table space creation or use the ALTER TABLESPACE statement for existing table spaces.
- Do not use STMM during the migration final test and cutover to ensure stable runtimes.
- To avoid failing R3load processes due to wait situations, set LOCKTIMEOUT to -1.
- Define one buffer pool using the remaining memory after utility heap and sort memory are configured.
- Leave all other configuration settings according to the SAP Notes for DB2 standard parameter settings.
- Create primary and secondary indexes before you load data.
- Allocate enough temporary space for the index build to avoid temp space overflow. Be aware that the amount of data may have increased since the last successful test migration. Therefore, increase temporary space to provide enough reserve for the final productive migration.
- As a starting point, use as many R3load processes as you have CPU cores on your server. Do not use many more R3load processes than available CPU cores to avoid hardware (CPU, memory, I/O) resource bottlenecks.

Optional optimizations

The following optimizations require more testing effort, as they may not show full effect in all environments:

- To determine optimized values, use Self Tuning Memory Management (STMM) during a test migration.
- Avoid many parallel index build processes by optimizing the package import order. The goal is to prevent I/O resource overload.
- Monitor the import process and adjust the configuration for the utility heap, buffer pool, and sorting.
- Monitor and optimize the configuration for I/O (for example, NUM_IOCLEANERS, DB2_PARALLEL_IO, or disable file system caching for logging).
- To optimize the number of parallel processes with respect to evenly distributed resource usage, analyze the resource usage of the system (CPU, I/O, and memory).

Advanced optimizations

The following optimization topics can be very complex and might have side effects that can influence the overall migration process. Test them in detail.

- Use the application server for import if the I/O subsystem can handle additional workload, the target system is short on CPU, and a fast network connection between the application server and the database server is available.
- You can use a DMS temporary table space to optimize the index rebuild phase, but be aware of the side effects (for example, no parallel REORG processes and the need to create the temporary table space manually). After a successful migration make sure to switch back to SMS temporary table spaces.
- Change the order of index creation for selected tables if the index build phase takes significantly longer compared with other large tables, which is hard to determine.
- Optimize the CHNGPGS_THRESH parameter together with buffer pool and index creation to optimize the I/O in this area.
- Create dedicated buffer pools for one or more table spaces (for example, for temporary table spaces).

2.4.2 DB2 row compression based on R3load 6.40

With R3load Release 6.40, you can compress table data during import in combination with the usage of the DB2 LOAD API by using the R3load parameter LOADCOMPRESS. The compression dictionary is built up by importing a defined number of rows and issuing a REORG on the table after this. Normally, a sample (which is about 1% of the number of rows of the table) is able to achieve a compression dictionary of a good quality. However, some tables might need a higher sample rate.

2.4.3 DB2 row compression based on R3load 7.00

R3load Version 7.00 and later offers a new SAMPLED compression option. This option loads a representative sample of data into a table, builds a compression dictionary, and then loads the complete set of data.

We recommend using the new R3load sampling method for compressing data while importing, as it shows an optimal combination of maximum compression ratio and minimized load runtime.

Nevertheless, keep in mind that this approach is not fully automatic. That is, it consists of two phases, and you must restart R3load manually after successfully finishing the first phase. Import into a compressed table can be faster than into an uncompressed table. Therefore, you might actually reduce the overall runtime of the import and may accept this manual process.

In addition, the two phases can be separated. You can perform phase one before a productive migration with data obtained from a test migration. This sample is still valid even after some time and the data will be replaced during to the final migration that is performed in phase two.

If you perform phase one up front in a productive migration, you must ensure that no structural change to the related tables takes place.

2.4.4 Collecting statistics

After a migration, all non-volatile tables should have valid statistics. For large tables, the statistics collection using the RUNSTATS command can take very long.

To extract statistics for such big tables after the final test migration, you can use the tool **db2look**. This generates update statements on the statistics tables of DB2 in an SQL script. This script can be executed to update the statistics for big tables much faster than a RUNSTATS.

Be aware that the definition of some tables may change after running **db2look** and before the production migration. Those tables may require RUNSTATS after the data is loaded.

For small tables, performing a RUNSTATS is faster than running this script.

To shorten the RUNSTATS runtime, you should consider parallel execution. To achieve this, a script with the RUNSTATS commands can be generated and split into several pieces.

Therefore, we recommend a mixed procedure for the statistics update. Use parallel RUNSTATS for small tables and **db2look** for big tables.

2.5 SAP NetWeaver BW migrations

SAP NetWeaver Business Warehouse (SAP NetWeaver BW) migrations follow the same rules and optimizations mentioned in this book. However, there is a major difference, as SAP NetWeaver BW implements more database-specific optimizations that cannot all be handled by R3load. This is true in particular for DB2 database partitioning feature (DPF) and multi-dimensional clustering (MDC). Both require special DDL statements and, therefore, you must do the following to succeed in migrations of SAP NetWeaver BW-based systems:

- Read the white paper *Heterogeneous SAP NetWeaver Business Intelligence (BI) System Copy to IBM DB2 for Linux, UNIX and Windows,* which is available in the SAP Community Network:

  https://www.sdn.sap.com/irj/sdn/db6
- Execute report SMIGR_CREATE_DDL on the source system.
- Execute report SAP_DROP_TMPTABLES on the source system.
- Execute report RS_BW_POST_MIGRATION on the target system.

3

Tools overview

In this chapter we provide an overview of available SAP system copy tools. You can use these tools both for homogeneous and heterogeneous system copies. Tools that are used for heterogeneous system copies are often also referred to as *migration tools*.

This chapter is not intended to replace the SAP training or documentation available. This chapter introduces the tools required for the DB2 optimizations and points our hints and tricks found during our tests.

3.1 SAPInst and R3SETUP

The SAP installation tools SAPInst and R3SETUP are used to install SAP systems or to unload or load systems during a system copy procedure. They invoke other tools (for example, R3ldctl, R3szchk, and R3load) and control the entire installation and migration process. Some of the tasks that the tools perform include:

- Creating users or groups on the operating system level
- Adapting file system rights
- Installing SAP binaries (Kernel)
- Triggering the database unload and load processes.
- Triggering post processing such as collecting database statistics

SAPInst is used as of SAP basis Release 6.10, whereas R3SETUP supports basis Releases 3.1I up to 4.6D.

To simplify the migration process, the SAP installation tool SAPInst provides software life-cycle options. For example, the following options are available for a source system during a heterogeneous system migration:

- Export preparation
- Table splitting preparation
- Database Instance export

3.2 R3load, R3szchk, and R3ldctl

R3load, R3szchk, and R3ldctl are the main SAP tools used in SAP operating system and database (OS/DB) migrations. In this section we describe their purposes.

3.2.1 R3load

R3load is the *core* migration tool. It exports SAP ABAP table data from the source database and imports it into the target database. Besides this base functionality, it offers advanced options, for example, using DB2 row compression while importing data.

Figure 3-1 illustrates the function of R3load.

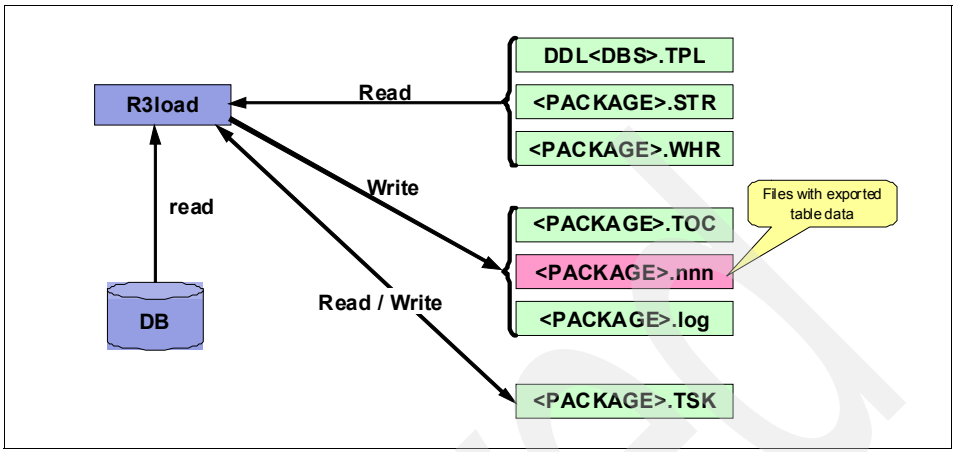


Figure 3-1 Tasks performed by R3load during the export

3.2.2 R3ldctl

R3ldctl unloads SAP ABAP data dictionary structures from the source system. It generates structure files (*.STR files) that describe the definition of tables, indexes, and views. In addition, it generates database-specific template files (*.TPL files) with definitions of DDL statements, for example, statements to create a table, to drop a table, and so on.

3.2.3 R3szchk

R3szchk computes the size of tables and indexes for the target database.

Figure 3-2 illustrates the tasks performed by R3ldctl and R3szchk.

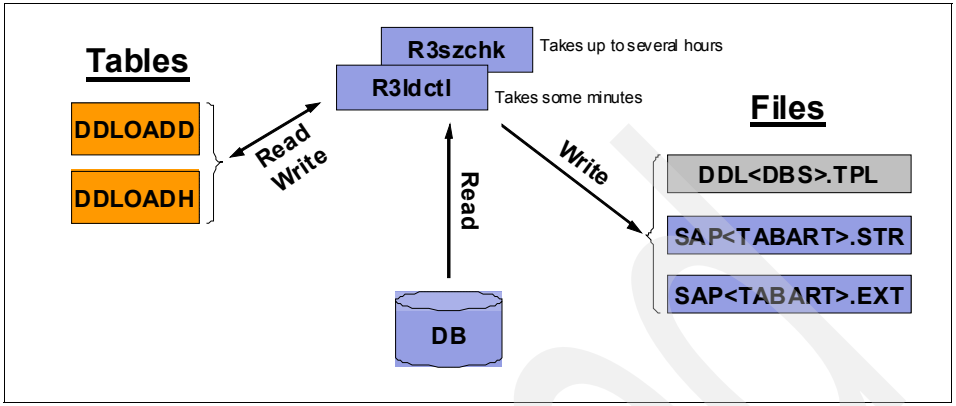


Figure 3-2 Tasks performed by R3ldctl and R3szchk

R3szchk may run for a long time in some cases, as it uses the `SELECT COUNT(*)` statement against all tables to calculate the target database size information. This is true in particular if the database management system is changed during the migration. If you experience a long runtime of the R3szchk, check whether R3szchk is using the `SELECT COUNT(*)` statement for the tables. If so, there are two optimizations available to overcome this bottleneck:

- You may run the R3szchk while the system is still up and running and used for production.

  Note: R3szchk requires that the STR files created by R3ldctl calculate the target size. If running R3szchk and R3ldctl while the system is up and running, no change of the data definition is allowed afterwards (for example, by new transports). These changes will not be reflected in the STR files, so the export will not be consistent or will fail.

- Another option for reducing the runtime is to use the option –r together with an input file to avoid the expensive statements. This file contains the name of the table and the number of records for this table. Running R3szchk with the option –r and the file containing the number of records will reduce the runtime by factors.

Example 3-1 shows a sample input file for R3szchk.

Example 3-1 File with table names and number of records

```
/B28/FCDIOEPE      12708488
/TST/FCDIOEPE      12704927
/BB1/FCDIOEPE      12704554
/BIO/B0001012000   11275128
WBCROSSGT          10946626
/BIO/AQP_DS3100    10697826
/BIC/B0000183009   10463648
/BBL/B0000183009   10463648
ZRSDRDHELP         10000000
```

You can determine the appropriate number of records up front by extracting the data from database statistics. You may use the DBACOCKPIT within the SAPGUI or the appropriate SQL statement. With DB2, you can use the following statement to retrieve the number of records for the largest 50 tables:

```
SELECT tabname, card FROM syscat.tables ORDER BY CARD DESC FETCH FIRST 50 ROWS
ONLY
```

The usage of the -r parameter is not supported by the SAPInst program at the time that the book was written. You must run the export preparation phase manually, which requires detailed knowledge of the migration procedure and a lot of experience in migrations.

3.3 Migration monitor: MigMon

The migration monitor, MigMon, is described in SAP Note 784118. Its parameters, functions, and control files are explained in more detail in the *Migration Monitor User's Guide*. This guide can be extracted from the DISTMON.SAR file, which is attached to SAP note 855772.

The main aspects and attributes of the MigMon are:

- Allow advanced control of R3load export and import.
- Automate dump shipping between source and target system.
- Support parallel unload and load processing.
- MigMon is controlled by properties files.
- Properties files are constantly reread after a defined time period, so some attributes, such as the number of parallel R3load processes, can be adjusted dynamically.

As of SAP NetWeaver 7.0, MigMon has been fully integrated into SAPInst. With older SAP releases, manual intervention may be necessary.

3.4 Time analyzer: MIGTIME

To have a baseline for optimization, you should have an overview of the import and export runtimes. For this analysis, SAP provides a toolset that is called migration time analyzer (MIGTIME). It is delivered as software archive file MIGTIME.SAR. You can find this archive on the installation master DVD or in SAP Service Marketplace (Support Packages and Patches). The archive comprises the content shown in Example 3-2.

Example 3-2 Content of MIGTIME.SAR archive

```
SAPCAR: processing archive /sapcd2/IM_AIX_PPC64/COMMON/INSTALL/MIGTIME.SAR
(version 2.01)
rw-r--r--      147883    17 Nov 2007 18:53 migtime.jar
Java-archive
rwxr-xr-x         516    17 Nov 2007 18:53 export_time.sh
UNIX-script for export analysis
rwxr-xr-x         516    17 Nov 2007 18:53 import_time.sh
UNIX-sript for import analysis
rwxr-xr-x         498    17 Nov 2007 18:53 time_join.sh
UNIX-script for time joining
rw-r--r--         699    17 Nov 2007 18:53 export_time.bat
WINDOWS-script for export analysis
rw-r--r--         699    17 Nov 2007 18:53 import_time.bat
WINDOWS-script for import analysis
rw-r--r--         681    17 Nov 2007 18:53 time_join.bat
WINDOWS-script for time joining
rw-r--r--       69828    17 Nov 2007 18:53 TimeAnalyzer.pdf
Documentation in PDF-format
```

You can find all the necessary information about how to start the scripts and choose the parameters in the documentation. Therefore, we only present some examples for the outcome of the analysis.

The result of every analysis is a text file and optionally an HTML file with the graphical representation of the runtimes. Additionally, a XML file can be created, which can be used as an input file for the time-join script.

3.4.1 Export times analysis

Example 3-3 shows an example for an export_time.txt file as a result of the *export_time* script. This file has two parts. The first part relates to the package export times. The second part relates to the table export times.

Example 3-3 Example of export_time.txt (part 1)

```
-------------------------------------------------------------------------------
package           time     start date        end date          size MB  MB/min
-------------------------------------------------------------------------------
REPOSRC           0:13:30  2007-09-20 20:58  2007-09-20 21:12   959.46   71.07
SAPAPPL2_1        0:04:46  2007-09-20 21:01  2007-09-20 21:06    27.48    5.76
SAPAPPL0_1        0:03:45  2007-09-20 20:59  2007-09-20 21:03    39.82   10.62
D010TAB           0:02:54  2007-09-20 20:49  2007-09-20 20:52    46.59   16.07
REPOTEXT          0:02:32  2007-09-20 20:58  2007-09-20 21:01   151.99   60.00
TRFCQDATA         0:02:29  2007-09-20 21:18  2007-09-20 21:21   155.01   62.42
SAPNTAB           0:02:02  2007-09-20 21:03  2007-09-20 21:05    41.02   20.17
SAPAPPL1          0:01:56  2007-09-20 21:00  2007-09-20 21:02    23.37   12.09
D010INC           0:01:49  2007-09-20 20:49  2007-09-20 20:51    19.55   10.76
PRCD_COND         0:01:45  2007-09-20 20:58  2007-09-20 21:00    13.52    7.73
SAPSSEXC          0:01:43  2007-09-20 21:06  2007-09-20 21:08    57.76   33.65
SMW3_BDOC2        0:01:32  2007-09-20 21:12  2007-09-20 21:13    41.81   27.27
DOKCLU            0:01:14  2007-09-20 20:53  2007-09-20 20:54   143.68  116.50
…
-------------------------------------------------------------------------------
                  1:12:05  2007-09-20 20:42  2007-09-20 21:21  2795.49
-------------------------------------------------------------------------------
```

For every package you get the following information:

- Package: The name of the package that was exported (may also be only one table, if split out).
- Time: The runtime needed to export the package.
- Start date: Date and time when the export of the package starts.
- End date: Date and time when the export of the package ends.
- Size MB: Size of the exported package in MB (for example, on disk).
- MB/min: Export rate in MB per minute (related to the export file size).

The list is sorted descending by the export time. Therefore, you get a good and fast overview of the packages that require the most time.

At the end of the output (see Example 3-3 on page 23), you have information about the following items:

- Sum of the export times
- Start time of the export process
- End time of the export process
- Size of the export (sum of all export files)

Example 3-4 shows the table that is related to the second part of the file. The `top`, `h` or `m` options of the export_time script control how many tables are shown.

Example 3-4 Example of export_time.txt (part 2)

```
-----------------------------------------
table             package           time
-----------------------------------------
REPOSRC           REPOSRC           0:13:29
D010TAB           D010TAB           0:02:54
REPOTEXT          REPOTEXT          0:02:31
TRFCQDATA         TRFCQDATA         0:02:29
D010INC           D010INC           0:01:49
PRCD_COND         PRCD_COND         0:01:45
SMW3_BDOC2        SMW3_BDOC2        0:01:32
DOKCLU            DOKCLU            0:01:13
MONI              MONI              0:01:11
DD03L             DD03L             0:01:11
DYNPSOURCE        SAPSSEXC          0:01:05
BALDAT            BALDAT            0:01:03
DDNTF_1B          SAPNTAB           0:00:55
SMWT_TRC          SMWT_TRC          0:00:50
…
-----------------------------------------
                                    0:57:47
-----------------------------------------
```

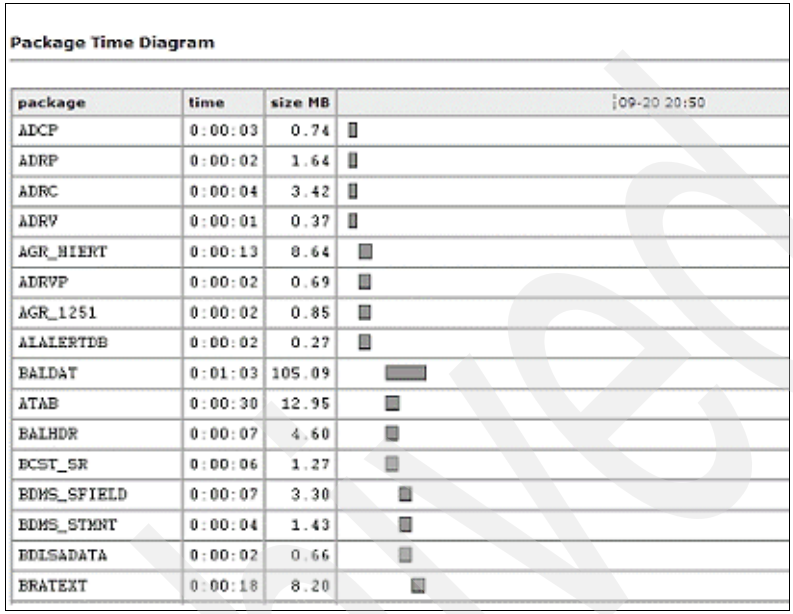
For every table listed you obtain the following information:

- Table: Name of the table exported
- Package: The name of the package containing the table (When split out, the table name and the package name are identical.)
- Time: The runtime needed to export the table

Using this information, you can easily identify the tables that contribute mostly to the export time of a package that comprises more than one table.

You obtain the graphical representation of the first part of the export_time.txt when you use the `html` option of the export_time script to generate a package time diagram.

Figure 3-3 shows a part of such a diagram. The diagram is sorted by the start time of the exported package and therefore provides a chronological overview of the export. The length of the bars represents the export time.



Package Time Diagram

package	time	size MB	09-20 20:50
ADCP	0:00:03	0.74	
ADRP	0:00:02	1.64	
ADRC	0:00:04	3.42	
ADRV	0:00:01	0.37	
AGR_HIERT	0:00:13	8.64	
ADRVP	0:00:02	0.69	
AGR_1251	0:00:02	0.85	
AIALERTDB	0:00:02	0.27	
BALDAT	0:01:03	105.09	
ATAB	0:00:30	12.95	
BALHDR	0:00:07	4.60	
BCST_SR	0:00:06	1.27	
BDMS_SFIELD	0:00:07	3.30	
BDMS_STMNT	0:00:04	1.43	
BDLSADATA	0:00:02	0.66	
BRATEXT	0:00:18	8.20	

Figure 3-3 Example of an export package time diagram

3.4.2 Import times analysis

Example 3-5 shows the partial content of an import_time.txt file as a result of the import_time script. This file consists of two parts. The first part relates to the package import times. The second part relates to the table import times.

Example 3-5 Example for Import_Time.txt (part 1)

```
-------------------------------------------------------------------------------
package          time     start date        end date          size MB  MB/min
-------------------------------------------------------------------------------
SAPAPPL2_1       0:33:19  2007-09-20 23:42  2007-09-21 00:16    27.48     0.82
SAPAPPL0_1       0:33:04  2007-09-20 23:39  2007-09-21 00:12    39.82     1.20
SAPAPPL1         0:18:37  2007-09-20 23:43  2007-09-21 00:01    23.37     1.26
SAPAPPL0_2       0:07:47  2007-09-20 23:55  2007-09-21 00:03     4.77     0.61
SAPSSEXC         0:07:36  2007-09-20 23:36  2007-09-20 23:43    57.76     7.60
REPOSRC          0:07:01  2007-09-20 23:35  2007-09-20 23:42   959.46   136.74
MONI             0:06:11  2007-09-20 23:36  2007-09-20 23:42   104.46    16.89
SAPAPPL2_2       0:03:14  2007-09-21 00:02  2007-09-21 00:06     3.92     1.21
D010TAB          0:02:26  2007-09-20 23:37  2007-09-21 00:35    46.59    19.15
SAPSDIC          0:02:21  2007-09-21 00:00  2007-09-21 00:02     4.14     1.76
```

```
SAPSSRC          0:01:33  2007-09-20 23:46  2007-09-20 23:48    11.11     7.17
SAPNTAB          0:01:07  2007-09-20 23:38  2007-09-20 23:39    41.02    36.74
DD03L            0:01:00  2007-09-20 23:40  2007-09-20 23:41    31.03    31.03
D010INC          0:00:48  2007-09-20 23:43  2007-09-20 23:44    19.55    24.44
DOKCLU           0:00:48  2007-09-20 23:35  2007-09-20 23:36   143.68   179.60
…
-------------------------------------------------------------------------------
                 2:27:27  2007-09-20 23:35  2007-09-21 00:36  2795.49
-------------------------------------------------------------------------------
```

For every package you receive the following information:

- Package: The name of the package that was imported (may also be only one table, if split out)
- Time: The runtime required to import the package (including index creation)
- Start date: Date and time when the import of the package starts
- End date: Date and time when the import of the package ends
- Size MB: Size of the imported package in MB (on disk)
- MB/min: Import rate in MB per minute (related to the export file size)

The list is sorted descending by the import time. Thus, you get a good and fast overview of the packages that require the most time.

At the end of the report you receive the following information:

- Sum of the import times
- Start time of the import process
- End time of the import process
- Size of the import (sum of all export files)

Example 3-6 shows the table that is related to the second section of the file. The `top`, `h`, or `m` options of the Import_time script control how many tables are shown.

Example 3-6 Example of Import_Time.txt (part 2)

```
-------------------------------------------
table            package          time
-------------------------------------------
REPOSRC          REPOSRC          0:07:01
MONI             MONI             0:06:11
DSYST            SAPSSEXC         0:05:12
D010TAB          D010TAB          0:01:11
DD03L            DD03L            0:01:00
D010INC          D010INC          0:00:48
DOKCLU           DOKCLU           0:00:48
DDNTT            SAPNTAB          0:00:47
PRCD_COND        PRCD_COND        0:00:43
```

```
TRFCQDATA          TRFCQDATA          0:00:40
STERM_LINK         STERM_LINK         0:00:32
REPOTEXT           REPOTEXT           0:00:30
DYNPSOURCE         SAPSSEXC           0:00:29
…
-------------------------------------------
                                      0:43:17
-------------------------------------------
```

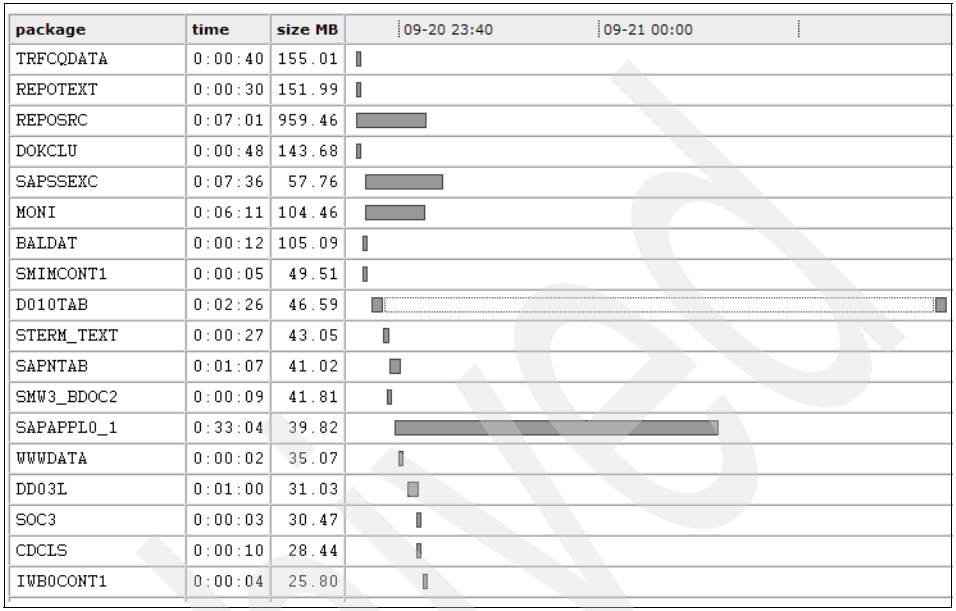
For every table listed you receive the following information:

- Table: Name of the table imported
- Package: The name of the package containing the table (When split out, the table name and the package name are identical.)
- Time: The runtime required to import the table

Using this information you can easily identify the tables that contribute mostly to the import time of a package that comprises more than one table.

You obtain the graphical representation of the first part of the import_time.txt when you use the `html` option of the import_time script to generate a package time diagram.

Figure 3-4 shows a part of package time diagram. The diagram is sorted by the start time of the imported package and, therefore, provides a chronological overview of the import. The length of the bar represents the import time.



package	time	size MB
TRFCQDATA	0:00:40	155.01
REPOTEXT	0:00:30	151.99
REPOSRC	0:07:01	959.46
DOKCLU	0:00:48	143.68
SAPSSEXC	0:07:36	57.76
MONI	0:06:11	104.46
BALDAT	0:00:12	105.09
SMIMCONT1	0:00:05	49.51
D010TAB	0:02:26	46.59
STERM_TEXT	0:00:27	43.05
SAPNTAB	0:01:07	41.02
SMW3_BDOC2	0:00:09	41.81
SAPAPPL0_1	0:33:04	39.82
WWWDATA	0:00:02	35.07
DD03L	0:01:00	31.03
SOC3	0:00:03	30.47
CDCLS	0:00:10	28.44
IWBOCONT1	0:00:04	25.80

Figure 3-4 Example of an import package time diagram

The dashed lines for table D010TAB mean that the import was interrupted and continued later.

3.4.3 Time join analysis

The third option of the time analysis tool is to join the export and import times. This produces the file `time_join.txt`. Choosing the `html` option also produces the file `time_join.html` with a graphical representation of the text version.

Example 3-7 shows the text versions. The records are on one line inside the text file. They are split into two lines here because of the document format and for readability.

Example 3-7 Example of time_join.txt

```
-----------------------------------------------------------------------------------------
package           total time  export time  start date        end date
                              import time  start date        end date          size MB
-----------------------------------------------------------------------------------------
SAPAPPL2_1           0:38:05      0:04:46  2007-09-20 21:01  2007-09-20 21:06
                                  0:33:19  2007-09-20 23:42  2007-09-21 00:16    27.48
```

```
SAPAPPL0_1          0:36:49      0:03:45  2007-09-20 20:59  2007-09-20 21:03
                                 0:33:04  2007-09-20 23:39  2007-09-21 00:12    39.82
SAPAPPL1            0:20:33      0:01:56  2007-09-20 21:00  2007-09-20 21:02
                                 0:18:37  2007-09-20 23:43  2007-09-21 00:01    23.37
REPOSRC             0:20:31      0:13:30  2007-09-20 20:58  2007-09-20 21:12
                                 0:07:01  2007-09-20 23:35  2007-09-20 23:42   959.46
SAPSSEXC            0:09:19      0:01:43  2007-09-20 21:06  2007-09-20 21:08
                                 0:07:36  2007-09-20 23:36  2007-09-20 23:43    57.76
SAPAPPL0_2          0:08:15      0:00:28  2007-09-20 21:00  2007-09-20 21:00
                                 0:07:47  2007-09-20 23:55  2007-09-21 00:03     4.77
MONI                0:07:23      0:01:12  2007-09-20 20:57  2007-09-20 20:58
                                 0:06:11  2007-09-20 23:36  2007-09-20 23:42   104.46
D010TAB             0:05:20      0:02:54  2007-09-20 20:49  2007-09-20 20:52
                                 0:02:26  2007-09-20 23:37  2007-09-21 00:35    46.59
SAPAPPL2_2          0:04:01      0:00:47  2007-09-20 21:02  2007-09-20 21:03
                                 0:03:14  2007-09-21 00:02  2007-09-21 00:06     3.92
SAPNTAB             0:03:09      0:02:02  2007-09-20 21:03  2007-09-20 21:05
                                 0:01:07  2007-09-20 23:38  2007-09-20 23:39    41.02
TRFCQDATA           0:03:09      0:02:29  2007-09-20 21:18  2007-09-20 21:21
                                 0:00:40  2007-09-20 23:35  2007-09-20 23:36   155.01
REPOTEXT            0:03:02      0:02:32  2007-09-20 20:58  2007-09-20 21:01
                                 0:00:30  2007-09-20 23:35  2007-09-20 23:35   151.99
SAPSDIC             0:03:00      0:00:39  2007-09-20 21:04  2007-09-20 21:04
                                 0:02:21  2007-09-21 00:00  2007-09-21 00:02     4.14
D010INC             0:02:37      0:01:49  2007-09-20 20:49  2007-09-20 20:51
                                 0:00:48  2007-09-20 23:43  2007-09-20 23:44    19.55
PRCD_COND           0:02:28      0:01:45  2007-09-20 20:58  2007-09-20 21:00
                                 0:00:43  2007-09-20 23:45  2007-09-20 23:46    13.52
DD03L               0:02:11      0:01:11  2007-09-20 20:51  2007-09-20 20:52
                                 0:01:00  2007-09-20 23:40  2007-09-20 23:41    31.03
DOKCLU              0:02:02      0:01:14  2007-09-20 20:53  2007-09-20 20:54
                                 0:00:48  2007-09-20 23:35  2007-09-20 23:36   143.68
…
--------------------------------------------------------------------------------
                    3:39:32      1:12:05  2007-09-20 20:42  2007-09-20 21:21
                                 2:27:27  2007-09-20 23:35  2007-09-21 00:36  2795.49
--------------------------------------------------------------------------------
```

Figure 3-5 shows a graphical representation of the package join time diagram.

package		time	size MB
ADRP	E	0:00:02	1.64
	I	0:00:04	
ADCP	E	0:00:03	0.74
	I	0:00:03	
ADRC	E	0:00:04	3.42
	I	0:00:05	
ADRV	E	0:00:01	0.37
	I	0:00:03	
AGR_HIERT	E	0:00:13	8.64
	I	0:00:11	
AGR_1251	E	0:00:02	0.85
	I	0:00:03	
ADRVP	E	0:00:02	0.69
	I	0:00:03	
ALALERTDB	E	0:00:02	0.27
	I	0:00:02	
BALDAT	E	0:01:03	105.09
	I	0:00:12	
ATAB	E	0:00:30	12.95
	I	0:00:15	

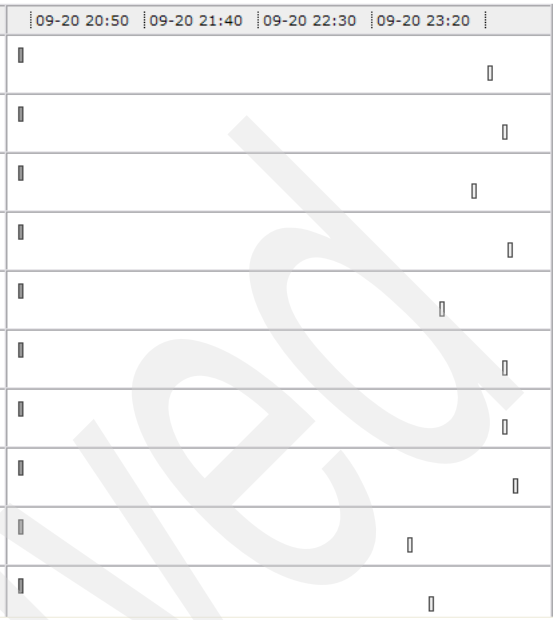


Figure 3-5 Example of a package join time diagram

3.5 Distribution monitor: DistMon

MigMon controls the export and import processing limited to a single server.

Using DistMon, the export and import processes can be performed by multiple servers, which is illustrated in Figure 3-6.

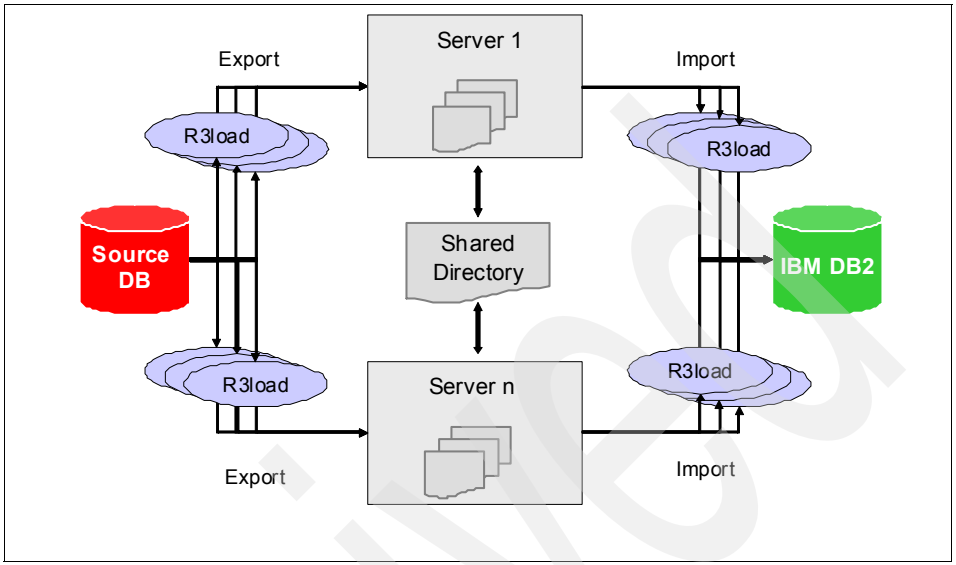


Figure 3-6 Function scheme of the distribution monitor

For a detailed description of the DistMon, see the SAP Notes 855772, 989116, and 1001383. The attachment of SAP Note 855772 contains the software archive DISTMON.SAR and the user guide *Distribution Monitor User's Guide,* which is stored in the zip-archive file *DistributionMonitorUserGuide.zip*. The user guide is also part of the SAR-archive DISTMON.SAR. It is named DistributionMonitorUserGuide.doc.

3.6 Tools for package and table splitting

To improve the speed of the SAP system copy, multiple R3load processes can export and import data in parallel. Different options are available to package the data during export and import, as shown Figure 3-7.

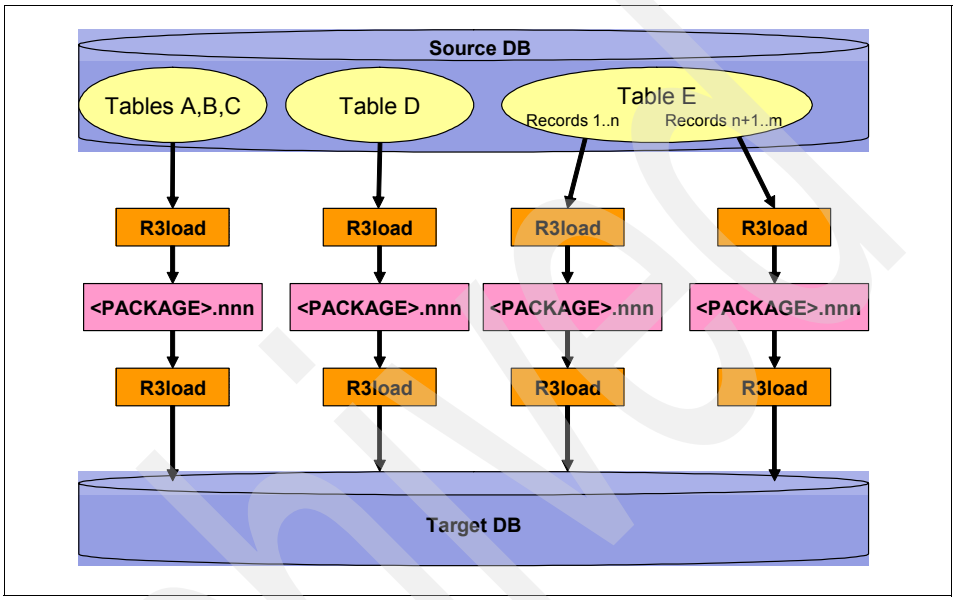


Figure 3-7 Package and table splitting options

The options for package and table splitting include:

- A package may contain data of multiple tables (for example, tables A, B, and C in Figure 3-7).
- A package may contain all data of a single table (for example, table D in Figure 3-7).
- A package may contain only a subset of a table (for example, as shown for table E in Figure 3-7). To export a single table into multiple data packages, the R3load process requires a WHERE file that defines the range of records that should be extracted from the table.

In most cases, the source database contains a small set of large tables. You can export these tables in parallel to multiple package files where each file contains the data of one table.

For very large tables, performance can be improved even further by exporting and importing a single table with multiple R3load processes in parallel.

SAP provides several tools for splitting packages or tables:

- Package Splitter

  This splits tables from existing structure files. You can explicitly specify the tables that should be split into separate structure files. You can also split tables into separate packages if their size exceeds a configurable threshold. Other split options are also available.

- R3ta

  This tool can generate multiple WHERE conditions for a table, which can then be used to export the data of this table with multiple R3load processes in parallel. Each R3load process requires a WHERE condition to select only a subset of the data in the table.

- SAPInst

  This tool can be used for the table splitting preparation. It invokes R3ta and the Package Splitter, and also automates some of the tasks that would otherwise need to be performed manually.

4

Export optimization techniques

This chapter describes different techniques to improve export performance during a heterogeneous system copy or Unicode conversion. We performed tests with these techniques and different options that are available for the corresponding tools. The test results are presented together with our recommendations.

4.1 A closer look at cluster tables and table clusters

Table clusters in many cases determine the overall migration time. This is due to the fact that they must be exported in a sorted manner when code page conversions are performed together with the fact that some of the table clusters are very large (for example, typical candidates are CDCLS, EDI40, and RFBLG).

In this section we discuss cluster tables and table clusters. The basics in this section can help you understand why table clusters must generally not be unloaded unsorted in certain situations, in particular when a Unicode conversion is performed.

This fact leads to a major pain point from the export point of view. The reason is that the sorted unload of table clusters can have runtimes that are by far higher than those of an unsorted export.

This is especially true if the source table and indexes are highly fragmented. With a Unicode conversion added, the export time can be long enough to consume a large portion of the entire downtime window.

Table 4-1 shows the effect of an unsorted export for a large table.

Table 4-1 Example from a customer migration (non-Unicode)

Table name	**CDCLS**
Number of rows	63,422,261
Sorted unload [hh:mm]	34:40
Unsorted unload [hh:mm]	03:28

Now we take a closer look at how cluster tables and table clusters are defined and what the implications are for Unicode conversions.

A SAP system stores table definitions in the data dictionary table DD02L. The structure of this table is shown in Figure 4-1.

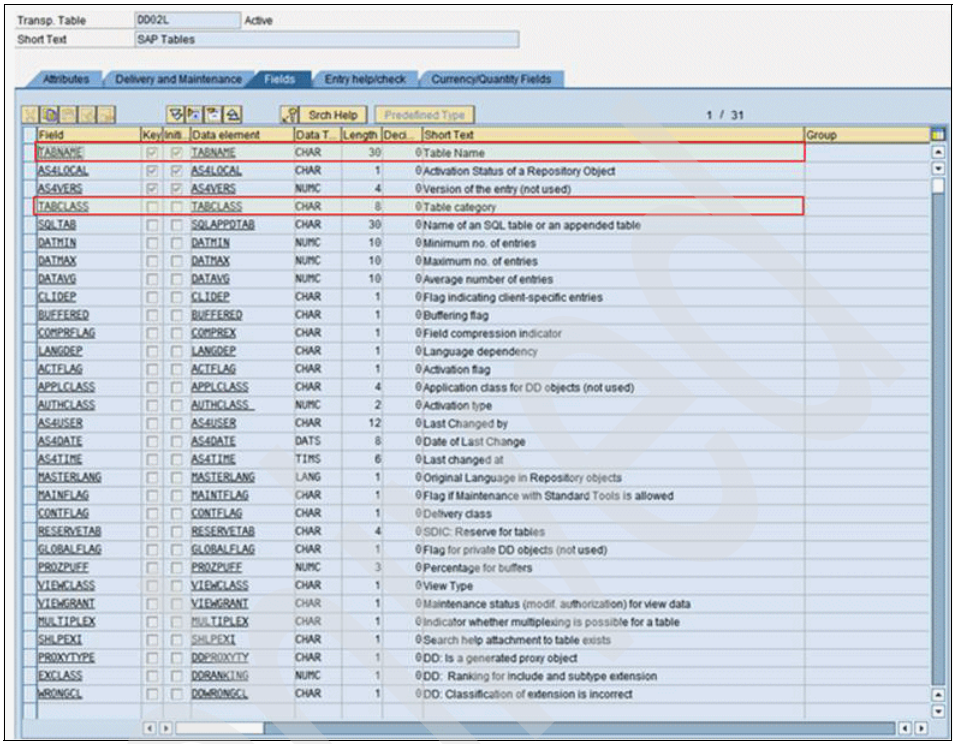


Figure 4-1 Structure of table DD02L

One important attribute of a table is the table class (field: TABCLASS). If you want to know how many tables are associated with a specific table class run the following SQL statement as the SAP database connect user (for example, `sap<sapsid>`).

```
SELECT tabclass, count(tabname) AS No_of_TABs_in_class
FROM DD02L
GROUP BY tabclass ORDER BY 2 DESC
```

For an SAP ERP 2004 IDES system the result looks like that shown in Example 4-1.

Example 4-1 Table class distribution for ERP 2004 IDES

```
TABCLASS  NO_OF_TABS_IN_CLASS
-------   -------------------
INTTAB                 127468            => Structures
TRANSP                  47078            => transparent tables (*)
```

```
VIEW                    30649            => views
POOL                     2022            => pool tables (*)
APPEND                   1114            => append structures
CLUSTER                    79            => cluster tables (*)

(*) contains data
```

Only the table classes TRANSP, POOL, and CLUSTER store data within database tables.

Tables that have a table class CLUSTER are called cluster tables. They are physically stored in a table that is called a table cluster. The definitions of these table clusters are stored in the data dictionary table DD06L.

To get a list of the cluster tables and their associated table cluster, use the following SQL statement:

```
SELECT tabname AS Cluster_Table, sqltab AS Table_Cluster
from dd02l
where tabclass='CLUSTER'
ORDER BY sqltab, tabname
```

The resulting list, shown in Example 4-2, is based on an SAP ERP 2004 IDES system.

Example 4-2 List of cluster tables with associated table cluster

```
CLUSTER_TABLE                  TABLE_CLUSTER
------------------------------ ------------------------------
AUAA                           AABLG
AUAB                           AABLG
AUAO                           AABLG
AUAS                           AABLG
AUAT                           AABLG
AUAV                           AABLG
AUAW                           AABLG
AUAY                           AABLG
CDPOS                          CDCLS
PCDPOS                         CDCLS
TACOPC                         CLU4
TACOPCA                        CLU4
TAB1                           CLUTAB
TAB2                           CLUTAB
CVEP11                         CVEP1
CVEP12                         CVEP1
CVEP13                         CVEP1
CVEP14                         CVEP1
TACOPA                         CVEP1
TACOPAB                        CVEP1
TACOPAC                        CVEP1
TACOPAD                        CVEP1
```

CVEP21	CVEP2
CVEP22	CVEP2
CVEP23	CVEP2
CVEP24	CVEP2
CVER11	CVER1
CVER12	CVER1
CVER13	CVER1
CVER14	CVER1
CVER15	CVER1
CVER21	CVER2
CVER22	CVER2
CVER23	CVER2
CVER24	CVER2
CVER25	CVER2
CVER31	CVER3
CVER32	CVER3
CVER33	CVER3
TACOPB	CVER3
TACOPBA	CVER3
CVT1	CVER4
CVT2	CVER4
CVT3	CVER4
CVER50	CVER5
CVER51	CVER5
DOKTL	DOKCLU
DSYOL	DSYO1
DSYOT	DSYO1
DSYGH	DSYO2
DSYGI	DSYO2
DSYGL	DSYO2
DSYXN	DSYO3
DSYXO	DSYO3
EDID2	EDI30C
EDID4	EDI40
EDIDD_OLD	EDIDOC
GLIDXB	EPIDXB
GLIDXC	EPIDXC
GLS2IDX	GLS2CLUS
MMIM_PRED	IMPREDOC
KONV	KOCLU
REGUP	REGUC
BSEC	RFBLG
BSED	RFBLG
BSEG	RFBLG
BSES	RFBLG
BSET	RFBLG
BSSEG	RFBLG
MHND	RFMHN
SFHOT	SFHOA
T512U	T512CLU
TERMC1	TERCL
TERMC2	TERCL2
TERMC3	TERCL3
UMG_TEST_B	UMG_TEST_C

```
UMG_TEST_D                      UMG_TEST_C
UMG_TEST_G                      UMG_TEST_F
CVERI_CLNT                      VER_CLUSTR
  79 record(s) selected.
```

To check the number of table clusters and the associated number of cluster tables, use the following SQL statement:

```
SELECT sqltab AS Table_Cluster, count(tabname) AS Cluster_Tables
FROM dd02l
WHERE tabclass='CLUSTER'
GROUP BY sqltab
ORDER BY 2 DESC
```

You can associate a table cluster to one or more cluster tables.

The way the cluster tables and table clusters relate according to physical storage of data is that a logical row in a cluster table is mapped to one or more physical rows in the table cluster.

Let us have a closer look at the definition of the table cluster DOKCLU and its contents. See Figure 4-2.

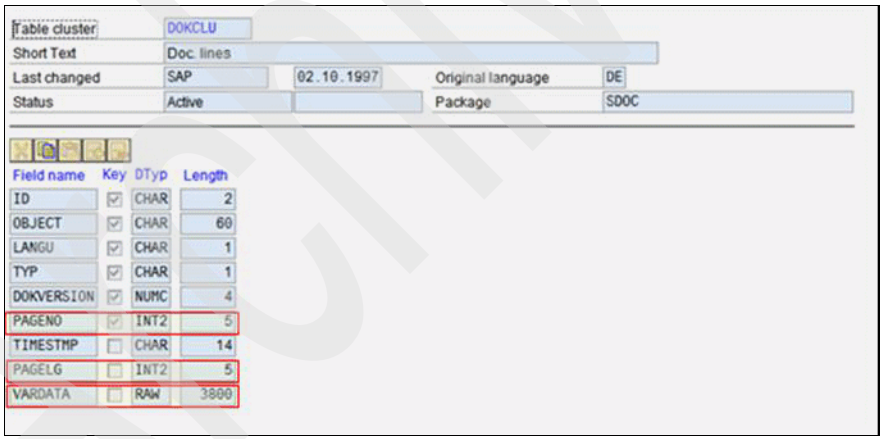


Figure 4-2 SE11 for table cluster DOKCLU

You can identify every logical row of a cluster table by the key fields of the table cluster. The data for the logical rows of a cluster table are stored in the column VARDATA. If the data is longer than the defined length of VARDATA, additional records for the same key are stored with an increased number of PAGENO. The field PAGELG informs you of the length of the data stored inside the VARDATA field.

The example in Figure 4-3 shows four records for the key DT RSMD_CONVERSION D E 0038 with an increasing PAGENO. The first three records are completely filled up with data in the field VARDATA. This can be seen from the PAGELG with a value of 3800, which matches the length of the VARDATA field. The fourth record with PAGENO 3 has a PAGELG of 60, meaning that this record is not filled up to the end.

Table: DOKCLU
Displayed Fields: 8 of 9 Fixed Columns: 6 List Width 0250

ID	OBJECT	LANGU	TYP	DOKVERSION	PAGENO	TIMESTMP	PAGELG
DT	RSMD_CONVERSION	D	E	0038	0	6)3UR6zzKQLkD4	3800
DT	RSMD_CONVERSION	D	E	0038	1	6)3UR6zzKQLkD4	3800
DT	RSMD_CONVERSION	D	E	0038	2	6)3UR6zzKQLkD4	3800
DT	RSMD_CONVERSION	D	E	0038	3	6)3UR6zzKQLkD4	60

Figure 4-3 Records of table cluster DOKCLU for one logical row

To handle cluster tables and the associated table clusters, SAP has created a cluster interface.

If we look into the mechanism that this interface is using when changing logical rows of a cluster table, we understand why the associated table clusters that are storing the data must not be unloaded unsorted when doing a Unicode conversion.

The following list shows the procedure that R3load uses when exporting a table cluster with Unicode conversion:

1. Reads all database rows belonging to one logical row (PAGENO is increased row by row. When PAGENO starts with 0 again, a new logical row starts.)
2. Concatenates RAW fields (for example, VARDATA)
3. Decompresses RAW fields
4. Identifies start and end offsets of character information
5. Identifies code page for character information.
6. Converts characters to Unicode.
7. Builds new RAW field content
8. Compresses RAW fields
9. Builds several database rows from the RAW field.
10. Writes database rows belonging to one logical row

Due to the Unicode conversion (or other code page conversions, for example, EBCDIC to ASCII), the contents and the length of the records may change. Even the number of the physical records belonging to a logical record may change.

Because the physical records are built together to a logical record, the data must be read in a sorted way to find all physical records that belong to a logical record. Therefore, an unsorted unload is not supported.

If no changes to the contents of the records are made (no code page conversion), the logical record does not have to be constructed and the table cluster can be unloaded unsorted.

4.2 Unsorted versus sorted export

There are two different ways to export the data from the source database:

- Sorted export
- Unsorted export

With a sorted export, the pages of a table are read in the sequence of the primary key. If the cluster ratio is not optimal, data pages will not be read continuously, but there will be mechanical movements of the disk heads on the spindles. In addition, database sort operations may occur, which will also waste time, and the export runtime will not be optimal.

Note: The unsorted export is one of the most powerful optimizations to improve the export time. However, the data is also imported unsorted and thus the database performance may not be optimal. You may see inefficient access to the disks, and in the worst case the DB2 optimizer may choose a wrong access plan. Use the unsorted export wisely and plan for a subsequent reorganization of unsorted tables.

With an unsorted export, the pages are read continuously and export time will decrease in most cases compared with sorted export. This improvement can be dramatic, so we recommend unsorted export for most of the tables. However, not all tables can be exported in this way. For exceptions, see SAP Note 954268.

Note: If a code page conversion is performed, table clusters must be unloaded sorted. As of R3load 6.40 patch level 55 (compile date February 10, 2006) and R3load 7.00 patch level 10 (compile date February 10, 2006), R3load automatically ensures that table clusters are exported in sorted order during a code page conversion.

4.2.1 Enabling unsorted export

The mode of export is controlled by the ORDER_BY_PKEY keyword in the *prikey* section of the DDL<DBS>.TPL file (Figure 4-4).

```
prikey: AFTER_LOAD ORDER_BY_PKEY
seckey: AFTER_LOAD

cretab: CREATE TABLE &tab_name
        ( /{ &fld_name &fld_desc /-, /} )
        TABLESPACE &location
        STORAGE (INITIAL     &init
                 NEXT        &next
                 MINEXTENTS  &minext
                 MAXEXTENTS  &maxext
                 PCTINCREASE &pctinc )
```

Figure 4-4 Example of DDL<DBS>.TPL file

If the keyword ORDER_BY_PKEY is deleted, the unloading is done unsorted.

When using the MigMon, you may use the `ddlMap` option in the export properties file, which names a file that contains the mapping between the package names and the DDL template files.

Example 4-3 shows an example of a mapping file. The DDL<DBS>_LRG.TPL template file that does not contain the ORDER_BY_PKEY keyword in the prikey section is generated by R3ldctl in addition to the DDL<DBS>.TPL file. The packages SAPCLUST and SAPSDIC are using the DDL<DBS>.TPL template file. The table MSEG (which has a package of its own) and the package SAPAPPL1 are unloaded by using the DDL<DBS>_LRG.TPL template file.

Example 4-3 Example of a MigMon DDL template mapping file

```
[ SORTED UNLOAD ]
# DDL file for sorted unload
ddlFile = <path>/DDL<DBS>.TPL
#package names
SAPCLUST
SAPSDIC
[ UNSORTED UNLOAD ]
# DDL file for unsorted unload
ddlFile = <path>/DDL<DBS>_LRG.TPL
# package names (may also contain only one table)
MSEG
SAPAPPL1
```

Note: When using DistMon, the packages that should be exported unsorted can also be controlled. This can be done by the parameter unsortedExport or the file unsorted_export.txt if the option is not specified.

4.2.2 Runtime comparison

In our test environment the export was done for the tables shown in Figure 4-5.

Table	No. of Rows	Size [MB]
ACCTIT	11.505.376	9.622
BKPF	17.520.379	9.516
BSIS	20.769.172	9.466
COEP	37.411.485	9.427
GLFUNCA	23.142.069	12.417
GLPCA	34.948.249	11.407
SWWLOGHIST	14.872.407	9.634
CDCLS	8.241.290	6.116
RFBLG	3.648.290	9.818
SOC3	40.666.845	9.443

Figure 4-5 Tables for export comparison

The tables were exported with the following options:

- Non-Unicode with sorted export
- Non-Unicode with unsorted export
- Unicode conversion with sorted export
- Unicode conversion with unsorted export (which is ignored for cluster tables)
- Four parallel processes were used for each case.

The results are shown in Figure 4-6.

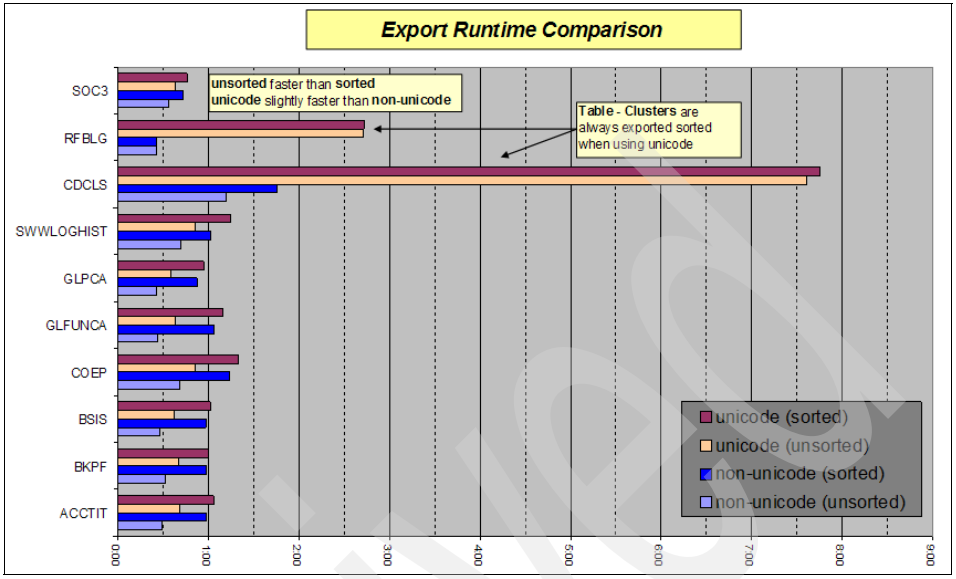


Figure 4-6 Export runtime comparison results

From Figure 4-6 we can conclude the following:

- Unsorted export is faster than sorted export for transparent tables.
- Non-Unicode export is in average faster than Unicode conversion export.
- Unicode conversion export of table clusters is much slower than non-Unicode export.

Table clusters in many cases determine the overall migration time. This is due to the fact that they must be exported sorted when converting to Unicode together with the fact that some of the table clusters are very large (for example, typical candidates are CDCLS, EDI40, and RFBLG).

To improve the export performance of those tables, they may be split into pieces using the table-splitting technique described in 5.1, “Table splitting” on page 62.

4.2.3 Resource consumption

During the export of the tables the resource consumptions were measured using NMON.

Figure 4-7 and Figure 4-8 on page 47 show the resource consumption for all four cases (non-Unicode/sorted, non-Unicode/unsorted, Unicode conversion/sorted, and Unicode conversion/unsorted).

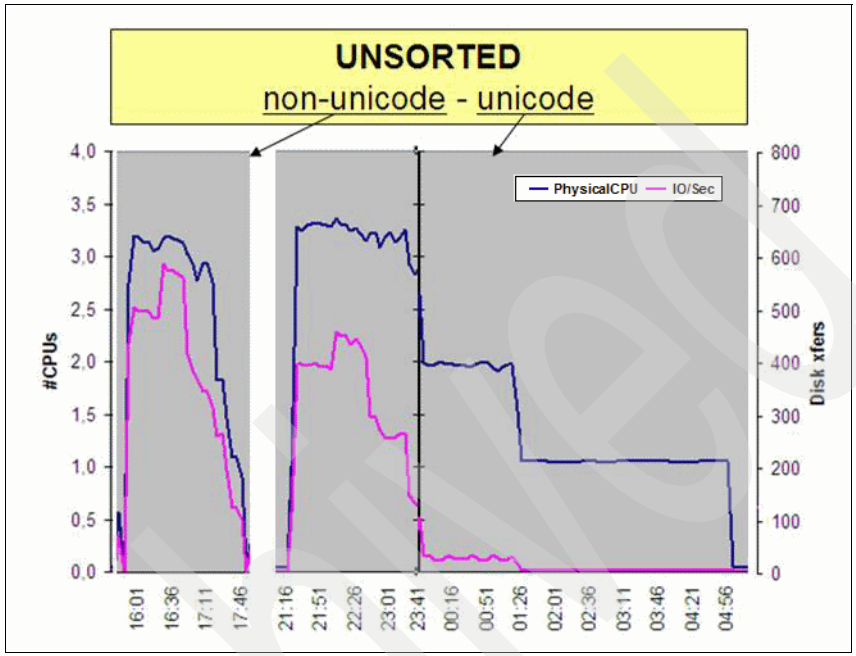


Figure 4-7 Resource consumption (CPU, I/O) for the unsorted export case

What are the most obvious differences of the two parts in these two figures? First of all, the runtime is much longer for the Unicode case. Even if we stated that the export should be performed unsorted, R3load exports the two table clusters RFBLG and CDCLS in a sorted manner. The CPU usage when running all four processes in parallel is quite similar. It is slightly higher for the Unicode case. The shrinkage of the CPU usage when only the table clusters remain can be clearly seen. The peek number of I/Os is smaller in the Unicode case, but spread over a longer time.

A remarkable result is the CPU usage starting at 23:41. The cluster tables RFBLG and CDCLS are running at this point in time and consuming two physical CPUs. At 01.26 only the table CDCLS is running and using one physical CPU. Seeing this, we can conclude that the export of such table clusters is fully using a single CPU and so the export runtime of those tables is also tightly related to performance of each single CPU for export.

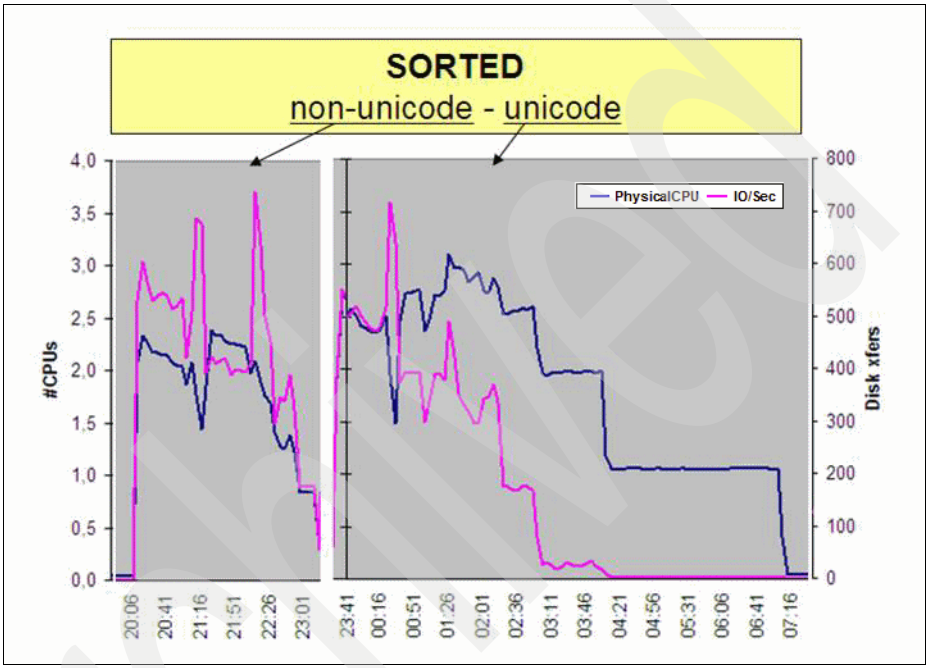


Figure 4-8 Resource consumption (CPU, I/O) for the sorted export case

What can be stated about Figure 4-8? AT first glance, the curves are less smooth in relation to the unsorted case. The difference in CPU usage between non-Unicode and Unicode case is bigger. The CPU consumption for the non-Unicode case is smaller and the I/O consumption is bigger.

Again, we can see the full usage of a single CPU by the two cluster tables in the Unicode export. For more information see 4.4, “Package splitting” on page 50, which gives more insight into the CPU usage during the export.

4.2.4 DB2 snapshot

Table 4-2 shows the results of database snapshots taken during a non-Unicode export, comparing unsorted and sorted exports. Most of the differences in the two snapshots are due to sort operations and use of temporary table space.

Table 4-2 DB2 snapshot comparison

	Sorted	Unsorted
Total Private Sort heap allocated	= 0	= 0
Total Shared Sort heap allocated	= 0	= 0
Shared Sort heap high water mark	= 0	= 0
Post threshold sorts (shared memory)	= 0	= 0
Total sorts	= 70	= 66
Total sort time (ms)	= 2528650	= 157
Sort overflows	= 2	= 0
Active sorts	= 0	= 0
Buffer pool data logical reads	= 12725094	= 14811069
Buffer pool data physical reads	= 6202833	= 6200044
Buffer pool temporary data logical reads	= 2438151	= 19
Buffer pool temporary data physical reads	= 1228390	= 0
Asynchronous pool data page reads	= 2799545	= 6165850
Buffer pool data writes	= 1218386	= 0
Asynchronous pool data page writes	= 1217750	= 0
Buffer pool index logical reads	= 2095225	= 5611
Buffer pool index physical reads	= 762257	= 614
Buffer pool temporary index logical reads	= 0	= 0
Buffer pool temporary index physical reads	= 0	= 0
Asynchronous pool index page reads	= 691405	= 3
Buffer pool index writes	= 0	= 0
Asynchronous pool index page writes	= 0	= 0
Buffer pool xda logical reads	= 0	= 0
Buffer pool xda physical reads	= 0	= 0
Buffer pool temporary xda logical reads	= 0	= 0
Buffer pool temporary xda physical reads	= 0	= 0
Buffer pool xda writes	= 0	= 0
Asynchronous pool xda page reads	= 0	= 0
Asynchronous pool xda page writes	= 0	= 0
Total buffer pool read time (milliseconds)	= 21442746	= 9702406
Total buffer pool write time (milliseconds)	= 4358579	= 0
Total elapsed asynchronous read time	= 9476897	= 9593233
Total elapsed asynchronous write time	= 3931277	= 0
Asynchronous data read requests	= 1977833	= 3086773
Asynchronous index read requests	= 384730	= 2
Asynchronous xda read requests	= 0	= 0
No victim buffers available	= 6972310	= 0
LSN Gap cleaner triggers	= 0	= 0
Dirty page steal cleaner triggers	= 450	= 0
Dirty page threshold cleaner triggers	= 10869	= 0
Time waited for prefetch (ms)	= 1946323	= 1023362
Unread prefetch pages	= 3474	= 0
Direct reads	= 4516	= 4460
Direct writes	= 0	= 0
Direct read requests	= 439	= 443
Direct write requests	= 0	= 0
Direct reads elapsed time (ms)	= 2463	= 2576
Direct write elapsed time (ms)	= 0	= 0
Database files closed	= 0	= 0
Host execution elapsed time	= 36376.929172	= 21345.196223

4.2.5 Conclusion and recommendation

Generally speaking, you should prefer an unsorted export whenever applicable. If you perform a Unicode conversion, you should check your big table clusters (for example, CDCLS, RFBLG, and EDI40) and consider a table splitting if they determine the overall export runtime.

We recommend the following:

- Use the MigMon with DDL-Mapping file when a combination of unsorted and sorted export should be used.
- Use the latest R3load that is available.
- If you are using R3load prior to Version 6.x or before compile date Feb. 10, 2006, refer to SAP Note 954268.
- Use unsorted export whenever applicable (at least for big transparent tables).
- Keep in mind that data is also imported unsorted and thus a subsequent reorganization may be needed.
- Unicode export times of table clusters are related to the performance of each single CPU and, therefore, new generation CPU should be used to export those tables (for example, by using application servers for those tables).

4.3 Export file size considerations

When dumping the exported data to files, consider the file size of both non-Unicode and Unicode conversion to better plan the disk space required.

Figure 4-9 shows the results for selected tables.

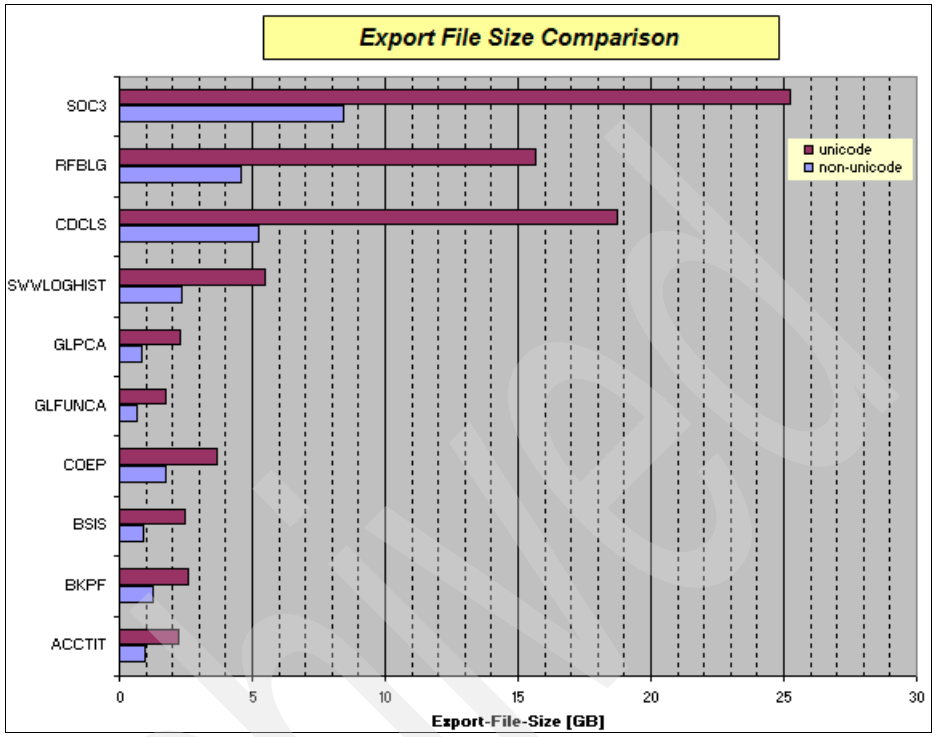


Figure 4-9 Export file size comparison

In case of non-Unicode, the size of the dump space can be estimated as 10–15% of the used database size.

The size of the export dump files in the case of a Unicode conversion exceeds those in the non-Unicode case by factors of 2 up to 3.5 in the above example. This means that you must provide at least two to four times more dump space in the case of a Unicode conversion. An increased amount of I/O also should be considered.

4.4 Package splitting

All tables in an SAP system are assigned to a data class (TABART). The relationship between a table and the data class is maintained in table DD09L. During the export preparation, R3ldctl generates one structure file (STR file) for each data class. Each structure file is processed by a single R3load process. You

can specify a size limit for the data packages that are generated by the R3load process in the corresponding cmd file, but all data packages of the structure file are created sequentially by one R3load process. Therefore, depending on the amount and size of tables defined in the structure file, the export can take a long time. Figure 4-10 shows the standard data classes usually used.

Type	TABART	Usage
Data	APPL0	Master data, transparent tables
	APPL1	Transaction data, transparent tables (big tables)
	APPL2	Organization and customizing
	USER, USER1	Customer data class
Special	CLUST	Cluster tables
	POOL	Pool Tables
System	SAUS	Exchange tables for Upgrades
	SDIC	ABAP Dictionary tables
	SDOCU	Documentation
	SLDEF	Repository switch (SAP Upgrade)
	SLEXC	Repository switch (SAP Upgrade)
	SLOAD	Screen and report loads
	SPROT	Spool and logs
	SSDEF	Repository switch (SAP Upgrade)
	SSEXC	Repository switch (SAP Upgrade)
	SSRC	Source of screens and reports
BW	DDIM	Dimension tables
	DODS	ODS, PSA tables
	DFACT	Fact tables

Figure 4-10 Data classes (TABART)

Usually, there are a lot of tables in the APPL data classes and some very large tables in data class APPL1.

When exporting such a system without splitting the structure files, the processing of packages that contain several large tables may dominate the total runtime of the export.

One solution is to split a single structure file into multiple files with the package splitting tool. There are two different versions available:

- Perl based
- Java™ based

Since only the Java tool is to be maintained in the future, this is the tool of choice.

The Java STR splitting tool provides the following features:

- Split out the largest <*n*> tables into separate structure files. Each file contains a single table.
- Split tables that exceed a defined size limit into separate structure files. Each file contains a single table.
- Split structure files for which the tables exceed a defined size limit. Each file may contain one or more tables.
- Split tables into separate structure files by specifying their names in an input file. Each line of this input file contains a single table name.

You can invoke the package splitter manually or by using SAPInst.

The parameters that are required to control the program flow can be supplied on the command line or by defining them in a file named `package_splitter_cmd.properties`.

Table 4-3 shows the options that are valid according to the Package Splitter User's Guide, which can be found in the SAP Service Marketplace at **service.sap.com/swdc** → **Support Packages and Patches** → **Additional Components** → **System Copy Tools** → **System Copy Tools 7.00** → **#OS independent**.

Table 4-3 Package splitter options

Option	Description	Comments
strDirs	List of STR file directories	Separator on Windows: ";" Separator on UNIX: ":"
extDirs	List of EXT file directories	Separator on Windows: ";" Separator on UNIX: ":"
outputDir	Output directory	If missing, the directories that contain the corresponding STR/EXT files are used.
top	Maximum number of tables	Largest N tables are extracted from packages.
tableLimit	Table size limit in MB	All tables larger than *tableLimit* are extracted from packages.
packageLimit	Package size limit in MB	All packages larger than *packageLimit* are split into packages smaller than this limit.

Option	Description	Comments
tableFile	File with the table names that are to be extracted	All tables from the file are extracted from the packages. This file must contain table names on separate lines (one name on each line).

On a UNIX system you can execute the package splitter as shown in Example 4-4.

Example 4-4 Command-line options for running the package splitter manually

```
./str_splitter.sh
        -strDirs /export_dump/DATA –extDirs /export_dump/DB/DB6
        -outputDir /split_output
        -top 20 –tableLimit 50 –packageLimit 200 –trace all
```

Before you start the export, you can optionally define the order in which the packages should be exported. This can be done by specifying the package names in a file that is provided to MigMon. You typically start the export beginning with those packages that contain the largest tables.

If you want to know how many tables belong to each data class in your system, you can use the SQL statement shown in Example 4-5.

Example 4-5 SQL statement to query number of tables for each TABART

```
SELECT tabart, count(tabart) AS "Tabs in TABART"
FROM dd09l
GROUP BY tabart
```

Example 4-6 is an example query output where the query was executed on an SAP ERP 2004 IDES system.

Example 4-6 Result set of SQL query for TABARTs and the number of tables

```
TABART TABS_IN_TABART
------ --------------
                  966
APPL0           13142
APPL1            7639
APPL2           25892
DDIM                1
DFACT               2
DODS                4
SDIC              337
SDOCU             112
```

```
SLEXC              11
SLOAD              37
SPROT             223
SSEXC             275
SSRC              174
USER              219
```

Some of the data classes, for example, CLUST and POOL, are missing in Example 4-6 on page 53. Those tables are included in the 966 tables without a data class in DD09L. R3ldctl assigns the data class to those tables based on the table class to which they belong. The corresponding STR file contains a valid table class for these tables. Another class of objects that have no TABART assignment in table DD09L are views. They are placed in the package SAPVIEW.STR.

4.5 Export server scenarios: Local or remote

Exporting the source system, especially in the case of Unicode conversions, consumes a lot of CPU resources.

Starting the export from one or more dedicated application servers is an option to free resources on the SAP database server. This section describes the export tests performed on the database server as well as on a remote application server. A setup from a customer migration can be found in 11.1, “Architecture overview” on page 238.

For more information regarding the code page specified during the export, see also 10.4, “Big and little endian” on page 233. It is important that the code page matches the endianness of each server where import processes are started.

4.5.1 Test setup

All exports for the following tests were performed with a conversion to Unicode, as the conversion adds additional load to the CPU.

For our test, we used the tables listed in Figure 4-5 on page 44. We used an unsorted export as the default for all tables. Nevertheless, table CDCLS was automatically exported in a sorted manner, as it is a table cluster.

The exports were performed using two different system setups:

- Exporting from the SAP database server

  The exporting R3load is started on the same machine where the DB2 database resides and thus competes with the database for system resources.

- Exporting from an SAP application server

  The exporting R3load is started on an SAP application server on a separate machine. The application server is remotely connected to a DB2 database. Export dump files are written to the application server machine. With this setup, R3load and DB2 use their own distinct system resources.

  Figure 4-11 illustrates these two test scenarios.

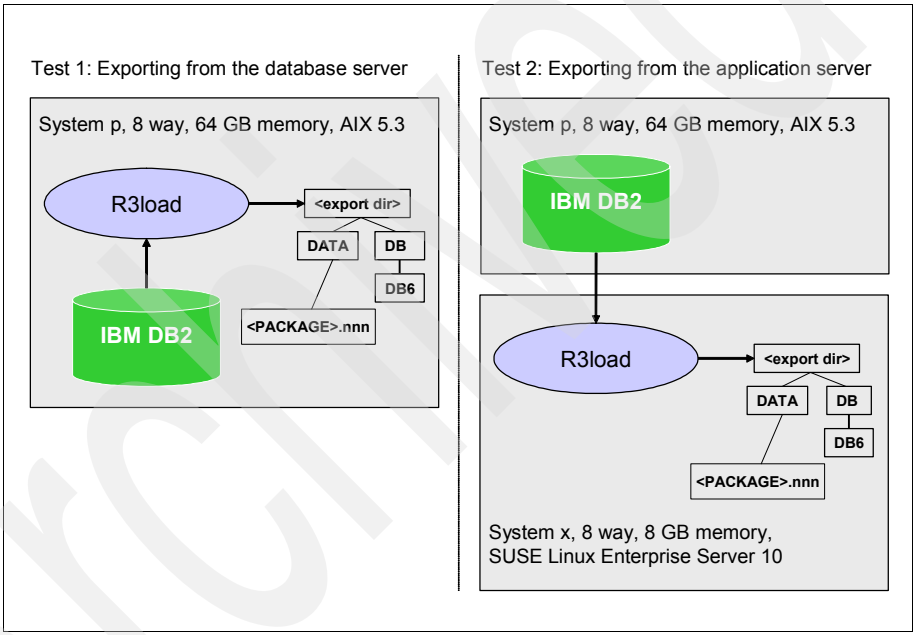


Figure 4-11 Export test scenarios

4.5.2 Test results

This section describes the test results for an export test. We compare the results of an export on the database server with the results of an export on dedicated application servers.

Exporting from the SAP database server

Figure 4-12 shows the export duration of each table involved. We see that the runtime for all tables except CDCLS was around one hour, whereas CDCLS export took nearly eight hours.

This difference can be explained with the mandatory sorted export of table clusters during Unicode conversions. This table is a potential candidate for the table splitting feature that is described in 5.1, “Table splitting” on page 62.

package	time	size MB	08-11 14:00	08-11 15:40	08-11 17:20	08-11 19:00
COEP	1:08:57	1033.29				
GLFUNCA	0:59:18	574.65				
BKPF	0:58:17	692.02				
BSIS	0:56:53	798.42				
GLPCA	0:53:14	422.71				
CDCLS	7:53:09	6921.85				
SWWLOGHIST	1:07:45	1605.92				
ACCTIT	1:00:16	689.09				

Figure 4-12 Table export runtimes

If we look into the CPU resource consumption (Figure 4-13), we clearly can see that R3load is using the main part of the available CPU, whereas DB2 is only using a minor part.

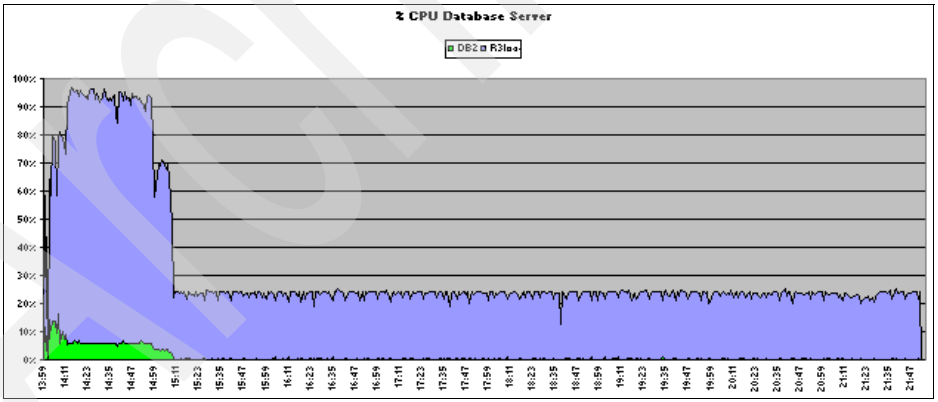


Figure 4-13 CPU resource consumption: SAP database server

We can see that at the beginning of the export, when eight R3load processes were running in parallel, the CPU utilization was nearly 100%. If the workload generated by R3load could be moved to another machine, this would free CPU resources on the DB server.

Exporting from an SAP application server

Figure 4-14 shows export runtimes. We can see that they decreased when comparing them with the ones in “Exporting from the SAP database server” on page 56. All tables except CDCLS were exported within 45–50 minutes, while CDCLS took 4.5 hours.

package	time	size MB
CDCLS	4:35:34	6921.39
COEP	0:49:31	1033.29
GLFUNCA	0:48:01	574.65
ACCTIT	0:46:28	689.09
SWWLOGHIST	0:46:27	1605.92
BSIS	0:45:13	798.42
BKPF	0:43:33	692.02
GLPCA	0:40:27	422.71

Figure 4-14 Table export runtimes

Figure 4-15 and Figure 4-16 on page 58 show the CPU resource usage for the SAP database server as well as for the separate application server.

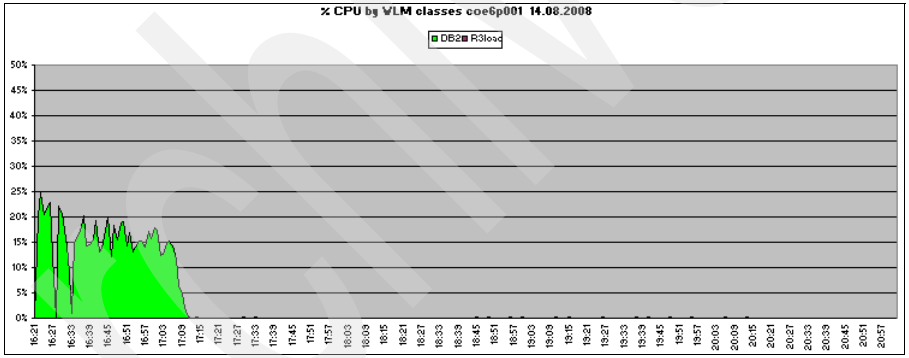


Figure 4-15 CPU resource consumption: SAP database server

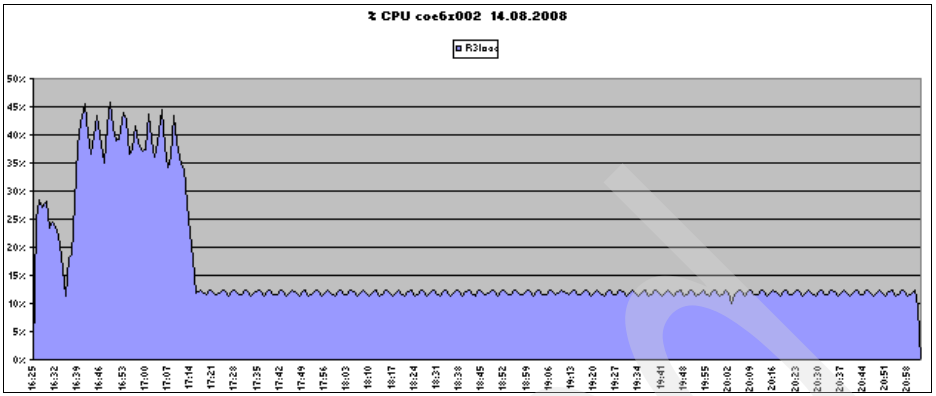


Figure 4-16 CPU resource consumption: SAP application server

When you compare the CPU utilization on the database server to the previous test performed in “Exporting from the SAP database server” on page 56, you can see that more CPU resources could be used by the DB2 database engine. This is due to the fact that the CPU load that was generated by R3load was completely moved to the application server.

4.5.3 Conclusion

If CPU resources become a bottleneck on the exporting machine, it makes sense to move the workload that is generated by R3load onto one or more dedicated SAP application servers.

When running R3load from the application server, more CPU resources could be used by DB2 on the SAP database server. In our test case, this helped to improve export runtimes, as shown in Figure 4-17.

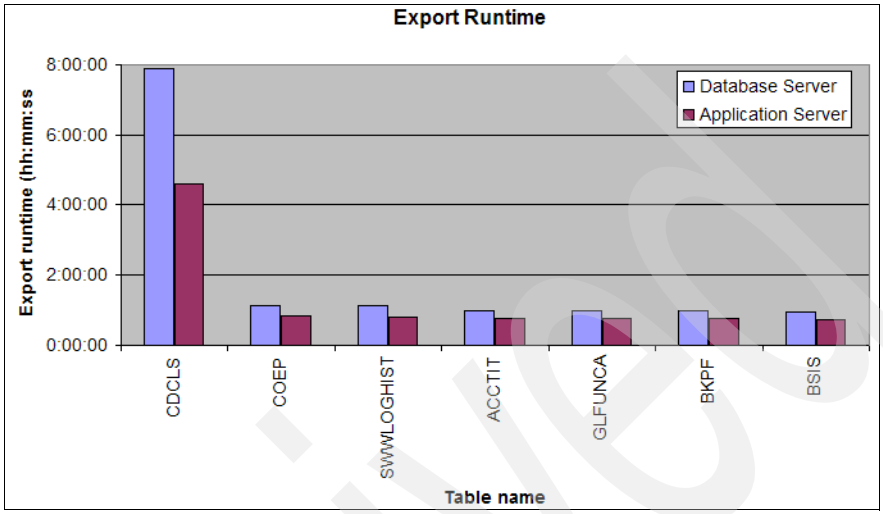


Figure 4-17 Export runtime tables

Keep in mind that when exporting from an application server, data must be transferred over the network. Ensure that the network connection does not turn into a bottleneck.

5

Advanced optimization techniques

This chapter describes advanced migration options. These options improve the migration throughput but must be planned and executed carefully. While the table splitting option is commonly used in large migrations, the socket transfer option is not widely used, as the operation has some risks. We describe those risks and also the benefits of the method in this chapter. For the table splitting option, we also highlight the specifics related to DB2 that should be taken into account when planning and executing this method.

5.1 Table splitting

The export and import runtime of very large tables can become a bottleneck during a heterogeneous system copy or a Unicode conversion. As introduced in 2.2.2, “Table splitting” on page 9, the contents of a single table can be exported and imported with multiple R3load processes in parallel. This *table splitting* approach is used for very large tables to improve the export and import runtime.

We performed several tests with the table splitting tools to figure out how the different options of exporting and importing a table effect performance.

5.1.1 How to split tables

To be able to export a single table with multiple parallel R3load processes, each process requires a specific WHERE file that specifies a distinct range of records in the source table. This set of records will be exported into a dedicated package file by a R3load process. You can create the WHERE files manually or with the available tools R3ta and where_splitter within SAPInst. Figure 5-1 illustrates the table splitting process.

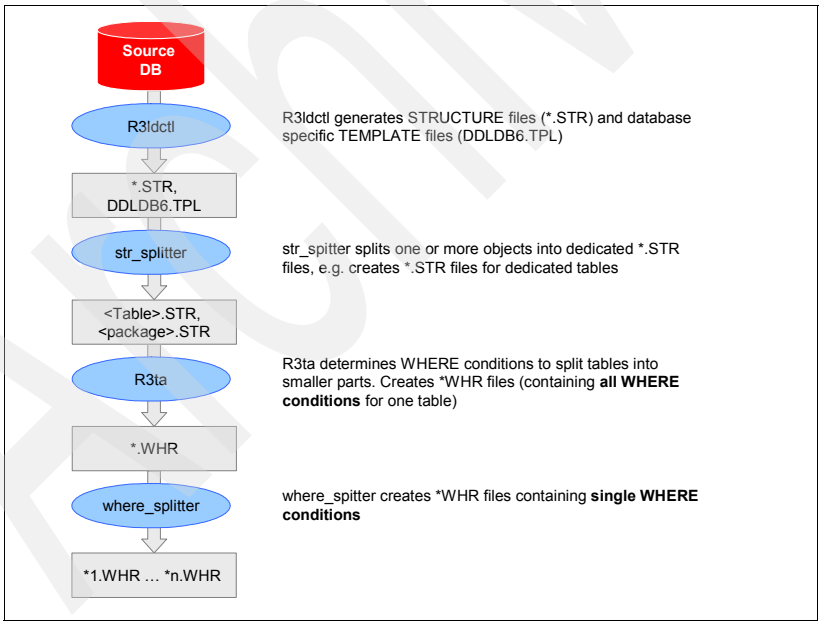


Figure 5-1 Table splitting: Part of overall process

The overall process is to generate the structure files by R3ldctl and use the package splitter to put the tables to dedicated STR files. Although the process in

principle works also when the table is included in a STR file that contains multiple tables, we recommend having each table in its dedicated STR files. Executing R3ta then determines the WHERE clauses and generates one WHR file for this table. It also generates the required files to create a temporary index on the source table to improve the export throughput.

You must split the generated WHR file into multiple WHR files, for example, using the where_splitter that is part of the str_splitter package (contained in the SAP archive SPLIT.SAR).

Using R3load you can create a temporary index on the table. R3load supports creation of the index using the STR file that R3ta generates.

Optionally, reorganize the table using the index. One of the important findings of our test was that the cluster factor of the index can strongly affect performance of unsorted exports. If the cluster factor is low, check whether it is possible to reorganize the table before starting the export.

The migration monitor finally uses those files to export the table in parallel.

In the following section we provide details about the configuration and setup of the process. The goal of this section is to discuss and explain the DB2-related optimizations and recommendations. Therefore, check the SAP information sources for details and the latest changes.

R3ta: Generate WHR files

To generate WHERE conditions, R3ta requires the following information as input:

- Name of the STR files that contain the structure definition of the tables that should be split
- Names of tables that should be split and hence which WHERE conditions should be generated
- Maximum number of rows in each data package or maximum number of data packages that should be generated for the table

Based on these parameters, R3ta connects to the source database and calculates appropriate WHERE conditions that you can use to split the tables in multiple packages. R3ta generates the following output:

- One WHR file for each table to be split. The WHR file typically contains multiple WHERE conditions. You use the WHR splitter to split the generated WHR file into multiple files by specifying the WHR Splitter Options.
- Task files for creating and dropping a temporary index, which should be created on the table before exporting the data.
- cmd files as input for R3load to create or drop the temporary index.
- STR files with the structure definition of the temporary index.

R3ta is part of the SAP kernel. It requires a *R3ta_hints.txt* file in the directory for which it is called. R3ta_hints.txt contains table names and suggestions for suitable column names to use in WHERE conditions. Example 5-1 shows sample content of the R3ta_hints.txt file.

Example 5-1 Content of R3ta_hints.txt

```
…
BALDAT          LOG_HANDLE
BKPF            BELNR
BSAD            BELNR
…
CDCLS           CHANGENR
```

R3ta is started as follows:

```
R3ta -f  <strfile> -table <tablename>%<number of splits>
```

Or:

```
R3ta -f  <strfile> -table <tablename>:<number of records>
```

Where:

- <*strfile*> indicates the fully qualified path and name of the STR file.
- <*tablename*> indicates the name of the table to split
- <*number of splits*> determines into how many parts the table should be split.
- <*number of records*> determines the maximum number of records in each package.

Although the R3ta utility delivers good results for the WHERE conditions to ensure similarly sized export packages, it is possible to identify the best statements manually. This can be done by analyzing the table content and its data distribution.

Note: You must ensure that the WHERE clauses export the entire table. Therefore, we recommend comparing the actual number of records on the source and target database. This comparison must be done on the database level and not by analyzing the R3load logs.

After we specified the number of packages for our tests, R3ta created <*number of splits*>+1 WHERE conditions. This behavior is quite usual. The last condition selects only a few rows, so the majority of data will be split into <*number of splits*> parts.

When calling R3ta, the output looks similar to Example 5-2.

Example 5-2 Output of R3ta

```
coe6p001:nzwadm 121> R3ta -f /export/split/EXP/DATA/CDCLS.STR -table CDCLS%8
R3ta: START OF LOG: Tue Aug 26 10:57:57 2008
R3ta -f /export/split/EXP/DATA/CDCLS.STR -table CDCLS%8
sapparam: sapargv( argc, argv) has not been called.
sapparam(1c): No Profile used.
sapparam: SAPSYSTEMNAME neither in Profile nor in Commandline
INFO: connected to DB
Processing /export/split/EXP/DATA/CDCLS.STR
INFO: reading R3ta_hints.txt
hint: ACCTCR    AWREF
hint: ACCTIT    AWREF
[…]
hint: ZZJVA     GL_SIRID
INFO: closing R3ta_hints.txt
trying to determine selectiveness of key column CHANGENR
SELECT COUNT(DISTINCT CHANGENR) FROM CDCLS
==> 522325 distinct values
Processing table CDCLS
trying SELECT with 0 key columns
SELECT  COUNT(*) FROM CDCLS
trying SELECT with 1 key column
SELECT CHANGENR,  COUNT(*) FROM CDCLS GROUP BY CHANGENR ORDER BY CHANGENR
INFO: disconnected from DB
R3ta: END OF LOG: Tue Aug 26 11:04:58 2008
```

The WHR-file created by R3ta and containing the WHERE clauses looks similar to Example 5-3.

Example 5-3 WHR file containing WHERE clauses

```
tab: CDCLS
WHERE "CHANGENR" <= '0000083166'
tab: CDCLS
WHERE ("CHANGENR" > '0000083166') AND ("CHANGENR" <= '0000151848')
tab: CDCLS
WHERE ("CHANGENR" > '0000151848') AND ("CHANGENR" <= '0000218459')
tab: CDCLS
WHERE ("CHANGENR" > '0000218459') AND ("CHANGENR" <= '0000284568')
tab: CDCLS
WHERE ("CHANGENR" > '0000284568') AND ("CHANGENR" <= '0000350715')
tab: CDCLS
WHERE ("CHANGENR" > '0000350715') AND ("CHANGENR" <= '0000414803')
tab: CDCLS
WHERE ("CHANGENR" > '0000414803') AND ("CHANGENR" <= '0000480490')
tab: CDCLS
WHERE ("CHANGENR" > '0000480490') AND ("CHANGENR" <= '0000546673')
tab: CDCLS
WHERE "CHANGENR" > '0000546673'
```

Where splitter

As mentioned, the where splitter is used to split a single WHR file (which was generated by R3ta and contains multiple WHERE conditions) into several WHR files (each containing one WHR condition). The where splitter is started as follows:

```
./where_splitter.sh -whereDir <whr_dir>
   -strDirs <strdir> -whereLimit 1
   -whereFiles <whr_file> -outputDir <outputdir>
```

Where:

- <*whr_dir*> indicates the location of the WHR files generated by R3ta, and <*strdir*> the path to the STR files.
- <*whr_file*> is the name of the WHR file to split and <*outputdir*> indicates the location for the where splitter output.

Example 5-4 shows a where splitter output.

Example 5-4 Where splitter output

```
coe6p001:nzwadm> ./where_splitter.sh -whereDir /export/split/splits -strDirs
/export/split/EXP/DATA -whereLimit 1 -whereFiles CDCLS.WHR -outputDir
/export/split/EXP/DATA
```

```
+ /usr/java14/jre/bin/java -showversion -cp ./.:./split.jar
com.sap.inst.split.PackageSplitter -where -whereDir /export/split/splits
-strDirs /export/split/EXP/DATA -whereLimit 1 -whereFiles CDCLS.WHR -outputDir
/export/split/EXP/DATA
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
Classic VM (build 1.4.2, J2RE 1.4.2 IBM AIX build ca142-20061124 (SR7) (JIT
enabled: jitc))
File not found:
/mnt/nas_p001/export/split/splitter/package_splitter_cmd.properties
Processing of 'CDCLS.WHR' WHERE file: OK
```

Create temporary index

To speed up the export, an index on the split tables can be created with R3load using the cmd and TSK files that were generated by R3ta. Be aware that creating additional indexes on the production system can effect the workload on the system and may also result in different access plans for the queries that are part of the production workload. Therefore, this kind of export optimization should be handled with care.

The following command is executed from the directory where the files created by R3ta reside:

```
R3load -dbcodepage 1100 -i <table>_IDX.cmd -l <table>.log
```

> **Note:** The creation of an additional index in the source system can have side effects on performance and database optimizer behavior.
>
> Creating an additional index or reorganizing the table might not be possible during productive operation.

Before exporting data, we recommend checking the cluster factor of the index that is used by R3load if a WHR file is specified and the tables are exported unsorted.

If tables are being exported unsorted and if the cluster factor of the created index is low, a significant reduction in unload performance is possible. In this case, you may reorganize the table using the index before you start the export. Be aware that reorganizing the table using this new index may have side effects on the production system. The reorg will increase the workload on the system. It may also affect the cluster factors of other indexes and hence potentially affect performance of queries that are using these indexes. Therefore, the reorg must be handled with care.

For more details about the availability and the restrictions of the table splitting feature, see SAP Note 952514.

5.1.2 Testing table split

The table splitting tests were performed with tables GLPCA and CDCLS on the System p® test environment. The characteristics of the test tables were already described in 4.1, “A closer look at cluster tables and table clusters” on page 36.

We started the test cases by running a single R3load process to export the data of the table. In this case no WHERE clause was used. We then repeated the export with an increasing number of R3load processes (2, 4, and 8 parallel jobs) and used WHERE clauses to perform the export. On the target database, we imported the data packages using one or multiple R3load processes according to the number of data packages that were generated during the export.

Multiple data packages can only be imported in parallel using SQL INSERT instead of the DB2 LOAD utility. When importing in a single table with multiple R3load processes using option `loadprocedure fast LOAD_FORCED`, the table is locked exclusively by the first R3load process and all other R3load processes must wait. Since INSERT is slower than DB2 LOAD, we want to compare import performance of a single R3load process using DB2 LOAD and multiple R3load processes using INSERT.

Figure 5-2 depicts an overview of export/import tests with different degrees of parallelism.

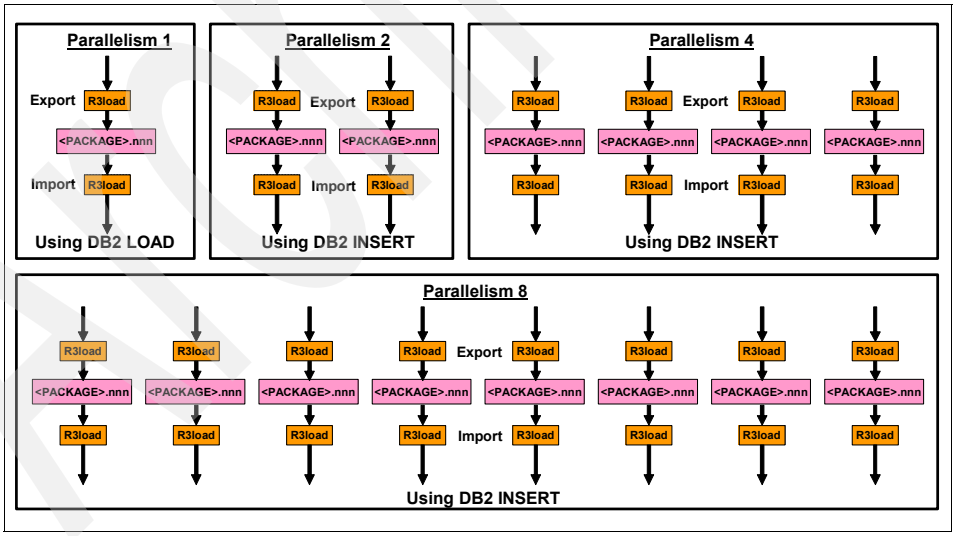


Figure 5-2 Overview of export/import tests with different degrees of parallelism

5.1.3 Results of export tests with table splitting: CDCLS

We started our export test based on the experience gathered from the tests in 4.4, “Package splitting” on page 50. Figure 5-3 shows the runtime of this non-optimized export.

package	time	size MB	08-11 14:00	08-11 15:40	08-11 17:20	08-11 19:00
COEP	1:08:57	1033.29				
GLFUNCA	0:59:18	574.65				
BKPF	0:58:17	692.02				
BSIS	0:56:53	798.42				
GLPCA	0:53:14	422.71				
CDCLS	7:53:09	6921.85				
SWWLOGHIST	1:07:45	1605.92				
ACCTIT	1:00:16	689.09				

Figure 5-3 Table export runtimes

During the test, we saw that the overall runtime was determined by table CDCLS (as this table cluster was exported in a sorted manner due to Unicode conversion). It is obvious that the overall export duration could be reduced by reducing the runtime of CDCLS, which was 7 hours and 53 minutes.

We tested whether splitting CDCLS can speed up the overall process.

In a first step, we performed standalone tests on table CDCLS, splitting this table into 1, 2, 4, and 8 packages and exporting it in parallel. For each split-off package, we started one R3load process.

Non-split export of CDCLS

In this standalone tests, a non-split export of CDCLS was slightly faster (7 hours and 35 minutes) than the one above, as there was no competing workload. See Figure 5-4.

package	time	size MB	08-26 19:30	08-26 21:10	08-26 22:50	08-27 00:30
CDCLS	7:35:06	6921.68				

Figure 5-4 CDCLS export runtime: Parallelism 1

From a CPU resource consumption perspective, the database server had 75% free capacity. See Figure 5-5.

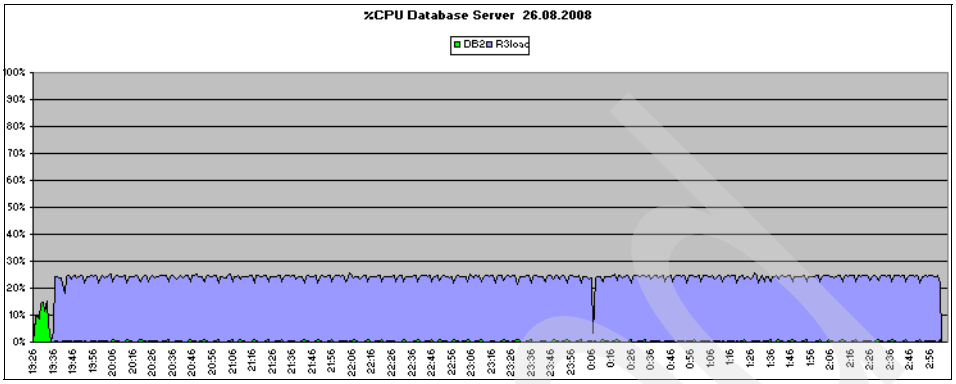
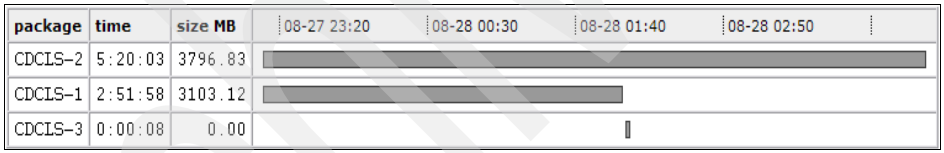


Figure 5-5 CDCLS export: Parallelism 1: CPU consumption

Split CDCLS: Parallelism 2

By splitting CDCLS into two parts, the export runtime already was reduced by 2 hours and 15 minutes to 5 hours and 20 minutes, as shown in Figure 5-6.



package	time	size MB
CDCLS-2	5:20:03	3796.83
CDCLS-1	2:51:58	3103.12
CDCLS-3	0:00:08	0.00

Figure 5-6 CDCLS export runtime: Parallelism 2

From a CPU resource consumption perspective, the database server had free capacity. See Figure 5-7.

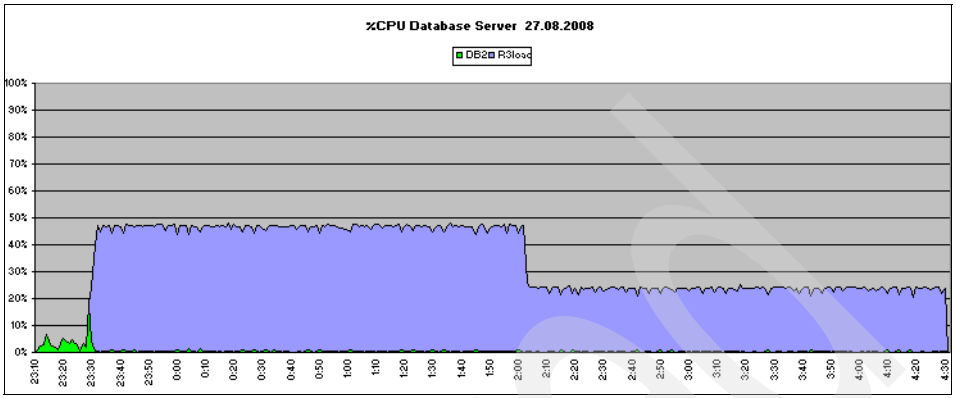
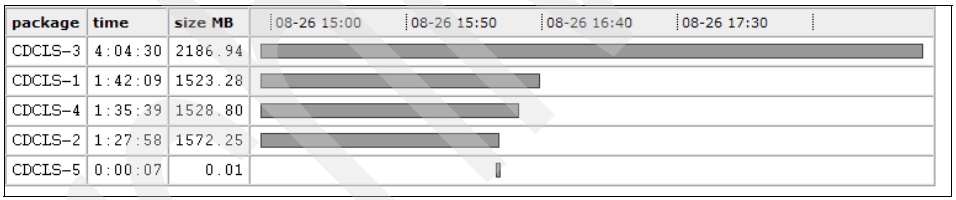


Figure 5-7 CDCLS Export: Parallelism 2: CPU consumption

Split CDCLS: Parallelism 4

Splitting CDCLS into four parts reduced the overall runtime even further to 4 hours and 5 minutes, as shown in Figure 5-8.



package	time	size MB
CDCLS-3	4:04:30	2186.94
CDCLS-1	1:42:09	1523.28
CDCLS-4	1:35:39	1528.80
CDCLS-2	1:27:58	1572.25
CDCLS-5	0:00:07	0.01

Figure 5-8 CDCLS export runtime: Parallelism 4

From a CPU resource consumption perspective, the database server was near its capacity limit. See Figure 5-9.

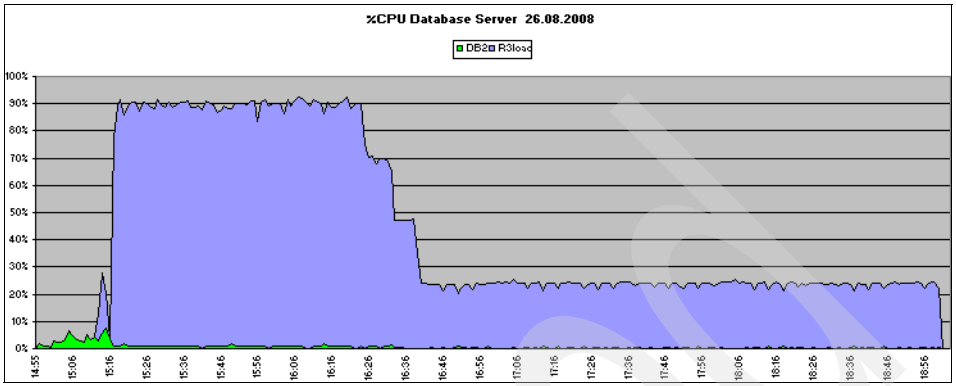
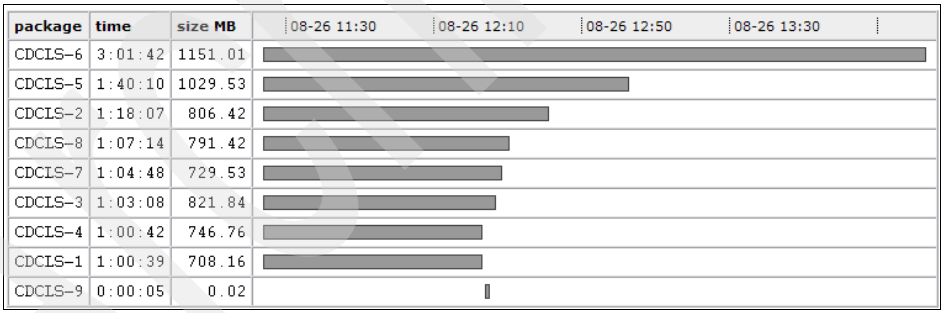


Figure 5-9 CDCLS export: Parallelism 4: CPU consumption

Split CDCLS: Parallelism 8

Splitting CDCLS into eight parts further reduced the overall runtime to 3 hours and 2 minutes, as shown in Figure 5-10. But from a CPU resource consumption perspective, the database server was at its capacity limit. It therefore would not make sense to start more R3load processes in parallel in our case. See Figure 5-11 on page 73.



package	time	size MB
CDCLS-6	3:01:42	1151.01
CDCLS-5	1:40:10	1029.53
CDCLS-2	1:18:07	806.42
CDCLS-8	1:07:14	791.42
CDCLS-7	1:04:48	729.53
CDCLS-3	1:03:08	821.84
CDCLS-4	1:00:42	746.76
CDCLS-1	1:00:39	708.16
CDCLS-9	0:00:05	0.02

Figure 5-10 CDCLS export runtime: Parallelism 8

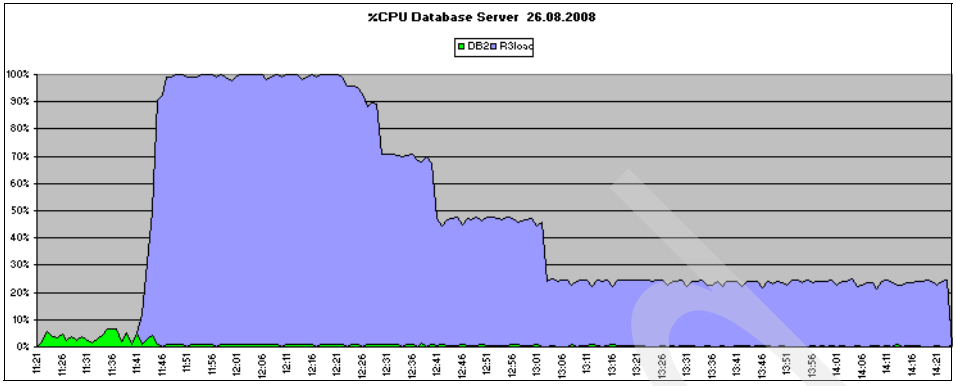


Figure 5-11 CDCLS Export; Parallelism 8: CPU consumption

Comparing test runs

When comparing the test with splitting CDCLS and exporting it with different degrees of parallelism, we get the result displayed in Figure 5-12. The runtime is reduced by running more R3load processes in parallel. However, runtimes for the table split scenarios are above their theoretical optimum value (see “Reference” in Figure 5-12).

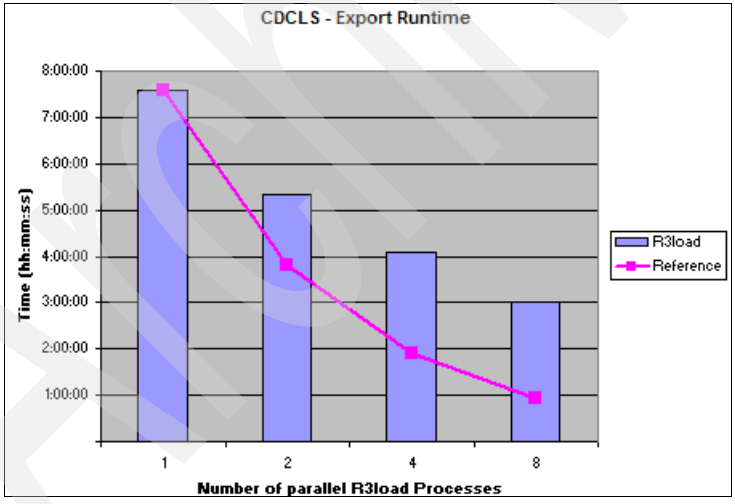


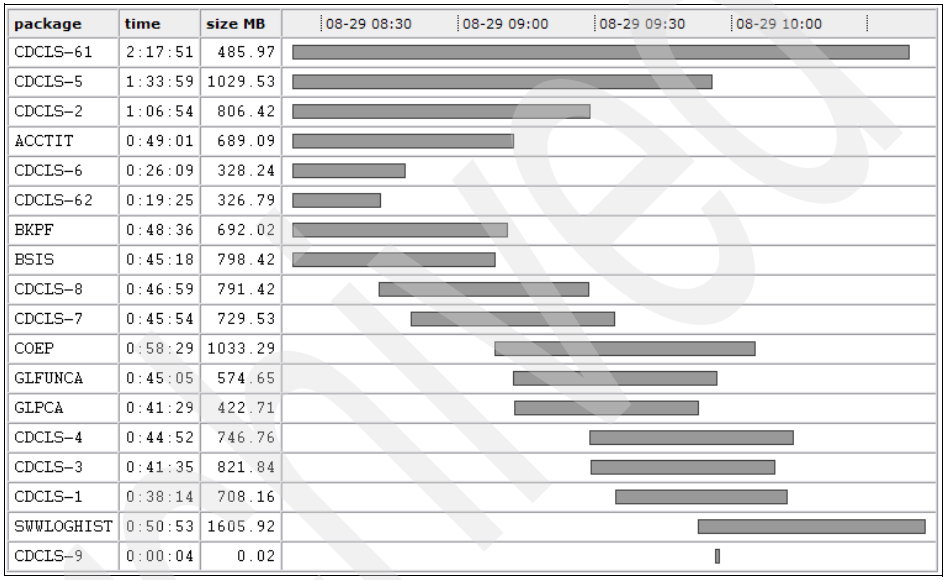
Figure 5-12 CDCLS export runtime summary

During all parallel tests, we observed that there was at least one package determining the overall runtime. By further splitting this package, the export duration could be further reduced.

Combined export

With the experience gathered from the previous tests, we started the export of the complete set of tables, while splitting table CDCLS into several parts.

We saw that even with splitting CDCLS into equal parts in respect to the number of rows by using R3ta, export runtimes of these packages are different. Therefore, we manually split package CDCLS-6 into two additional parts prior to the test and overall export was completed after 2 hours and 22 minutes. Figure 5-13 shows the runtime result of this test.



package	time	size MB
CDCLS-61	2:17:51	485.97
CDCLS-5	1:33:59	1029.53
CDCLS-2	1:06:54	806.42
ACCTIT	0:49:01	689.09
CDCLS-6	0:26:09	328.24
CDCLS-62	0:19:25	326.79
BKPF	0:48:36	692.02
BSIS	0:45:18	798.42
CDCLS-8	0:46:59	791.42
CDCLS-7	0:45:54	729.53
COEP	0:58:29	1033.29
GLFUNCA	0:45:05	574.65
GLPCA	0:41:29	422.71
CDCLS-4	0:44:52	746.76
CDCLS-3	0:41:35	821.84
CDCLS-1	0:38:14	708.16
SWWLOGHIST	0:50:53	1605.92
CDCLS-9	0:00:04	0.02

Figure 5-13 Table export runtimes: Optimized

We were able to reduce the overall runtime significantly when optimizing the export with table splitting. See Figure 5-14.

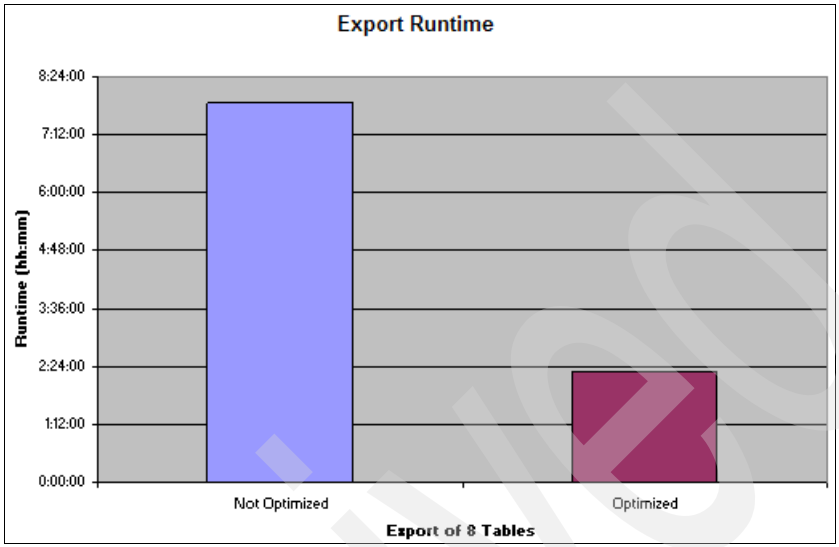


Figure 5-14 Table export runtimes comparison

5.1.4 Results of export tests with table splitting: GLPCA

The option to export a table in several packages is not only available during a code page conversion and a sorted export of a table. Although the unsorted export is the most powerful export optimization, it can be further improved with split export for the table.

Figure 5-15 summarizes the results of the export tests with table GLPCA. This table was exported unsorted and thus the index on the WHERE clause is important.

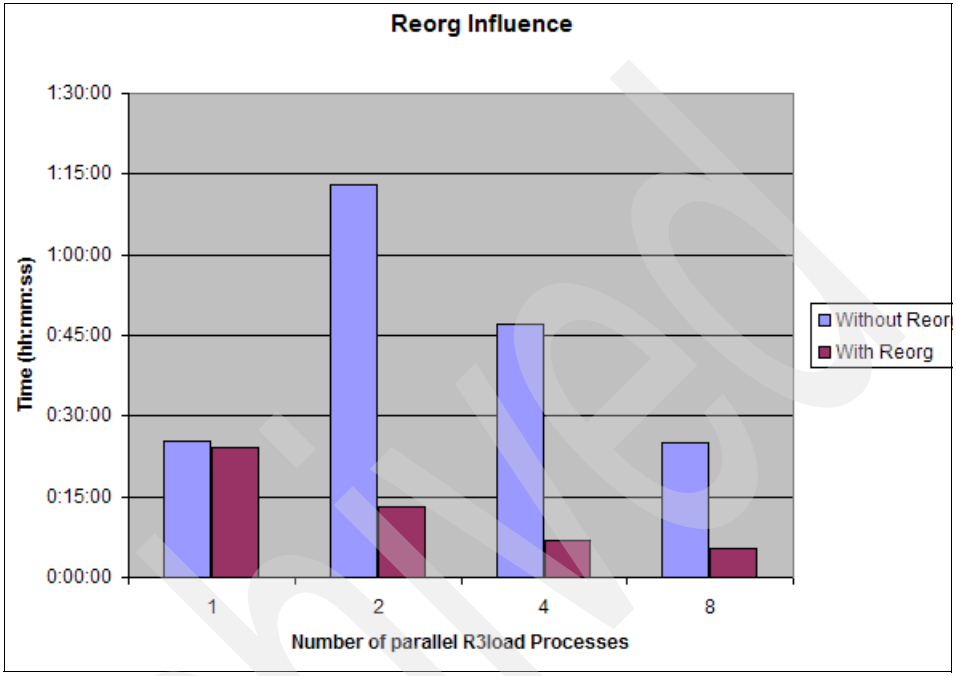


Figure 5-15 REORG influence on export time

The first test cases were executed before table GLPCA was reorganized according to this index. Before the REORG, the index had a cluster ratio of 4%, which resulted in poor performance in those test cases.

Reorganizing GLPCA according to this index resulted in a significant runtime reduction. Export performance improved by up to factor 6.7 (comparing four parallel exports) due to the REORG.

The second important result is that the export scales almost linearly when increasing the number of parallel jobs from 1 to 2 and 4 parallel jobs. When increasing from 4 to 8 parallel jobs, the performance gain was only factor 1.3. With eight parallel jobs the CPU utilization was 30% and I/O wait time was up to 50%. We therefore conclude that in the test case with eight parallel R3load jobs the system performance was bound by the limited I/O capacity of the storage.

The export performance of this table can be nearly doubled by exporting it unsorted, as shown in Figure 4-6 on page 45. The export can be further

improved by factors with using parallel export, as shown in Figure 5-15 on page 76.

5.1.5 Results of import tests with table splitting

R3load has several arguments to influence the behavior of how to import data. By default, R3load imports the data using SQL INSERT (DBSL INSERT). The first step toward improvement is to use a function called FAST INSERT, which is enabled with the R3load parameter "`-loadprocedure fast`".

The most powerful improvement is to use the DB2 LOAD API with R3load. This is described in detail in Chapter 7, "Import optimization" on page 109. Unfortunately, the LOAD API serializes the load on the table. This means that a split table cannot be imported in parallel when using DB2 LOAD. However, the LOAD API is usually by factors faster than the INSERT, and so the question is whether it is beneficial to use several parallel inserts instead of a serialized R3load with DB2 LOAD API.

Figure 5-16 shows the results of the import tests with table GLPCA using INSERT. In the first test cases we used SQL INSERT to load data and then we continued with DB2 LOAD. We also modified the order of data load and index build to figure out the fastest way of loading.

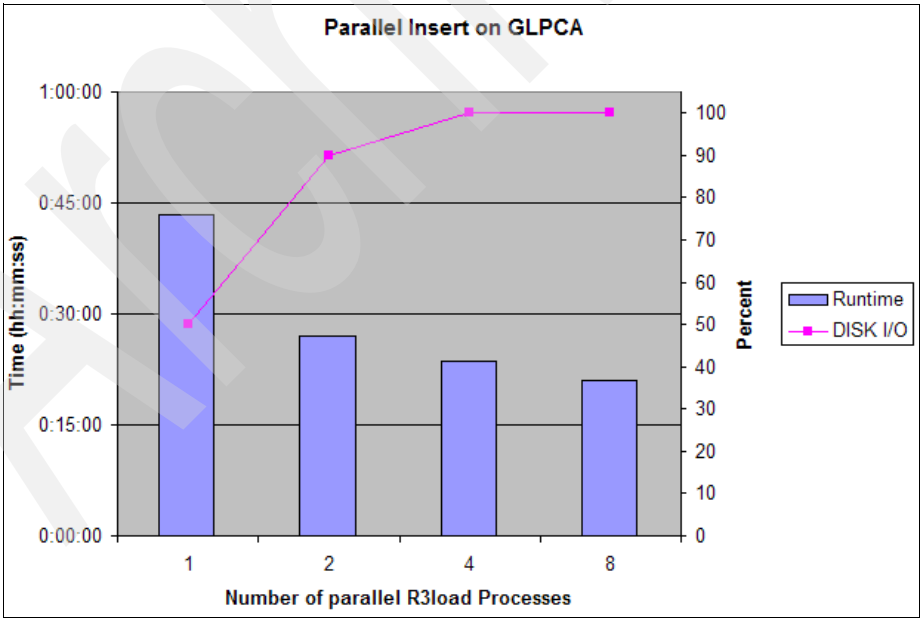


Figure 5-16 Import of GLPCA using parallel insert

A first test result, not documented in the chart, was that INSERT performance is better if the indexes are created up front. For the remaining INSERT tests we therefore first created indexes and then imported data. Next we tested FAST INSERT and found that this improves performance by 6% compared with the default DBSL INSERT. Based on this, we continued INSERT testing with different amounts of parallel R3load process (1, 2, 4, and 8). With an increasing number of parallel R3load processes, performance improved below linear scalability. For example, when doubling the amount of processes from two to four the performance improved only by 15%. But it is also obvious that the machine was already using 90% of the I/O throughput with two processes and running at 100% I/O usage with four and eight parallel processes.

Next we tested the DB2 LOAD. By default, the R3load will switch back to inserts even if the LOAD API is specified in the R3load load arguments when a split table is imported. To force the R3load to use the LOAD API, the following load argument must be specified:

```
-loadprocedure fast LOAD_FORCED
```

Alternatively, you can use the environment variable `DB6LOAD_FORCE_LOAD` to force R3load using the LOAD API. For more information see 7.2, “Using and optimizing DB2 LOAD with R3load” on page 114. When using this option, be aware that it reduces the performance for small or empty tables. Those tables usually are imported using inserts with a lower initialization overhead, resulting in a faster import. Use this option only in a dedicated migration monitor instance.

We started with a single R3load process and then with eight parallel processes. The first process locks the table, and all other processes must wait until it has completed its work. The next R3load process continues loading and two or four R3load processes would also wait, and so we do not display the results. The surprising result is that the load is taking much longer when eight processes are running in parallel. See Figure 5-17.

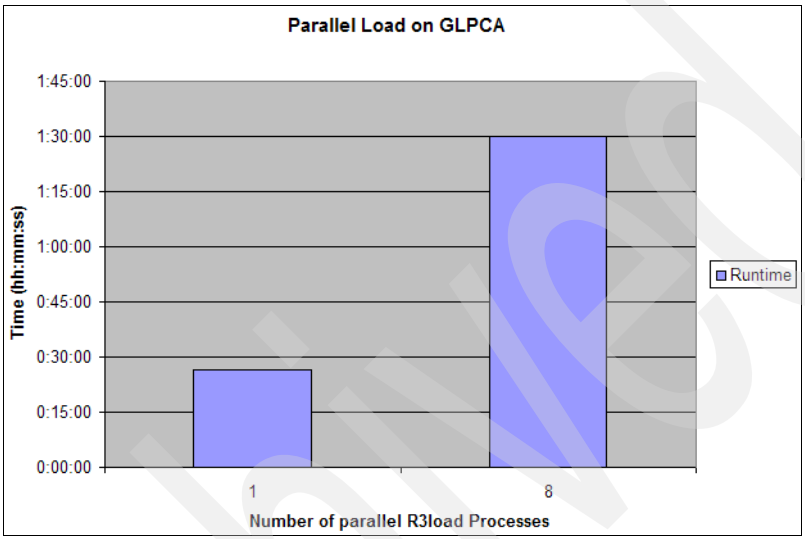


Figure 5-17 Runtime of parallel LOAD on GLPCA

To understand this, we must look closer at the behavior of DB2 LOAD. The processes are locked, as you can see in the db6util – sl output shown in Example 5-5.

Example 5-5 Lock situation documented by db6util -sl

```
LOCK WAITS:
----------
          17                  21                  28
    (PID:462)  <--       (PID:495)  <--       (PID:622)
      R3load               R3load               R3load
                            ^                    23
                            |----------       (PID:550)
                                                R3load
                            ^                    22
                            |----------       (PID:501)
                                                R3load
DETAILED INFORMATION ABOUT LOCKED PROCESSES:
--------------------------------------------
  ID      PID       APPL-NAME     HOSTNAME(PART)  MODE RMODE OBJECT TABLE
  17      462          R3load       DB-Server(0)
  Status   : UOW Waiting
```

```
Wkstn    : coe6x002
Last SQL : INSERT INTO "CDCLS" VALUES( ?, ?, ?, ?, ?, ?, ?, ? )
21      495          R3load       DB-Server(0)     Z     Z  TABLE SAPBN7.CDCLS
Status   : Lock Waiting (186 seconds)
Wkstn    : coe6x002
Last SQL : SET CURRENT ISOLATION RESET
28      622          R3load       DB-Server(0)     U     U    ROW SYSIBM.SYSTABLES
Status   : Lock Waiting (4462 seconds)
Wkstn    : coe6x002
Last SQL : SET CURRENT ISOLATION RESET
```

The lock wait does not explain the longer runtime compared with the single R3load process. The reason can be found in the mode of index creation. By default, DB2 rebuilds the indexes after each portion of the load. So, the index build is in fact performed several times. The DB2 LOAD API allows configuring the indexing mode and switching to incremental index build.

To activate the incremental index build when using the DB2 LOAD utility with R3load, we defined the following environment variable before starting the import:

```
setenv DB6LOAD_INDEXING_MODE 2
```

You can check the current indexing mode using the following DB2 command:

```
list utilities show detail
```

Figure 5-18 shows test results using different indexing modes.

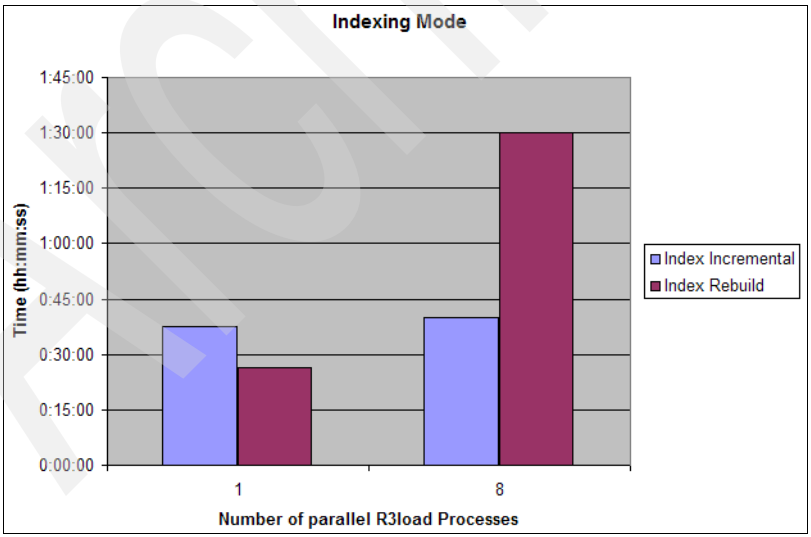


Figure 5-18 Importing GLPCA using different indexing modes

Incremental indexing has a negative impact when using a single R3load process, while it is beneficial to parallel R3load processes using the DB2 LOAD API. However, the overall runtime using the LOAD API is slower (40 minutes) compared with the parallel R3loads using inserts (21 minutes. See result above).

Next we look closer at the effect of the index creation order when importing table CDCLS. Figure 5-19 shows the results of the import tests with table CDCLS. The throughput is increasing, but we also found that for table CDCLS the INSERT did not scale very well, especially when building the indexes after the data was inserted. The reason may be related to the high utilization of the storage subsystem. With one R3load process the storage was 50% utilized, with two R3load processes it was 75% utilized, and with four and eight R3load processes it was 95% utilized. Therefore, when going from four to eight parallel R3load jobs no performance improvement was achieved.

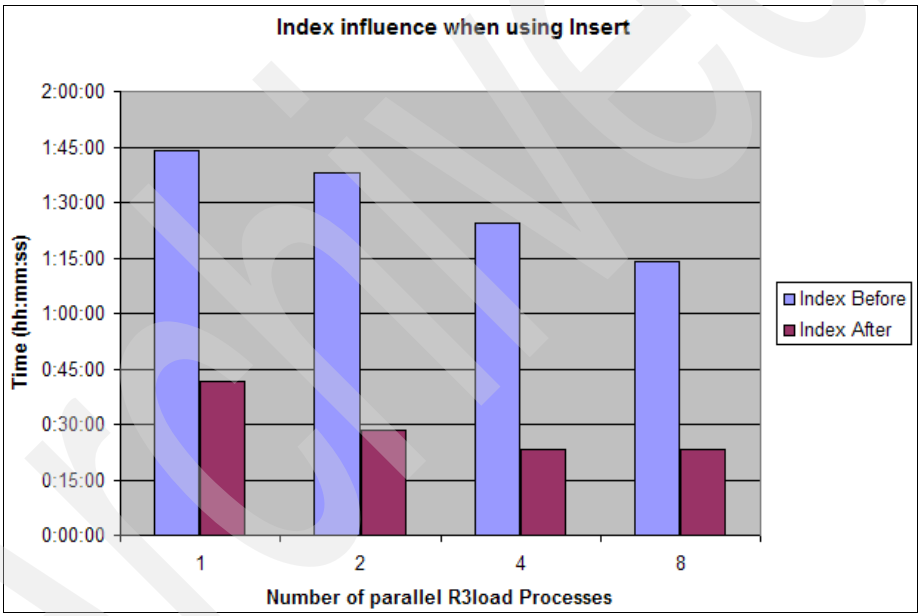


Figure 5-19 INSERT of CDCLS with different parallelism and index creation order

When using SQL INSERT the best performance was achieved by first inserting the data and then creating the indexes. However, this could change with a different hardware environment, as described in the test in 7.4, “Order and configuration of index creation” on page 122.

For the tests with DB2 LOAD the optimal order of loading and index creation depends on the number of parallel R3load jobs. For a single R3load job, it was

more efficient to first create the index and load the data. For multiple R3load jobs, we achieved the best runtime by building the indexes after the load.

Figure 5-20 shows this result. It is no surprise that the R3loads creating the indexes after the import show the same runtime, as this is basically the same process.

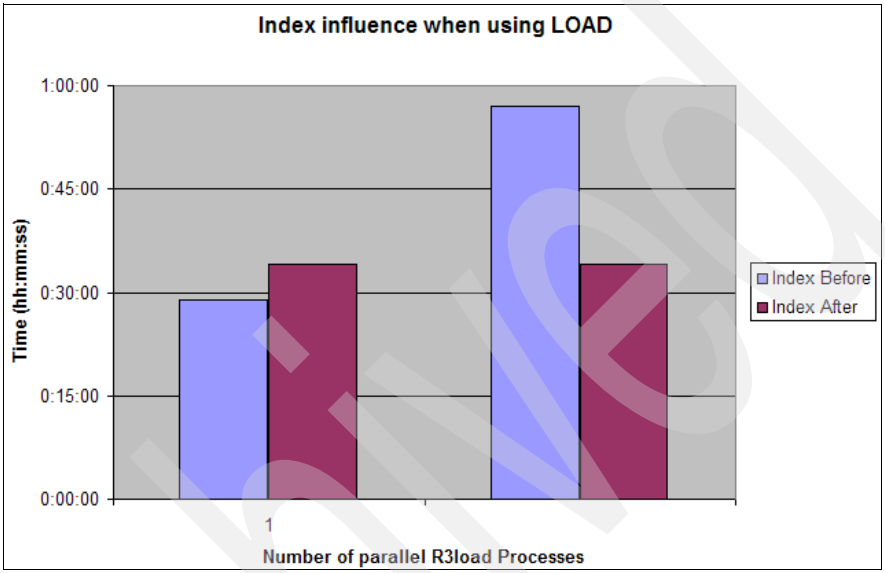


Figure 5-20 LOADING CDCLS with different parallelism and index creation order

Comparing the test results again shows a performance advantage for the parallel inserts. While the best run with inserts is finished after 23 minutes, the fastest run with LOAD API and parallel import is 34 minutes.

The major difference from the tests with the table GLPCA is that here a change in the order of the index creation was beneficial for overall performance.

Comparing the export and import runtimes of the table CDCLS, you can see that the export takes 7 hours and 35 minutes with a single process and can be optimized with the table splitting to 2 hours and 18 minutes. So the import in any case is by factors faster and must be placed in the context of the overall migration process. So, besides performance, you must consider the resource consumption and ease of use.

To evaluate the resource usage, we compared two test runs with almost the same duration and configuration.

Figure 5-21 and Figure 5-22 show the CPU usage and the disk usage during tests with LOAD and INSERT using just one process. Especially in the second graph (Figure 5-22), the peeks related to the different processes are obvious. In general, the resource usage with LOAD shows a more unsteady behavior, but both are in the same order of magnitude.

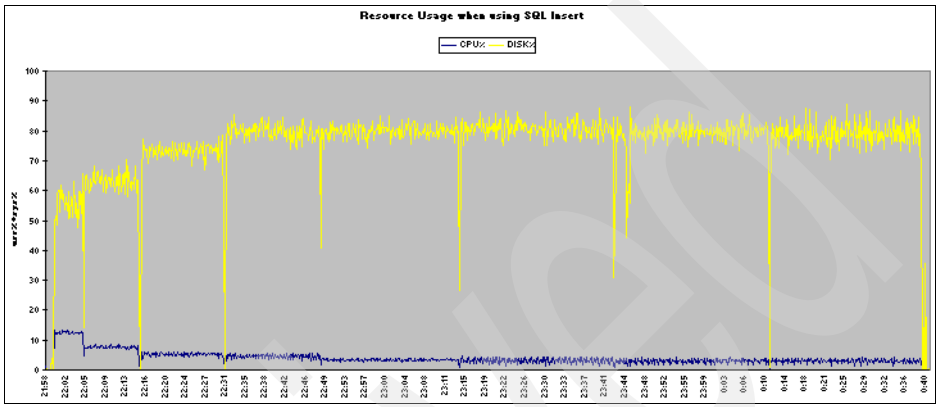


Figure 5-21 Resource usage during split import with SQL INSERT

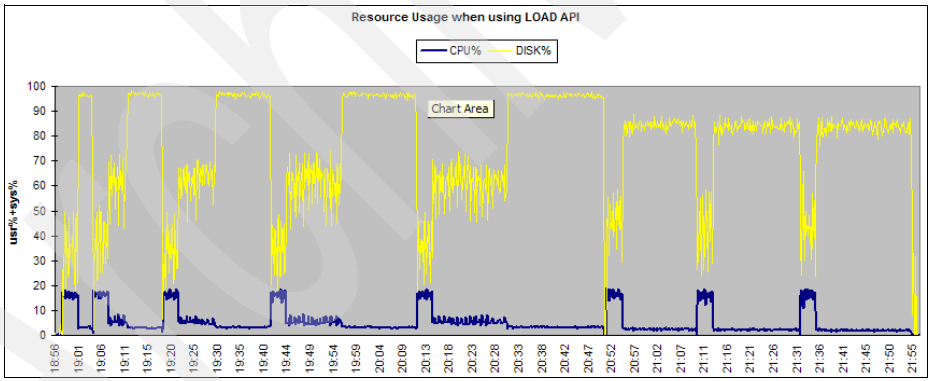


Figure 5-22 Resource usage during split import with LOAD

Both tests show an average disk usage of 75%, but the test with LOAD has higher peeks (up to 100%). The disk throughput limits the overall performance in both tests and results may be different with other I/O configurations. The test shows that a single LOAD process is using only slightly more resources. The consumption is in the same order of magnitude as a single INSERT process. The additional I/O due to the logging is not significant in this test, but this may change when using many parallel inserts.

Out of this test we can conclude that if the import performance and logging I/O is not the bottleneck, it is a good decision to use INSERT instead of LOAD.

When using DB2 LOAD that is serializing on the table level, another effect shows up that is related to this behavior. The R3load processes are started but locked on the database level, and thus several R3load processes are started but are actually idle. The migration time diagram shown in Figure 5-23 can be used to show this effect.

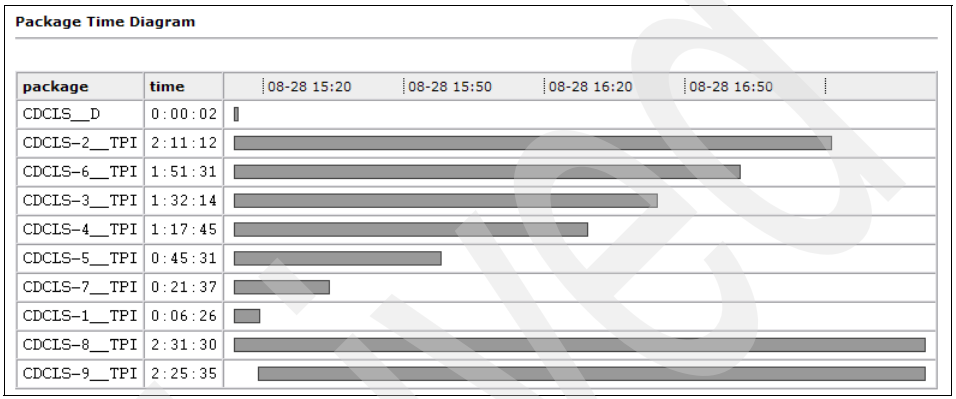


Figure 5-23 Import process with parallel load on single table

At a first glance, it looks like almost all processes are busy in parallel, but in fact only one process is active. Each time that a process finishes it releases the table lock, and the next R3load process can acquire a lock and start to load data. So with eight parallel processes defined in the MigMon, all are started but only one is loading data. The others consume memory resources but do not load any data.

So if you are using the DB2 LOAD API, the table must be placed into the orderBy file used in the `import_monitor_cmd.properties` file or by using a dedicated migration monitor instance. For the CDCLS, such a file could look like the one shown in Example 5-6.

Example 5-6 OrderBy file

```
[CDCLS]
jobNum=1
CDCLS-1
CDCLS-2
CDCLS-3
CDCLS-4
CDCLS-5
CDCLS-6
CDCLS-7
```

```
CDCLS-8
CDCLS-9
```

5.1.6 Conclusions

Recommendations for the export with WHERE splitting are:

- If single tables determine the overall export runtime, WHERE splitting can improve export duration by factors.
- WHR packages generated by R3ta and where_splitter may show different runtime behavior. You might have to further split long-running packages manually.
- We recommend that the index, which is used to select the data, has a high cluster factor. If this is not the case, consider performing a reorganization using this index before starting the export.

> **Note:** The creation of an additional index in the source system can have side effects on performance and database optimizer behavior.
>
> Creating an additional index or reorganizing the table might not be possible during productive operation.

Recommendations for the import with WHERE splitting:

- The optimal configuration for index order, number of parallel processes, and the import method depends on various factors such as infrastructure, table size, and definition.
- With respect to ease of use, R3load using INSERT is a good starter.
- DB2 LOAD may be an option for certain tables but must be understood.
  - You should understand the required configuration and behavior for R3load with the DB2 LOAD API.
  - Enable the incremental index build.
  - Serialize the LOAD using the orderBy entry in the import monitor properties file. Alternatively, use a dedicated MigMon instance.
  - Set the `LOAD_FORCED` option to force the use of LOAD. Ensure that this is done for the selected tables only.

- If you have a table with many or large indexes, it is more effective to create the indexes *before* the data is loaded. For tables with only a single index the import might be faster by first loading the data and then creating the index.
- Typically, *multiple* large tables are imported concurrently during an SAP heterogeneous system copy. Therefore, consider the resource consumption of the different options of table splitting.

5.2 Socket option

Data that is exported from the source system must somehow be transported to the target system. You can, for example, store the data on disk and transport it to the target server using one of the following methods:

- Transportable device (tape, disk)
- Network share
- ftp

However, the most efficient way is to use the socket option, which works similarly to a network pipe and is realized by using TCP/IP sockets. The data is sent to the target server by the exporting process using these sockets. The importing process receives the data and adds it to the target tables at once. This method saves time since you do not have to store the data on a device and you can immediately start the import. If you want to use this option, make sure that you have a stable network connection. Figure 5-24 shows the process.

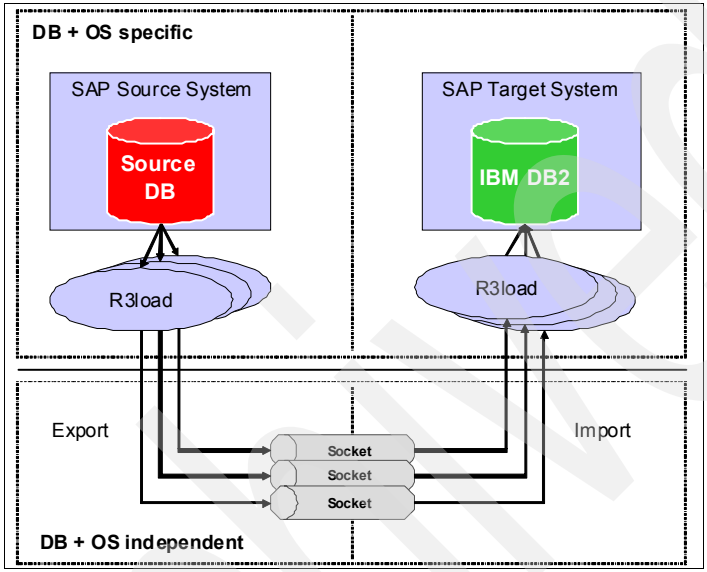


Figure 5-24 Using the socket option

To configure the socket option, you must set the parameters in the MigMon properties file. Table 5-1 lists the export socket options.

Table 5-1 Export socket options

Option	Description	Comments
socket	Socket operating mode	R3load will not write dump files to the file system, but the export and import work through the socket connection.
Host	Remote import host	Name or IP address of the import host.
Port	Host port number	Must be the same as the port number on the import host. Any free port on the import host from 1024 to 65535.

If you are using the socket option for import, Table 5-2 shows the parameters that must be defined for the import properties file used by the importing MigMon instance.

Table 5-2 Import socket options

Option	Description	comments
socket	Socket operating mode	
Port	Host port number	Any free port on the import host from 1024 to 65535

5.2.1 Comparing disk to socket migration method

Table BKPF was exported from a non-Unicode SAP system and imported to a Unicode SAP system.

Note: You can use the socket option for Unicode conversions as of R3load Version 6.40 patch level 98 (for more information see SAP Note 971646). For older R3load versions, you must not use the socket option for Unicode conversions.

Figure 5-25 shows the comparison of the socket feature to the disk dump method.

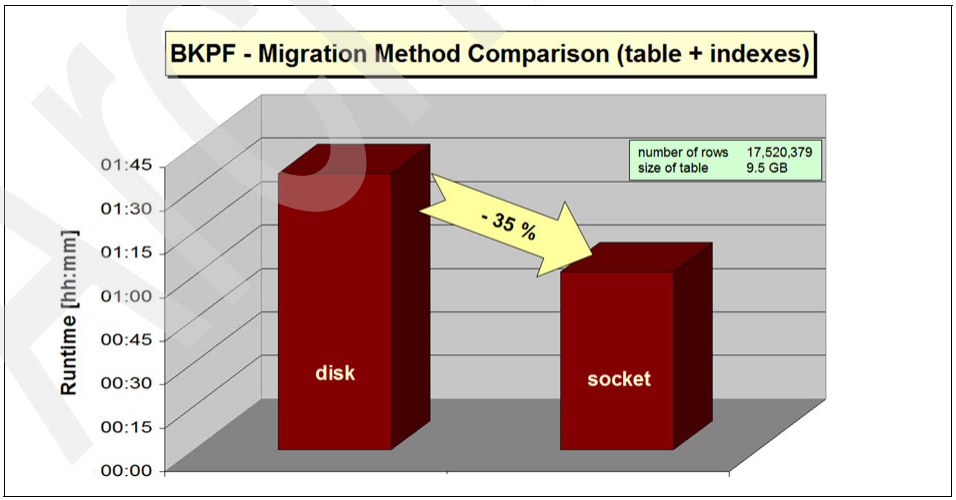


Figure 5-25 Migration method comparison (disk dump versus socket)

In our test environment we measured a reduction of the runtime by 35%. This can lead to a substantial reduction of downtime during a migration and Unicode conversion.

Looking at more tables, we differentiate between the behavior of table clusters and transparent tables, as shown in Figure 5-26.

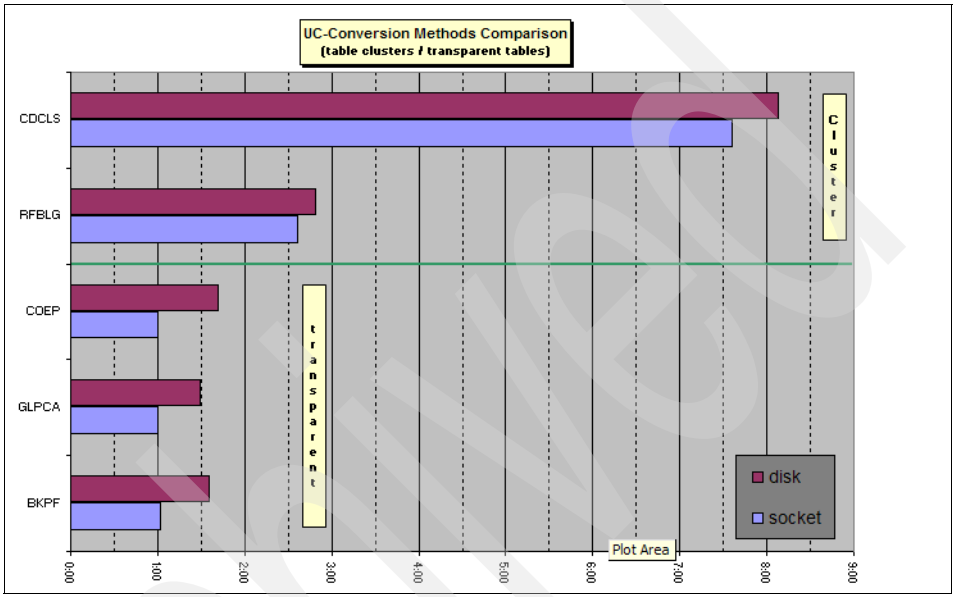


Figure 5-26 UC-conversion methods comparison (table clusters/transparent tables)

The transparent tables benefit a lot more (on average 30–35%) from the socket option than the table clusters (about 7–8%). When looking at the table clusters, the export time is dominating the runtime. The import is very fast in relation to the export. The time that can be saved by the socket feature can only be the runtime of the shorter part—either the export or import runtime.

Regarding the transparent table group, the difference between export and import runtime is not as extreme, so the benefit is bigger in this case. If export and import runtimes are equal, the time saving could theoretically come near to 50%.

5.2.2 Comparison of resources

During the migration of table BKPF, the resource usage was monitored with NMON. Figure 5-27 shows how the different methods behaved.

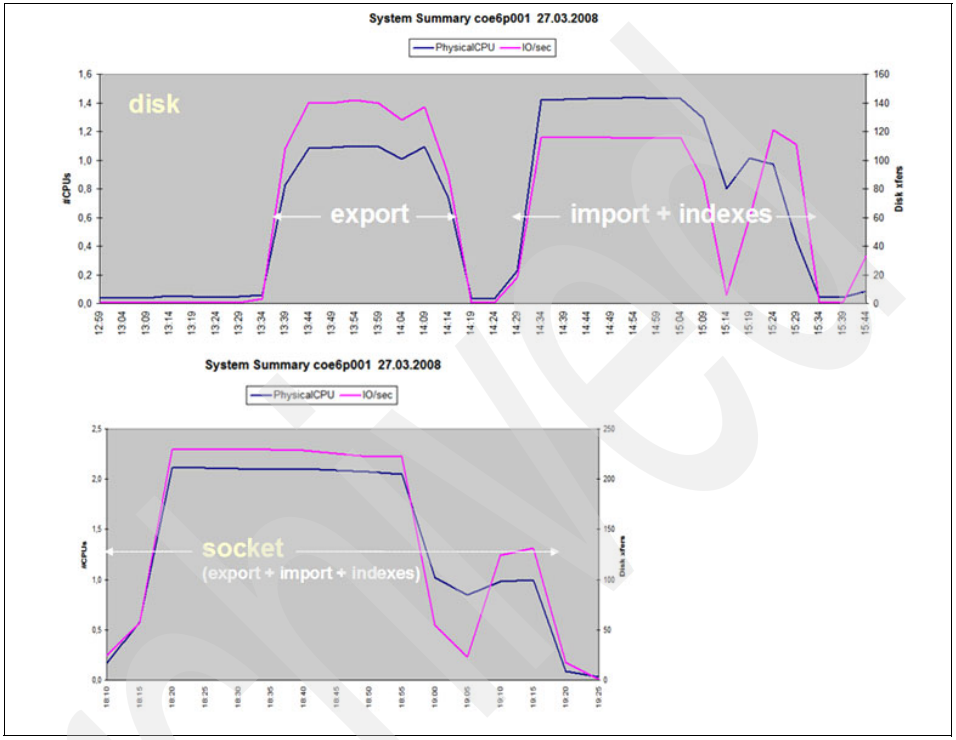


Figure 5-27 Resource usage comparison (disk dump <=> socket)

The disk dump method is a sequential process. First, the export is done, and then the data is imported to the database. In the last step, an index build is performed.

If you are using the socket feature, the export and import are done in parallel. The advantage in the above case is that you can save nearly the complete export time. The CPU usage and the I/O are slightly higher when using the socket feature. This is due to the fact that in the described test, both systems resided on the same LPAR using the same resources.

5.2.3 Migration monitor (MigMon) configuration

Figure 5-28 shows how you can configure the migration monitor (MigMon) when you are using the socket feature for a Unicode migration.

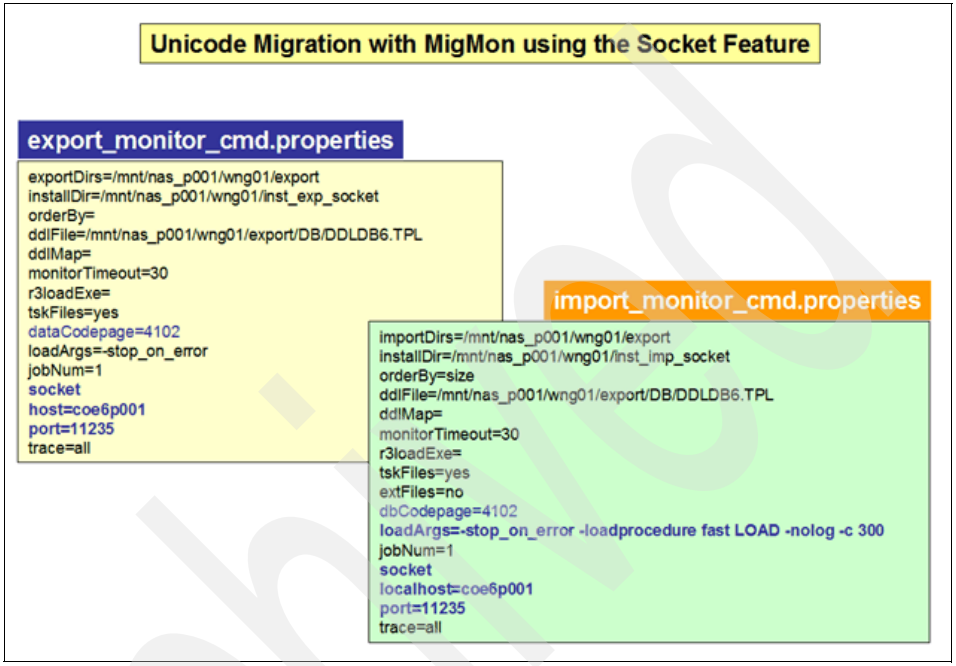


Figure 5-28 MigMon configuration for unicode migration with socket feature

To use the socket feature, you must use the socket keyword in the properties files of the MigMon together with a host name and a port. Be aware that the code page settings depend on the endianness of the server.

5.2.4 Export log

The log of the R3load export looks similar to the one shown in Example 5-7. The port definition on the export and import side must match. Otherwise, there will be no connection between them.

Example 5-7 Export log (Unicode conversion using the socket feature)

```
/usr/sap/NZW/SYS/exe/run/R3load: START OF LOG: 20080327181447
/usr/sap/NZW/SYS/exe/run/R3load: sccsid @(#) $Id:
//bas/640_REL/src/R3ld/R3load/R3ldmain.c#20 $ SAP
/usr/sap/NZW/SYS/exe/run/R3load: version R6.40/V1.4
Compiled Aug 13 2007 18:10:02
```

```
/usr/sap/NZW/SYS/exe/run/R3load -socket -e BKPF.cmd -datacodepage 4102 -l
BKPF.log -stop_on_error
(DB) INFO: connected to DB
Database table TCPDB empty. Guessing! See note 42305
(SYSLOG) INFO: k CP2 : TCPDB&O&                                          rscpll   17
(RSCP) WARN: UMGSETTINGS nametab creation: status not available, no table entry.
(RSCP) INFO: Global fallback code page = 0000
(RSCP) INFO: Common character set is 7-bit-ASCII
(RSCP) INFO: Collision resolution method is unexpected !
(RSCP) INFO: R3trans code pages = AskSPUMG
(RSCP) INFO: EXPORT TO ... code pages = AskSPUMG
(RSCP) INFO: Check for invalid language keys: active, by default
(RSCP) INFO: I18N_NAMETAB_NORM_ALLOW = 0
(RSCP) INFO: I18N_NAMETAB_NORM_LOG   = 0
(RSCP) INFO: I18N_NAMETAB_ALT_ALLOW  = 0
(RSCP) INFO: I18N_NAMETAB_ALT_LOG    = 0
(RSCP) INFO: I18N_NAMETAB_OLD_ALLOW  = 0
(RSCP) INFO: I18N_NAMETAB_OLD_LOG    = 0
(RFS) INFO: connection established to port 11236 on coe6p001
(GSI) INFO: dbname  = "NZW
"
(GSI) INFO: vname    = "DB6                          "
(GSI) INFO: hostname = "UNKNOWN
"
(GSI) INFO: sysname  = "AIX"
(GSI) INFO: nodename = "coe6p001"
(GSI) INFO: release  = "3"
(GSI) INFO: version  = "5"
(GSI) INFO: machine  = "0001DBD7D600"
(GSI) INFO: instno   = "INITIAL   "
(RTF) ########## WARNING ###########
   Without ORDER BY PRIMARY KEY the exported data may be unusable for some databases
(EXP) TABLE: "BKPF"
(DB) INFO: disconnected from DB
/usr/sap/NZW/SYS/exe/run/R3load: job completed
/usr/sap/NZW/SYS/exe/run/R3load: END OF LOG: 20080327191618
```

5.2.5 Import log

Example 5-8 shows the log of the R3load import. The import part is started first and works as a listener. When the export part is started, the connection is established. The DDL template file is missing the `ORDER_BY_PKEY` keyword. Therefore, the export was done unsorted. The index creation strategy was set to `BEFORE_LOAD`.

Example 5-8 Import log (Unicode conversion using the socket feature)

```
/usr/sap/ZWU/SYS/exe/run/R3load: START OF LOG: 20080327181441
/usr/sap/ZWU/SYS/exe/run/R3load: sccsid @(#) $Id:
//bas/700_REL/src/R3ld/R3load/R3ldmain.c#14 $ SAP
```

```
/usr/sap/ZWU/SYS/exe/run/R3load: version R7.00/V1.4 [UNICODE]
Compiled Mar  5 2008 08:27:05
/usr/sap/ZWU/SYS/exe/run/R3load -socket -i BKPF.cmd -dbcodepage 4102 -l
BKPF.log -stop_on_error -loadprocedure fast LOAD -nolog -c 300
(DB) INFO: connected to DB
(RFS) INFO: listening at port 11236 on coe6p001
(RFS) INFO: connection established
(RFS) INFO: connection established
(GSI) INFO: dbname  = "ZWU
"
(GSI) INFO: vname    = "DB6                                  "
(GSI) INFO: hostname = "UNKNOWN
"
(GSI) INFO: sysname  = "AIX"
(GSI) INFO: nodename = "coe6p001"
(GSI) INFO: release  = "3"
(GSI) INFO: version  = "5"
(GSI) INFO: machine  = "0001DBD7D600"
(RTF) ########## WARNING ###########
   Without ORDER BY PRIMARY KEY the exported data may be unusable for some
databases
(RTF) WARNING: CREATE PSEUDO TABLE template not found in control file
"/mnt/nas_p001/wng01/export/DB/DDLDB6.TPL"
(DB) INFO: BKPF dropped #20080327181453
(DB) INFO: BKPF created #20080327181453
(DB) INFO: BKPF~0 created #20080327181453
(DB) INFO: BKPF~1 created #20080327181453
(DB) INFO: BKPF~2 created #20080327181453
(DB) INFO: BKPF~3 created #20080327181454
(DB) INFO: BKPF~4 created #20080327181454
(DB) INFO: BKPF~5 created #20080327181454
(DB) INFO: BKPF~6 created #20080327181454
(DB) INFO: BKPF~BUT created #20080327181454
(DB6) Using LOAD API for table BKPF
(IMP) INFO: import of BKPF completed (17520379 rows) #20080327191617
(DB) INFO: disconnected from DB
/usr/sap/ZWU/SYS/exe/run/R3load: job completed
/usr/sap/ZWU/SYS/exe/run/R3load: END OF LOG: 20080327191618
```

5.2.6 Conclusion

If you have big tables that define the overall runtime of the migration, you can save time using the socket option. The maximum time that can be saved by the socket feature can only be the runtime of the shorter part—either the export or import runtime.

Note: Make sure that the network between servers does not turn into a bottleneck, for example, by using multiple network connections.

Ensure that you use stable network connections. A failed connection means restarting a failed LOAD process from the beginning.

You can also use this option on only one server. A possible implementation scenario is to use the importing server also as the application server for the export and to define the socket communication locally. If you want to combine this option with other techniques of data transport, you must use multiple MigMon instances due to different configurations in the properties file.

You can also use the socket option for Unicode conversions if you obey the restrictions described in SAP Note 971646.

6

Basic DB2 layout and configuration options

Before you start the import into the DB2 target system, you should carefully plan the database layout and database configuration.

The heterogeneous system copy or the Unicode migration also includes the possibility to assign tables and indexes to dedicated table spaces. With this, you are able to optimize the layout with respect to manageability and performance.

In this chapter we briefly describe basic DB2 recommendations regarding table space layout and DB2 configuration.

6.1 Table space principles and configuration

DB2 supports multiple ways to store data in table spaces. The two different basic types are the system managed space (SMS) and database managed space (DMS) table spaces. As an enhancement to these types, automatic storage was introduced with DB2 Version 8.2.2. It uses the best of these two types by reducing the effort to manage the table spaces. At the same time it results in reducing control regarding the placement and administration of table space containers. Depending on your priorities, you may choose one type or the other.

During the migration, the database size as well as the size of table spaces and containers are defined in the `DBSIZE.XML` file that is created during the export. The SAP installation tool SAPInst provides several options to choose from, for example, to use automatic storage or to use DMS table spaces and to specify the number of file systems (sapdata1–sapdata<n>) for DB2. The tool also lets you create table spaces manually. Use the `createTablespaces.clp` script in the SAPInst directory as the base for a custom configuration.

The size and the number of containers should be configured wisely, as they have an impact on performance and you cannot easily change this setup while the system is running in production. An appropriate disk layout also has a dependency on the underlying disk subsystem and is beyond the scope of this book.

However, we would like to provide basic recommendations. These are valid for all implementations:

- Use separate disks for logging and for table spaces.
- Do not configure operating system I/O (for example, swap, paging, or heavily spool) on DB2 data disks or disks that are used for logging.
- Spread the containers of heavily used table spaces across as many spindles as possible.
- Avoid putting multiple containers of one table space onto the same device. If you add additional containers onto the same device, they should be added in a new stripe set. But each stripe set should have no more than one container per device.

Besides the configuration parameters described in this chapter, the appropriate layout and data placement is essential for the further operation of the database. The migration also lets you assign large tables to dedicated table spaces and so optimize the layout. As the growth of the SAP database is monitored in production, a good estimation of the future growth is possible and should be taken into account.

Figure 6-1 shows a sample list of the largest table spaces in an SAP database. We use this example to show how a table space layout should *not* look. In this case the BTABD table space is by factors larger than the next smaller table space. This may not have an impact on the performance for the daily workload but it does have an impact on administration. For example, DB2 is backing up table spaces in parallel to improve performance. This cannot be fully exploited if one table space is by factors larger compared to the others.

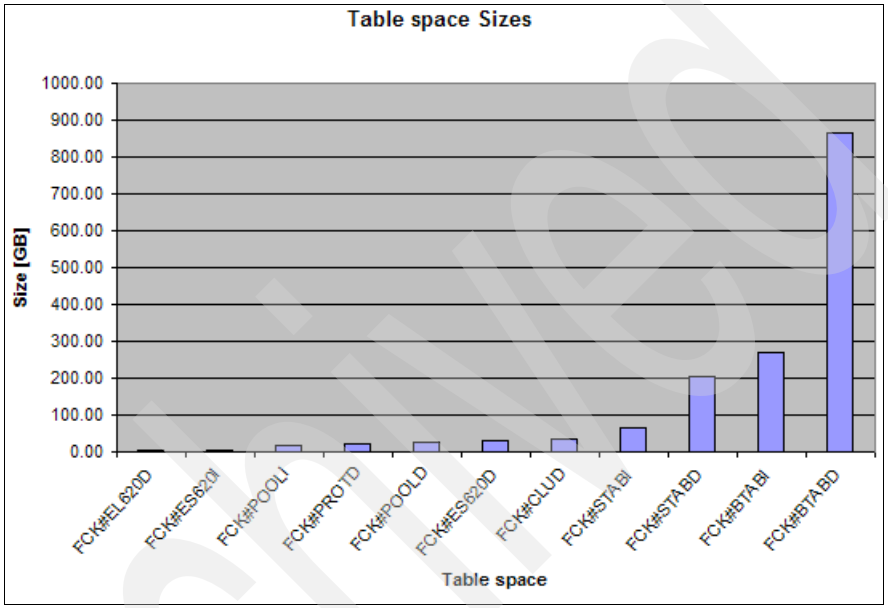


Figure 6-1 table space sizes (not optimized)

To optimize the layout, the largest tables can be assigned to dedicated table spaces. So the table space sizes can look as shown in Figure 6-2.

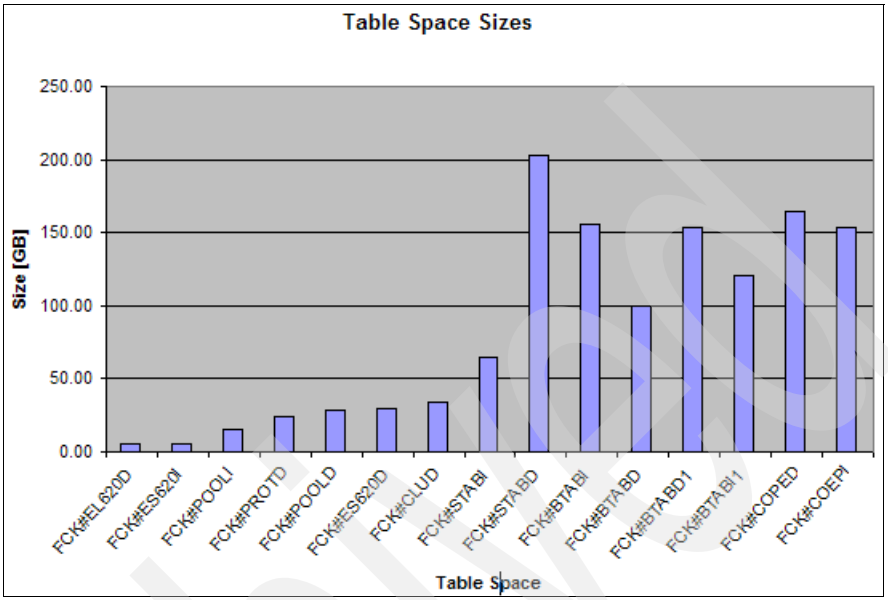


Figure 6-2 Table space sizes (optimized)

The picture shows a much better distribution. If the future growth rates are incorporated, this will be a good base for DB2 operation in production.

The size of the DB2 containers for the table spaces are determined by the R3szchk utility during the export and calculated to a more or less appropriate value for the target database. With the automatic resize function of DB2, the size estimation is no longer an issue, as container sizes are automatically increased. However, prior to a final migration we recommend creating the table spaces with the appropriate size as found during the migration test. This ensures consecutive blocks on disks and also saves time during the import.

Also check the test results for DMS temporary table spaces in 7.11, “Temporary table spaces” on page 146.

Note: Using DB2 V9.x or later running on AIX® with JFS2 or on Windows platforms, table space creation and automatic container size extension is very fast, while the allocation on other operating systems or file system may be slower.

6.1.1 Data placement and SAP data classes

As described above, the separation of large tables is one of the unique opportunities during a migration to establish an appropriate layout for future operation.

There are several ways to implement this while running a migration. One way is to adopt the technical settings of a table and to create new data classes and table spaces on the source system without actually moving the data to the new table spaces. The SAP migration tools then generate the control files accordingly and the tables are created in new table spaces in the target system. The basic procedure for this is:

1. In table TSDB6, you must add the names of the data table spaces and the according index table spaces.
2. In table DDART, you must add the new data class.
3. In table DARTT, you can set the description of the data class in other languages.
4. In table TADB6, the data table space names and the data classes are mapped. You must create new entries for the new data classes in TADB6.
5. In table IADB6, the index table space names and the data classes are mapped. You must create new entries for the new data classes in IADB6.
6. In table DD09L, you must change the TABART (data class) manually or by using SAP transaction “SE11”.

You can find details for this procedure in SAP Note 46272: *Implement new data class in technical settings* or in SAP Note 136702: *DB6: Move tables to other DB2 table spaces*.

If you are using DB2 already on the source system, you might want to change the data class mapping to table spaces and tables using the DBA Cockpit, as described in SAP Note 515968: *DB6: Creating data classes and table spaces in DBA cockpit*.

The new assignment of tables to table spaces and data classes takes effect during the migration.

Alternatively, you can edit the file `DDLDB6.TPL` that is used to specify the mapping between data class and table spaces. In Example 6-1, the new data class ZAPP1 and its corresponding table spaces are added in the file.

Example 6-1 Modification in DDLDB6.TPL to include new data class

```
# table storage parameters
loc:    USER6 NZW#ES40AD      NZW#ES40AD     NZW#ES40AD
        APPL1 NZW#BTABD       NZW#BTABI      NZW#BTABD
        ZAPP1 NZW#BTAB1D      NZW#BTAB1I     NZW#BTAB1D
```

In addition, you must adopt the .STR file containing the table to be moved. Example 6-2 shows a sample file to move the table COEP_L to the above-defined table space.

Example 6-2 Modified STR file for table COEP_L to be assigned to data class ZAPP1

```
tab: COEP_L
att: ZAPP1 4  ?N T all    COEP_L~0                    ZAPP1 4
fld: MANDT                          CLNT     3     0   0 not_null  1
```

Be aware that the second solution actually creates the tables in the new table spaces, but the data dictionary on the target will not be adopted and therefore will be inconsistent. The SAP system can be operational, but tables could be moved back, for example, during a subsequent system copy. Therefore, the first solution is the preferred one and the data dictionary should be checked and adopted on the source system prior to running the SAP migration tools.

6.1.2 File system caching

If file system caching is enabled the operating system attempts to minimize the frequency of disk access. This is done in main memory within a structure called the file system buffer cache. With many applications this leads to increased performance. However, certain classes of applications derive no benefit from the file system buffer cache. Some workloads never reuse data due to the sequential nature of their data access, which results in poor buffer cache hit rates. One example of this workload is loading a large amount of data. The use of a file system buffer cache results in undesirable overheads since data is first moved from the disk to the file system buffer cache and from there to the application buffer.

DB2 is able to bypass the file system buffer cache and directly access the file (container) to write or read data. It is possible to disable file system caching with the `–dio` or `–cio` mount options (this also eliminates the need for serialization of write operations) or by the appropriate DB2 command. The preferred solution is

to use the DB2 command, as file system caching can be switched on or off for dedicated table spaces that may be defined on the same file system.

Although the DB2 LOAD API is not using the DB2 buffer pool and thus not caching the data, the nature of the workload during the migration allows increased performance when file system caching is disabled.

Figure 6-3 shows the test results. It compares the runtimes of an import with R3load and the influence of file system caching.

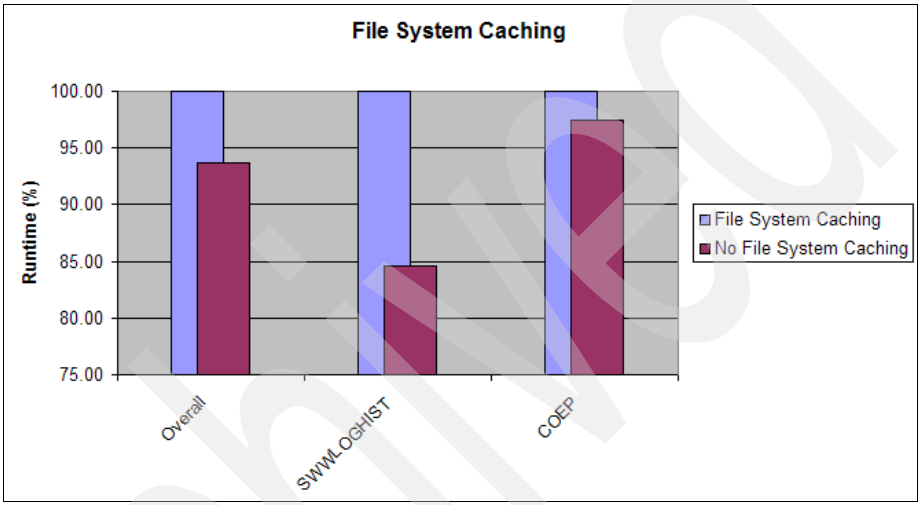


Figure 6-3 File system caching

Figure 6-3 shows a runtime improvement of about 6% in the overall process. The measurements were done on the tables COEP and SWWLOGHIST. Comparing the results for the two tables, we see that the benefit of switching off the file system caching is quite different. The tests were performed several times and the result displayed is the average runtimes of the tests. There is a noticeable improvement possible that also depends on the file system and operating system used. For the tests we have used a LINUX environment with local disks and reiserfs. Therefore, in other system environments the performance benefit can be higher.

With DB2 V9.5, the default I/O mechanism for newly created table space containers on most AIX, Linux, Solaris™, and Windows platforms is CIO/DIO (concurrent I/O or Direct I/O) and should be used for normal production also.

6.1.3 Page size

Table spaces are created based on a certain page size. Four supported page sizes exist:

- 4 KB
- 8 KB
- 16 KB
- 32 KB

There are two types of table spaces:

- Regular table spaces
- Large table spaces

The type defines the maximum size of the table space. Prior to DB2 V8.2.2, the DB2 system catalog was created with a page size of 4 KB pages. It was not possible to change this, often resulting in a database using different page sizes for different table spaces. As each distinct page size requires a buffer pool with the same page size, the buffer pool configuration was complex. This has changed with DB2 V8.2.2 and therefore SAP recommends that you use a uniform page size of 16 KB. The migration should be performed with these recommendations. Besides the page size, you should check that table spaces are created using the LARGE keyword. As of DB2 V9.1, LARGE is the default type when creating DMS table spaces. Large table spaces are enabled to use large record identifiers (RID). Large RIDs allow table spaces to grow beyond previous limits, as shown in Table 6-1.

Table 6-1 Size limits

Table space page size	Max. table size in DB2 V8	Max. table size in DB2 V9
4 KB	64 GB	2048 GB
8 KB	128 GB	4096 GB
16 KB	256 GB	8192 GB
32 KB	512 GB	16384 GB

In particular when using compression, the part of the 6-byte RID called SLOT becomes important. This 2-byte part of the RID addresses the record within a page. It was 1-byte before and now has changed to 2-bytes. This means that the number of records that can be addressed within a page is increasing from 255 to 32.768 in theory. The numbers are slightly lower in reality, but the values listed in Table 6-1 should give you an idea of the differences between large and regular table spaces.

We recommend that you use LARGE table spaces of page size 16 KB including the system catalog. Only in special cases can you use a different page size, but this should be checked with SAP.

6.1.4 Extent size

Workload tests and numbers collected from customer systems have shown that I/O performance is fairly insensitive to the extent sizes for production workloads. This is mostly due to the fact that modern storage subsystems employ sequential detection and that the prefetching can be influenced by the prefetch size parameter for DB2 table spaces. Even if you pick an extent size such as 128 KB (which is typically smaller than a full RAID array stripe size), the storage subsystem sequential prefetch keeps the other disks in the array busy. On the other side, the extent size influences the size of the database, as it is the minimum allocation unit for tables. A large extent size may result in wasted disk space for small or empty tables, and also for tables using multi dimensional clustering (MDC).

The current SAP recommendation is to use an extent size of 2, which together with the recommendation of 16 KB pages is 32 KB. This is a good starting point for the majority of the table spaces, but you may use a larger extent size if wasting space due to empty tables or MDC tables is not an issue, as larger extent sizes have other benefits like improved backup performance. The extent size could have an influence on the import also.

Figure 6-4 shows this influence. The extent size for the table spaces varies from 2 to 128 with a page size of 16 KB. As you can see, the import time is improving up to an extent size of 64 and slightly decreasing with 128.

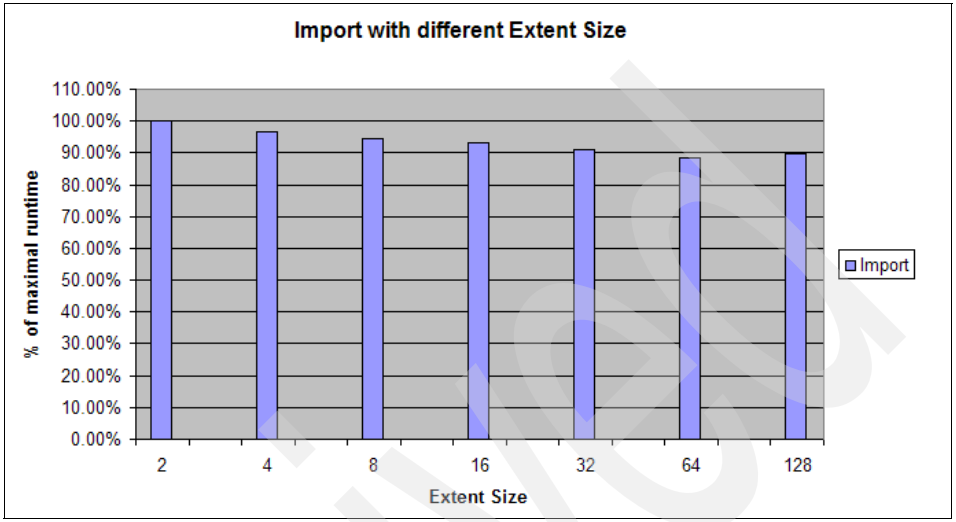


Figure 6-4 Import with different extent sizes

With an extent size of 64, the import was about 11% faster than the default of extent size 2.

You may create table spaces with larger extent sizes if they contain only large tables. To implement this, you must manually create the table spaces as described in 6.1, “Table space principles and configuration” on page 96.

6.1.5 NUM_IOCLEANERS

This parameter allows specifying the number of asynchronous page cleaners for a DB2 database. Page cleaners write changed pages from the buffer pool to disk before a database agent requires space in the buffer pool.

As we use the LOAD API to load the biggest tables, this parameter is not that important for the overall migration process. However, for index creation (if temporary data is written from buffer pool to temporary table space) this will have some impact on performance.

During the migration evaluation project, our tests showed no additional improvement by defining more than 20 I/O cleaners for index creation. Unless the disk throughput is below the expected performance for a given I/O configuration,

we recommend that you set this configuration parameter to AUTOMATIC and let DB2 determine the appropriate number.

6.1.6 Prefetch size

The table space prefetch size determines the degree to which separate containers can operate in parallel and how much data is read ahead in case prefetching is initiated by the database. Typically, prefetching occurs during the index build phase (within the LOAD API or in a dedicated create index statement). Index builds benefit from a large prefetch size.

DB2 configures the prefetch size when set to AUTOMATIC. The formula shown in Example 6-3 is used.

Example 6-3 Calculation of prefetch size

```
Prefetch size =
   (number of containers) * (number of physical disks per container) * (extent size)
```

The value for the number of containers and the extent size is directly derived from the table space definition, while the number of disks per container can be specified. The value defaults to 1 disk per container unless a different value is specified using the DB2_PARALLEL_IO registry variable.

Setting the prefetch size to AUTOMATIC is a good starting point and may only be changed if the I/O performance is below the expected rate for the given disk subsystem.

6.1.7 DB2 parallel I/O

DB2 assumes that each container has one physical disk that is assigned on the underlying disk subsystem. If you are using modern disk subsystems, this probably is not true. To specify the number of disks for each container in a table space use the DB2 registry variable DB2_PARALLEL_IO. This variable has a direct influence on the prefetch performance and should be set according to the disk configuration.

You can set this variable for each table space to a different value, as shown in Example 6-4.

Example 6-4 DB2_PARALLEL_IO

```
DB2_PARALLEL_IO=<tn1>:<nd1>,<tn2>:<nd2>, ... ,<tnN>:<ndN>
tn = table space number
nd = number of spindles per table space container
```

If you set it to the default value (DB2_PARALLEL_IO=*), each container is treated as though it has six dedicated disks.

We recommend that you adopt this configuration parameter according to the number of disks available for the containers. However, if the exact number of disks cannot be determined, the variable should be set to the default value.

6.2 Logging configuration

The migration process is basically a process of moving data from one database to another using the SAP utilities, so database logging may be a potential bottleneck during the migration. If you are using the DB2 LOAD functionality, logging is reduced to a minimum. Therefore, the logging configuration is not as critical as during production use of the database.

DB2 has two different modes of using log files:

- Circular logging mode
- Archival logging mode

While archival logging allows point-in-time recovery, the circular logging mode allows only offline backups and no rollforward after a restore because log files are not archived, but overwritten.

Database load is a very special operation and a recovery situation during the migration is very unlikely (this almost always means failure of the migration and a restart). There is no need to configure archival logging. In addition, the circular logging can be beneficial for performance, as it does not log LOB data. Therefore, the amount of logging I/O can be reduced.

We recommend that you run the database in circular logging mode by setting the configuration parameters as shown in Example 6-5.

Example 6-5 Database configuration for circular logging

```
Log retain for recovery enabled (LOGRETAIN) = OFF
User exit for logging enabled   (USEREXIT) = OFF
First log archive method        (LOGARCHMETH1) = OFF
Options for logarchmeth1        (LOGARCHOPT1) =
Second log archive method       (LOGARCHMETH2) = OFF
Options for logarchmeth2        (LOGARCHOPT2) =
```

The DB2 database configuration variable LOGINDEXBUILD enables the database manager to log the creation of indexes and will generate a significant amount of logging. By default, this variable is turned off and you should ensure

that this is true for your environment. However, a certain amount of logging occurs during the migration as some tables are imported by INSERT and the DB2 LOAD utility logs the allocation of new extents. Although the amount of logging is small compared with the production use, you can optimize it. The available configuration parameters are SOFTMAX, LOGBUFSZ, or the registry variable DB2_LOGGER_NON_BUFFERED_IO, which enables direct I/O on the log file system. But as logging is typically not critical during the migration, there is no need for special attention to this.

By providing a sufficient amount of disk space, an appropriate number and size of primary and secondary log files, and enabling circular logging, the optimization for the migration in this area should be fine.

6.3 Conclusions

An SAP system copy gives you the unique opportunity to redesign or adjust the database layout and setup. As preparation, spend some time on the table space layout and define the configuration parameters accordingly. Try to find an appropriate balance between the number and size of containers for the table spaces to avoid large files and a large number of files. This is particularly important if you are running an SAP NetWeaver BW with DB2 DPF using logical partitions.

One option for optimizing the layout is to separate the largest tables into dedicated table spaces by adapting the SAP data dictionary on the source system or adapting the data class settings in the STR files and the tables space assignment for the data classes in the appropriate DDLDB6.TPL file during the migration.

Incorporate the future growth of the system in the layout planning, as a subsequent relocate will impact production.

During the migration, you have the option to disable file system caching by using the DB2 commands to improve the import. If buffering of LOB fields is required, you might want to enable the file system caching for selected table spaces again.

Since a database rollforward recovery during a migration does not make sense, leave the logging in circular logging mode but specify enough primary log files, secondary log files, and disk space to accommodate the logging workload, especially if INSERTs are used.

7

Import optimization

The data import phase is a long-running task. Together with the export, it influences the overall downtime phase of a heterogeneous system copy or Unicode migration. Therefore, it is important to optimize the import using SAP tools and DB2 configuration settings. This chapter provides information about optimization techniques and some insight into and background information about configuration settings.

Although all information provided here is based on real-life migrations and intensive tests, there is no hard coded set of parameters and values available that apply to all migrations. The parameters must be adopted for a given system environment.

7.1 DB2 LOAD versus INSERT

R3load prior to SAP kernel Release 4.6 uses only INSERT to import data. This procedure is sufficient for installations and migrations of small databases. However, if a large amount of data must be moved from one system to another system the DB2 LOAD utility is much faster because it writes formatted pages directly into the database.

7.1.1 Data processing

Figure 7-1 shows the simplified process model of the two different methods to import data. As you can see, DB2 LOAD deploys multiple parallel processes to parse and format the data and also multiple processes to write the data to disk.

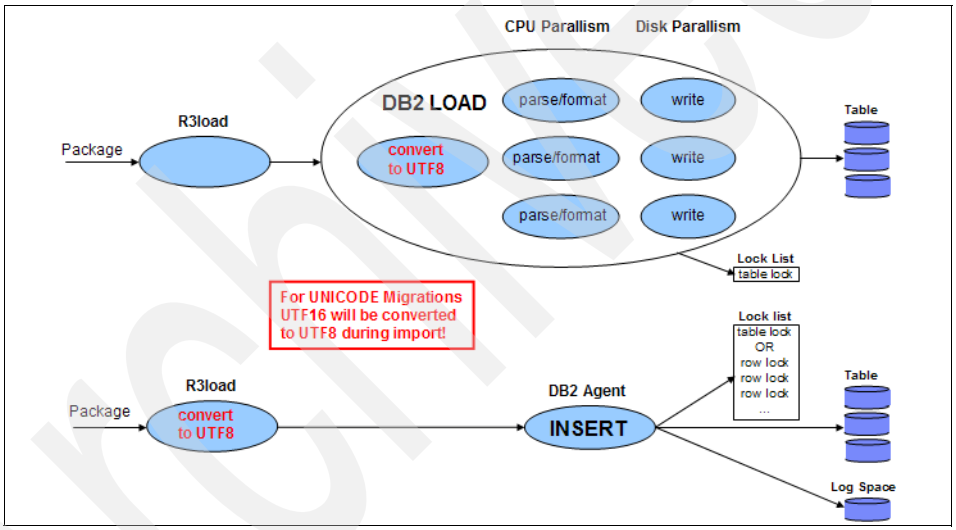


Figure 7-1 Simplified process model: DB2 LOAD versus INSERT

The locking behavior of DB2 LOAD and INSERT is different:

- DB2 LOAD locks the table exclusively.
- INSERT uses row-level locks.

If multiple R3load processes use the DB2 LOAD utility to import data *into the same table* (for example, in case of a splitted table (WHERE splitter)), only one of them can import data at any point in time. If multiple R3load processes use INSERT to import data, they can import the data concurrently.

In case the data contains UTF16 encoded character data (Unicode), this data must be converted from UTF16 to UTF8. When using INSERT this conversion is performed by the R3load process. When using DB2 LOAD this conversion is performed by the DB2 LOAD utility.

When using DB2 LOAD, the amount of logging is drastically reduced. This further improves performance. For each new extent that is created in the table, only one log entry will be written. When using INSERT each loaded record will be logged.

The processing with DB2 LOAD is optimized on throughput. The overall load process is divided into several distinct phases:

- Loading phase
- Building indexes and statistics phase
- Delete phase
- Index copy phase

R3load is using the DB2 LOAD API with a subset of the available DB2 LOAD features. Therefore, only two phases are normally shown in the db2diag.log when using R3load:

- Load: Phase during which the data is written to the table.
- Build: Phase during which indexes are created

The delete phase should not appear, as data consistency is guaranteed by the SAP application and the index copy phase is only used if the index is created in a system temporary table space.

The DB2 LOAD utility does not fire triggers, and does not perform referential or table constraints checking other than checking the uniqueness of indexes. The utility is also able to load data into partitioned databases (DB2 DPF feature). For example, DPF is supported for SAP NetWeaver Business Warehouse systems.

7.1.2 Performance comparison

DB2 LOAD is typically much faster than INSERT. The increase of throughput with DB2 LOAD depends on the underlying infrastructure and to some extent also on the table structure. If your system is I/O constraint, DB2 LOAD can show less improvement, as the disks might not be able to write the data as fast as requested.

In a test with table FUNCA (60 million records, 25 GB data) we compared an import using INSERT with an import using DB2 LOAD. It took approximately 12 hours and 30 minutes to insert all data into the table, while the same table was loaded within only around 55 minutes. Figure 7-2 shows different test results on an I/O constraint Linux system for various tables.

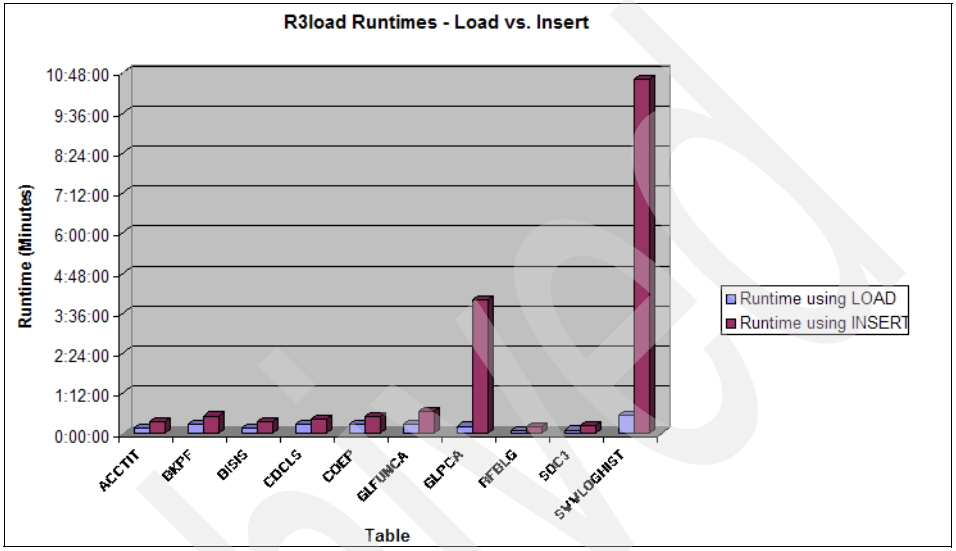


Figure 7-2 Runtime comparison: DB2 LOAD versus INSERT

Multiple concurrent R3load using DB2 LOAD scales well. We have seen imports of up to 700 GB per hour. However, this depends on various factors, for example, table structure, I/O configuration, available memory, and CPUs. These results were achieved on Power 4 machines and therefore you can expect a higher throughput with up-to-date hardware.

7.1.3 R3load options

There are different R3load options available to use the different options to import data. Table 7-1 shows the options that can be specified to use INSERT.

Table 7-1 R3load options for insert

R3load -i ... - loadprocedure dbsl	Standard INSERT using the SAP DBSL layer
R3load -i ... - loadprocedure fast	Optimized INSERT with reduced code path and other optimizations

The options provided in Table 7-2 can be specified to use DB2 LOAD.

Table 7-2 R3load options for DB2 LOAD

R3load -i ... - fast LOAD	This option must be used with SAP Kernel 4.6D.
R3load -i ... - loadprocedure fast LOAD	This option must be used with SAP Kernel 6.x or later releases.

Even if you instruct R3load to use DB2 LOAD by specifying the options above, for some tables R3load still uses INSERT. This is the case for the following tables:

- Tables with LOB data
- Tables that contain less than 200 KB of data
- Splitted tables (WHERE splitting)

However, you can override these exceptions when using R3load as follows:

- Use R3load option fast `LOAD_FORCED` for R3load 7.0.
- Specify the environment variable `DB6LOAD_FORCE_LOAD` for older releases.

If you want to use DB2 LOAD for *splitted tables* (instead of INSERT):

- Force usage of DB2 LOAD for splitted tables and use a separate MigMon instance to load those tables to make sure that small tables are *not* imported with DB2 LOAD.
- Make sure that packages for each splitted table are loaded sequentially by defining a package group for each table in the orderBy file. Example 7-1 shows the syntax of the `orderBy` file to load the splitted table CDCLS.

Example 7-1 Syntax of the orderBy file to load the splitted table CDCLS

```
[CDCLS]
jobNum=1
CDCLS-1
CDCLS-2
CDCLS-3
...
```

- Enable incremental index build for splitted tables. This can be achieved by modifying the `import_monitor.sh` file. Example 7-2 shows what should be modified in the `import_monitor.sh` file.

Example 7-2 Modified version of import_monitor.sh file

```
#!/bin/sh
#
# Import Monitor startup on UNIX
#
export DB6_LOAD_INDEXING_MODE=2
...
```

7.2 Using and optimizing DB2 LOAD with R3load

The following chapters provide general rules of thumb on how to configure the various DB2 LOAD-related parameters.

7.2.1 Configuring the DB2 LOAD API

There are several options for R3load to control the behavior of the DB2 LOAD API. In addition to the options mentioned above, the following environment variable settings are supported:

- DB6LOAD_CPU_PARALLELISM=<n>

  This variable controls the DB2 LOAD `CPU_PARALLELISM` parameter.

- DB6LOAD_DATA_BUFFER_SIZE=<n>

  This variable controls the DB2 LOAD `BUFFER_SIZE` parameter.

- DB6LOAD_DISK_PARALLELISM=<n>

  This variable controls the DB2 LOAD `DISK_PARALLELISM` parameter.

- DB6LOAD_INDEXING_MODE =<n>

  This variable controls the DB2 LOAD `INDEXING_MODE` parameter, where 0 equals `AUTOSELECT` (Default), 1 equals `REBUILD`, 2 equals `INCREMENTAL`, and 3 equals `DEFERRED`.

- DB6LOAD_FORCE_LOAD

  This variable forces DB2 to use DB2 LOAD even for small or split tables if it is set to any value (for example, 1). With R3load 7.xx, you can specify the force option directly in the R3load arguments.

7.2.2 CPU_PARALLELISM

Use this parameter to exploit intra-partition parallelism (if this is part of your machine's capability). The parameter specifies the number of processes or threads used by the DB2 LOAD utility to parse, convert, and format data records. The maximum value allowed is 30. If this parameter is not specified, the DB2 LOAD utility selects a default value that is based on the number of CPUs on the system, which normally is a good choice.

Figure 7-3 shows the result of two different tests on an 8-way Linux machine. The red line indicates the import of a single table (BKPF) with a different degree of CPU parallelism. This did not show any effect on the runtime for this table.

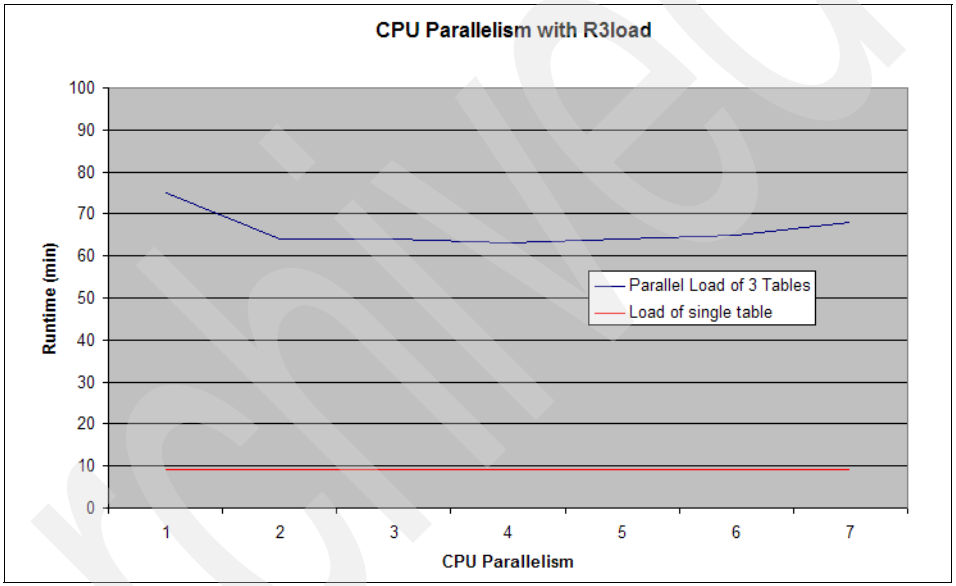


Figure 7-3 Influence of CPU_PARALLELISM on DB2 LOAD runtime

The blue line indicates the overall runtime of a set of tables that were loaded with three concurrent loads. In this scenario, the overall runtime decreases when specifying a larger number of CPU parallelism for a load process. The maximum throughput is reached if you use four out of the eight available CPUs and the throughput slightly decreases when more CPUs are used. In this concurrent test scenario, DB2 uses a CPU parallelism of seven, which does not give full performance but that is only 5% below the optimal runtime. So in certain cases, it might be useful to limit the degree of CPU parallelism when running multiple R3load operations on a large SMP machine. A larger number of CPU parallelism only increases the throughput to a small extent, while the CPU resources might be better used for other operations like RUNSTATS or index builds.

In summary, the DB2 default setting usually is a good choice, but you should monitor CPU parallelism during the test migrations and you should ensure that it is not decreasing below two or three.

7.2.3 DATA_BUFFER_SIZE

The `DATA_BUFFER_SIZE` parameter specifies the total amount of memory that is allocated to the DB2 LOAD utility as a buffer. The data buffer is allocated from the utility heap. Depending on the amount of storage available on your system, you should consider allocating more memory for use by the DB2 utilities. The maximum value for `DATA_BUFFER_SIZE` is limited by the utility heap available and determined by an algorithm that takes into account the available memory and also the table structure. Therefore, you must modify the database configuration parameter `UTIL_HEAP_SZ` accordingly.

Figure 7-4 displays the memory usage if you are running several load processes in parallel. The utility heap is set to 100,000 pages (400 MB) and the memory that is used by a single process is decreasing. While the first LOAD process is using about 180 MB, the third LOAD process is using about 100 MB, and the sixth process is using 40 MB.

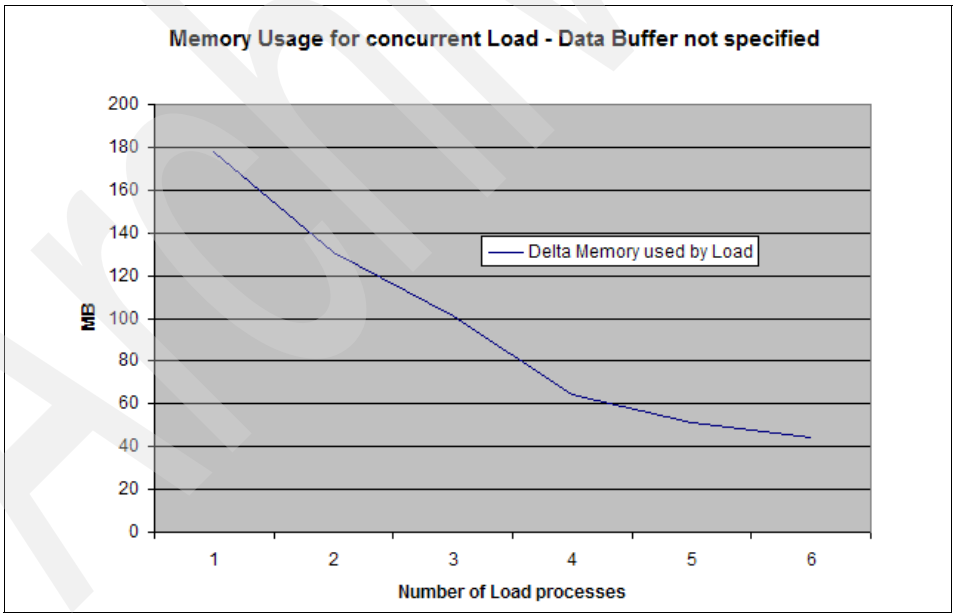


Figure 7-4 Variable buffer usage

The test and experience during migrations show that the amount of memory available does not have a large impact on performance as long as the parallelism

of the load is not reduced due to memory constraints. You can verify the parallelism of the load process in the db2diag.log and you should check whether subsequent load processes reduce the number of CPUs used. In the DB2 diagnostic log file, entries similar to Example 7-3 are displayed for each load process.

Example 7-3 Messages in DB2 diagnostic log file created by DB2 LOAD

```
2008-03-07-05.41.18.778164+060 I363319E493         LEVEL: Warning
PID     : 24645                TID  : 47427818088768PROC : db2sysc 0
INSTANCE: db2bn6               NODE : 000          DB   : BN6
APPHDL  : 0-13                 APPID: *LOCAL.db2bn6.080307044116
AUTHID  : SAPBN6
EDUID   : 65                   EDUNAME: db2lrid 0
FUNCTION: DB2 UDB, database utilities, DIAG_NOTE, probe:0
DATA #1 : String, 76 bytes
LOADID: 40.2008-03-07-05.41.18.655881.0 (10;5)
Load CPU parallelism is: 6, 0
```

You should configure the utility heap to at least the number of parallel R3load processes multiplied by 30,000 pages (117 MB) with sufficient memory. 200,000 pages is a good starting point for this if the memory is available. See also 7.3, “UTIL_HEAP_SZ” on page 121.

If the machine is under memory constraint, consider setting the `DB6LOAD_DATA_BUFFER_SIZE` environment variable to a fixed value. Unlike the automatic memory allocation where each DB2 load process gets a different amount of memory, this environment variable allows a more deterministic configuration of the load process. However, DB2 still influences the behavior of the memory allocation and ensures that each load process gets the minimum memory required according to the internal algorithm. On the other side, when having large SMP machines and many parallel running R3load processes, it could be beneficial to limit both the data buffer and the CPU parallelism. We have seen in some cases that even with enough memory available, R3load processes decrease in performance or are stalled without entries in the error logs.

Figure 7-5 shows the memory usage of load processes if the data buffer is set to a fixed value. About 200 MB are used for each LOAD process, resulting in a total of 1.2 GB used for the six LOAD processes. This amount of memory exceeds the specified utility heap of 390 MB or 100,000 pages. The DB2 configuration parameter specifies the guaranteed minimum, and if memory is still available, this is used for the utility heap to satisfy the DB2 LOAD processes.

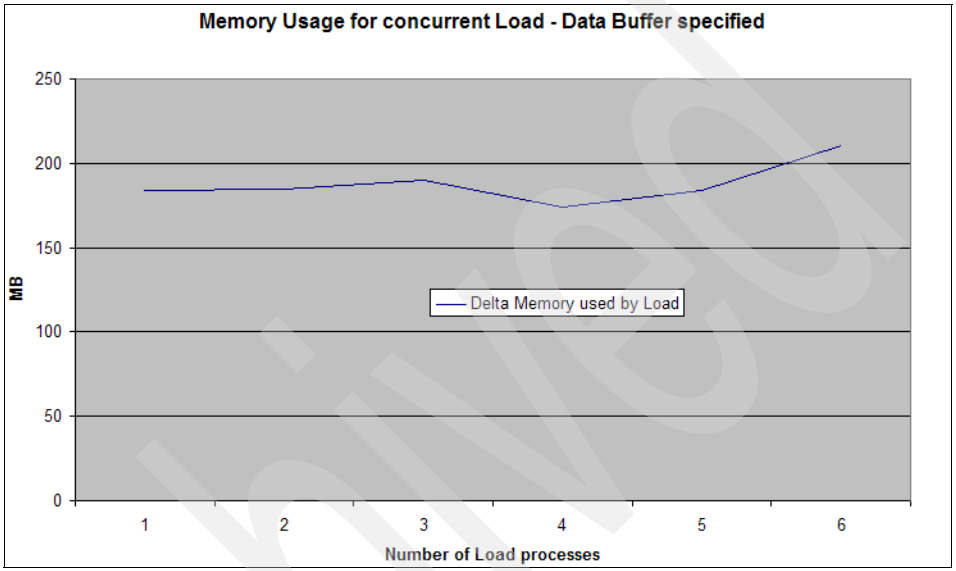


Figure 7-5 Fixed buffer usage

The conclusion of this test and our experience from real customer migrations is to use the DB2 default for `DATA_BUFFER_SIZE`.

Note: In large SMP environments with many parallel running R3load processes, you may limit both the DATA_BUFFER_SIZE and CPU_PARALLELISM to avoid stalled R3load processes.

7.2.4 DISK_PARALLELISM

The `DISK_PARALLELISM` parameter specifies the number of processes or threads used by the LOAD utility to write data records to disk. Use this parameter to improve load performance. The maximum number allowed is the higher value of either four times the CPU_PARALLELISM value (used by the LOAD utility) or 50.

By default, `DISK_PARALLELISM` equals the sum of the table space containers on all table spaces that contain objects for the table being loaded, except where this

value exceeds the maximum number allowed. Typically, there is no need to change this value manually. This is supported by the chart shown in Figure 7-6.

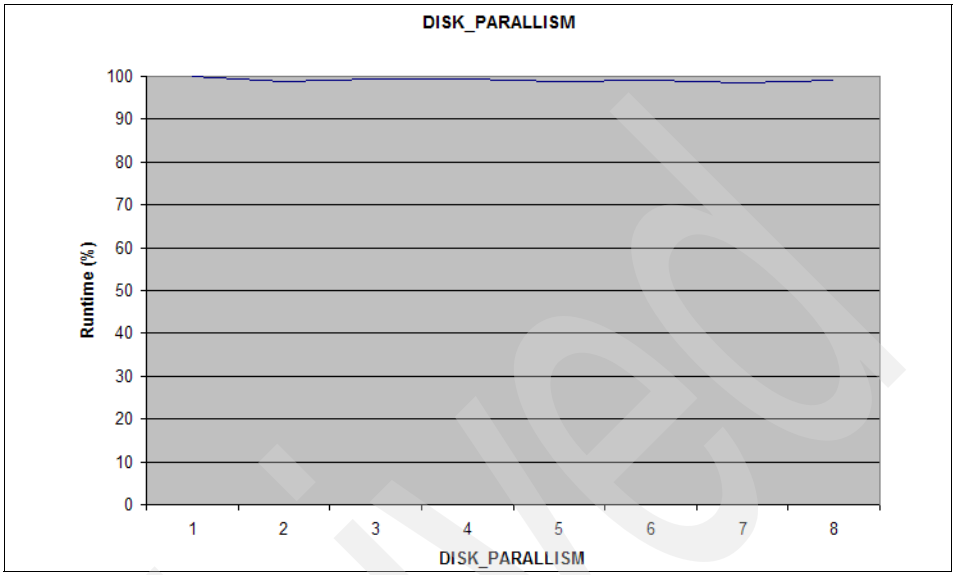


Figure 7-6 Influence of disk parallelism

7.2.5 INDEXING MODE

Specifies whether the LOAD utility rebuilds indexes or extends them incrementally. Valid values are:

- AUTOSELECT (default)

  The DB2 LOAD utility automatically decides between `REBUILD` or `INCREMENTAL` mode.

- REBUILD

  All indexes are rebuilt. The utility must have sufficient resources to sort all index key parts for both old and appended table data.

- INCREMENTAL

  Indexes are extended with new data.

- DEFERRED

  The LOAD utility does not attempt to create an index if this mode is specified. Indexes are marked as needing a refresh. The first (not LOAD related) access to such indexes forces a rebuild, or indexes might be rebuilt when the database is restarted. However, this option is ignored in many cases, as almost all SAP tables have unique indexes that should be defined prior to

loading. SAP tables without unique indexes can be F-fact tables in SAP NetWeaver BW systems.

We typically do not recommend changing any of the above values, and the default should be used unless needed for specific optimizations. One of this specific optimization is the usage of DB2 LOAD API when sequentially importing splitted tables. In this case, you should use indexing mode `INCREMENTAL` to load splitted tables.

7.2.6 Summary

In summary, you do not have to set the environment variables that control the LOAD. Use the defaults. There is one exception: If you want to use DB2 LOAD to import splitted tables you should use indexing mode `INCREMENTAL` for the splitted tables (use separate MigMon instance for splitted tables).

In complex and highly optimized environments, you may want to test different parameter settings to optimize import runtime.

7.2.7 Clean up a failed DB2 LOAD

If an R3load using the DB2 LOAD API fails, you might have to clean up the table from a LOAD pending state. To terminate a failed LOAD, issue the following commands:

```
db2 connect to <db_name> user <connect user - e.g. sapr3>
db2 load from <empty file> of ixf terminate into <schema>.<tabname>
```

To see which table is in LOAD pending state you can use the following SQL statement:

```
db2 "select tabschema, tabname from SYSIBMADM.ADMINTABINFO where load_status =
'PENDING'"
```

If for any reason the LOAD terminated abnormally, the table spaces in rare cases remain in *quiesce exclusive* state (db2 list table spaces shows state 0x0004). To remove this status, use the following commands:

```
db2 connect to <db_name> user <connect user – e.g. sapr3>
db2 quiesce table spaces for table <failed table> exclusive
db2 quiesce table spaces for table <failed table> reset
```

This procedure should unquiesce the table spaces. The table for which the load failed remains in a LOAD pending state and you must execute the steps as described above.

Usually, it is a good idea to drop the table afterwards and let R3load recreate it. (Verify that the R3load task file is also recreated/edited.)

7.3 UTIL_HEAP_SZ

The database parameter utility heap size (`UTIL_HEAP_SZ`) indicates the maximum amount of memory that can be used simultaneously by the DB2 utilities such as BACKUP, RESTORE, or LOAD. If the parameter is set too low, you may not be able to concurrently run utilities. You must set this parameter large enough to accommodate all of the buffers that you want to allocate for the concurrent utilities. Based on the number of processes defined for loading data and the page size and extent size of the table space the table resides in, the memory is calculated for a load operation. This memory is allocated from the utility heap. If there is not enough memory available, the number of CPUs used for LOAD will be reduced.

You should configure the utility heap at least to the number of parallel R3load processes multiplied by 30,000 pages with sufficient memory. 200,000 pages is a good starting point for this if the memory is available.

Figure 7-7 indicates that the utility heap increases the import runtime if it is configured too small. The longest runtime (100%) for the test was achieved using 15,000 pages for the utility heap. The runtime is improving by 5% if you are using 33,750 pages but it does not improve with a larger size.

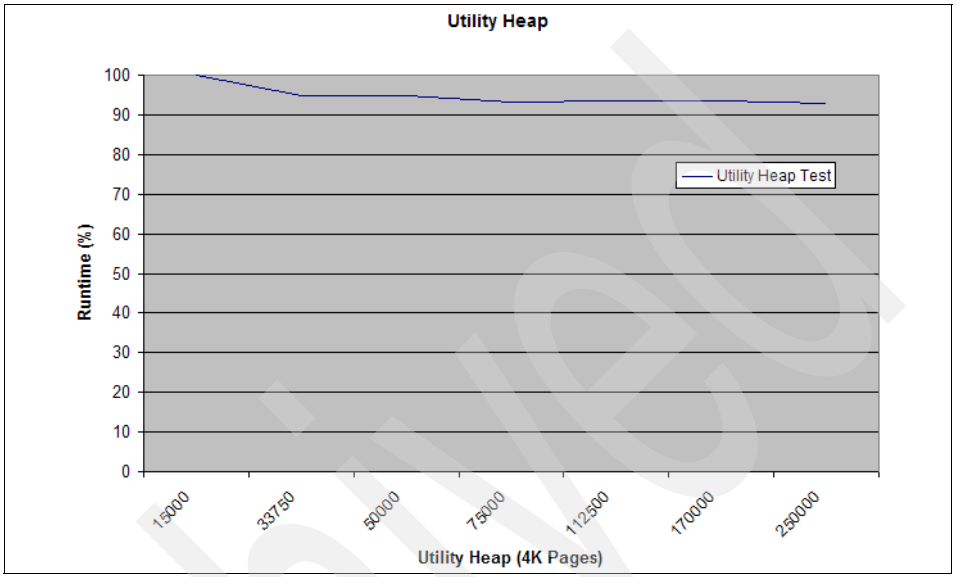


Figure 7-7 Utility heap

7.4 Order and configuration of index creation

Besides loading the data into the database, it is important to optimize the creation of indexes. Depending on their structure and size, index creation can consume a large portion of the overall migration time. The goal must be to minimize the time for index creation. Things to consider in this area are the order of creating indexes and memory usage.

To achieve best performance for the index creation, avoid concurrent index creation and avoid spilling data to disk. This can be influenced by controlling the order of creating indexes, mode of index creation, and choice of appropriate DB2 parameters (for example `SORTHEAP`). In addition, you must minimize the disk contention during index creation.

The R3load tool allows you to specify whether the indexes should be created before or after the data load. If you create the indexes before the load, they are created during the build phase of DB2 LOAD.

Example 7-4 is an example of a db2diag.log with entries for the build phase of the DB2 load.

Example 7-4 Messages in db2 diagnostic log file created by DB2 LOAD

```
2001-07-20-03.46.17.195667   Instance:db2fcp   Node:000
PID:5721(db2lrid)   Appid:*LOCAL.db2fcp.010719161634
database_utilities  sqlulPrintPhaseMsg   Probe:0   Database:FCP
Starting BUILD phase at 07-20-2001 03:46:17.193097.
2001-07-20-03.56.41.297931   Instance:db2fcp   Node:000
PID:5721(db2lrid)   Appid:*LOCAL.db2fcp.010719161634
database_utilities  sqlulPrintPhaseMsg   Probe:0   Database:FCP
Completed BUILD phase at 07-20-2001 03:56:41.288616.
```

Figure 7-8 shows results of testing index creation:

- Incremental index creation.
- Load data, then create primary key and index.
- Create primary key, load data, and then create index.
- Create primary key, index, and then load data.

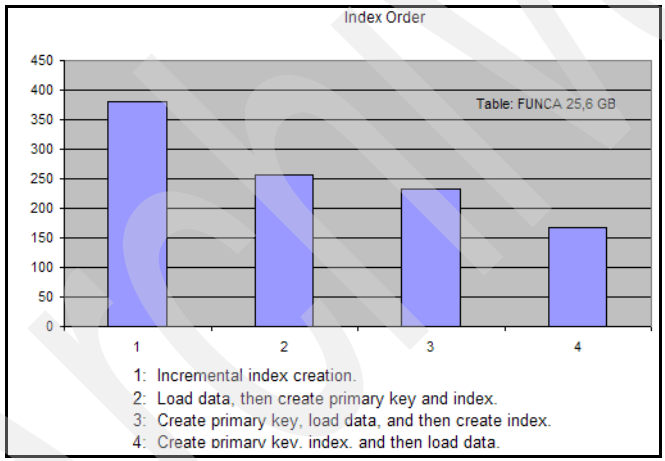


Figure 7-8 Index order

When looking at single processes, the combination of upfront creation of primary key and secondary indexes is the fastest solution.

The test for incremental index creation (`DB6LOAD_INDEXING_MODE=2`) showed the worst performance. Additionally, it consumed more space for storing the index pages. Therefore, we do not recommend using incremental index creation except for table splitting (see 5.1.5, “Results of import tests with table splitting” on page 77).

Figure 7-9 shows results with different tables that are in line with the result above except for table SWWLOGHIST, where the index creation after loading data is slightly faster.

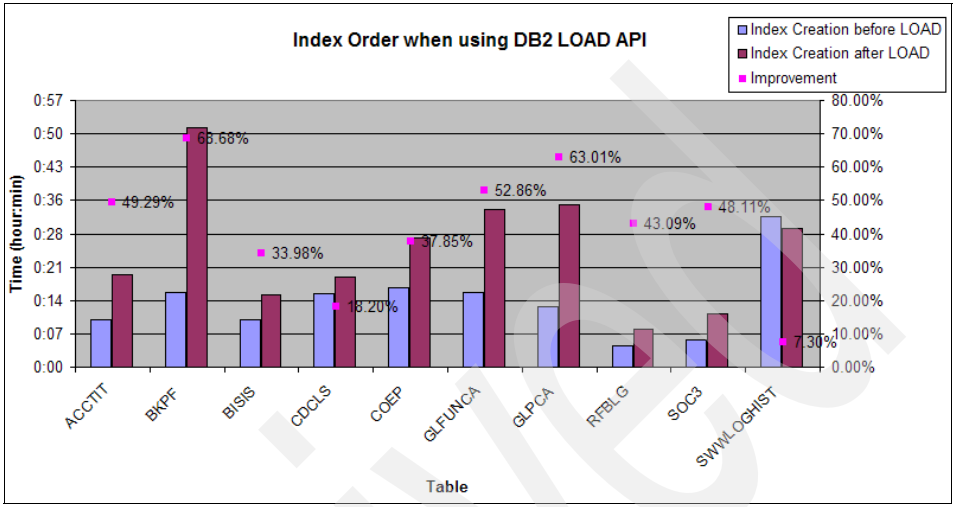


Figure 7-9 Index creation order

Starting with SAP releases using kernel Version 7.00, the default for indexes creation is `BEFORE_LOAD`. Editing the first two lines in the `DDLDB6.TPL` file can influence the order of index creation for R3load, as shown in Example 7-5.

Example 7-5 DDLDB6.TPL file is used to define the order of index creation

```
prikey: BEFORE_LOAD ORDER_BY_PKEY
seckey: BEFORE_LOAD
cretab: CREATE TABLE &tab_name&
        ( /{ &fld_name& &fld_desc& /-, /} )
        IN &location&
        INDEX IN &locationI&
        LONG IN &locationL&
drptab: DROP TABLE &tab_name&
```

The index creation typically generates the highest I/O load during the migration process. A good optimization strategy, therefore, is to distribute the index creation uniformly over the entire migration process to be able to use all available resources, for example, CPU, disks, and memory.

In some cases, for example, parallel import of a splitted table, it might be useful to change the order of index creation. In this case, an appropriate configuration for creating indexes is of some importance. Figure 7-10 shows the influence of the index creation order on the overall result. In general, the index creation before inserting data is the best choice. However, it might result in bad performance for certain tables, for example, GLPCA and SWWLOGHIST in our test case.

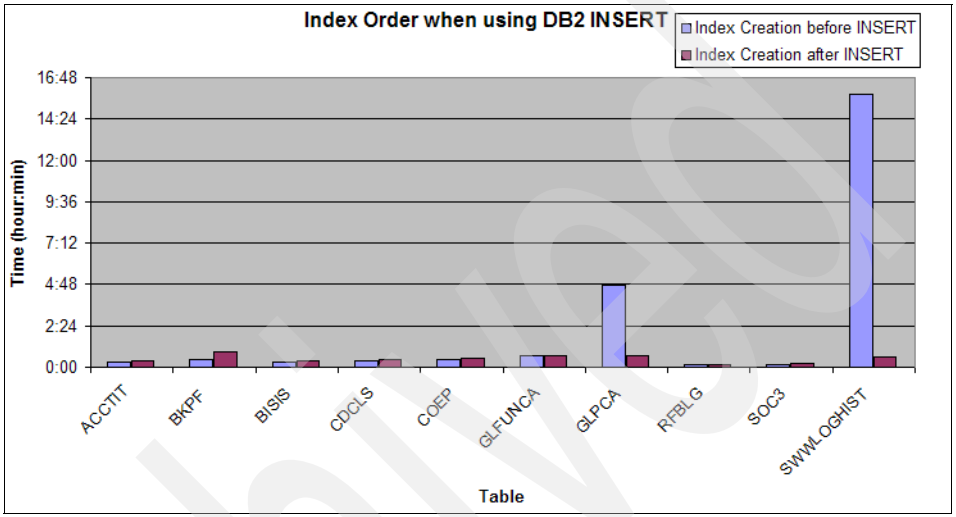


Figure 7-10 Performance impact of index creation before and after INSERT

The index creation before inserting or loading data is in most cases the best choice from a performance point of view. However, it can introduce an additional risk to the migration process. In case an R3load process fails during the index build phase (when using the DB2 LOAD API), the entire import fails and the LOAD must be restarted from scratch.

If an index is created after the table has been loaded, only the index must be recreated while the data in the table is still usable.

> **Note:** To avoid problems during index creation, ensure that there is enough space available for the temporary table space. As the final migration is performed some time after the test migration, tables and their indexes may have grown. We recommend adding additional space for the temporary table space to address this issue.

Especially when using DB2 LOAD, it is typically most beneficial to create the indexes before loading data. This means that the indexes are built during the build phase of the DB2 LOAD API.

In some cases it can be useful to perform the import with two MigMon runs to first load data and then build indexes. Each run can be performed with different configurations. See also 7.8, “Optimal number of R3load processes” on page 133, for additional information about the optimal number of R3load processes.

We provide guidelines for how to improve the index creation. The overall goal of the index creation phase is to limit the amount of I/O. In particular, sorting should occur in memory and the remaining I/O must be optimized.

Figure 7-11 shows the result of a series of tests with a DB2 release lower than DB2 V9.5.

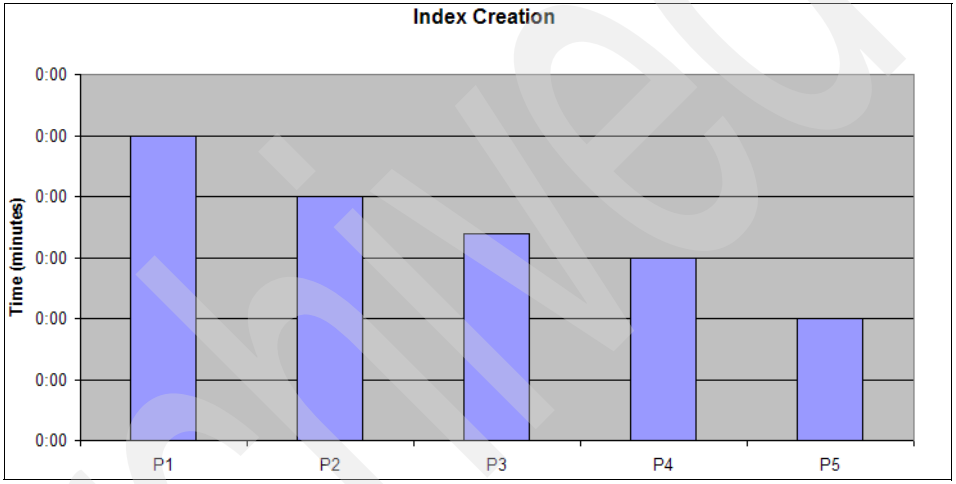


Figure 7-11 Utility heap

The tests shown in Figure 7-12 on page 128 are:

P1	Default configuration
P2	INTRA_PARALLEL enabled with parallelism of 5
P3	INTRA_PARALLEL enabled with parallelism of 12
P4	P3 + decreased CHNGPGS_THRESH
P5	P4 + increased SORTHEAP and SHEAPTHRES_SHR => no sort overflow

The tests P1 and P2 aimed to show the influence of parallelism by enabling `INTRA_PARALLEL`, which is the default with DB2 V9.5. The test P4 has optimized the I/O, and the test P5 finally configured DB2 to avoid a large portion of the I/O generated by a create index statement.

Other important parameters and configuration options that are related to the index optimization are discussed in the following sections. However, these parameters are not only valid for the index build process but also for INSERT or LOAD and are therefore of global importance.

7.5 LOCKTIMEOUT

During production operation you must set this parameter to an appropriate value to avoid deadlocks and to ensure that there are no applications waiting indefinitely.

During the migration process, we recommend setting the value of `LOCKTIMEOUT` to -1.

If you set this parameter to -1, lock timeout detection is turned off. It avoids the abortion of an index creation that is waiting for a lock held by another long-running index creation job. This may also be useful if you import multiple data packages in parallel using the DB2 LOAD API (WHERE splitter). In this case, the first LOAD process is locked and the other processes must wait until the data package is imported.

7.6 DB2 buffer pools

The LOAD API does not use the buffer pool for asynchronous writes. However, buffer pools are used during index creation and inserts. As buffer pools are competing with other memory areas like utility heap or sort heap, they should be configured wisely.

Unlike a *normal* workload, the migration process is somehow special, as only data is loaded and indexes are built during the import phase of the migration. To analyze and optimize the use of buffer pools, we must look closer at how buffer pools are used during the import. Figure 7-12 indicates that the use of buffer pools is dominated by the temporary data, and in particular it becomes clear that we have a high amount of physical I/O resulting in a poor buffer pool quality.

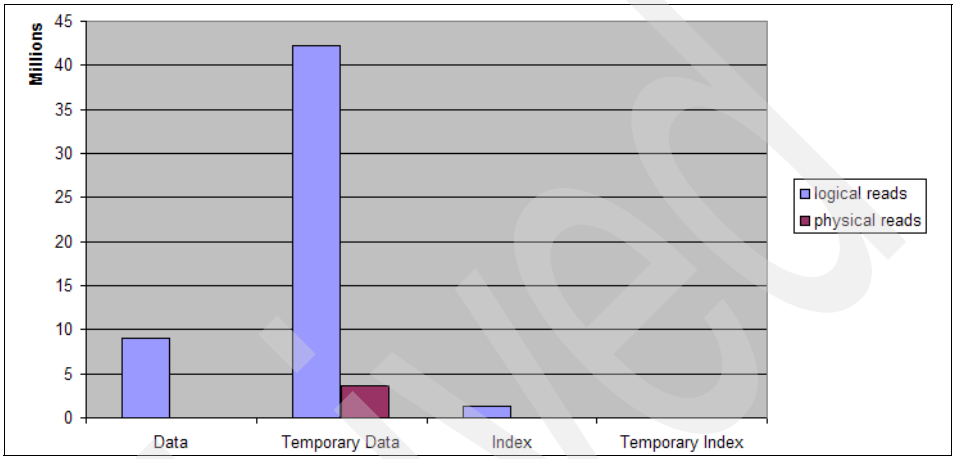


Figure 7-12 I/O distribution

Normally, you would increase the buffer pool to reduce the physical I/O. However, in our situation the temporary I/O is generated by the index build. Besides increasing the buffer pool, you can alternatively optimize the sort memory areas to reduce the I/O activities to a minimum.

You can specify small buffer pools for regular data and define one big buffer pool for temporary table spaces to assign the memory where it is best used. The success of this optimization must be tested.

Define the utility heap and the sort configuration with enough memory and assign the remaining memory to the buffer pools and the other DB2 memory areas.

The buffer pool configuration is of less importance if you use DB2 LOAD.

7.7 Import using DB server versus remote client

Typically, you can optimize the export by using application servers because R3load itself uses a significant portion of the available CPU resources. This applies specifically to Unicode migrations where the conversion is done during export.

In general, the benefit of using application servers for the import is not as big as during the export. Figure 7-13 shows the result of four tests with local and remote R3load processes.

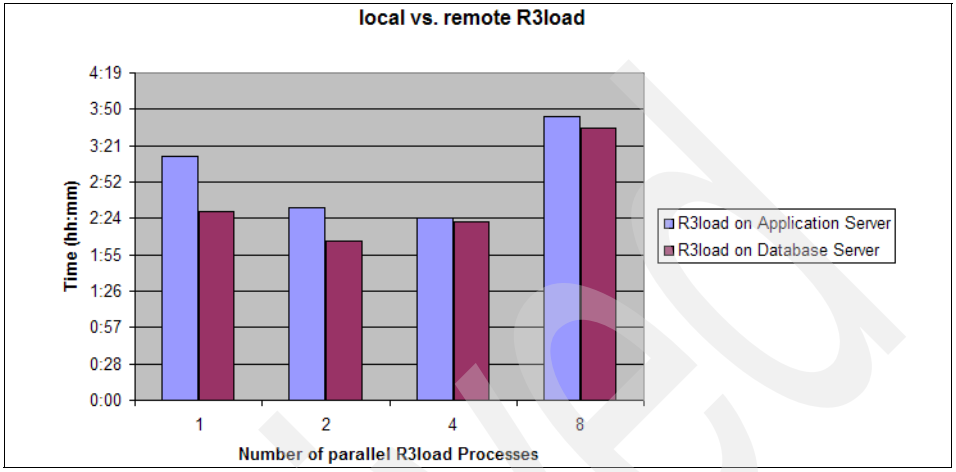


Figure 7-13 Import with local and remote R3load processes

The MigMon properties files are identical in every test case and differ only by the number of parallel load processes. Surprisingly, the setup with the local R3load is faster in all test cases. When we look at the CPU usage on the database server, it becomes obvious that fewer CPU resources are used if R3load processes run on the application server. See Figure 7-14.

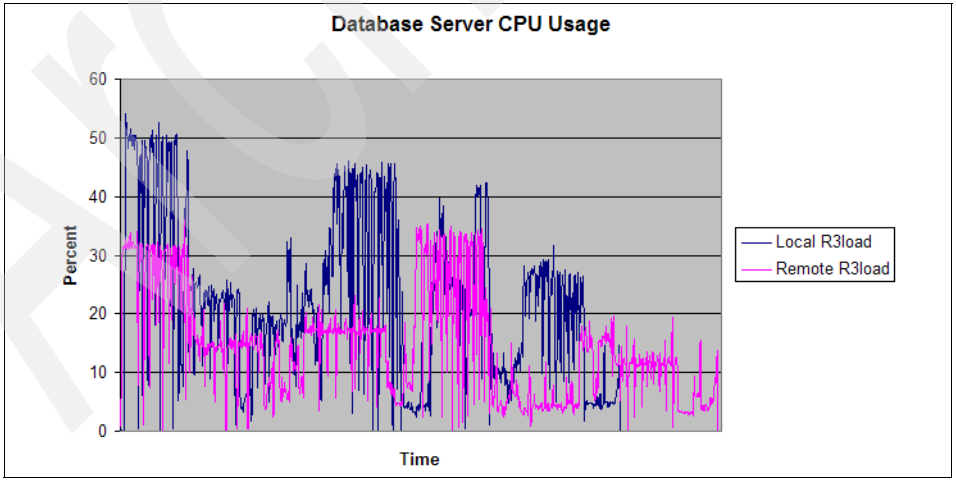


Figure 7-14 CPU usage on DB server with local and remote R3load

While the average CPU usage with local R3load is 20.3%, the average CPU usage decreases to 12.5% if R3load is running on the application server. Looking at the usage in detail, you can see that two CPUs are used on average 70% serving the two R3load processes, which means that the application server is using about 20% of its overall CPU capacity for the R3load process. See Figure 7-15.

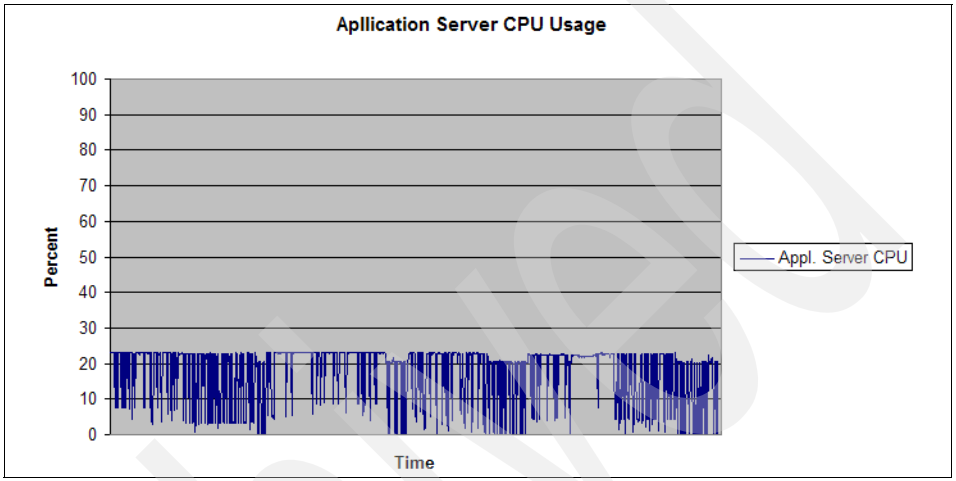


Figure 7-15 CPU usage on the application server

These results indicate that the majority of the workload during the database import is related to DB2 processes, while the R3load itself only generates a fraction of the workload. Looking at a typical workload distribution during a database load, there is a clear ratio between the database process and the R3load process.

Figure 7-16 shows the workload distribution between the database and R3load gathered from a migration of a customer system. The machine was constantly using about 100% CPU capacity and the database was consuming 80–90% of the CPU resources.

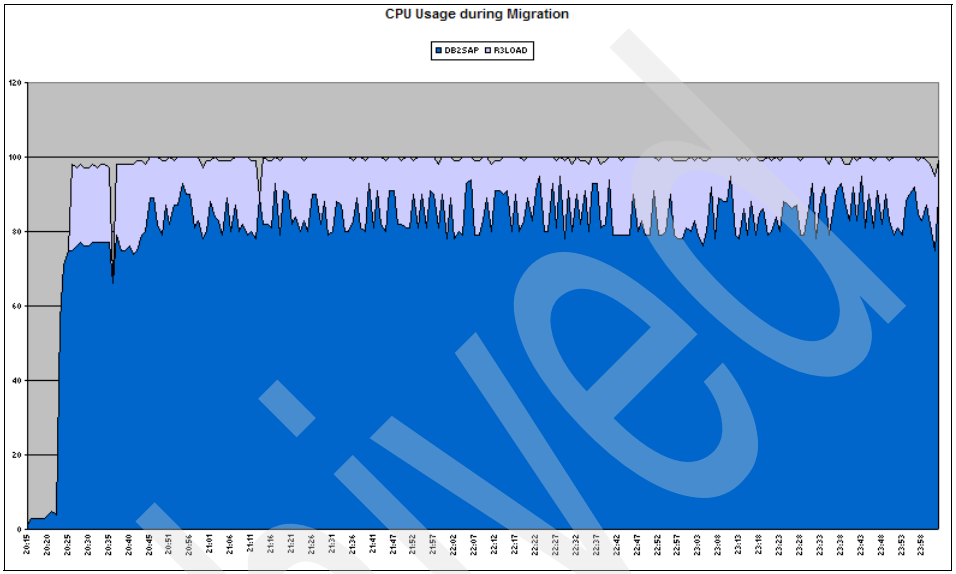


Figure 7-16 Workload distribution: Database versus R3load

Based on these results, a rule of thumb is that for the import about 10–15% of the CPU workload can be shifted away from the database server by using a separate application server for R3load. However, the question is still open as to why the database load was not faster in our test case when using remote R3load. Looking at resources like memory and disk I/O, our test case shows a similar workload profile as for the customer case.

Figure 7-17 shows the disk utilization of the test runs with local and remote R3load. The disks are very busy and partly at their capacity limit (100% usage).

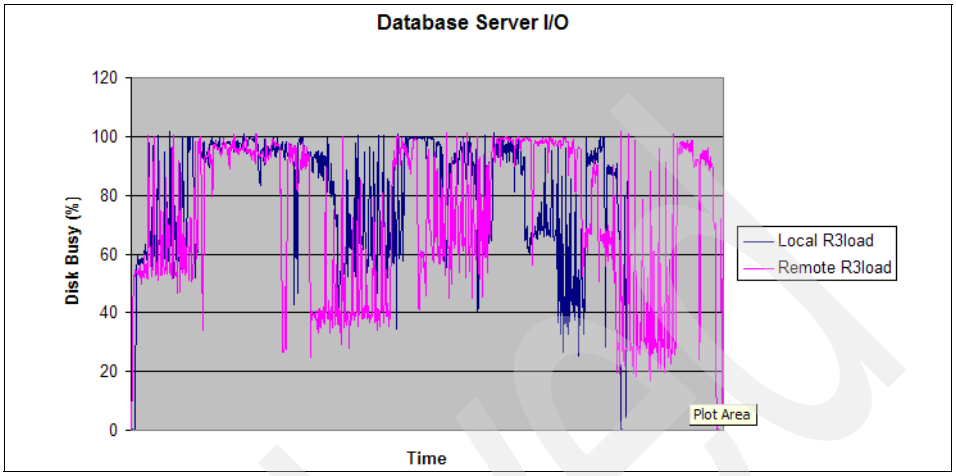


Figure 7-17 I/O Profile: Import with local and remote R3load

Looking at the network workload, you can clearly see the difference between local and remote R3load. See Figure 7-18.

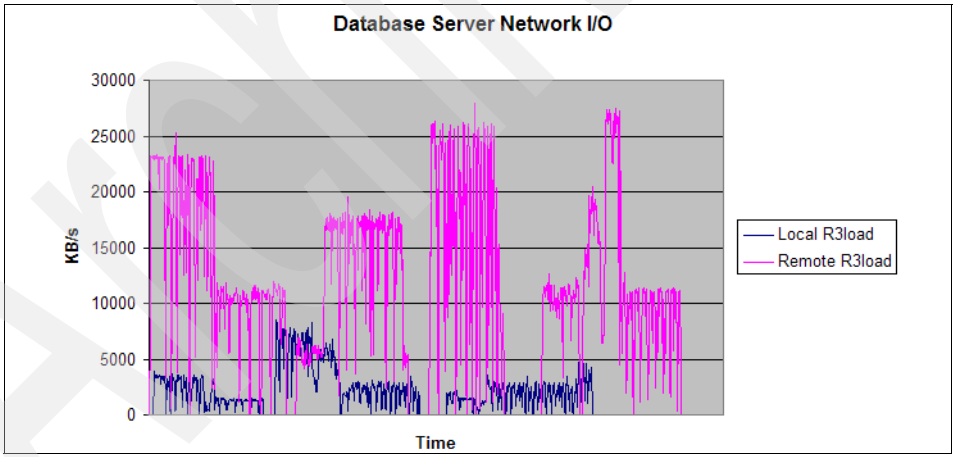


Figure 7-18 Network I/O during import with local and remote R3load

The network I/O is by factors higher if you use remote R3load because data is shipped across the network. In our test case we used Network Attached Storage. I/O operations are handled via the same adapter that is used for communication with the applications servers. The maximum amount of data that is transferred per second is 25 MBps, which is not the limit of the used Gigabit Ethernet

adapter. But the impact on the overall performance of the LOAD process is obvious. Collisions may appear and the latency in the network communication leads to slower performance. This is the reason for the reduced performance when running R3load on the application server.

This test result might not apply to all environments. If CPU on the target database server is the bottleneck, shifting the R3load workload to the application server can improve performance by around 10–15% if a fast network connection between application server and database server is available.

Hint: When starting R3load from a remote application server, note that the DB2 LOAD API cannot be used when the Thin Client is installed. To use LOAD functionality the DB2 RUNTIME Client is necessary. More information about this topic can be found in the DB2 section in the SAP COMMUNITY NETWORK:

https://www.sdn.sap.com/irj/sdn/db6

7.8 Optimal number of R3load processes

One important tuning option is to select the optimal number of import processes as well as the optimal number of export processes. As usual, the goal is to maximize the usage of all available resources like CPU, disks, memory, and network. While the optimization of the memory usage is mainly related to the DB2 configuration, the usage of CPU and disks is mainly influenced by the number of R3load processes that are running in parallel.

At first we consider the overall runtime of the import tests with different numbers of parallel R3load processes.

The tests ran on an 8-way machine, and Figure 7-19 shows an interesting result. The optimum number of parallel R3load processes is two (so just ¼ of the available CPU).

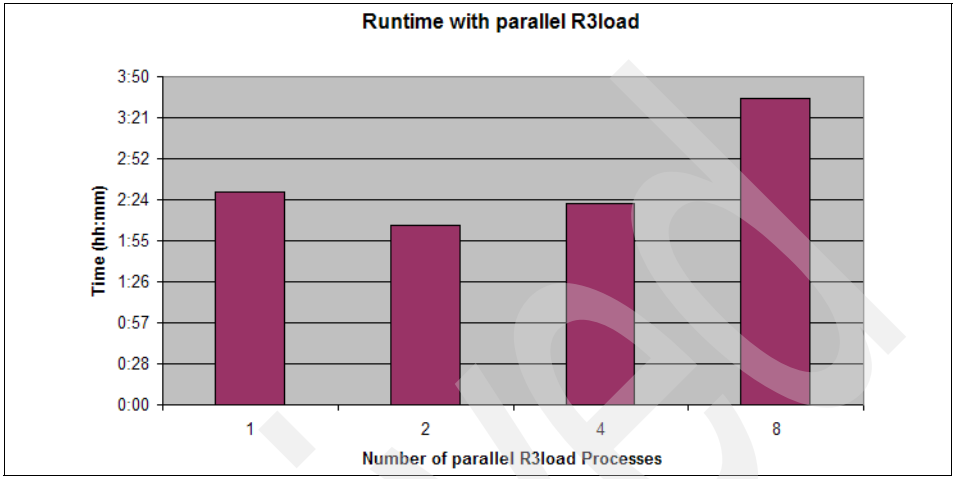


Figure 7-19 Runtime with different number of parallel processes

Let us look at runtime diagrams created for the different import runs. Figure 7-20 shows the different behavior of the R3load packages when using different levels of parallelism. Looking at the BKPF package as an example, you can see that the runtime of this package is increasing if several processes run in parallel. Therefore, the overall runtime of the import expands by using many parallel processes.

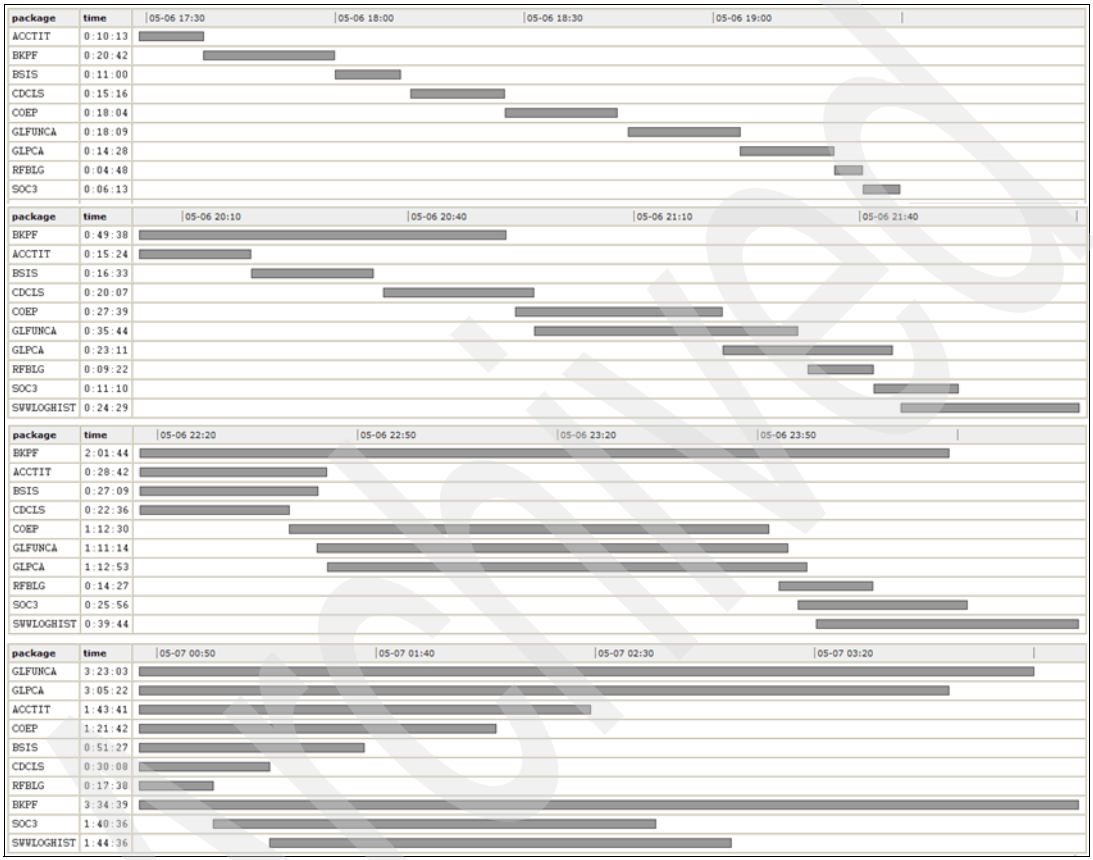


Figure 7-20 Import time comparison

Figure 7-21 shows this correlation between elapsed time and the sum of the time that all processes are running. While the overall elapsed time is slightly increasing with the number of processes, the sum of the single runtimes for each package increases significantly. Therefore, no additional scaling is possible if more than two parallel processes are used.

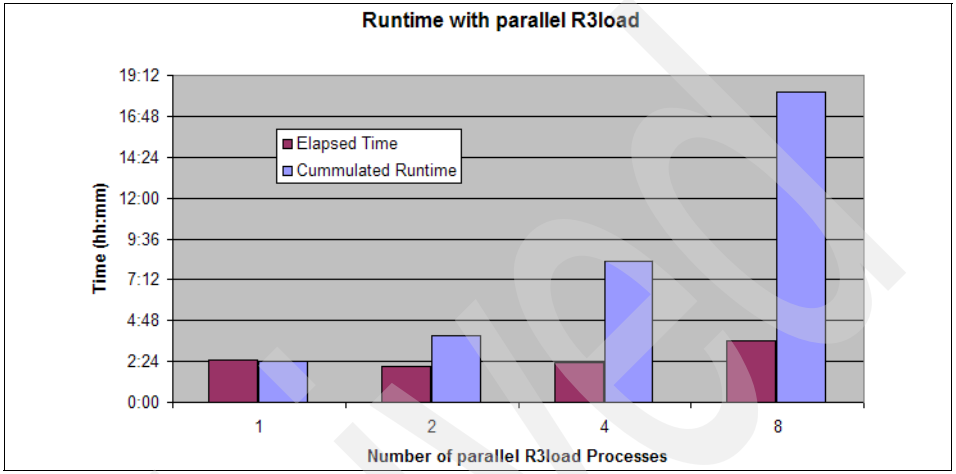


Figure 7-21 Elapsed time versus sum of all import runtimes

The CPU does not seem to be the bottleneck. But based on the setup of the disks (three SATA disks with RAID0 array), I/O could potentially be the bottleneck.

Figure 7-22 shows that the disks are busy at 100% if four and eight processes run in parallel. This is clearly a bottleneck in this test scenario. Running two parallel processes gives maximum performance.

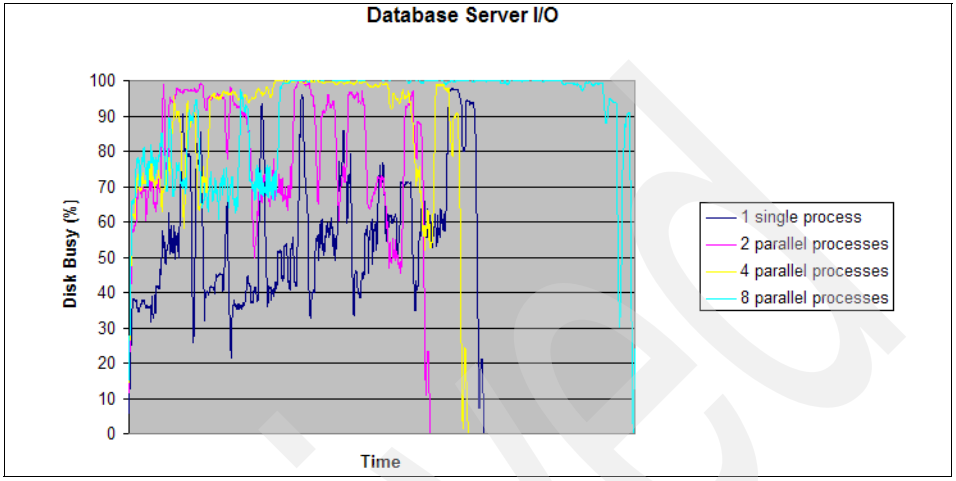


Figure 7-22 I/O workload of parallel import processes

Although it is the goal to use the available resources as efficiently as possible, you should avoid overloading your disks. They should not operate at 100% of their capacity limit over a longer period of time.

If you want to figure out the optimal number of parallel processes NMON (see 8.3, “NMON” on page 215) is a good idea to measure the resource usage during the test migration and to optimize the number of parallel processes accordingly. As explained, the optimal number of parallel processes depends on the specific hardware in use. We next provide additional hints for optimizing the import that are independent of the given hardware.

Figure 7-23, which correlates the CPU and I/O usage with the schedule of the packages during the migration test, explains one rule of thumb. The I/O workload profile changes during table import. Since DB2 LOAD is running in several phases, especially the I/O profile changes. Figure 7-23 indicates that the CPU usage during a table load stays stable but that the I/O is increasing at the end for some tables.

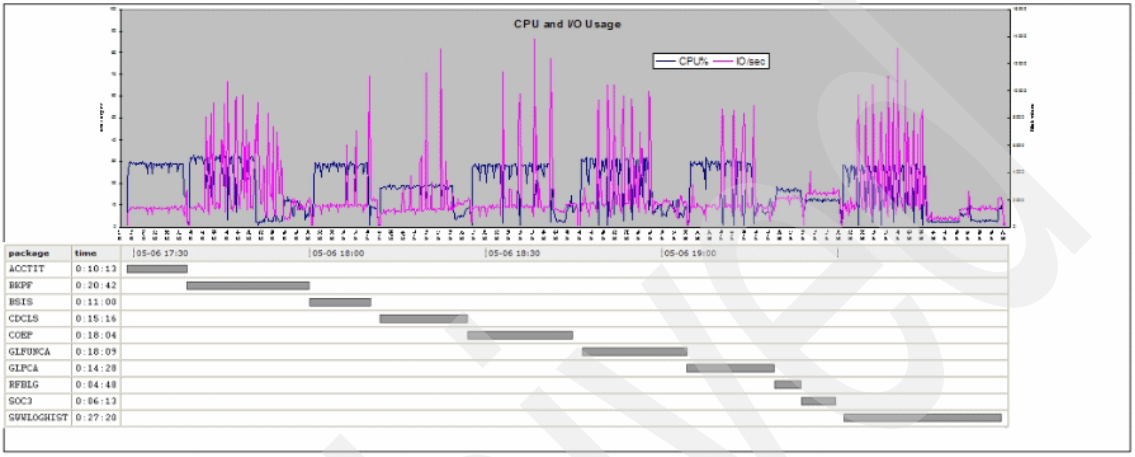


Figure 7-23 Correlation of workload and package runtimes

This is the impact of the build phase that is starting as the last phase of a LOAD. The index build phase can add more I/O and can overload the machine, which results in decreased performance. By changing the order of the import, performance can vary without changing any other configuration parameter or the underlying infrastructure.

In some cases it could be beneficial to force the index creation of all tables to the end of the migration. For example, if there is a clear CPU or memory bottleneck, a distinct index creation phase can help. This is a major change in the overall process and should be tested in detail.

We can formulate a second general rule of thumb that is related to the CPU usage of the different processors in the system.

As indicated in Figure 7-24, one CPU has a much higher workload compared with the others. A typical scenario is that one CPU is used up to 100% while others are using between 10% and 20% for a single load process. The CPU workload distribution depends on the parallelism used for DB2 LOAD, the speed of a single CPU, the number of CPUs, and whether other system limits are reached.

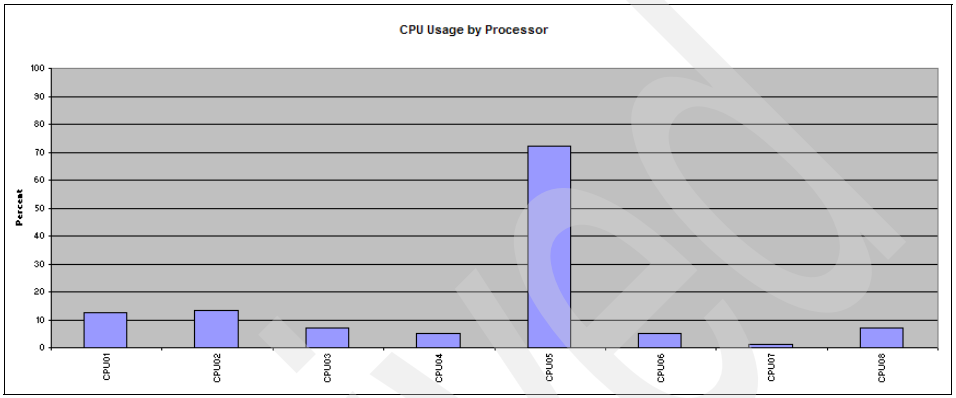


Figure 7-24 CPU distribution

To summarize the most important rule of thumb from this chapter, calculate the number of parallel R3load processes based on the number of available CPUs. A good start is typically to use the same number of R3load processes as CPUs or cores that are available.

This can be used as starting configuration, but you should monitor the resource usage during the migration process and ensure that no resource is overloaded and that no resources are idle.

7.9 Using DB2 self tuning memory management

With DB2 9.1, self tuning memory management (STMM) was introduced, which balances memory areas between several consumers based on their requirements and costs. These costs are associated with changing the amount of memory for the consumers.

STMM decreases the need for detailed memory configuration. It can tune the following memory consumers from the DB2 database global memory:

- Database `LOCKLIST` and `MAXLOCKS`
- Package cache size (`PCKCACHESZ`)
- Sort memory (`SHEAPTHRES_SHR`, `SORTHEAP`)
- Buffer pools

As discussed earlier in this book, some of the configuration parameters that effect the memory consumption are also important during a heterogeneous system copy or Unicode conversion. The values of `LOCKLIST`, `MAXLOCKS`, and `PCKCACHESZ` do not play a significant role for performance. The configuration for sorting and buffer pools is important. Out of these, the sorting configuration is more important, as the DB2 LOAD utility loads data into the database, bypassing the buffer pools. However, the buffer pool is also important and therefore STMM can help to find the optimal configuration.

To understand the capabilities of STMM in the context of the migration, we performed a series of tests. The first test series was to import 10 large tables. Since one test run takes about two hours, the import was repeated five times to give STMM a chance to adapt to the workload.

Figure 7-25 shows the configuration changes performed by STMM for the buffer pool, package cache, and sort configuration. The sizes for `MAXLOCKS` and `LOCKLIST` have never been changed during the tests and are not part of Figure 7-25. On the secondary axis, you can see the runtime of each of the runs as a blue line, so the x-axis does not represent the time but only the STMM change points, that is, every value represents an STMM change to the DB2 configuration. During the tests, STMM made 150 changes.

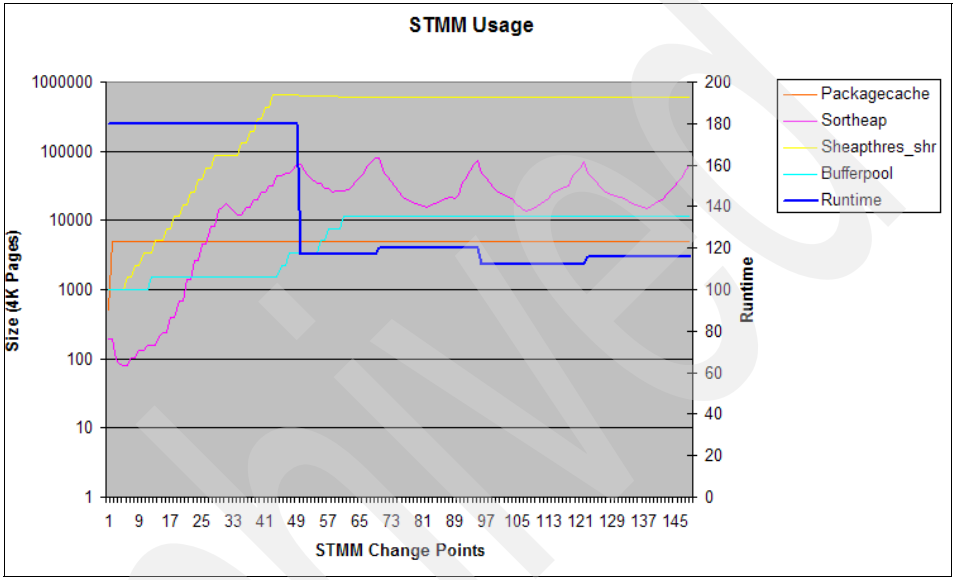


Figure 7-25 STMM configuration changes

You can see in Figure 7-25 that STMM rapidly adapts the size for the package cache, followed by the sort configuration, while the size of the buffer pool is mainly changed during the second test run. Already in the second test run, the runtime of the imports is near the optimal time (that was reached in run four). Starting with run three, the configuration remains stable except for the configuration for the `SORTHEAP`, which is increasing at the end of a run (influenced by the index builds) and decreasing again at the beginning of the next run. The runtimes also remain stable around the optimum and the changes in the runtime are varying between 112 and 120 minutes.

The value for `SHEAPTHRES_SHR` is stable at about 600,000 four KB pages while the value for `SORTHEAP` alternates between 15,000 and 60,000 pages. During the test, we used four parallel processes, and therefore the best practice for sort configuration is more or less validated by this test.

Looking at the reason for the improvement of the import time, we must look into the results in more detail. On our I/O-bound test system, we would most likely find the answer in the I/O configuration.

Figure 7-26 shows the difference in data read and written during the test. Approximately 100 GB less data is written, and only half of the data is read if you are using the optimal configuration by STMM. This again shows the importance of the appropriate sort configuration.

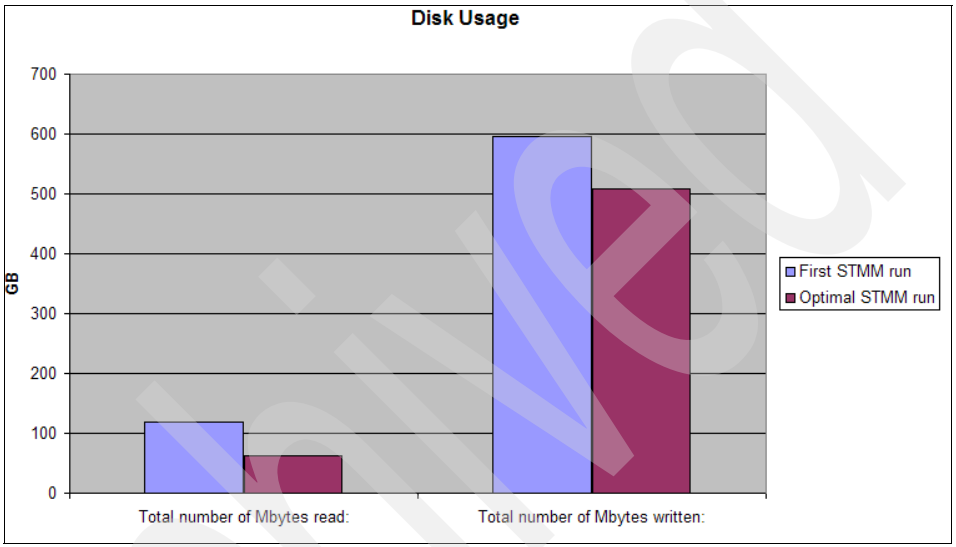


Figure 7-26 I/O reduction by STMM

Should STMM be used during the migration? There is some trade-off, for example, higher CPU usage for the STMM itself and, perhaps more importantly, the runtimes of the import process vary as small changes in the database or infrastructure configuration can have a huge impact on the overall runtime.

Figure 7-27 is similar to the STMM chart above, but here we started with a best practice configuration and a fixed buffer pool. As above, STMM is adapting the SORTHEAP, while other parameters are stable.

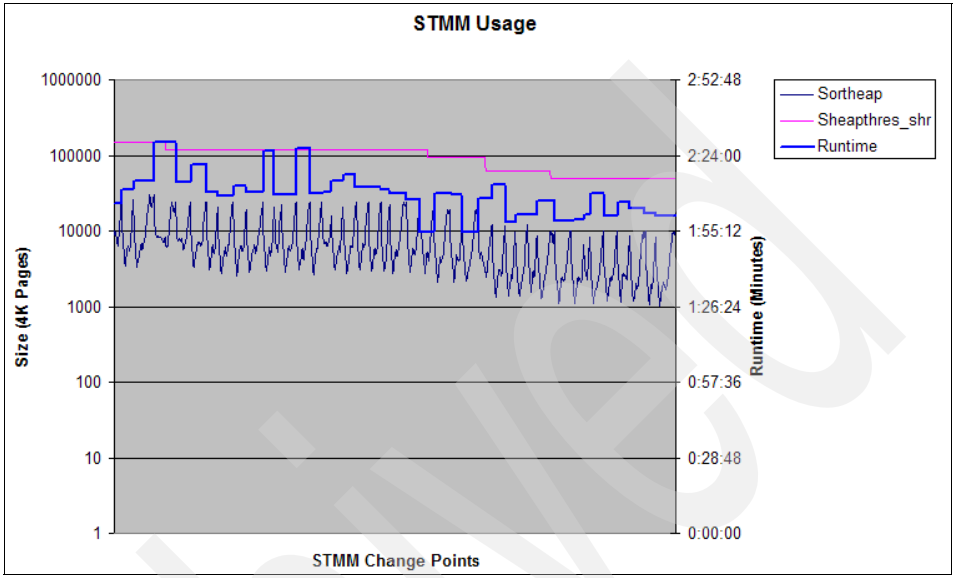


Figure 7-27 Long term STMM monitoring

As you can see in Figure 7-27, the runtimes are changing from run to run, and it could happen during a final cutover migration that the runtime is increasing while DB2 STMM is adopting the configuration.

Therefore, STMM is a good option to start during a test migration and to monitor the changes. We recommend that you start STMM with a best practices configuration. During subsequent test migrations, you can use the configuration found by STMM. A practical approach is to use the maximum value for SHEAPTHRES_SHR and SORTHEAP and to adapt the buffer pool according to the remaining memory.

Unfortunately, it is hard to extract the STMM changes from the DB2 diagnostic log file or the STMM logs. In the appendix, you can find a simple script that parses the db2diag.log for STMM changes and creates a file with the essential information.

7.10 Unicode influence on LOAD

As seen in Chapter 4, “Export optimization techniques” on page 35, the Unicode conversion has some influence on the export runtime, the CPU, and disk usage. This section describes the influence of the Unicode conversion on the import.

To validate the influence, the 10 large test tables were exported to a Unicode and to a non-Unicode dump and were imported into two databases that were set up and configured identically.

Figure 7-28 shows the result of the comparison.

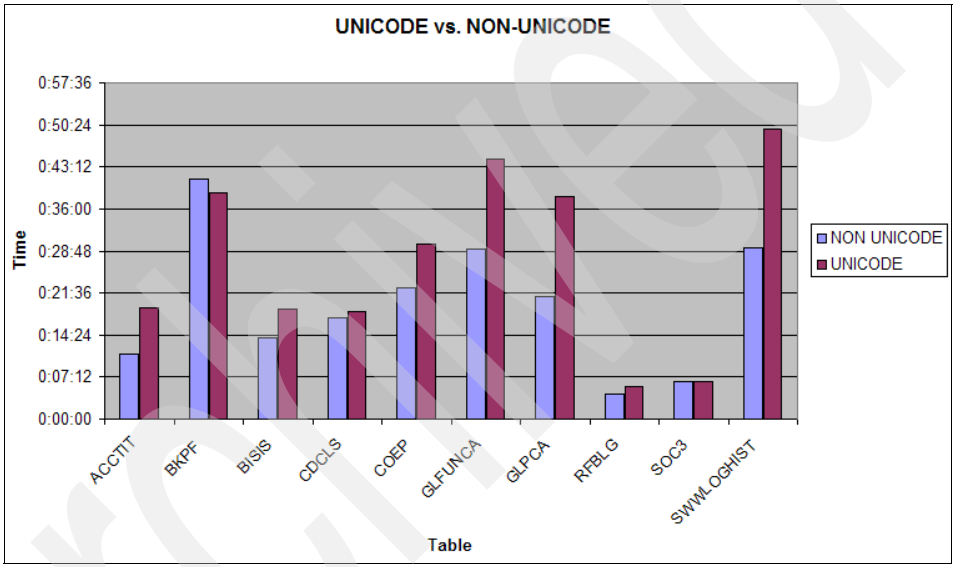


Figure 7-28 Import runtime comparison Unicode versus non-Unicode

Figure 7-28 shows that the import of the Unicode export files on average takes longer than the non-Unicode case. In particular, the tables ACCTIT, GLPCA, SWWLOGHIST, and GLFUNCA show significantly longer runtimes that, in the worst case, are almost doubled.

DB2 LOAD typically shows significant performance improvements over INSERT for most of the tables like GLPCA, GLFUNCA, or SWWLOGHIST. However, the improvement is less on the table ACCTIT (factor two in our tests), and so this table could be a candidate for insert and the table split option that allows multiple R3load processes to populate the table in parallel.

The above results lead to the question of whether you should use different migration approaches for those tables and (what is perhaps more important) how to identify those tables.

Typically, the goal of a system copy is to export and import the database within a given downtime window that is defined by the business needs. The goal is not to migrate as fast as possible for the sake of a fast migration. Therefore, you should only use those tuning options that are necessary to meet your downtime requirement. In our case, as long as there is not an issue with LOAD and Unicode, use the LOAD API.

Identifying the potential tables for special treatment during the Unicode import might be difficult. The easiest approach is to compare the import rates (GB/hour) of the largest tables after a test migration and, if there is a significant difference in the throughput, these tables are potential candidates for alternative approaches.

Based on our test results some upfront analysis can be made by checking the data structures of critical tables. The tables that show the above effect during a Unicode import typically have text fields. For example, the tables COEP, ACCTIT, GLPCA, and GLFUNCA all have a VARCHAR(150) defined (Field SGTXT) that contains strings. But it is not enough to compare the defined fields. You must actually look into the data that is stored inside the tables. For example, table BKPF also has a VARCHAR(120) field, but it is not influenced by the Unicode import.

The difference is in the actual usage of the defined field. In our test, this field in BKPF holds only one character per record, while the SGTXT field in the other tables is used up to its defined maximum. The difference in runtime and CPU usage is related to the fact that DB2 must convert the data from the UTF-16 format (coming from SAP) to the internally used UTF-8 format. For details about this, check Chapter 10, “Unicode details” on page 225.

For the sizing of the target landscape, some details about the resource usage might be of interest.

Figure 7-29 shows the CPU usage during the import. You can see an increase in CPU usage from 28.4 to 33.7%, which in fact is an increase of almost 16% that must be incorporated into the sizing.

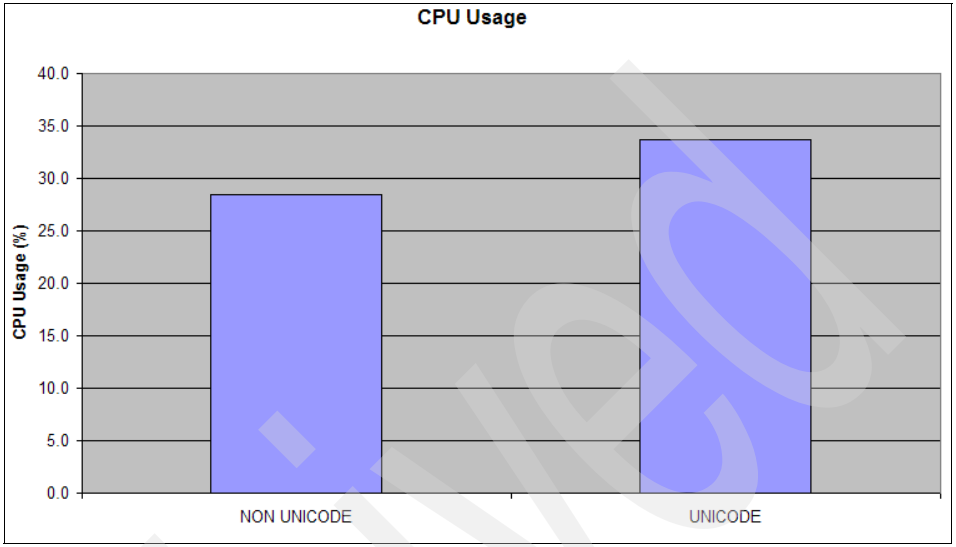


Figure 7-29 CPU usage during unicode import

7.11 Temporary table spaces

For SAP databases, the data and index table spaces are by default defined as database managed space (DMS) table spaces. Temporary table spaces are defined as system managed space (SMS) table spaces. This setup also applies if you use automatic storage management (ASM). ASM automatically creates SMS or DMS table spaces.

While DMS table spaces for data and index are the best option to use, you may want to change the type for temporary table spaces in certain cases. In general, temporary table spaces in an SAP environment are created as SMS table spaces. However, if a system encounters a significant amount of large I/O operations, for example, large sorts on the temporary table spaces, DMS could be an option to use.

Figure 7-30 compares runtimes of a test import using SMS or DMS temporary table spaces.

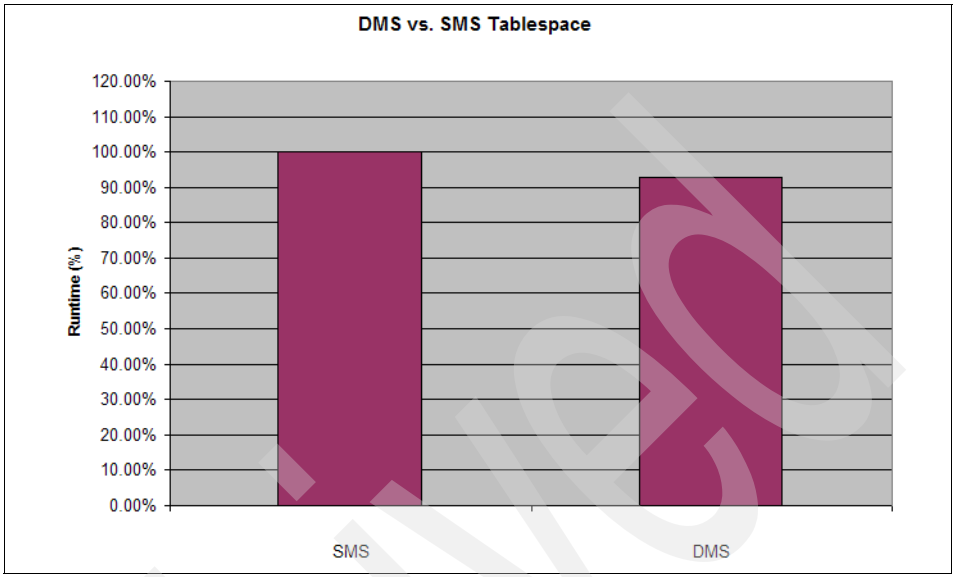


Figure 7-30 Performance of import with DMS and SMS temporary table space

The runtime of the test import with DMS temporary table spaces is about 5% faster compared with the test import based on SMS temporary table spaces. Therefore, DMS temporary table spaces may be used to optimize the import runtime.

There are some restrictions though. The creation of DMS temporary table spaces has not been implemented in SAPInst and therefore you must create these table spaces manually. The existing SMS temporary table spaces should be dropped to ensure the use of the DMS temporary table spaces.

Example 7-6 shows the DDL to create a DMS temporary table space.

Example 7-6 Syntax to create a DMS temporary table space

```
CREATE TEMPORARY TABLE SPACE PSAPTEMPDMS IN DATABASE PARTITION GROUP
IBMTEMPGROUP PAGESIZE 16384 MANAGED BY DATABASE USING (
FILE '/db2/BN6/saptemp1/NODE0000/tmpdms/PSAPTEMP.container000' 148800,
FILE '/db2/BN6/saptemp1/NODE0000/tmpdms/PSAPTEMP.container001' 148800,
FILE '/db2/BN6/saptemp1/NODE0000/tmpdms/PSAPTEMP.container002' 148800,
FILE '/db2/BN6/saptemp1/NODE0000/tmpdms/PSAPTEMP.container003' 148800)
EXTENTSIZE 2
PREFETCHSIZE AUTOMATIC
BUFFERPOOL IBMDEFAULTBP
```

```
OVERHEAD 7.500000
TRANSFERRATE 0.060000
AUTORESIZE YES
MAXSIZE NONE
NO FILE SYSTEM CACHING
DROPPED TABLE RECOVERY OFF;
```

Another and probably more important restriction is that only one DB2 REORG process is allowed per DMS table space. This restriction probably interferes with deployment of DB2 compression during import. A compression dictionary is created by reorganizing the respective table. As R3load uses the temporary table space during the REORG, a second parallel R3load might fail with the following error:

```
SQL2216N  SQL error "-294" occurred while reorganizing a database table or its
indexes.
```

Therefore, if you want to use a DMS temporary table space to perform the import and DB2 data compression in one step, make sure that the required reorganization of the corresponding tables does not overlap.

Figure 7-31 shows a series of import tests that were performed sequentially. The first series (test 1 to 4) was performed with the SMS temporary table space, and the second series (test 5 to 8) was performed with DMS temporary table space.

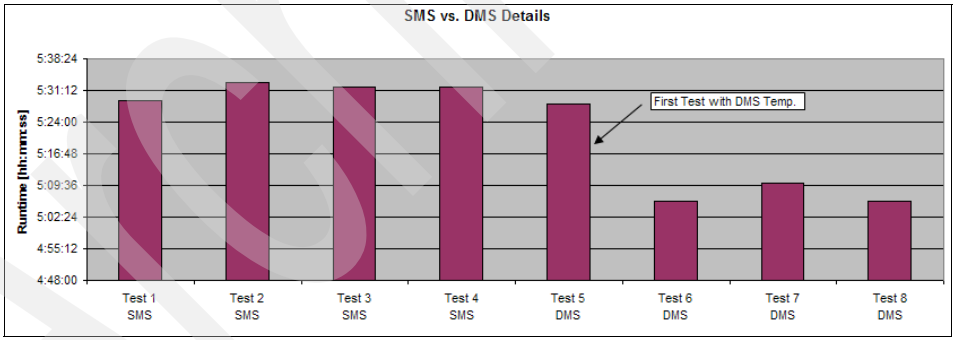


Figure 7-31 Details for DMS temporary table space

If you look at the test results in more detail, you can see that the first test with a DMS temporary table space (test 5) has almost the same runtime as the tests with SMS temporary table space. Subsequent tests with DMS ran faster. This is due to the fact that the DMS temporary table space was not allocated to its maximum required size at the beginning of the first run and therefore performance decreased because the table space was extended automatically (autoresize).The test was running on Linux with reiserfs. On AIX with JFS2 or Windows NTFS, you will probably see a different behavior.

We therefore recommend creating the DMS containers with the maximum required size to avoid autoresize activities. You can run the first test migration starting with small containers to determine the appropriate size and extract this information by using the `db2look` utility (`db2look –d <SID> -l`), or you can use the file createTable spaces.clp in the SAPInst directory as a basis to create the DMS temporary table space.

7.12 CHNGPGS_THRESH

You can use the changed pages threshold (`CHNGPGS_THRESH`) parameter to specify the level (percentage) of changed pages at which the asynchronous page cleaners will be started if they are not currently active. If the page cleaners are started, they build a list of the pages to write to disk. Once they have completed writing those pages to disk, the page cleaners become inactive again and wait for the next trigger to start. The `CHNGPGS_THRESH` parameter becomes important for large tables that are imported by INSERT, for example, parallel insert of splitted tables. Since all the pages in the buffer pool must be flushed to disk after a commit, this operation could become a bottleneck, as a large portion of data is written to disk at once. If you set `CHNGPGS_THRESH` to a lower value, the dirty pages can be written earlier and this could speed up the import process.

On the other hand, lowering this value could influence the performance of index build. The `CHNGPGS_THRESH` parameter can force temporary data for index creation to be written to disk even if it would fit into the buffer pool. Therefore, overall performance decreases, as seen in Figure 7-32.

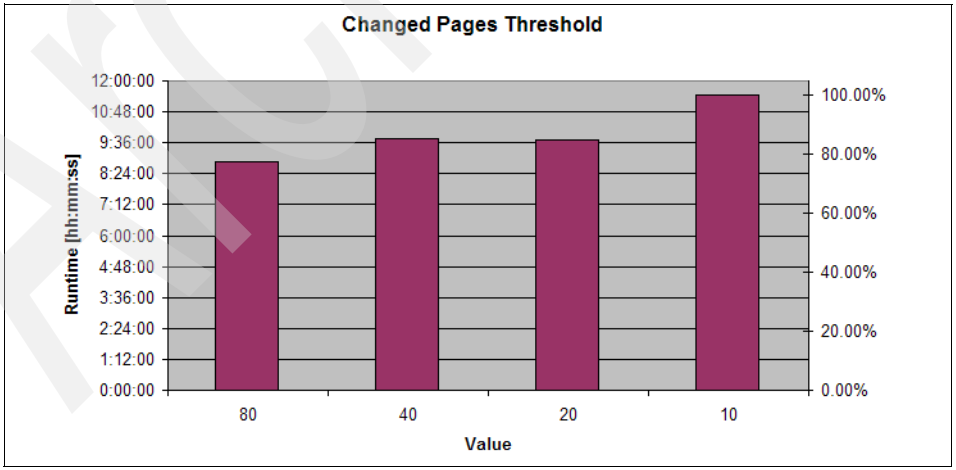


Figure 7-32 Import times with different changed page threshold

Tuning this parameter is difficult, and we can give no general recommendation. If you are using the default method of creating the indexes before the LOAD, the default value of 40 is a good starting point. In special cases you might want to consider changing CHNGPGS_THRESH if indexes are created after the table has been populated.

7.13 SORTHEAP

This parameter defines the maximum number of private or shared memory pages to be used for a single sort. You should define this parameter to be as large as possible for the index creation, which might avoid sort overflows and spilling data to disk. This parameter can be set significantly higher compared with a typical value during production. While the recommendation for productive operation of ERP systems is 2048, SORTHEAP should be set to a significantly larger number during migrations. Although it is usually not possible to avoid spilled sorts during a migration of large tables, the goal is to avoid as many sort overflows as possible. So if enough memory is available, setting SORTHEAP to a value of 50,000 pages is common. To optimize this value, monitor the database for sort overflows and adjust the parameter accordingly together with the SHEAPTHRES_SHR configuration parameter.

7.14 SHEAPTHRES_SHR and SHEAPTHRES

SHEAPTHRES_SHR represents a soft limit for the total amount of database shared memory that can be used by sort memory consumers at any time, whereas SHEAPTHRES defines the limit on the instance level. As per the SAP recommendation, SHEAPTHRES is set to 0. When the instance-level SHEAPTHRES is set to 0, the tracking of sort memory consumption is done at the database level only and memory allocation for sorts is constrained by the value of the database-level SHEAPTHRES_SHR configuration parameter.

Automatic tuning of SHEAPTHRES_SHR is allowed only when the database manager configuration parameter sheapthres is set to 0.

Figure 7-33 shows the results of tests to find the optimal combination of SORTHEAP and SHEAPTHRES_SHR for different tables.

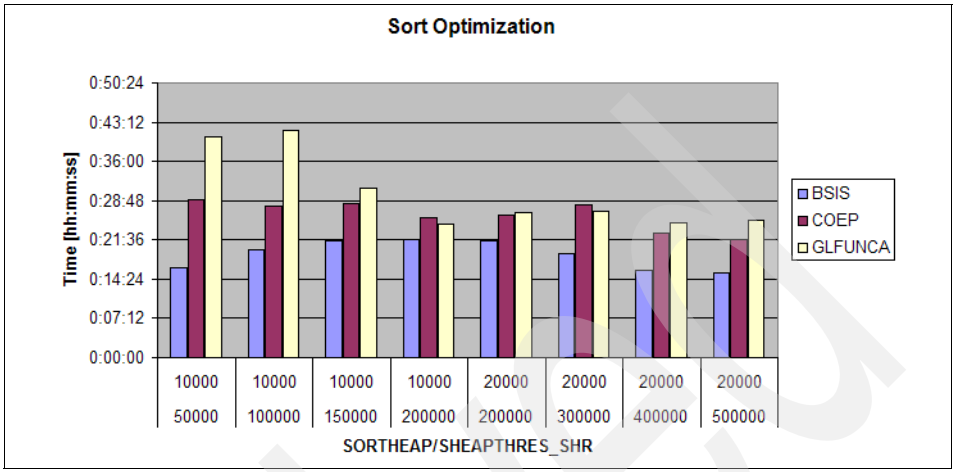


Figure 7-33 Sort optimization

In this test, all tables were loaded sequentially. You can see that for the table GLFUNCA the increase of the SHEAPTHRES_SHR parameter from 150,000 to 200,000 pages already gives maximum performance, whereas the optimum for tables COEP and BSIS is reached with a combination of 20,000 pages for SORTHEAP and 400,000 pages for SHEAPTHRES_SHR. Optimizing for the latter means reserving the memory and not giving it to other memory consumers. This might result in an overall performance decrease. Therefore, the goal is to avoid sort overflows to assign sufficient memory for sorts but not to waste memory.

Ideally, you should set SHEAPTHRES_SHR to a reasonable multiple value of SORTHEAP.

A good starting configuration is to set the value of SHEAPTHRES_SHR to twice the number of parallel R3load processes multiplied by the SORTHEAP value. However, the optimal value could be higher, as indexes are created in parallel and so multiple parallel sorts for a single index create may run in parallel.

7.15 INTRA_PARALLEL

This parameter specifies whether the database manager can use intra-partition parallelism. This parameter can have an effect on index creation up to DB2 Version 9.1. Starting with DB2 Version 9.5, index creation automatically uses parallelism, so this parameter has no effect in this case. For older versions of

DB2 the parameter can be enabled to improve performance of the index creation phase if the I/O subsystem is capable of handling the additional workload.

> **Note:** `INTRA_PARALLEL` should be used carefully, as it effects the CPU usage, the memory usage, and the I/O behavior of the database significantly. Additional testing is required to use this functionality.

7.16 Disable database logging

The R3load offers the parameter `–nolog` to avoid the logging of data while inserting it into the database. With this R3load option, you can reduce logging to a very minimum. However, the best practice is to use the DB2 LOAD API. In this case the `–nolog` option only affects small tables or exceptional tables that are not using the DB2 LOAD API.

If a table was created with the option NOT LOGGED INITIALLY, any changes made to this table by an INSERT, DELETE, UPDATE, CREATE INDEX, DROP INDEX, or ALTER TABLE operation in the same unit of work in which the table is created are not logged. You can deactivate logging during R3load, which could avoid disk contention and logging overhead during inserts. It also can effect the asynchronous writes (see `CHNGPGS_THRESH`) during insert. In addition, if you run the rollforward utility and it encounters a log record that indicates that a table in the database was populated with the `NOT LOGGED INITIALLY` option, the table will be marked as unavailable. Therefore, perform a full offline backup after the migration to ensure that the database and all tables are recoverable.

The various options to compress the tables while using R3load with DBSL Insert are not available if you are using the -nolog option (except DB2 9.5 automatic dictionary creation).

The `-nolog` option reduces the amount of logging but is typically not faster than logging enabled. So this option may only be used when logging I/O is limiting the performance.

7.17 Conclusions

To summarize the findings and recommendations in this section, we have grouped optimizations into three different classes:

- The default optimizations
- The optional optimizations
- The other functionality that is either complex to implement, needs significantly more testing, or shows side effects that may influence the overall migration process.

The default optimizations include:

- Use the DB2 LOAD API with default settings for R3load.
- Specify enough utility heap to ensure parallelism of the LOAD API. Use 200,000 as a starting point.
- Configure `SORTHEAP` and `SHEAPTHRES_SHR` to accommodate the large sorts during index build. A good starting point is 50,000 pages for `SORTHEAP` and `2 * (SORTHEAP) * (number of R3load processes)` for `SHEAPTHRES_SHR`.
- Do not use STMM during final migration test and migration cutover.
- Set `LOCKTIMEOUT` to `-1` to avoid failing R3load processes due to wait situations.
- Define one buffer pool that is using the remaining memory after utility heap and sort heap has been configured.
- Leave the other configuration settings according to the SAP Notes for DB2 standard parameter settings.
- Create primary and secondary indexes before data is loaded and ensure sufficient disk space for temporary table spaces.
- Start with the same number of R3load processes as available CPU cores.
- Do not use the `-nolog` option in R3load unless logging I/O is becoming an issue during the migration.

The optional optimizations are:

- Use STMM during a test migration to determine optimized values.
- Optimize the package import order to avoid many parallel index build processes on large tables.
- Monitor the import process and adjust the configuration for utility heap, buffer pool, and sorting.
- Monitor and optimize the configuration for I/O, for example, `NUM_IOCLEANERS`, `DB2_PARALLEL_IO`, or disabling file system caching for logging.

- Analyze the resource usage of the system (CPU, I/O, and memory) to optimize the number of parallel processes with respect to evenly distributed resource usage.
- Consider the influence of an Unicode import in the sizing and check potential candidates for switching from LOAD to parallel inserts.

The advanced optimizations are:

- Use dedicated application servers to perform the import if:
  - The I/O subsystem can handle additional workload.
  - The target system is short on CPU.
  - A fast network connection between application server and database server is available.
- You can use a DMS temporary table space to optimize the index rebuild phase, but you should be aware of the side effects, for example, only a single REORG process is possible and the need to create the temporary table space manually.
- Change the order of index creation for selected tables if the index build phase takes significantly longer compared with other large tables.
- If you run the index build after data load with a separate MigMon run. For example, if you use INSERT and table splitting, you may enable `INTRA_PARALLEL` with DB2 9.1 and optimize the `CHNGPGS_THRESH` parameter together with buffer pool size to optimize I/O for the index build.
- Create dedicated buffer pools for one or more table spaces (for example, for temporary table space).

7.18 R3load with DB2 compression

This chapter explains how data can be compressed during import into the database with DB2 9.1 or later.

7.18.1 Introduction to DB2 row compression

The row compression feature, also called deep compression, introduced with DB2 9.1 uses a dictionary-based approach to replace repeating patterns with short symbols. The dictionary stores patterns that occur most often and corresponding symbols that are used to replace these patterns. This compression approach is depicted in Figure 7-34.

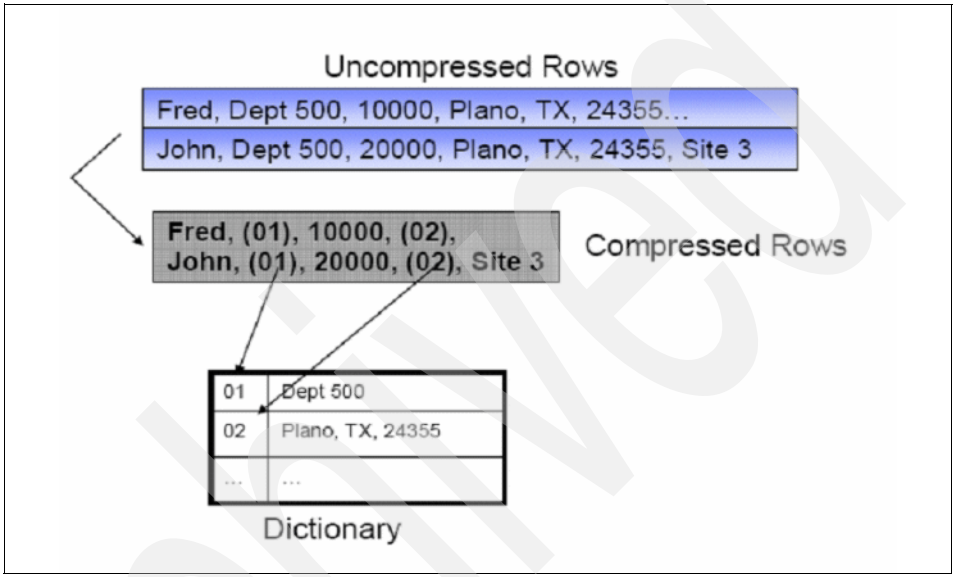


Figure 7-34 DB2 compression

To prepare the table for compression the compression attribute must be enabled. This can be done as follows:

- During the creation of a table using the following command:

```
CREATE TABLE <tabname> COMPRESS YES
```

- By altering an existing table using the following command:

```
ALTER TABLE <tabname> COMPRESS YES
```

You can then create the dictionary and compress the data with the following command:

```
REORG TABLE <tabname>
```

This command starts an offline REORG that creates the compression dictionary and also compresses existing rows. All rows that are inserted or loaded subsequently into the table will be examined. If a matching pattern exists in the dictionary, this pattern will be replaced by the corresponding symbol.

The compression dictionary can also be generated online using the following command:

```
INSPECT ROWCOMPESTIMATE TABLE NAME <table_name> RESULTS KEEP <file_name>
```

Existing data remains unchanged. All rows, however, that are inserted or loaded into the table subsequently are compressed according to the existing dictionary. The INSPECT command only creates a dictionary if the compression flag of the table is switched on and no dictionary exists. If a dictionary already exists, you must use the REORG method. In this case, you can choose between two options:

- REORG TABLE <tabname> KEEPDICTIONARY

  Keeps the current existing compression dictionary
- REORG TABLE <tabname> RESETDICTIONARY

  Creates a new compression dictionary

7.18.2 Automatic dictionary creation

Automatic dictionary creation (ADC) was introduced with DB2 9.5. It creates a compression dictionary automatically when the following conditions apply:

- The `COMPRESS` flag was enabled.
- No compression dictionary exists for the table.
- The table contains a sufficient amount of data.

ADC is triggered after a table was populated with approximately 2 MB of data. DB2 automatically creates a compression dictionary and compresses all rows that are added subsequently.

You can use ADC in combination with R3load to create a compression dictionary while data is loaded.

7.18.3 R3load options to compress data during import into DB2

A very elegant way of enabling DB2 row compression during SAP system copies, Unicode conversions, or database migrations is to compress the data at the time that it is loaded into the database. The R3load tool provides several options to deploy DB2 compression and to compress the data when it is loaded into the tables.

Depending on the underlying R3load version, several options are available. For information about R3load options for Version 6.40, see 7.18.4, “DB2 row compression based on R3load 6.40” on page 157. For information about R3load

options that are available for Version 7.00 or later, see 7.18.5, “DB2 row compression based on R3load 7.00” on page 163.

Using the socket option of R3load together with its compression option *Bernoulli sampling* can provide significant runtime advantages. For more information see 7.18.6, “Bernoulli sampling method” on page 182.

7.18.4 DB2 row compression based on R3load 6.40

R3load Version 6.40 introduced a feature that you can use to compress the data during the import into the DB2 database. For more information see SAP Note 905614. In this section this technique is also called *compressed import*.

To generate a dictionary, R3load first loads a certain number of rows into the table without compressing them. R3load creates the compression dictionary that is based on these rows by running an offline reorganization. You can use the environment variable `DB6LOAD_COMPRESSION_THRESHOLD` to define the amount of rows that are initially loaded and that are used to create the dictionary. By default, the variable is set to 10.000.

Afterwards, R3load imports the remaining rows into the table and DB2 compresses the rows based on the previously created dictionary. As a result the major part of the table is compressed.

The compression rates that can be achieved depend on the quality of the compression dictionary. The test results in the following chapters provide hints on how to choose the sample size to create the compression dictionary and how this approach of creating a compression dictionary and compressing data compares with the REORG/compression of the entire table data.

To compress tables during an import with R3load, you can use one of the following commands:

- `R3load –i …… -loadprocedure fast COMPRESS`

  With this option R3load uses ARRAY INSERT.

- `R3load –i …… -loadprocedure fast LOADCOMPRESS`

  With this option R3load uses the DB2 LOAD API.

> **Note:** Make sure that the load procedure arguments are written in capital letters.

Test setup

The following tests were performed with a customer system. We assume that the content of the investigated tables is quite typical for a productive client environment. Other migrations of customer systems have shown similar compression results.

The tables shown in Figure 7-35, which are used in an SAP ERP system, were selected for the investigation.

Table name	Table Size (uncomp)	Table rows
CE1SIAL	6.816.878	74.602.471
VBRP	3.571.785	39.282.766
VBFA	3.140.659	217.443.538
GLPCA	3.084.205	73.974.228
GLFUNCA	3.078.345	153.417.726
LIPS	2.971.492	33.512.810
KONP	2.470.421	135.971.957
MSEG	1.870.987	31.802.883
VBAP	1.780.197	19.579.511
NAST	1.493.277	59.317.229
LTAP	1.455.066	30.067.932
COEP	1.190.393	41.591.449
BKPF	973.026	30.382.943
VBPA	884.489	108.457.703
VBEP	764.161	30.562.334
MBEW	618.187	17.924.492
STXH	591.900	43.308.153
SWWLOGHIST	442.829	26.710.924
VBUP	227.895	53.092.357

Figure 7-35 Tables used for DB2 compression tests with R3load 6.40

In a first step, the tables were imported without compression. Figure 7-35 shows the table size. The unit of measure was 16 KB pages. The overall size of the tables was 571.1 GB.

The Table rows column contains the number of rows in each table. Since the tables have different structures, the ones with the largest number of rows are not necessarily the biggest ones.

In a second step, a full table reorganization was performed to compress the tables. The compression ratios were used as a baseline for the tests with DB2 compression and R3load 6.40.

For the actual tests with DB2 compression based on R3load 6.40, the import was performed with two sequential MigMon runs:

1. In the first run data was loaded with the DB2 LOAD API.
2. In the second run the indexes were created.

DB2 compression during import with different sample sizes

The *sample size* is defined as the *percentage of rows* that were imported before the data was reorganized to build the compression dictionary. Additionally, the default value of 10,000 rows was also taken into account.

The compression rates with the following sample sizes were measured:

```
10.000 rows – 1% - 5% - 25% - 50% - 75%
```

For each of these sample sizes, the table was imported. A script controlled the import. The Appendix A, “Background information” on page 251, provides an example of the script that was used.

Figure 7-36 provides a graphical overview of the results achieved in this investigation.

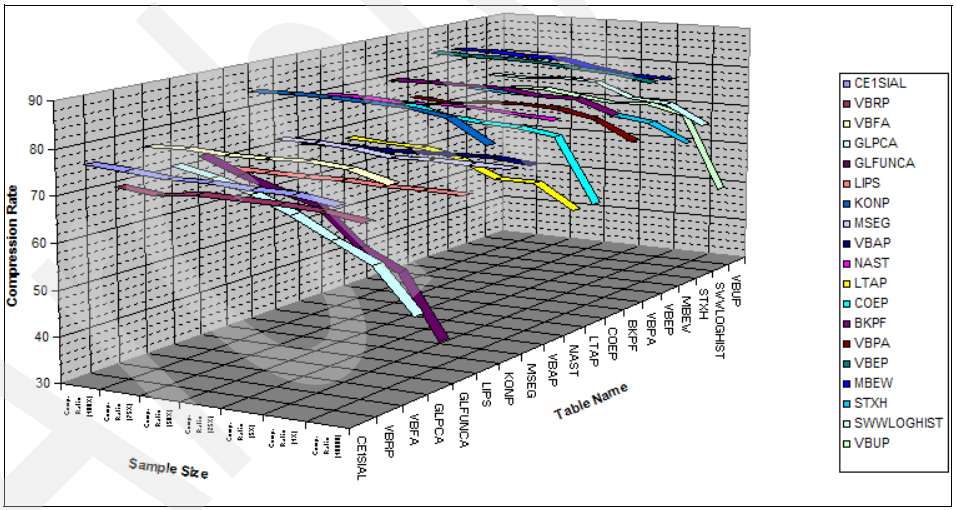


Figure 7-36 Compression rates with different sample sizes

The axis depicted as Sample Size starts on the right (where it is cutting the Table Name axis) with the default sample size of 10,000 rows. Going to the left side of this axis (where it is cutting the vertical Compression Rate axis), the sample sizes increase from 1%, 5%, 25%, 50%, 75%, up to 100%.

The graph can be interpreted as follows:

- Almost straight horizontal stripes

  Even with small data samples a good compression dictionary was built and hence good compression ratios were achieved (for example, CE1SIAL, VBRP, LIPS, and MSEG).

- Low compression rate at the beginning but fast convergence to high compression rates

  The default sample size for the creation of the compression dictionary does not result in good compression rates. But a 1% sample is enough for tables that belong to this group (for example, COEP and VBUP).

- Low at the beginning with slow convergence to high compression rate

  For this type of table a large sample is required to get a satisfactory compression rate (for example, GLPCA and GLFUNCA).

For most of the tables, a 1% sample would be enough to achieve a compression rate that is very close to the optimal compression rate.

As a best practice, we recommend that you choose a sample of 1%. If a table shows the characteristic of the third group that is mentioned above, the sample size should be increased or you might even want to consider a full compression.

Comparison of import times: Compressed versus uncompressed

We also investigated the impact of DB2 compression on the import runtime. In particular, we measured the runtime of data import and index creation both with and without DB2 compression.

Figure 7-37 shows the comparison of import and index build without compression and with compression. The complete process (importing and index build) is normally faster when the data is compressed during the import.

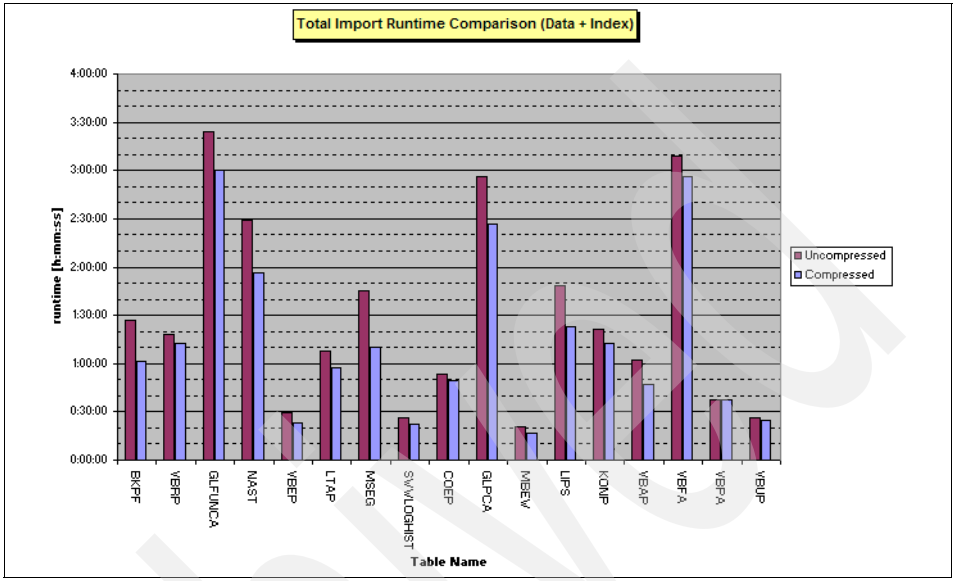


Figure 7-37 Runtime of import and index creation with and without DB2 compression

Figure 7-38 provides details about the runtimes of import and index build.

	COMPRESSED			UNCOMPRESSED						
	DATA	INDEX	D+I	DATA	INDEX	D+I	Diff DATA Runtime	Diff INDEX Runtime	Diff total Runtime	%
TableName	runtime	runtime		runtime	runtime					
BKPF	0:18:40	0:42:41	**1:01:21**	0:17:31	1:09:41	**1:27:12**	0:01:09	0:27:00	0:25:51	29.6
VBRP	1:02:16	0:10:21	**1:12:37**	0:58:59	0:19:19	**1:18:18**	0:03:17	0:08:58	0:05:41	7.3
GLFUNCA	0:49:18	2:10:31	**2:59:49**	0:44:34	2:39:26	**3:24:00**	0:04:44	0:28:55	0:24:11	11.9
NAST	0:35:30	1:20:53	**1:56:23**	0:34:20	1:54:54	**2:29:14**	0:01:10	0:34:01	0:32:51	22.0
VBEP	0:11:43	0:11:30	**0:23:13**	0:11:28	0:18:07	**0:29:35**	0:00:15	0:06:37	0:06:22	21.5
LTAP	0:30:43	0:26:18	**0:57:01**	0:28:51	0:39:14	**1:08:05**	0:01:52	0:12:56	0:11:04	16.3
MSEG	0:37:23	0:32:42	**1:10:05**	0:35:38	1:09:15	**1:44:53**	0:01:45	0:36:33	0:34:48	33.2
SWWLOGHIST	0:07:19	0:14:58	**0:22:17**	0:07:14	0:19:26	**0:26:40**	0:00:05	0:04:28	0:04:23	16.4
COEP	0:21:24	0:28:19	**0:49:43**	0:20:21	0:32:57	**0:53:18**	0:01:03	0:04:38	0:03:35	6.7
GLPCA	0:52:11	1:34:37	**2:26:48**	0:47:25	2:09:07	**2:56:32**	0:04:46	0:34:30	0:29:44	16.8
MBEW	0:12:32	0:03:55	**0:16:27**	0:12:06	0:08:39	**0:20:45**	0:00:26	0:04:44	0:04:18	20.7
LIPS	0:57:23	0:25:51	**1:23:14**	0:54:29	0:53:36	**1:48:05**	0:02:54	0:27:45	0:24:51	23.0
KONP	0:51:35	0:21:07	**1:12:42**	0:50:09	0:31:03	**1:21:12**	0:01:26	0:09:56	0:08:30	10.5
VBAP	0:34:25	0:12:41	**0:47:06**	0:32:38	0:29:23	**1:02:01**	0:01:47	0:16:42	0:14:55	24.1
VBFA	0:55:26	2:01:10	**2:56:36**	0:52:57	2:16:20	**3:09:17**	0:02:29	0:15:10	0:12:41	6.7
VBPA	0:17:23	0:19:58	**0:37:21**	0:16:44	0:21:04	**0:37:48**	0:00:39	0:01:06	0:00:27	1.2
VBUP	0:10:54	0:13:56	**0:24:50**	0:10:45	0:15:20	**0:26:05**	0:00:09	0:01:24	0:01:15	4.8
			20:57:33			**25:03:00**	**0:29:56**	**4:35:23**	**4:05:27**	**16.3**

Figure 7-38 Runtime of import and index build with and without compression

The import tests with DB2 compression were performed with a sample size of 1% of the number of rows in the table to build the compression dictionary.

From the test results we can conclude the following:

- The import time is nearly the same for both compressed and uncompressed tables. The difference is less than 5%, with a little runtime advantage for the uncompressed import.
- The build phase of the indexes is significantly faster if you are using the compressed import. The advantage is between 1% and 33%. The average of the examined tables is about 16%.

Although we generally recommend that you build indexes before data is loaded, in some cases it might be beneficial to build indexes afterwards (for example, Bernoulli compression and subsequent parallel insert of split tables).

Based on the tests above, if you create indexes after importing data the index creation will be significantly faster if the data is compressed.

Compressed import setup using the migration monitor

This section explains how you can set up the migration monitor (MigMon) configuration to compress data during import with R3load.

First of all you must make sure that the DB2 compression attribute is activated for the corresponding tables. To activate it for all tables, you can modify the `CREATE` statement in the DDL template file. If we only want to compress a subset of all tables, we need two different template files. You can use, for example, DDLDB6.TPL for the tables without compression and DDLDB6C.TPL for the tables with compression.

Starting with Version 6.40, the migration monitor (MigMon) supports a feature called *DDL mapping* to use more than one TPL file. You can use this feature to build a mapping between a package and a DDL template file (for example, DDLDB6.TPL, DDLDB6C.TPL). This also defines the DDL syntax to be used. You can find an example of such a file in Appendix A, “Background information” on page 251.

You configure the MigMon using the following properties files:

- export_monitor_cmd.properties (for the export)
- import_monitor_cmd.properties (for the import)

These files define all necessary parameters for the migration monitor to work properly.

Appendix A, “Background information” on page 251, provides an example of a properties file to compress data during import into DB2. As discussed above, a sample size of 1% of the total number of rows in the table is a good choice. Because the number of rows in each table differs, you may not want to use the same value for variable `DB6LOAD_COMPRESSION_THRESHOLD` for all tables. Once again, this variable defines the sample size as the *N* first number of rows to be taken as the sample to build the dictionary, not as the percentage of records in the table.

To be able to use different sample sizes for different sets of tables, we recommend using multiple instances of the migration monitor (in different directories) and using different values for `DB6LOAD_COMPRESSION_THRESHOLD` for each instance. We recommend that you define this variable in the import_monitor.sh script. Thus, you avoid confusion with the different settings.

7.18.5 DB2 row compression based on R3load 7.00

R3load Version 7.00 introduces new options related to DB2 compression that help to improve the quality of the compression dictionary and hence the compression rates. They also automate some of the necessary steps to compress data during import.

Introduction of available options

The following sections describe options and tests based on R3load Version 7.00 and DB2 V9.5[1]. Table 7-3 lists the available compression options for SAP kernel 7.00[2] or later. For information about the functionality provided by R3load 6.40, see 7.18.4, “DB2 row compression based on R3load 6.40” on page 157.

Table 7-3 R3load 7.00 options for DB2 compression

R3load option	Impact
`COMPRESS`	Ignored with DB2 V9.5 and later. With DB2 V9.1, `COMPRESS` is compatible with the option of R3load 6.40.
`COMPRESS_ALL`	R3load creates tables with the `COMPRESS YES` option.
`FULL_COMPRESS`	If tables were created with the `COMPRESS YES` option, R3load first loads all data into tables. A subsequently triggered reorganization compresses the tables.

[1] All described functionality (except the Automatic Dictionary Creation (ADC)) is also valid for DB2 V9.1.

[2] For the latest information check SAP Note 1058437.

SAMPLED	R3load imports only a sample of data into tables, for example, every hundredth row from the export file. A subsequently triggered reorganization compresses the partially loaded table. R3load stops on error after this step and must be restarted without the COMPRESS option to load the complete set of data.

You can combine the options above within one single R3load command:

```
R3load […] –loadprocedure fast  LOAD:FULL_COMPRESS_ALL
```

This option creates tables with the COMPRESS YES option, loads all data, and performs a subsequent reorganization.

Compression tests with R3load 7.00

To evaluate R3load options with respect to both compression ratio and import runtime, several tests were performed. These tests are described in the following sections, together with a description of each R3load option. Each test run consisted of eight R3load processes that were running in parallel, whereas each process loaded one single table.

Table 7-4 provides an overview of which database tables were used during the tests and lists their main characteristics (size and number of rows).

Table 7-4 Compression tests setup with R3load 7.00 options

Table name	Size (KB) uncompressed	Number of rows
ACCTIT	8,854,048.00	11.505.376
BKPF	8,604,032.00	17.520.379
BSIS	9,380,928.00	20.769.172
CDCLS	9,499,456.00	37.411.485
COEP	11,554,624.00	23.142.069
GLFUNCA	10,695,616.00	34.948.249
GLPCA	8,567,488.00	14.872.407
SWWLOGHIST	9,668,672.00	40.666.845

Uncompressed load with optional subsequent compression

If a table should either remain uncompressed or if you want to compress it manually, you can run R3load without any compression option. R3load then creates tables without the COMPRESS YES option, loads all data, and stops successfully.

To compress the fully loaded table, issue the appropriate DB2 commands, for example, activate the `COMPRESS` option and schedule a table reorganization afterwards (you can also schedule the table reorganization using the DBA Cockpit).

With the described approach, tables grow to their maximum size before the data is compressed. If you need to reduce the allocated space, you must subsequently resize the table spaces. We recommend that you at least reduce the high water mark (HWM) of table spaces that have been compressed after loading data. This influences the backup behavior on DB2 table spaces. A DB2 backup operation processes all pages up to the high water mark as part of the backup image, regardless of whether they contain data.

Figure 7-39 illustrates the process of the uncompressed load with optional subsequent manual compression.

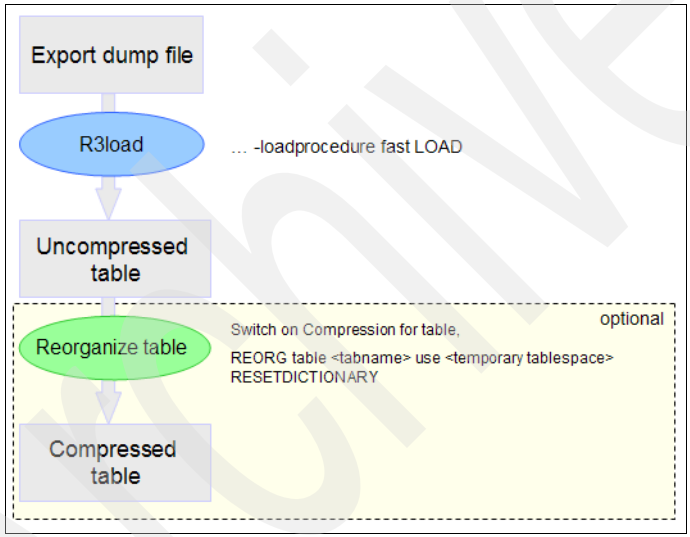


Figure 7-39 Uncompressed LOAD and optional subsequent manual compression

Advantage

The major advantage of this scenario is the *optimal compression ratio* (if you decide to compress tables subsequently).

Disadvantage

The disadvantages of the uncompressed load are:

- You must manually compress tables.
- Tables grow to their maximum size before compression. We recommend that you reduce the high water mark after compression.

Test description and results

In this test R3load was used to create the tables (without the `COMPRESS YES` option) to load the data. The tables were not compressed subsequently. This test forms the baseline to calculate the compression ratio for the other tests. The indexes were created before loading the data. The following R3load option was used:

```
R3load -i <package>.cmd -dbcodepage 1100 -l <package>.log -loadprocedure fast
LOAD
```

You can find the complete MigMon properties file in Appendix A, “Background information” on page 251.

Table 7-5 table shows the table size and runtime of data load.

Table 7-5 Runtimes of LOAD without compression

Table name	Size (KB)	Runtime load (min)
ACCTIT	8,854,048.00	131
BKPF	8,604,032.00	146
BSIS	9,380,928.00	115
CDCLS	9,499,456.00	125
COEP	11,554,624.00	149
GLFUNCA	10,695,616.00	159
GLPCA	8,567,488.00	164
SWWLOGHIST	9,668,672.00	168

The R3load log files contained the following information:

```
[…]
((DB) INFO: ACCTIT created #20080307155957
(DB) INFO: ACCTIT~0 created #20080307155958
(DB) INFO: ACCTIT~1 created #20080307155958
(DB6) Using LOAD API for table ACCTIT
(IMP) INFO: import of ACCTIT completed (11505376 rows) #20080307181028
(DB) INFO: disconnected from DB
```

```
/usr/sap/ZWN/SYS/exe/run/R3load: job completed
/usr/sap/ZWN/SYS/exe/run/R3load: END OF LOG: 20080307181028
```

Full load and subsequent compression triggered by R3load

To achieve an optimal compression ratio together with automation of the compression process, you can use the following R3load options:

```
-loadprocedure fast LOAD:FULL_COMPRESS_ALL
```

In this case, R3load creates the tables with the `COMPRESS YES` option, loads all data, and then performs a subsequent reorganization of all tables to rebuild the dictionary and compress all data.

With DB2 9.5 the automatic dictionary creation will create a compression dictionary at some point in time during the import. So tables do not grow to their maximum sizes, but might grow to a size that is above the optimal compressed size. If needed, you might consider resizing table spaces to free up space on disks. We recommend that you reduce at least the high water mark (HWM) of table spaces that have been compressed after loading data. This influences the backup behavior on DB2 table spaces: A DB2 backup operation processes all pages up to the high water mark as part of the backup image, regardless of whether these pages contain data.

Figure 7-40 illustrates the process of the uncompressed load and subsequent compression triggered by R3load.

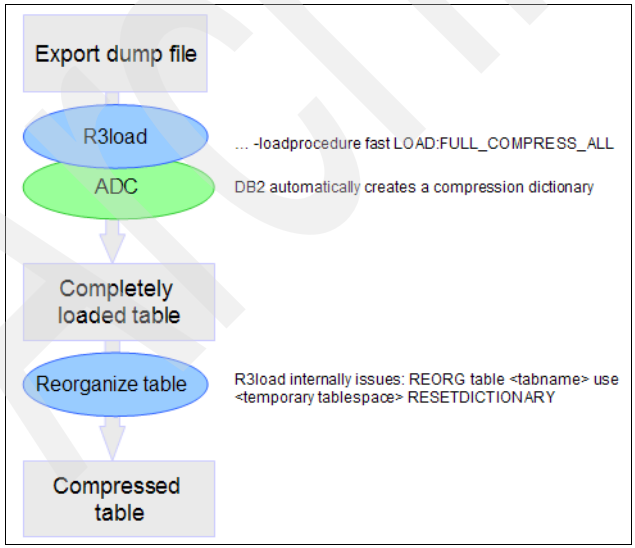


Figure 7-40 Uncompressed load and subsequent compression triggered by R3load

Advantages

Two advantages of this uncompressed load are:

- Automatic process for compressing tables
- Optimal compression ratio

Disadvantages

The disadvantages of this load procedure are:

- Tables grow beyond their optimal compressed sizes before the final compression step. We recommend that you reduce the high water mark after compression.
- Higher import runtime (full import + full reorganization).

Test description and results

For the test the R3load was executed as follows:

```
R3load -i <package>.cmd -dbcodepage 1100 -l <package>.log -loadprocedure fast
LOAD:FULL_COMPRESS_ALL
```

You can find the complete MigMon properties file in Appendix A, “Background information” on page 251.

Table 7-6 provides information about the table size, the compression ratio, and the data load runtimes.

Table 7-6 Runtimes of LOAD with subsequent compression triggered by R3load

Table name	Size (kb)	Compression ratio	Runtime load (min)
ACCTIT	2,194,560.00	75.2	351
BKPF	1,689,344.00	80.4	285
BSIS	2,127,104.00	77.3	277
CDCLS	7,109,792.00	25.2	285
COEP	2,892,544.00	75.0	375
GLFUNCA	3,089,792.00	71.1	400
GLPCA	2,465,568.00	71.2	378
SWWLOGHIST	2,921,344.00	69.8	397

The R3load log files contained the following information:

```
[…]
(DB) INFO: connected to DB
(DB6) INFO: COMPRESS YES is set during table creation.
(DB6) INFO: If row compression is active, REORG will be triggered after all
data has been inserted.
[…]
(DB6) Using LOAD API for table ACCTIT
(DB6) INFO: Table ACCTIT reorganized to create compression dictionary
(IMP) INFO: import of ACCTIT completed (11505376 rows) #20080317221627
(DB) INFO: disconnected from DB
/usr/sap/ZWN/SYS/exe/run/R3load: job completed
/usr/sap/ZWN/SYS/exe/run/R3load: END OF LOG: 20080317221628
```

Compressed load with automatic dictionary creation

With DB2 9.5 you can use *automatic dictionary creation* (ADC) in combination with R3load to create a compression dictionary and compress the data while the data is loaded. This can be achieved with the following R3load options:

```
-loadprocedure fast LOAD:COMPRESS_ALL
```

In this case, R3load enables the `COMPRESS` flag for all tables before the data is loaded. ADC then creates dictionaries while the data is loaded and all data that is loaded after dictionary creation is automatically compressed during load.

ADC creates a compression dictionary automatically when the following conditions apply:

- COMPRESS flag is enabled for the table.
- No compression dictionary exists for the table.
- The table contains a sufficient amount of data.

ADC is triggered after a table was populated with approximately 2 MB of data. DB2 automatically creates a compression dictionary and compresses subsequently added data.

Figure 7-41 illustrates the compressed load with automatic dictionary creation.

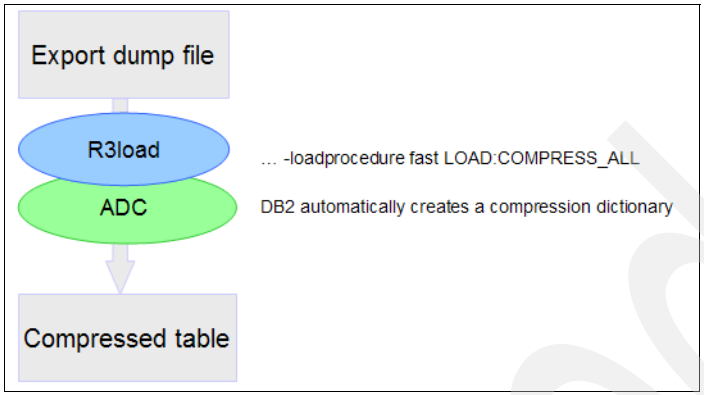


Figure 7-41 Compressed load with automatic dictionary creation

Especially when the underlying R3load exports were performed in sorted order, this approach could lead to suboptimal compression rates. The data that is used for ADC might not be representative for the entire table.

Advantages

The advantages of this load method include:

- Automatic process to compress tables during LOAD.
- Tables do not grow to their uncompressed sizes before data is compressed.
- Almost no runtime overhead.

Disadvantage

The only disadvantage is that compression rates may not be optimal.

Test description and results

In this test R3load was executed as follows:

```
R3load -i <package>.cmd -dbcodepage 1100 -l <package>.log  -loadprocedure fast
LOAD:COMPRESS_ALL
```

You can find the complete MigMon properties file in Appendix A, “Background information” on page 251.

Table 7-7 shows the table size, compression ratio, and runtime of data load.

Table 7-7 Runtimes of LOAD with subsequent compression triggered by R3load

Table name	Size (kb)	Compression ratio	Runtime load (minutes)
ACCTIT	6,473,568.00	26.9	128
BKPF	2,721,408.00	68.4	147
BSIS	4,552,384.00	51.5	124
CDCLS	8,504,128.00	10.5	138
COEP	5,846,144.00	49.4	145
GLFUNCA	6,750,976.00	36.9	155
GLPCA	4,626,528.00	46.0	163
SWWLOGHIST	6,818,080.00	29.5	173

The R3load log files contained the following information:

```
[…]
(DB) INFO: connected to DB
(DB6) INFO: COMPRESS YES is set during table creation.
(DB6) DB2 9.5 automatic dictionary creation does not require an explicit REORG.
(DB6) Compression threshold is ignored.
[…]
(DB6) Using LOAD API for table ACCTIT
(IMP) INFO: import of ACCTIT completed (11505376 rows) #20080317153527
(DB) INFO: disconnected from DB
/usr/sap/ZWN/SYS/exe/run/R3load: job completed
/usr/sap/ZWN/SYS/exe/run/R3load: END OF LOG: 20080317153527
```

Compressed load based on R3load sampling

To optimize the quality of compression dictionaries, R3load Version 7.00 and later offers a new option, `SAMPLED`. To understand how this option improves compression, we should first review the `COMPRESS` option available with the older R3load Version 6.40. With this option the first *n* rows are loaded. Then, with DB2 versions earlier than 9.5, R3load creates a compression dictionary by performing a reorganization, and all subsequently loaded rows are automatically compressed. When using DB2 Version 9.5 with Automatic Dictionary Creation, the `COMPRESS` option of R3load is ignored since the dictionary will be automatically created by DB2. This approach of building a compression dictionary based on the first *n* records in the table (either by an explicit REORG or by ADC) can lead to suboptimal compression rates. Especially when the underlying export dump

files contain data in sorted order, the selected sample might not be representative of the complete table data.

Loading data with the new `SAMPLED` option offers optimal data compression while avoiding time-consuming table reorganizations on the entire set of data. This new option can be deployed by running the migration monitor two times, as follows:

- Phase 1

  R3load is executed with option `SAMPLED` to build the compression dictionary. R3load imports only each nth row of the export file into the target table. Afterwards, it automatically performs a reorganization with the `RESETDICTIONARY` option.

  The environment variables shown in Table 7-8 can be used to control the R3load behavior.

  Table 7-8 Variables influencing the R3load SAMPLED behavior

Variable	Description
DB6LOAD_SAMPLING_FREQUENCY=n	Only one out of <*n*> rows is imported. The default is 100.
DB6LOAD_MAX_SAMPLE_SIZE=x	R3load inserts a maximum of *x* (MB) data. The default is infinite.

  In contrast to the approach described in the previous sections, the compression dictionary is built based on a fairly representative subset of the data. After the load of the data sample has finished, R3load stops with an error.

Figure 7-42 illustrates the process of the sampled data load.

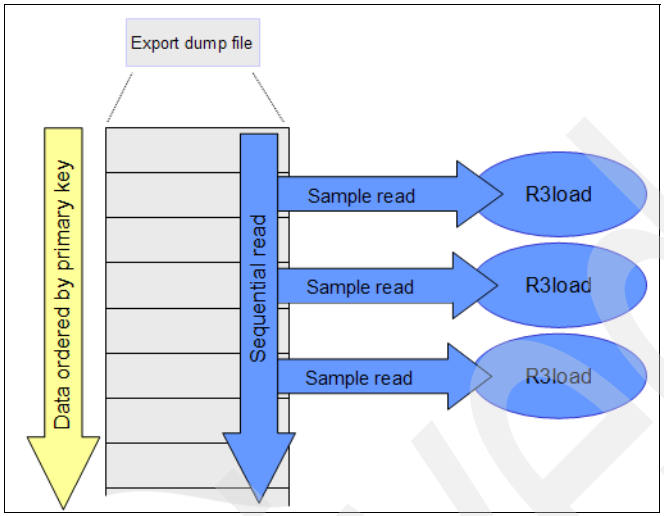


Figure 7-42 Sampled data load

- Phase 2

  R3load must be restarted by omitting the compression options. The second run truncates all existing data while the compression dictionary is maintained. All data is loaded into the tables in a compressed format.

Figure 7-43 illustrates the process to use R3load option `SAMPLED` to load and compress data.

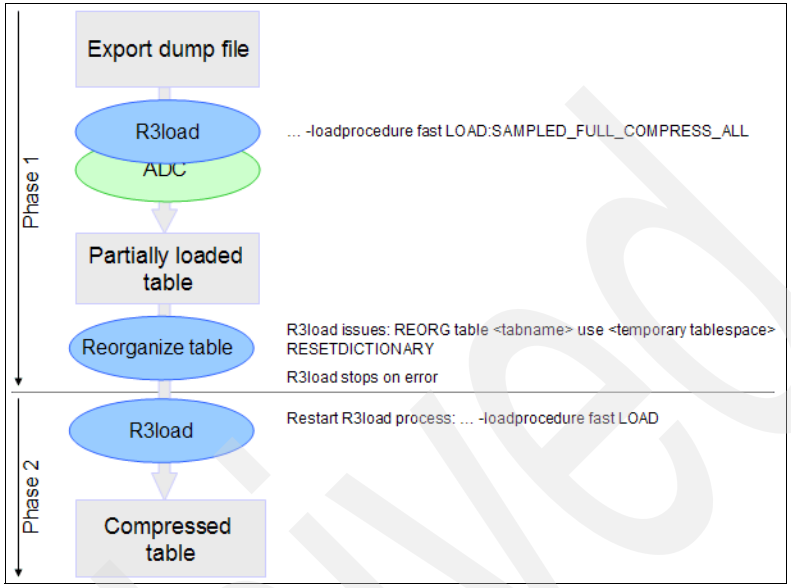


Figure 7-43 Process to use R3load option SAMPLED to load and compress data

Advantages

The advantages of this process include:

- Optimal compression ratio.
- Tables do not grow to their maximum sizes before compression.

Disadvantages

The disadvantages of this process are:

- Two-step approach (that is, more effort).
- R3load must be restarted.

Test description and results

In phase 1 R3load was executed as follows:

```
R3load -i <package>.cmd -dbcodepage 1100 -l <package>.log
   -loadprocedure fast LOAD:SAMPLED_FULL_COMPRESS_ALL
```

In phase 2 R3load was executed as follows:

```
R3load -i <package>.cmd -dbcodepage 1100 -l <package>.log -loadprocedure fast
LOAD
```

You can find the complete MigMon properties file in Appendix A, “Background information” on page 251.

Table 7-9 shows the table size, compression ratio, and data load runtime[3].

Table 7-9 Total runtime of load with option SAMPLED (sum of both load phases)

Table name	Size (KB)	Compression ratio	Runtime load (min)
ACCTIT	2,089,344.00	76.4	116
BKPF	1,677,408.00	80.5	124
BSIS	2,100,256.00	77.6	104
CDCLS	7,075,296.00	25.5	131
COEP	2,876,608.00	75.1	140
GLFUNCA	3,092,128.00	71.1	148
GLPCA	2,572,224.00	70.0	151
SWWLOGHIST	2,937,248.00	69.6	159

[3] Load runtime is calculated as sum of phase 1 and phase 2

The information contained in the R3load log files is shown in Figure 7-44.

```
[...]
(DB) INFO: connected to DB
(DB6) INFO: COMPRESS YES is set during table creation.
(DB6) INFO: If row compression is active, REORG will be triggered after all data has
been inserted.
(DB6) INFO: Inserting SAMPLED data with frequency 100 and maximum sample size 0
MB.
[...]
(DB6) Using LOAD API for table ACCTIT
(DB6) INFO: Table ACCTIT reorganized to create compression dictionary
(DB6) WARNING: Sampling was active and data for table ACCTIT was not completely
loaded.
115054 out of 11505376 rows with maximum row size 1944 have been sampled.
(IMP) ERROR: EndFastload: rc = 2
(DB) INFO: disconnected from DB

[...]

(DB) INFO: ACCTIT deleted/truncated #20080317093337
(DB6) Using LOAD API for table ACCTIT
(IMP) INFO: import of ACCTIT completed (11505376 rows) #20080317112209
(DB) INFO: disconnected from DB
/usr/sap/ZWN/SYS/exe/run/R3load: job completed
/usr/sap/ZWN/SYS/exe/run/R3load: END OF LOG: 20080317112209
```

Phase 1 ends – R3load stopped on error

Phase 2 has to be started manually – Table will be truncated

Figure 7-44 R3load log files content

Summary of tests with R3load 7.00

This section summarizes the test results of R3load 7.00 functionality.

Load runtime

Figure 7-45 shows the runtime of each type of R3load option that was tested. The runtime is shown as the sum of all tables involved. Figure 7-45 shows comparable runtimes for the following approaches:

- Uncompressed load
- Compressed load with automatic dictionary creation (ADC)
- Compressed load based on R3load sampling

An uncompressed load that is followed by a subsequent compression (full table reorganization) shows a significant increase in runtime.

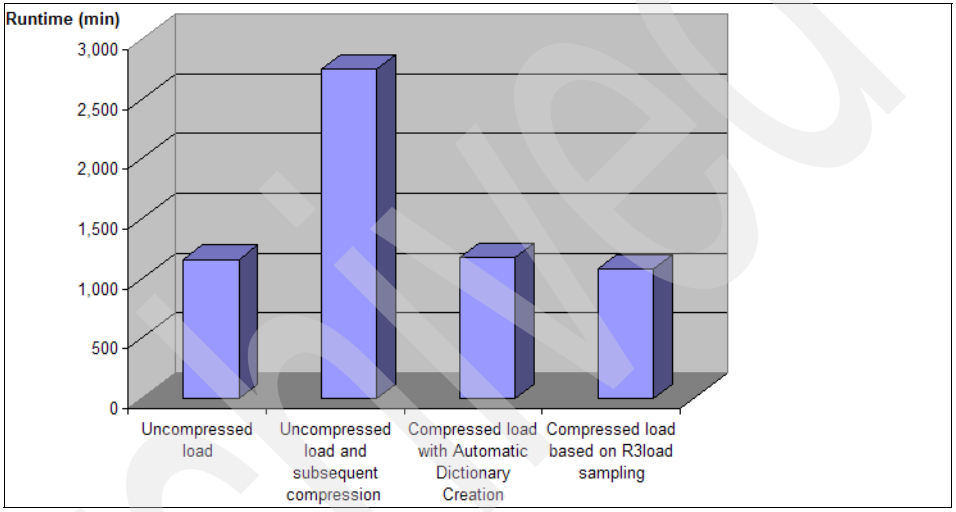


Figure 7-45 Runtime comparison of different approaches to load and compress data

Compression ratio

Figure 7-46 summarizes the compression ratios that were achieved with the different R3load options. The compression ratio is shown as an average of all tables involved.

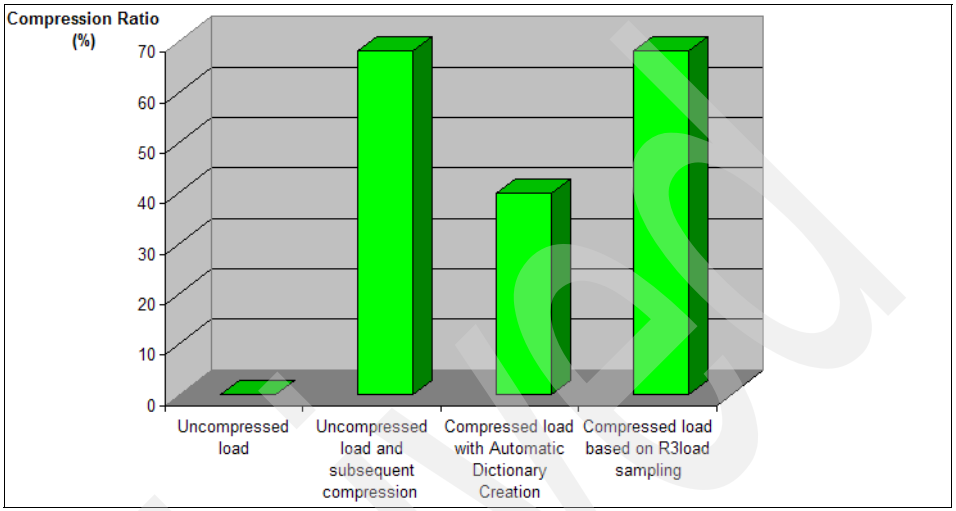


Figure 7-46 Summary of compression ratios

Figure 7-46 shows optimal compression ratios for the following options:

- Compressed load based on R3load sampling
- Uncompressed load followed by a subsequent compression

Using R3load in combination with DB2 automatic dictionary creation results in a suboptimal compression ratio.

Resource consumption

When comparing compressed load based on R3load sampling and uncompressed load with respect to utilization of resources, it becomes clear that sampling adds additional CPU workload. At the same time, sampling reduces disk I/O. See Figure 7-47 and Figure 7-48 on page 180.

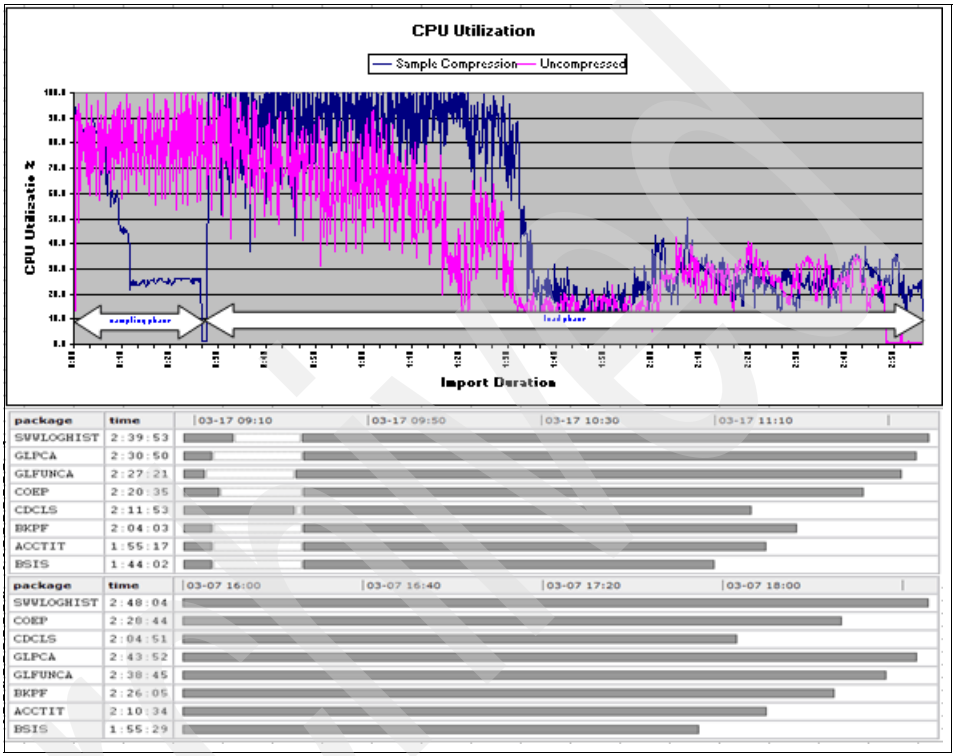


Figure 7-47 CPU usage: Sampled load versus uncompressed load

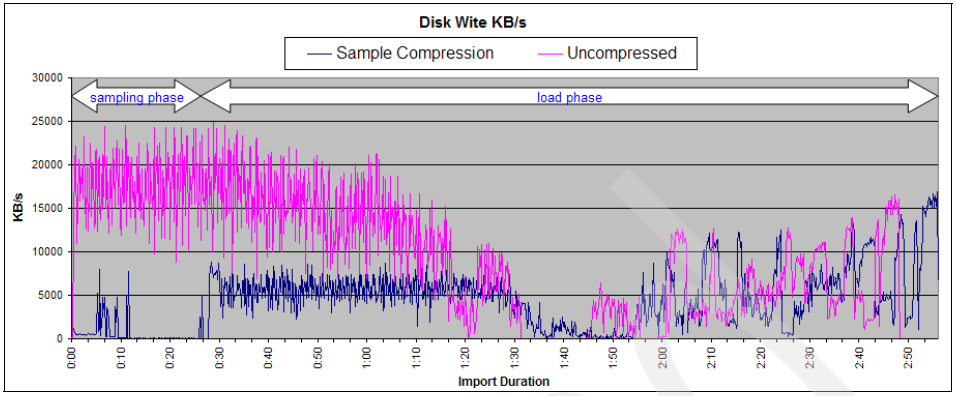


Figure 7-48 Disk usage: Sampled load versus uncompressed load

Depending on the bottlenecks in your system, you should consider this in your planning. For example, if you are short on CPU resources, reduce the number of parallel R3load processes.

Conclusion and recommendation

We recommend that you use the new R3load sampling method for compressing tables while data is being imported. Table 7-10 summarizes our findings.

Table 7-10 Summary: Comparison of R3load options

R3load technique	Optimal compression ratio	Optimal load runtime	Automatic process	Tables grow to max size before compression
Uncompressed load	No	Yes		Yes
Compressed load with DB2 Automatic Dictionary Creation	No	Yes	Yes	No, but suboptimal compression ratio
Uncompressed load, subsequent compression by R3load	Yes	No	Yes	Partially[a]

Compressed load based on R3load sampling	Yes	Yes	Partially	No

a. With DB2 9.5, automatic dictionary creation compresses data to a certain extent during the load process. A second compression step triggered by R3load leads to a higher compression ratio, while leaving table spaces (high water mark) at a higher size.

The results show that R3load with option `SAMPLED` provides an optimal combination of compression ratio and load runtime. All other options either lead to suboptimal compression ratios, longer runtimes, or an increased table space size. We therefore recommend that you use the `SAMPLED` option to compress data during migrations.

Keep in mind, however, that this approach is not fully automatic. It consists of two phases, and R3load must be restarted manually after successfully finishing the first phase. Loading a data sample that is truncated before loading the complete data can also increase the import runtime.

You can further optimize the process as follows. Phase 1 can be performed before a productive migration with data obtained from a test migration. Because import with data compression (phase 2) can be faster than without compression, this approach might further reduce the overall runtime of the import.

This approach also works when using the R3load socket feature. In this case we recommend exporting to local files first, performing import phase 1 with this export, and then running phase 2 with the socket feature. If you would also run phase 1 with the socket option, a complete export must be performed to build the sample. And as no export files are persistent on disk, a subsequent import for phase 2 could not be done without performing a second export. When using a combination of a local export and socket feature, make sure that you rename or move all cmd files on the import side before starting phase two. They will be regenerated with the correct content.

Note: If you perform phase 1 upfront of a productive migration, you must ensure that no structural change to the related tables takes place.

Finally, we also recommend that you reduce the high water mark (HWM) of table spaces that have been compressed *after loading data.*

7.18.6 Bernoulli sampling method

As discussed in the previous chapter, the best compression dictionary is either built based on the entire data in the table or a representative sample that is loaded up front.

The `SAMPLED` option for R3load already delivers an excellent compression rate that is very close to the optimum. However, there are some trade-offs when using this method. First of all, the sample size is static and applies to all tables that are exported. It can only be changed globally for all tables. Secondly, the R3load sampling method requires scanning the entire export file to retrieve the sample, which could take some time and requires the export of the entire data.

DB2 offers an alternative solution that has been integrated as tool /ISIS/ZCOMP into SAP systems. See *SAP Note 980067*. DB2 itself offers efficient sampling techniques, particularly Bernoulli sampling.

With row-level Bernoulli sampling, a sample of *P* percent of the table rows is returned. Each row in the table is considered independently and included in the sample with a probability of *P*/100. Since all rows in the table have an equal probability of being included in the sample, Bernoulli sampling always gets a representative random sample regardless of data clustering.

For details on the Bernoulli function, refer to the DB2 documentation at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.regvars.doc/doc/c0007340.html

Note: The basic concept behind the Bernoulli sampling procedure and tools is to export Bernoulli sampled data on the DB2 target system after a test migration. These data samples can then be used to build nearly optimal DB2 compression dictionaries during the productive migration on the DB2 target system. The advantage of this method is the generation of optimal samples for each table based on statistic data in the DB2 catalog.

The tool determines the percentage necessary to yield a certain sample size. If statistics are available for the table, it determines the sample size using the `AVGROWSIZE` and table cardinality from DB2 system tables. That is, it uses the following two equations:

$$NumRows = \frac{(SampleSize(MB) \times 1024) \times 1024}{AVGROWSIZE}$$

$$SamplePercentage = \left(\frac{NumRows}{TableCardinality} \right) \times 100$$

If table statistics do not exist, the tool calculates the sample percentage by using the physical size of the object, as shown in the following equation:

$$SamplePercentage = \frac{(SampleSize(MB) \times 1024) \times 100}{DATA\_OBJECT\_L\_SIZE}$$

Using these equations, you can exactly define the sample size by specifying the appropriate percentage in the select statement. Example 7-7 shows how to select a Bernoulli sample.

Example 7-7 SQL statement to select a Bernoulli sample

```
SELECT * FROM SAPR3.ZVBAP_001 TABLESAMPLE BERNOULLI ( 8.1833060556465E-01);
```

By using this function as a WHERE clause for the table splitter (5.1, “Table splitting” on page 62), you can export only a subset of data from a table.

To enable compression from a Bernoulli sample, the overall procedure is:

1. Run a test migration with or without compression and build the DB2 target system.
2. After a test migration was performed to DB2 you can export a subset of the data on the DB2 target system with Bernoulli sampling, as described above.
3. In subsequent test migrations or the final migration, perform the following steps:
   a. Import the sample data into the new table in the target system.
   b. Enable compression on the new table and create the compression dictionary.
   c. Delete the sample data from the target table, but leave the compression dictionary.
   d. Import all the data into the table.

The results of the above steps can look like those shown in Figure 7-49.

Table	Orignal Size (KB)	Number of Rows	Number of Rows Sample	CR Bernoulli	CR Full REORG
BKPF	4050464	10331351	15553	81.31%	81.55%
COSP	20058464	18799934	5684	87.79%	87.72%
VBFA	25412416	112736531	21920	77.12%	76.67%

Figure 7-49 Compression ratios with Bernoulli sampling

The last column of the table shows the compression ratios achieved when the compression dictionary is created after all the data has been loaded into the table.

The test results show that compression dictionaries that are based on Bernoulli sampled data yield excellent compression rates. The compression rates are within 0.3% of a compression rate that was achieved from a full table reorganization.

The ABAP program /ISIS/ZCOMP supports the definition of SQL statements to retrieve Bernoulli sampled data from selected tables. *SAP Note 980067* provides further information about the corresponding tools.

The ABAP program generates all necessary scripts to perform the export of the data sample. It also generates the scripts necessary to perform the import, create the DB2 compression dictionary, and compress the data. A prerequisite for using this tool is R3load with SAP Kernel 7.00 patch 151 or higher.

Tools to compress data based on Bernoulli sampling

This section describes the available tools to deploy Bernoulli sampling. As already mentioned, the data samples must be retrieved from a DB2 target system that has been built during a test migration. Two alternative methods are supported to retrieve the Bernoulli sample from this DB2 system:

- R3load
- DB2 export/import

Selecting the tables to be compressed

Run the tool /ISIS/ZCOMP via SAP transaction SE38 and select the tables to be compressed.

As shown in Figure 7-50, select the tables that will be compressed from either the Compressed tab or the Candidates tab and click **Add to Export List**.

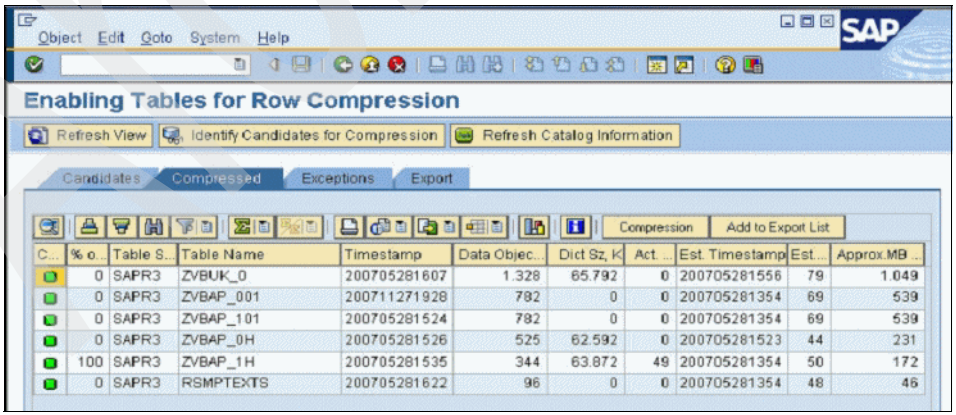


Figure 7-50 Report /ISIS/ZCOMP: Selecting tables for export

Clicking the **Export** tab shows the complete list of selected tables, as shown Figure 7-51.

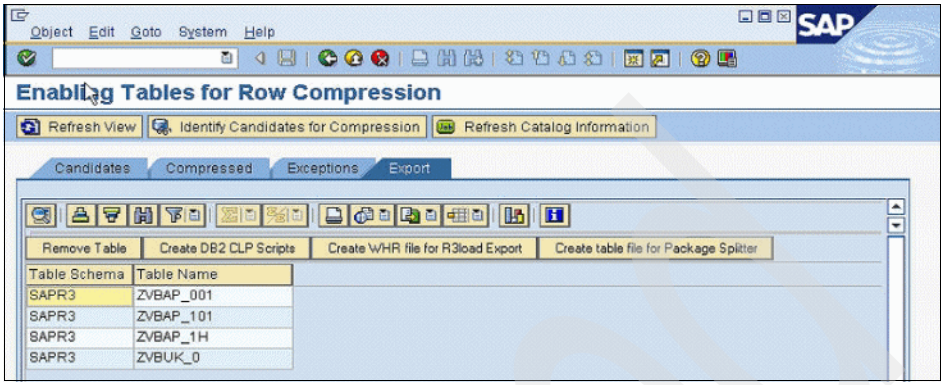


Figure 7-51 Report /ISIS/ZCOMP: Available export options

The following functions are available to prepare the Bernoulli sampling process:

- Create DB2 CLP scripts to export/import the Bernoulli sample with native DB2 tools.
- Create WHR files to export/import the Bernoulli sample with R3load.
- Create a file with the names of tables to be sampled as input for the Package Splitter. This is to prepare the R3load-based approach.

Preparing the Bernoulli sampling

The graphical user interface and available functions to prepare the Bernoulli sampling process are described in more detail in this section.

R3load-based export/import of the Bernoulli samples

The Create table file for Package Splitter button creates a file tabfile.txt that contains the list of tables to be exported in separate packages. Typically, the largest tables are exported in dedicated packages anyway. However, if the tables chosen for compression are not already assigned to dedicated packages, you can use this function to prepare the package splitting for those tables. This creates a file called `tabfile.txt` that contains a list of all the tables that were selected for export.

Example 7-8 shows an example table file for Package Splitter.

Example 7-8 Table file for Package Splitter

```
ZVBAP_001
ZVBAP_101
ZVBAP_1H
ZVBUK_0
```

You can use the Package Splitter to split the selected tables into separate packages by using option `tableFile`. This creates new .STR files for the selected tables. You must copy (not move) these .STR files to a dedicated directory structure, as explained later in this document.

The Create WHR file for R3load Export button generates the WHR-files needed by R3load to export the Bernoulli samples of the selected tables.

The tool generates WHR files in the following format:

```
tab: <table name>
TABLESAMPLE BERNOULLI (<P>)
```

<P> is the percentage of data that is sampled. The syntax is also shown in Example 7-9.

Example 7-9 Content of a WHR file used to export a Bernoulli sample

```
tab: ZVBAP_001
TABLESAMPLE BERNOULLI(8.1833060556464798E-01)
```

For each table, the WHR-file is created, and in addition a DB2 CLP file `truncate_table_script.clp` that is used to delete the data samples from the DB2 target tables after the compression dictionaries is created. If tables with outdated or no statistics exist, they will be reported in file `TABLES_WITHOUT_CURRENT_STATISTICS.txt`.

In the Set Script Criteria dialog box you can specify a different sample size for all the selected tables. In general, the default sample size of 6 MB yields an excellent compression rate that is close to the optimum that can be achieved with a full table reorganization. However, the sample sizes should not be larger than 10 MB. Larger samples do not yield better compression rates but will increase processing time. Therefore, choose a sample size between 6–10 MB.

You can also change the schema name. The target system can have a different schema name. Therefore, the generated import scripts should contain the correct target schema.

To generate the scripts select the **Generate Scripts** push button (Figure 7-52). They are stored on the presentation server, not on the database server.

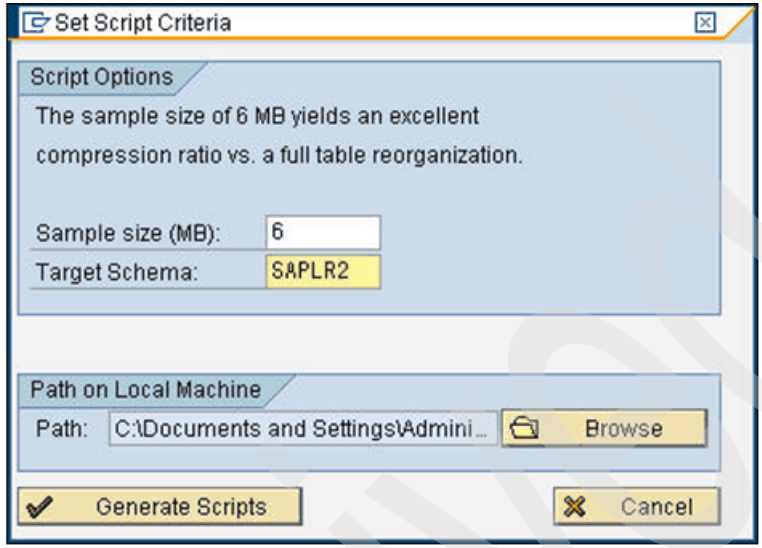


Figure 7-52 Report /ISIS/ZCOMP: Set Script Criteria dialog

After the WHR files have been generated, copy them together with the corresponding STR files to a new directory on the source system. Ensure that the `exportDir` property of the migration monitor points to this directory. Now you can run the migration monitor to export the data.

Creating files for DB2-based sampling

You can use the Create DB2 CLP Scripts button to generate DB2 CLP scripts to export a Bernoulli sample with native DB2 commands instead of using R3load. This option is available for DB2 experts to use the available native DB2 tools. An alternative use case for this option is to create a set of scripts to move tables to a new table space with optimal compression rate (as an alternative to DB6CONV or Online Table Move[4]).

After choosing the Create DB2 CLP Scripts push button a dialog box appears. Here you specify the sample size, the target schema, and location to save the scripts. The dialog box is identical to the one for the R3load-based sampling, but it generates different scripts. For each table, a script for exporting the sample is created.

[4] See SAP note 362325.

Example 7-10 shows an export script that was generated by the tool.

Example 7-10 Script to export a Bernoulli sample with native DB2 tools

```
-- Execute db2 -tvf <script>.clp to run the script.
-- The script will export to the current directory.
-- change the export directory if necessary.
EXPORT TO SAPR3.ZVBAP_001.ixf OF IXF SELECT * FROM SAPR3.ZVBAP_001 TABLESAMPLE
BERNOULLI ( 8.1833060556465E-01);
```

Another script will be generated to import the data, create the compression dictionary, and finally delete the data form the table. This is shown in Example 7-11.

Example 7-11 DB2 script to import a Bernoulli sample and build a compression dictionary

```
-- Execute db2 -tvf <script>.clp to run the script.
-- The script will import from the current directory.
--change the import directory if necessary.
--The script assumes that the table has already been created.
--Import the Bernoulli sample.
LOAD FROM SAPR3.ZVBAP_001.ixf OF IXF REPLACE INTO SAPLR2.ZVBAP_001;

-- Enable compression on the table.
ALTER TABLE SAPLR2.ZVBAP_001  COMPRESS YES;

-- Create the compression dictionary.
REORG TABLE SAPLR2.ZVBAP_001  RESETDICTIONARY;

-- Delete the data from the table but keep the dictionary.
DELETE FROM SAPLR2.ZVBAP_001;
```

As you can see, the script does not create the table to be imported. So the table must exist or must be created by R3load. You can create it by using the `OMIT` options of the migration monitor or R3load. The details are described later in this section.

After the files have been generated, copy the files to a new directory on the source system. Also adapt permissions if necessary, as these scripts should be executed by the instance owner `db2<sid>`.

Running the Bernoulli compression process

The previous chapters have described the tools and different approaches (R3load or native DB2 tools) to use Bernoulli sampling. This section describes the necessary steps of the entire procedure to use Bernoulli sampling to build the DB2 compression dictionaries.

Directory structure and migration monitor properties

We recommend that you create the directory structure as shown in Figure 7-53 to perform the migration and the Bernoulli sampling procedure.

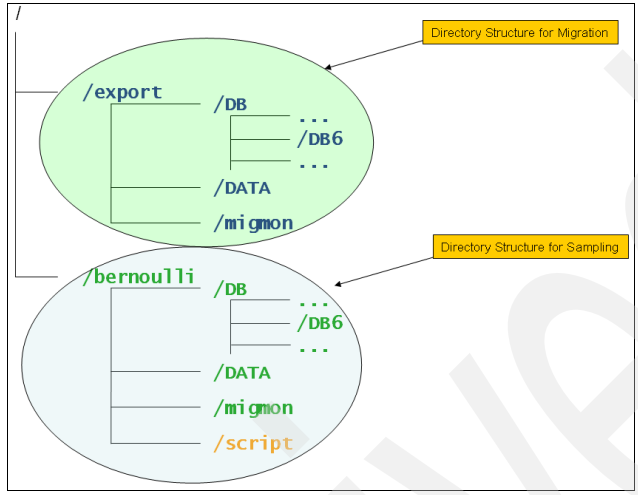


Figure 7-53 Directory structure for migrations with DB2 compression based on Bernoulli sampling

You also must specify the appropriate properties in the migration monitor (MigMon) properties files. When using R3load for sampling, you must adapt both the import monitor properties and the export monitor properties. See Example 7-12.

Example 7-12 MigMon properties file to deploy Bernoulli sampling

```
tab: ZVBAP_001
# List of export directories, separator on Windows ; on UNIX :
exportDirs=/bernoulli
# Installation directory
installDir=/bernoulli/migmon
# DDL control file, default is DDL<DB_TYPE>.TPL
ddlFile=/bernoulliP/DB/DB6/DDLDB6.TPL
```

Exporting the sample data

To export the data, simply run the script that is generated by the /ISIS/ZCOMP tool if you want to use the DB2 native method or run the export monitor and use R3load to export the data.

Import using DB2 CLP script

The ABAP program /ISIS/ZCOMP generates the DB2 CLP scripts needed to import the Bernoulli samples. However, before you can run the script, the target tables must be created using R3load. You can create them with the migration monitor by using parameter `OMIT` and setting this to `DIVP`. When running the import monitor only the table creation step will be executed.

After the tables have been created, you can use the DB2 CLP scripts created by /ISIS/ZCOMP to import the sample data. Before you run the CLP scripts, you might have to change the location of the export IXF file.

The time to load the data is minimal, usually less than 1 minute, as the default sample size is 6 MB. After importing the sample data, the script also enables compression by using the `ALTER TABLE` command, followed by table reorganization. The table reorganization creates the compression dictionary. The time needed to reorganize 6 MB of data is minimal, too (that is, less than one minute).

After the compression dictionary is created the script truncates the table. This removes all the data from the table but preserves the compression dictionary. When new data is inserted or loaded later on it will be compressed automatically.

Import using R3load

If the sample data was exported using R3load, you can use R3load with the new `SAMPLED` options to import the sample data into the DB2 target system. The new `SAMPLED` option is used in this case not to sample this data (not required since the data was already sampled during the export). The option is just used for convenience to make sure that the import stops on error after the sample was imported. So you can easily restart the import afterwards to actually load the data.

To import the sample data using the MigMon, you must change the `import_monitor_cmd.properties` file as follows:

```
loadArgs=-stop_on_error –loadprocedure fast SAMPLED_FULL_COMPRESS_ALL
```

The `importDirs` parameter must point to the directory that contains the STR files and the data files that contain the data samples. Before running the import monitor as user `<sid>adm`, the environment variable `DB6LOAD_SAMPLING_FREQUENCY` must be set to 1. This forces R3load to load all the data from the sample export.

Now you can start the import monitor to import the sample data.

After the sample data is imported, R3load automatically enables compression and creates the dictionary (through a table reorganization). It then stops on error.

The entire table data can then be loaded by restarting the migration monitor with the LOAD option. This truncates the existing data samples and loads the data in compressed format. Make sure that the `importDirs` parameter points to the directory that contains the entire export (not the sample export). The OMIT parameter must be changed to omit the table creation, since all the tables were created in the first execution of the import monitor. The import monitor can be restarted as follows to load the entire data set:

```
loadArgs=-stop_on_error –loadprocedure fast LOAD omit=T
```

Conclusion

DB2 compression based on Bernoulli sampling results in optimal compression rates. This is due to optimized sample sizes. It also reduces the amount of I/O to retrieve the data sample from disk compared with the R3load option `SAMPLED`.

However, our tests and experiences from customer migrations have shown that the sampling based on the R3load option `SAMPLED` results in similar compression results and is less complex to use. The scanning of the export dump files is also very fast and can be performed during a test migration if required. Therefore, we recommend this approach.

However, the Bernoulli sampling method is a valid option for highly optimized migrations. For example, when optimal compression is needed, downtime is critical and the sampling cannot be shifted outside the migration downtime window. Another use case could be a migration using the socket feature where no full export dump is available during the migration.

7.19 Deferred table creation

SAP systems consist of tens of thousands of tables. Many of them are empty. Every table created in a database consumes a certain amount of space, even if it is empty. In the case of a DB2 database, creating a table in a DMS table space allocates the following amount of extents or pages:

- Two extents for the table
- Three extents for the first index on the table
- Two extents for long fields (if applicable)
- Four extents for LOB objects (if applicable)

In other words, unused tables inside SAP applications lead to unnecessarily allocated disk space. To address this, SAP has introduced the concept of virtual tables. When installing or migrating an SAP system, database virtual tables can be created instead of physical tables. Virtual tables are read-only views that are defined on base tables with no physical representation in the database. As soon

as an application attempts to write data into one of these views, it gets materialized into a physical table.

More information about deferred table creation can be found in the SAP Paper: "Deferred Table Creation in SAP Systems with IBM DB2 for Linux, UNIX, and Windows" written by Johannes Heinrich. The paper can be found at the following link:

https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/e02a282a-868c-2b10-0e93-da0bb5abf461

R3load as of Version 7.00 supports the creation of virtual tables. Therefore, this concept can be utilized during database migrations or Unicode conversions.

The R3load option `DEF_CRT` enables deferred table creation. It can be combined with other options, for example, compression. In the following section we describe some examples of how to enable deferred table creation.

7.19.1 Enabling deferred table creation

Example 7-13 shows an example of how to use the deferred table creation option with R3load.

Example 7-13 R3load with deferred table creation option

```
R3load -i ATAB.cmd -dbcodepage 4103 -l ATAB.log –loadprocedure fast
LOAD:DEF_CRT
```

At first R3load creates a view ATAB. When loading data, DB2 returns an SQL error:

```
SQL0150N  The target fullselect, view, typed table, materialized query table,
or staging table in the INSERT, DELETE, UPDATE, or MERGE statement is a target
for which the requested operation is not permitted.
```

Based on this error, R3load materializes the view into a real table. We can see this from the log file shown in Example 7-14.

Example 7-14 R3load with deferred table creation option: Log file

```
(DB) INFO: ATAB created #20090427130836
(DB) INFO: ATAB~0 created #20090427130837
(DB6) [IBM][CLI Driver][DB2/LINUX] SQL0150N  The target fullselect, view, typed
table, materialized query table, or staging table in the INSERT, DELETE,
UPDATE, or MERGE statement is a target for which the requested operation is not
permitted.  SQLSTATE=42807
(DB6) INFO: Virtual Table ATAB has been successfully converted ( ignore
SQL0150N )
```

```
(DB6) Using LOAD API for table ATAB
(IMP) INFO: import of ATAB completed (750622 rows) #20090427130911
```

Virtual tables that are not loaded with any data are not converted, as shown in Example 7-15.

Example 7-15 R3load with deferred table creation option: No data loaded

```
/target/sapmnt/TAR/exe/R3load -i SWWLOGHIST.cmd -dbcodepage 4103 -l SWWLOGHIST.log
-loadprocedure fast LOAD:DEF_CRT

(DB) INFO: connected to DB
(GSI) INFO: dbname   = "TAR                                                    "
(GSI) INFO: vname    = "DB6                              "
(GSI) INFO: hostname = "UNKNOWN                                  "
(GSI) INFO: sysname  = "Linux"
(GSI) INFO: nodename = "n4shost"
(GSI) INFO: release  = "2.6.16.21-0.8-smp"
(GSI) INFO: version  = "#1 SMP Mon Jul 3 18:25:39 UTC 2006"
(GSI) INFO: machine  = "i686"


(RTF) ########## WARNING ###########
   Without ORDER BY PRIMARY KEY the exported data may be unusable for some databases

(DB) INFO: SWWLOGHIST created #20090427130908

(DB) INFO: SWWLOGHIST~0 created #20090427130908

(DB) INFO: SWWLOGHIST~1 created #20090427130908

(DB) INFO: SWWLOGHIST~2 created #20090427130908

(IMP) INFO: import of SWWLOGHIST completed (0 rows) #20090427130908

(DB) INFO: disconnected from DB
```

7.19.2 Enabling deferred table creation in conjunction with compression

As already mentioned in the previous sections, the R3load option Deferred Table Creation can be combined with compression.

Example 7-16 shows a sampled compression approach including deferred table creation.

Example 7-16 R3load with deferred table creation option and sample compression

```
target/sapmnt/TAR/exe/R3load -i ATAB.cmd -dbcodepage 4103 -l ATAB.log -loadprocedure
fast LOAD:SAMPLED_FULL_COMPRESS_ALL_DEF_CRT

(DB) INFO: connected to DB
(DB6) INFO: COMPRESS YES is set during table creation.
(DB6) INFO: If row compression is active, REORG will be triggered after all data has
been inserted.
(DB6) INFO: Inserting SAMPLED data with frequency 100 and maximum sam-ple size 0 MB.
(GSI) INFO: dbname  = "TAR
"
(GSI) INFO: vname    = "DB6                                "
(GSI) INFO: hostname = "UNKNOWN                                                        "
(GSI) INFO: sysname  = "Linux"
(GSI) INFO: nodename = "n4shost"
(GSI) INFO: release  = "2.6.16.21-0.8-smp"
(GSI) INFO: version  = "#1 SMP Mon Jul 3 18:25:39 UTC 2006"
(GSI) INFO: machine  = "i686"


(RTF) ########## WARNING ###########
   Without ORDER BY PRIMARY KEY the exported data may be unusable for some databases

(DB) INFO: ATAB created #20090427133801

(DB) INFO: ATAB~0 created #20090427133801

(DB6) [IBM][CLI Driver][DB2/LINUX] SQL0150N  The target fullselect, view, typed table,
materialized query table, or staging table in the INSERT, DELETE, UPDATE, or MERGE
statement is a target for which the requested operation is not permitted.
SQLSTATE=42807

(DB6) INFO: Virtual Table ATAB has been successfully converted ( ignore SQL0150N )
(DB6) Using LOAD API for table ATAB
(DB6) INFO: Table ATAB reorganized to create compression dictionary
(DB6) WARNING: Sampling was active and data for table ATAB was not com-pletely loaded.
7507 out of 750622 rows with maximum row size 574 have been sampled.
(IMP) ERROR: EndFastload: rc = 2
(DB) INFO: disconnected from DB

/target/sapmnt/TAR/exe/R3load: job finished with 1 error(s)
/target/sapmnt/TAR/exe/R3load: END OF LOG: 20090427133806

/target/sapmnt/TAR/exe/R3load: START OF LOG: 20090427134028

/target/sapmnt/TAR/exe/R3load: sccsid @(#) $Id:
//bas/700_REL/src/R3ld/R3load/R3ldmain.c#18 $ SAP
/target/sapmnt/TAR/exe/R3load: version R7.00/V1.4 [UNICODE]
Compiled Mar 11 2009 00:55:25
```

```
/target/sapmnt/TAR/exe/R3load -i ATAB.cmd -dbcodepage 4103 -l ATAB.log -loadprocedure
fast LOAD

(DB) INFO: connected to DB
(GSI) INFO: dbname  = "TAR
"
(GSI) INFO: vname    = "DB6                              "
(GSI) INFO: hostname = "UNKNOWN
"
(GSI) INFO: sysname  = "Linux"
(GSI) INFO: nodename = "n4shost"
(GSI) INFO: release  = "2.6.16.21-0.8-smp"
(GSI) INFO: version  = "#1 SMP Mon Jul 3 18:25:39 UTC 2006"
(GSI) INFO: machine  = "i686"

(RTF) ########## WARNING ###########
   Without ORDER BY PRIMARY KEY the exported data may be unusable for some databases

(DB) INFO: ATAB deleted/truncated #20090427134029

(DB6) Using LOAD API for table ATAB
(IMP) INFO: import of ATAB completed (750622 rows) #20090427134103

(DB) INFO: disconnected from DB

/target/sapmnt/TAR/exe/R3load: job completed
/target/sapmnt/TAR/exe/R3load: END OF LOG: 20090427134103
```

7.20 Optimizing statistics collection

After a database migration, the database statistics for all non-volatile tables must be collected with the RUNSTATS command to support the cost-based optimizer in generating good access plans. For large tables this can be an expensive and time-consuming procedure. But DB2 delivers some functions to improve the runtime of this step and therefore reduces the downtime of a migration. The following options are available:

- Use sampled statistics.
- Run multiple parallel RUNSTATS processes in parallel.
- Extract statistics from a previous test migration using DB2LOOK and apply those using the DB2 CLP.
- Start RUNSTATS when the last tables are imported into the database and free CPU and I/O resources are available.

Finally, you can combine the various options to find the optimal way.

The following section describes the advantages of the methods mentioned here and explains how to interrupt the normal migration process so that you can use these options.

7.20.1 Sampled statistics

You can collect sampled statistics both for tables and for indexes. For SAP systems the collection of distribution statistics for tables and detailed statistics for indexes is the recommended default. The runtime of these types of statistics can be significantly improved with the following sampling options:

- TABLESAMPLE SYSTEM

  Only a subset of the table's pages is analyzed. The performance improvement depends on the sampling rate. Use 10% as a starting point. Be aware that with this sampling option the execution plans of SQL statements may not be optimal.

- SAMPLED DETAILED

  This option applies to statistic information that is collected for indexes. It can be orders of magnitude faster than non-sampled DETAILED for large tables. It provides much higher quality statistics than non-DETAILED with a modest amount of extra CPU and memory.

7.20.2 Parallel runstats

To further improve performance more than one RUNSTATS process can be executed in parallel. Figure 7-54 shows the effect on the runtime.

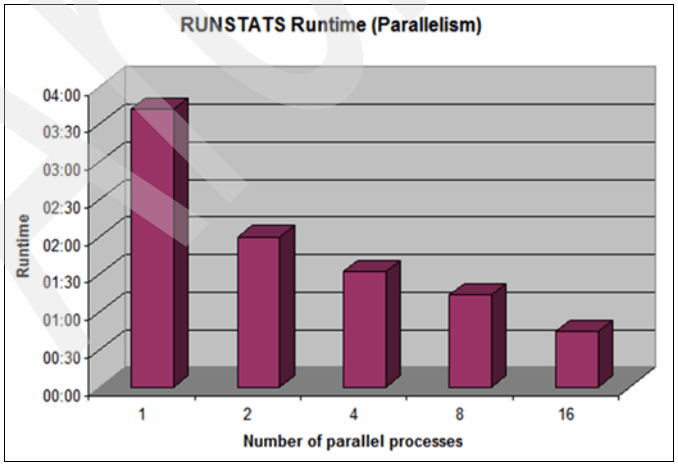


Figure 7-54 Runtime comparison of runstats with different numbers of parallelism

The tests were performed on an 8-way machine. In this case, the largest relative gain was reached when using two parallel RUNSTATS processes. Nevertheless, 16 parallel processes gave the shortest overall runtime. As the import is typically finished before collecting the database statistics, you can run many parallel processes to leverage the available hardware resources. However, you should monitor for lock wait situations on the system catalog, as those can degrade the overall performance.

To run massive parallel RUNSTATS against the database, we recommend that you generate a script with the RUNSTATS command that can be split into multiple smaller parts.

The statement in Example 7-17 can be useful. `<sap-schema>` must be replaced by the actual name of the sap schema.

Example 7-17 SQL statement to generate a script to collect statistics

```
SELECT 'RUNSTATS ON TABLE <sap-schema>.' || tabname ||
       ' WITH DISTRIBUTION AND DETAILED INDEXES ALL ; '
FROM syscat.tables
WHERE TYPE='T'
  AND TABSCHEMA='<SAP-SCHEMA>'
  AND VOLATILE != 'C'
ORDER BY stats_time asc, tabname
;
```

7.20.3 db2look

Once the statistical values have been calculated, they are stored in DB2 catalog tables that can be updated. For a final migration, you can extract these values from the statistics tables using the db2look tool. The tool generates an SQL file that contains SQL statements to update the statistics tables. As statistics values usually exist from a previous test migration, you can extract them using db2look. Although the statistic information is typically based on an older snapshot of the production database and therefore is outdated, you can use it to start production on the target system. We have not seen any issues due to these outdated statistics, but you should run the RUNSTATS against those tables as soon as possible after the system is restarted and used for production.

Note: Only use this method if the content of the corresponding tables does not change significantly after statistics have been extracted. Typically, the content of PSA and ODS tables in SAP NetWeaver BW systems is very volatile. Since the DB2LOOK approach to collect statistics is more complex to use we recommend using it only if really required.

Example 7-18 shows how to generate a db2look SQL-file for the update of table EDIDS in database NZW with schema name sapnzw. The name of the generated file is `edids_db2look_stats.sql`.

Example 7-18 db2look command to extract statistics for a table

```
db2look         -d nzw
                -m -r -c
                -z sapnzw -t edids
                -o edids_db2look_stats.sql
```

When you execute the above command, the content of the resulting SQL file is as shown in Example 7-19.

Example 7-19 DB2CLP file generated by db2look

```
-- This CLP file was created using DB2LOOK Version 9.1
-- Timestamp: Wed Mar 19 17:41:38 NFT 2008
-- Database Name: NZW
-- Database Manager Version: DB2/AIX64 Version 9.1.3
-- Database Codepage: 819
-- Database Collating Sequence is: IDENTITY
-- COMMIT is omitted. Explicit commit is required after executing the script.
------------------------------------------------------------
-- Mimic Tables, Columns, Indexes and Column Distribution
------------------------------------------------------------
-- Mimic table EDIDS
UPDATE SYSSTAT.TABLES
SET CARD=455396,
    NPAGES=6529,
    FPAGES=6529,
    OVERFLOW=0,
    ACTIVE_BLOCKS=0
WHERE TABNAME = 'EDIDS' AND TABSCHEMA = 'SAPNZW  ';
UPDATE SYSSTAT.COLUMNS
SET COLCARD=4,
    NUMNULLS=0,
    SUB_COUNT=-1,
    SUB_DELIM_LENGTH=-1,
    HIGH2KEY='811',
    LOW2KEY='810',
    AVGCOLLEN=3
WHERE COLNAME = 'MANDT' AND TABNAME = 'EDIDS' AND TABSCHEMA = 'SAPNZW  ';

...

UPDATE SYSSTAT.COLDIST
SET COLVALUE = NULL, VALCOUNT= -1
WHERE VALCOUNT <> -1 AND COLNAME = 'MANDT' AND TABNAME = 'EDIDS'
```

```
      AND TABSCHEMA = 'SAPNZW  ';
UPDATE SYSSTAT.COLDIST
SET COLVALUE='800',
    VALCOUNT=375046
WHERE COLNAME = 'MANDT' AND TABNAME = 'EDIDS'
      AND TABSCHEMA = 'SAPNZW  '
      AND TYPE      = 'F'
      AND SEQNO     =  1;
UPDATE SYSSTAT.COLDIST
SET COLVALUE='811',
    VALCOUNT=70196
WHERE COLNAME = 'MANDT' AND TABNAME = 'EDIDS'
      AND TABSCHEMA = 'SAPNZW  '
      AND TYPE      = 'F'
      AND SEQNO     =  2;
UPDATE SYSSTAT.COLDIST
SET COLVALUE='810',
    VALCOUNT=10094
WHERE COLNAME = 'MANDT' AND TABNAME = 'EDIDS'
      AND TABSCHEMA = 'SAPNZW  '
      AND TYPE      = 'F'
      AND SEQNO     =  3;

...

UPDATE SYSSTAT.INDEXES
SET NLEAF=360,
    NLEVELS=2,
    FIRSTKEYCARD=4,
    FIRST2KEYCARD=71,
    FIRST3KEYCARD=42649,
    FIRST4KEYCARD=-1,
    FULLKEYCARD=42649,
    CLUSTERFACTOR=0.931574,
    PAGE_FETCH_PAIRS='100 33400 258 29724 1032 26022 1806 25009 2580 20284 3354
19201 4386 11693 4644 7534 5160 6587 6529 6529 6529 6529 ',
    CLUSTERRATIO=-1,
    SEQUENTIAL_PAGES=359,
    DENSITY=88,
    AVERAGE_SEQUENCE_GAP=0.000000,
    AVERAGE_SEQUENCE_FETCH_GAP=-1.000000,
    AVERAGE_SEQUENCE_PAGES=359.000000,
    AVERAGE_SEQUENCE_FETCH_PAGES=-1.000000,
    AVERAGE_RANDOM_PAGES=0.000000,
    AVERAGE_RANDOM_FETCH_PAGES=-1.000000,
    NUMRIDS=455396,
    NUMRIDS_DELETED=0,
    NUM_EMPTY_LEAFS=0,
    AVGPARTITION_CLUSTERRATIO=-1,
```

```
    DATAPARTITION_CLUSTERFACTOR=1.000000,
    INDCARD=455396
WHERE INDNAME = 'EDIDS~2' AND INDSCHEMA = 'SAPNZW  '
      AND TABNAME = 'EDIDS' AND TABSCHEMA = 'SAPNZW  ';
```

When executing the **db2look** command, make sure that you do not forget the -r option. Otherwise, the generated script contains RUNSTATS commands.

The `-t` option allows you to specify up to 30 table names, but we recommend that you generate a dedicated file for each table to better control the process.

To identify large tables for which you want to apply extracted statistics, you can use the statement shown in Example 7-20 (replace the <*sap-schema*> with the actual sap schema name).

Example 7-20 SQL statement to retrieve names of largest tables from the database

```
SELECT tabname,
       data_object_l_size,
       long_object_l_size,
       lob_object_l_size
FROM   sysibmadm.admintabinfo
WHERE  tabschema = <sap-schema>
AND    data_object_l_size > 400000
```

Finally, you must apply the extracted statistics information to the DB2 target system as part of the productive migration. You can use the following command:

```
db2 –tvf <tabname>_db2look_stats.sql
```

You must perform this step for every table that will be handled with the db2look approach. Typically, the execution of the update commands on the catalog tables takes only sub seconds and so this is very fast.

Note: You can also apply the statistics outside the downtime window of the system migration. If you create the database before the downtime window, you can create the tables based on already existing test export files and then apply the statistics as described above. A prerequisite is that the table structure is not changed between the test migration and the final migration.

Currently, the following restrictions apply to the db2look approach to collect statistics:

- db2look can only apply *basic* statistics if runstats was not executed before. This means that before you extract the statistics with db2look, you must make sure that the corresponding tables have basic statistics, not distribution statistics.
- Automatic runstats is disabled when the generated script is executed on the DB2 target system for the corresponding tables.

To re-enable automatic runstats after the import of the statistics use the following command for each corresponding table:

```
db2 "RUNSTATS ON TABLE <tabname> SET PROFILE NONE"
```

7.20.4 Exclude the normal statistics collection from the process flow

Before you use the above-mentioned methods, you should consider the following. By default. SAPInst or R3SETUP automatically collect statistics. Depending on the SAPInst or R3SETUP version, this is done using dmdb6srp, the DB2 runstats command, by enabling automatic runstats, or a combination of these methods.

This must be avoided if you want to optimize statistic collection with the described methods. The next chapter explains how this can be achieved.

For SAP versions greater than or equal to SAP Basis 6.20, SAP supports only SAPInst versions located on the Patch Collection DVDs. For these versions the post-processing statistics collection is done only on the following tables:

- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSINDEXES
- SYSIBM.SYSTABLES
- SYSIBM.SYSUSERAUTH
- <SAP-SCHEMA>.ALCLASTOOL
- <SAP-SCHEMA>.ATAB
- <SAP-SCHEMA>.DD01L
- <SAP-SCHEMA>.DD03L
- <SAP-SCHEMA>.DD04L
- <SAP-SCHEMA>.DD07L
- <SAP-SCHEMA>.DD12L
- <SAP-SCHEMA>.DD25L
- <SAP-SCHEMA>.DDNTF
- <SAP-SCHEMA>.DDFTX
- <SAP-SCHEMA>.DDNTT
- <SAP-SCHEMA>.DYNPSOURCE

- <SAP-SCHEMA>.REPOSRC
- <SAP-SCHEMA>.REPOTEXT
- <SAP-SCHEMA>.RSMPTEXTS
- <SAP-SCHEMA>.TMDIR

The AUTO_RUNSTATS configuration parameter will be turned on after the import. To avoid this, adapt the file keydb.xml accordingly. SAPInst must have been executed once, so all necessary sections are available in keydb.xml.

For SAP Basis 4.6D or earlier releases, you can skip the statistics job (running the dmdb6srp executable) by adapting the appropriate section in the R3S-file. Change the ACTION variable to SKIP. In addition, the database configuration for automatic runstats must be disabled in the R3S file. You can do this by deleting the following two lines from the steps in the R3S file:

```
PARAMETER=AUTO_RUNSTATS
230_VALUE=OFF
```

7.20.5 Influence statistics collection using table DBSTATC

When using SAP statistics collection methods like *dmdb6srp* with previous releases of SAP or *DB13*, the process is controlled by the table DBSTATC. Although this is no longer the preferred method by SAP, you may have this enabled for specific reasons.

To exclude large tables from this method of statistics collection, appropriate entries must be made in the table DBSTATC.

Let us assume that you do not want SAP to gather statistics for a table named <*table*>. To avoid this you must check whether a row for <*table*> exists in DBSTATC. If a row exists, change the column ACTIV to the value I or R. If no row exists for the table, add a row for table <*table*> to DBSTATC with column ACTIV containing I or R.

> **Note:** Do not set the value of column ACTIV to N, as that marks <*table*> as volatile in the DB2 catalog table.

To include the table in the statistics collection process later on, either delete the row for <*table*> in DBSTATC or set the column ACTIV to A.

7.20.6 Choosing the correct method

Running the db2look script is much faster for large tables than executing the RUNSTATS command to generate statistics, and vice versa for small tables.

If we set the two runtimes in relation, we can define a runtime-factor (RF), as shown here:

$$RF = \frac{runtime(RUNSTATS)}{runtime(DB2LOOK)}$$

If the runtime-factor equals 1, RUNSTATS is as fast as updating the system catalog with the script generated by DB2LOOK. If the runtime factor is significantly higher than 1, it is beneficial to use the DB2LOOK method.

To decide which method to use, Figure 7-55 provides a rough guideline.

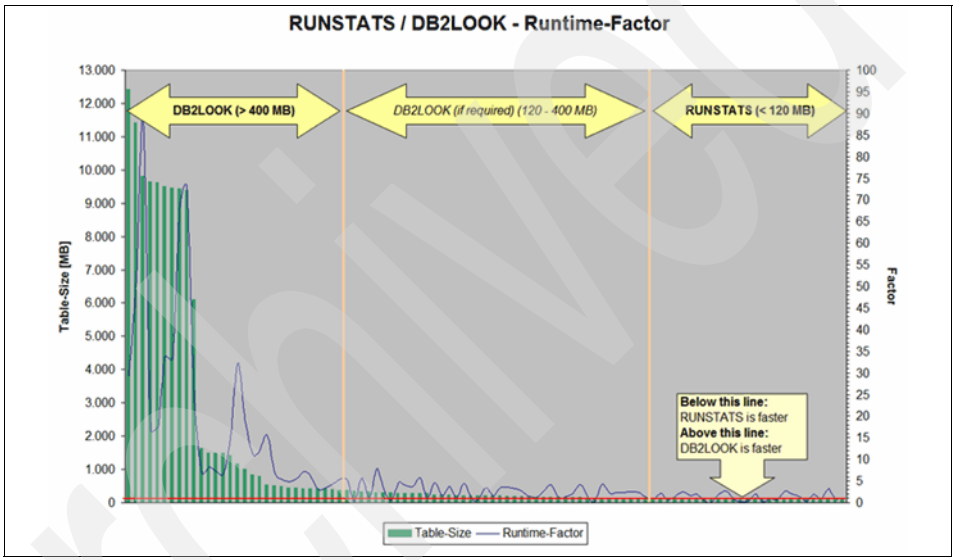


Figure 7-55 Runtime comparison RUNSTATS versus db2look

To decide which method is more appropriate to use you may test with both methods. Be aware that the runtime-factor is not only a function of the table size but also of the table structure and the underlying IT infrastructure. Especially for tables with many columns, the db2look method needs more time than for tables with fewer columns. Therefore, the break-even point for such a table is higher. The above test results indicate that tables with more than 400 MB of data are good candidates to handle with the db2look-based method to generate statistics.

Note: The above is the result based on the specific customer system that was used for these tests. The results will be different in a different environment. Nevertheless, these results provide a rule of thumb for the size category of tables to be used with db2look.

Using a combination of both methods can significantly improve overall performance of the generation of statistics during a migration to DB2.

Figure 7-56 shows the effects of different ways of generating DB2 statistics.

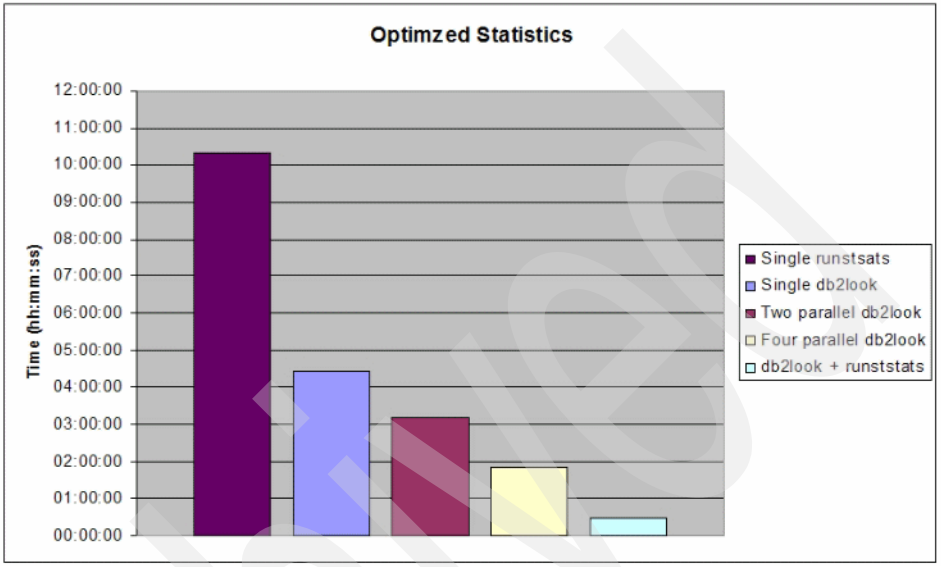


Figure 7-56 Performance comparison of different approaches to generate DB2 statistics

The very left bar shows the runtime of a single RUNSTATS process. The next three bars show the runtime with the db2look-based approach with 1, 2, and 4 parallel scripts. Finally, we used four parallel db2look scripts to generate statistics for the largest tables and a single script with the RUNSTATS command for the small tables.

In the final stage of the import, resource utilization typically declines as only a few R3load processes are still running. In this phase you can execute the prepared RUNSTATS or db2look scripts against tables that have already been loaded successfully. This way the statistics collection phase can overlap with the import phase. Running a script generated by db2look does not generate a high workload. However, you should monitor locks on the system catalog that can occur and interfere with the import phase.

To summarize the results, parallel RUNSTATS can optimize the statistic collection and you can use it to reduce the downtime window. The db2look method can further reduce the runtime dramatically and is a valid option for large tables. The performance-optimized solution could be to combine both methods and run the statistics collection while the import is still running.

8

Tips for monitoring the system copy process

In this chapter we describe monitoring tools and helpful SQL statements to gather useful data for monitoring the database during the migration. We do not provide a complete list of tools or monitoring information, but introduce some of the tools that our team found useful. Besides the tools described in this chapter, SAP delivers the remote monitoring functionality for DB2. The monitoring can be done from another SAP system, for example, SAP Solution Manager to fully monitor the DB2 system with the DBACOCKPIT and other SAP transactions.

8.1 db2top

db2top is a powerful, snapshot-based monitoring tool for DB2 for Linux, UNIX, and Windows databases. The tool is shipped as of DB2 Version 8 FP17, DB2 Version 9.1 FP6, and DB2 Version 9.5 FP2. Although it was initially intended to monitor partitioned databases, you can use the db2top tool to monitor DB2 databases during SAP system migrations. In this section we describe the most prominent options.

8.1.1 Applications

If you select the interactive command **1** in db2top, the sessions / applications screen is displayed, as shown in Example 8-1. For each R3load process, there is a R3load as well as a load entry (when using the fast LOAD option of R3load).

Example 8-1 db2top applications

```
[\]14:55:00,refresh=2secs(1.502)                Sessions                                AIX,part=[1/1],DB2ZWN:ZWN
[d=Y,a=N,e=N,p=ALL]                                                                      [qp=off]

  Local sessions...:         10                          25%         50%         75%       100%
  Remote sessions..:          4     ActSessions   -----------------------
  FCM buffers sent.:          0     Sys/Usr Cpu%  uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
  FCM buffers rec..:          0     r+w/sec%
  Piped sorts req..:         28
  Max Agents.......:          0      Agents stolen....:          0      Agent overflows..:          0
  Sheapthres.......:      78.1M      Max Sort Heap....:     171.4M      Sort heap alloc..:     171.4M
  Dynamic SQL stmts:      1,963      Static SQL stmts.:         21      Post hash joins..:          0
  Bufferpool reads.:          0      Bufferpool writes:          0      Direct reads.....:          0
     Application     Cpu%     IO%    Mem% Application                          Application
     Handle(Stat)   Total   Total   Total Status                                 Name
    ------------ ------- ------- ------- ---------------------------------- ---------------
            19(*)   0.00%   0.00%   8.14% UOW Executing                        R3load
            20(*)   0.00%   0.00%   8.72% UOW Executing                        R3load
            28(*)   0.00%   0.00%   9.30% UOW Executing                        R3load
             1(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             2(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             3(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             4(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             5(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             6(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             7(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             8(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
             9(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
            10(*)     N/A     N/A     N/A Setup/load[*] : 99%                  Load
            17(*)   2.70%   0.00%  11.63% UOW Executing                        R3load
            26(*)   2.70%   0.00%   9.30% UOW Executing                        R3load
            29(i)  10.81%   0.00%   9.30% UOW Waiting in the application       R3load
            27(i)  16.22%   0.00%   8.72% UOW Waiting in the application       R3load
            30(*)  24.32%   0.00%   8.72% UOW Executing                        R3load
            18(i)  43.24%   0.00%   9.30% UOW Waiting in the application       R3load
            21(i)   0.00%   0.00%   6.40% UOW Waiting in the application       R3load
            22(c)   0.00%   0.00%   3.49% Connected                            db2stmm
            23(c)   0.00%   0.00%   2.33% Connected                            db2taskd
            24(c)   0.00%   0.00%   2.33% Connected                            db2wlmd
           25(c)   0.00%   0.00%   2.33% Connected                           db2evmg_DB2DETA
```

If you select the interactive command a and if you provide an agent-ID, details about a certain R3load process are displayed. See Example 8-2.

Example 8-2 db2top: Application details

```
(*]14:57:06,refresh=14!secs(0.033)                    Sessions                        AIX,part=[1/1],DB2ZWN:ZWN
[d=Y,a=N,e=N,p=ALL]                                                                                   [qp=off]
*LOCAL.db2zwn.080603124835, UOW Waiting in the application
        ConnTime..: 14:48:35.409    UOW Start.: 14:48:36.542 Appl name.:  R3load    DB2 user..:   SAPZWN
        OS user...:       zwnadm    Agent id..:        18    Coord DBP.:       0    Coord id..:     6465
        Client pid:       745574    Hash joins:         0    Hash loops:       0    HJoin ovf.:        0
        SQL Stmts.:           20    Sorts.....:         4    Sort time.:   0.000    Sorts ovf.:        0
        Rows Read.:            0    Rows Sel..:        12    Read/Sel..:       0    Rows Wrtn.:        0
        Rows Ins..:            0    Rows Upd..:         0    Rows Del..:       0    Locks held:        0
        Trans.....:           20    Open Curs.:         0    Rem Cursor:       0    Memory....:   832.0K
        Dyn. SQL..:         3293    Static SQL:        28    Cpu Time..:  55.436    AvgCpuStmt:    0.166
---+----+----+----+----+----+----+----+ Dynamic statement [Execute] -+----+----+----+----+----+----+----
        Start.....: 14:57:06.239    Stop......: 14:57:06.319    Cpu Time..: 0.164    Elapse....:    0.800
        FetchCount:            0    Cost Est..:            8    Card Est..:     1    AgentTop..:        1
        SortTime..:            0    SortOvf...:            0    Sorts.....:     0    Degree....:        1
        Agents....:            1    l_reads...:            0    p_reads...:     0    DataReads.:        0
        IndexReads:            0    TempReads.:            0    HitRatio..: 0.00%    MaxDbpCpu.: 0.000[0]
        IntRowsDel:            0    IntRowsUpd:            0    IntRowsIns:     0
INSERT INTO "GLFUNCA" VALUES( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,  ?, ?, ?,
?, ?, ?, ?, ?, ?, ? )
```

8.1.2 Utilities

If you select the interactive command **u** of db2top, the utilities screen is displayed (Example 8-3). If you use the fast LOAD option of R3load, the underlying db2load processes are displayed.

Example 8-3 db2top: Utilities

```
[-]14:56:12,refresh=2secs(0.022)                  Utilities                          AIX,part=[1/1],DB2ZWN:ZWN
[d=Y,a=N,e=N,p=ALL]                                                                                   [qp=off]
Hash       # of     Utility        Utility     Uti Utility  Invoker  Completed        Work        Phase
Value      entries  Start Time     Type        Pri State    Type     Work             Unit   rog% Start Time
--------- ------- --------------- ----------- ---------- ---------- -------------- ------- ----- ---------------
        32       1  14:48:37.063978 Load        Execute    User           4,893,587 Rows     100% 14:48:37.820626
        73       1  14:48:37.822106 Load        Execute    User           1,430,901 Rows     100% 14:48:38.410790
   3359535       1  14:48:36.171723 Load        Execute    User             216,746 Rows     100% 14:48:36.242501
   4259913       1  14:48:36.606035 Load        Execute    User           4,903,748 Rows     100% 14:48:36.718483
   4277119       1  14:48:43.731411 Load        Execute    User           1,924,065 Rows     100% 14:48:44.475456
   4608047       1  14:48:44.431682 Load        Execute    User           1,920,457 Rows     100% 14:48:45.444587
   4672621       1  14:48:36.180905 Load        Execute    User             623,898 Rows     100% 14:48:36.246057
   5260591       1  14:48:36.554057 Load        Execute    User           1,212,508 Rows     100% 14:48:36.640082
   5458223       1  14:48:36.545191 Load        Execute    User           3,194,803 Rows     100% 14:48:36.640104
   5459052       1  14:48:36.175761 Load        Execute    User           1,394,956 Rows     100% 14:48:36.246437
```

8.2 Detailed monitoring

For a good overview of all of the activities going on during an SAP operating system and database (OS/DB) migration, you must look at various sources. The

following sections provide an insight into the details of monitoring and show where and how to obtain this important information.

8.2.1 Monitoring a single R3load process

In the log file of a dedicated R3load process, the last entries show which table is currently loaded. Example 8-4 is shows the log for loading the BKPF table.

Example 8-4 R3load <package>.log

```
/usr/sap/ZWN/SYS/exe/run/R3load -i BKPF.cmd -dbcodepage 1100 -l BKPF.log -loadprocedure fast
LOAD:FULL_COMPRESS_ALL
(DB) INFO: connected to DB
(DB6) INFO: COMPRESS YES is set during table creation.
(DB6) INFO: If row compression is active, REORG will be triggered after all data has been inserted.
(GSI) INFO: dbname   = "ZWN"
(GSI) INFO: vname    = "DB6                                  "
(GSI) INFO: hostname = "UNKNOWN                                                  "
(GSI) INFO: sysname  = "AIX"
(GSI) INFO: nodename = "coe6p001"
(GSI) INFO: release  = "3"
(GSI) INFO: version  = "5"
(GSI) INFO: machine  = "0001DBD7D600"
(RTF) WARNING: CREATE PSEUDO TABLE template not found in control file
"/mnt/nas_p001/export/hjm3/nuc_sort/EXP/DDLDB6.TPL"
(DB) INFO: BKPF created #20080603144841
(DB) INFO: BKPF~0 created #20080603144842
(DB) INFO: BKPF~1 created #20080603144842
(DB) INFO: BKPF~2 created #20080603144842
(DB) INFO: BKPF~3 created #20080603144842
(DB) INFO: BKPF~4 created #20080603144843
(DB) INFO: BKPF~5 created #20080603144843
(DB) INFO: BKPF~6 created #20080603144843
(DB) INFO: BKPF~BUT created #20080603144844
```

Information about how many rows must be loaded is located in the R3load <*package*>.TOC file, as shown in Example 8-5.

Example 8-5 R3load <package>.TOC

```
vn: R6.40/V1.4
id: 7d572a8d00000038
cp: 1100
data_with_checksum
tab: [HEADER]
fil: BKPF.001 1024
1 1
eot: #0 rows 20080228154724
tab: BKPF
fil: BKPF.001 1024
2 1024000
fil: BKPF.002 1024
1 268144
eot: #17520379 rows 20080228164403
eof: #20080228164403
```

Note: When performing a Unicode conversion, the number of rows stated in the <*package*>.TOC may differ from the number of rows actually loaded into the target table.

The `LIST UTILITES SHOW DETAIL` command of DB2 displays information about each load process. A load process consists of multiple phases. The current phase is marked in the output. In Example 8-6 we see that for table BKPF, Phase #2 (LOAD) is the current one and that 3,305,066 rows (out of 17,520,379) have been loaded.

Example 8-6 DB2 LIST UTILITIES command

```
coe6p001:db2zwn 34> db2 list utilities show detail
ID                                = 10
Type                              = LOAD
Database Name                     = ZWN
Partition Number                  = 0
Description                       = OFFLINE LOAD Unknown file type AUTOMATIC INDEXING
INSERT NON-RECOVERABLE SAPZWN  .BKPF
Start Time                        = 06/03/2008 14:48:44.431682
State                             = Executing
Invocation Type                   = User
Progress Monitoring:
   Phase Number                   = 1
      Description                 = SETUP
      Total Work                  = 0 bytes
      Completed Work              = 0 bytes
      Start Time                  = 06/03/2008 14:48:44.431700
   Phase Number [Current]         = 2
      Description                 = LOAD
      Total Work                  = 3305066 rows
      Completed Work              = 3305066 rows
      Start Time                  = 06/03/2008 14:48:45.444587
   Phase Number                   = 3
      Description                 = BUILD
      Total Work                  = 8 indexes
      Completed Work              = 0 indexes
      Start Time                  = Not Started
```

Furthermore, the DB2 diagnostic log file (db2diag.log) contains information about each LOAD job. See Example 8-7.

Example 8-7 db2diag.log

```
2008-06-03-14.48.44.747567+120 I17313A548         LEVEL: Warning
PID     : 1908802              TID  : 45015       PROC : db2sysc 0
INSTANCE: db2zwn               NODE : 000         DB   : ZWN
APPHDL  : 0-30                 APPID: *LOCAL.db2zwn.080603124847
AUTHID  : SAPZWN
EDUID   : 45015                EDUNAME: db2lfrm0 0
```

```
FUNCTION: DB2 UDB, database utilities, sqlulPrintPhaseMsg, probe:311
DATA #1 : String, 123 bytes
LOADID: 44758.2008-06-03-14.48.43.611458.0 (1;11)
Starting LOAD phase at 06/03/2008 14:48:44.595330. Table SAPZWN  .GLPCA
2008-06-03-14.48.45.450253+120 I17862A492         LEVEL: Warning
PID     : 1908802              TID  : 49641       PROC : db2sysc 0
INSTANCE: db2zwn               NODE : 000         DB   : ZWN
APPHDL  : 0-29                 APPID: *LOCAL.db2zwn.080603124846
AUTHID  : SAPZWN
EDUID   : 49641                EDUNAME: db2lrid 0
FUNCTION: DB2 UDB, database utilities, DIAG_NOTE, probe:0
DATA #1 : String, 79 bytes
LOADID: 40903.2008-06-03-14.48.44.330935.0 (1;10)
Load CPU parallelism is: 2, 0
2008-06-03-14.48.45.560370+120 I18355A547         LEVEL: Warning
PID     : 1908802              TID  : 48870       PROC : db2sysc 0
INSTANCE: db2zwn               NODE : 000         DB   : ZWN
APPHDL  : 0-29                 APPID: *LOCAL.db2zwn.080603124846
AUTHID  : SAPZWN
EDUID   : 48870                EDUNAME: db2lfrm0 0
FUNCTION: DB2 UDB, database utilities, sqlulPrintPhaseMsg, probe:311
DATA #1 : String, 122 bytes
LOADID: 40903.2008-06-03-14.48.44.330935.0 (1;10)
Starting LOAD phase at 06/03/2008 14:48:45.547420. Table SAPZWN  .BKPF
```

8.2.2 Lock wait situations

Run the SAP utility `db6util` with option `sl` to check for lock wait situations. Example 8-8 illustrates an output of the utility.

Example 8-8 db6util: Monitor lock wait situations

```
coe6p001:db2zwn 71> db6util -sl
This is the DB6 (DB2 UDB) utility program for SAP kernel 700.
(I) Option "-sl 1 1" .
(I) Checking for lock wait situations.
SNAPSHOT TAKEN AT: 20080513140838
--------------------------------
No Deadlocks were detected
LOCK WAITS:
---------
          28                  66
 (PID:1634366)  <--  (PID:1749106)
      R3load               db2bp
       ^                     75
       |----------  (PID:1892590)
                          db6util
DETAILED INFORMATION ABOUT LOCKED PROCESSES:
-------------------------------------------
  ID      PID       APPL-NAME     HOSTNAME(PART)  MODE RMODE OBJECT TABLE
  28  1634366         R3load       DB-Server(0)
  Status   : UOW Executing
  Wkstn    : coe6p001.wdf.sap.corp
  Appl.    : ptype UNKNOWN
  Last SQL : INSERT INTO "COEP" VALUES( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
             ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
```

```
            ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
            ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
            ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )
  66  1749106           db2bp       DB-Server(0)     Z     U  TABLE SAPZWN.COEP
  Status   : Lock Waiting (215 seconds)
  75  1892590         db6util       DB-Server(0)     Z    IN  TABLE SAPZWN.COEP
  Status   : Lock Waiting (16 seconds)
  Last SQL : CALL ADMIN_CMD( ( SELECT 'RUNSTATS ON TABLE sapzwn.coep AND
             DETAILED INDEXES ALL' FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1
             ROWS ONLY ))
(I) Connecting to ZWN as user zwnadm.
(I) Successfully connected to database ZWN .
(I) Application snapshot taken at 20080513140838.
(I) Disconnected from database.
db6util  successfully finished (exit 0).
```

8.2.3 Reorganization status

If you use R3load together with a COMPRESS option, table reorganization is triggered to build the compression dictionary. You can monitor the reorganization process, as shown in Example 8-9 and Example 8-10.

Example 8-9 db2pd: Monitor reorganization status

```
coe6p001:db2zwn 54> db2pd -db zwn –reorg
Database Partition 0 -- Database ZWN -- Active -- Up 0 days 01:36:32
Table Reorg Information:
Address   TbspaceID TableID PartID MasterTbs MasterTab TableName Type    IndexID    TempSpaceID
...  2DA8 6         5       n/a    n/a       n/a       RFBLG              Offline 0          5
...  22A8 1         4       n/a    n/a       n/a       SOC3               Offline 0          5
Table Reorg Stats:
Address   TableName Start               End  PhaseStart          MaxPhase Phase CurCount ...
Z4AA12DA8 RFBLG     06/03/2008 16:05:58 n/a  06/03/2008 16:05:59 4        Sort  173686   ...
04AA122A8 SOC3      06/03/2008 16:09:44 n/a  06/03/2008 16:09:46 4        Sort  100975   ...
```

Example 8-10 SQL: Monitor reorganization status

```
coe6p001:db2zwn 47>  db2 "SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA,
1, 15) AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,REORG_STATUS,
DBPARTITIONNUM FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM"
TAB_NAME     TAB_SCHEMA      REORG_PHASE      REORG_TYPE           REORG_STATUS DBPARTITIONNUM
----------- --------------- ---------------- -------------------- ------------ --------------
RFBLG           SAPZWN          BUILD            RECLAIM+OFFLINE+ALLO STARTED                0
  1 record(s) selected.
```

If you used the COMPRESS option, you can use the statement shown in Example 8-11 to obtain information about how many rows were compressed.

Example 8-11 SQL: Monitor compression status

```
coe6p001:db2zwn 83> db2 "SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, REORG_PHASE,
SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE, REORG_STATUS, REORG_ROWSCOMPRESSED FROM
SYSIBMADM.SNAPTAB_REORG"
TAB_NAME        REORG_PHASE      REORG_TYPE           REORG_STATUS REORG_ROWSCOMPRESSED
--------------- ---------------- -------------------- ------------ --------------------
RFBLG           BUILD            RECLAIM+OFFLINE+ALLO STARTED                     21844
SOC3            SORT+DICT_SAMPLE RECLAIM+OFFLINE+ALLO STARTED                         0
  2 record(s) selected.
```

8.2.4 Log space usage

When importing data or in the index build phase, database logging occurs. In this case, you can evaluate the amount of log space used and the application holding the oldest log entry, as shown in Example 8-12.

Example 8-12 SQL: Monitor log space usage

```
coe6p001:db2zwn 51> db2 "SELECT TOTAL_LOG_AVAILABLE AS LOG_AVAILABLE,TOTAL_LOG_USED as
LOG_USED, APPL_ID_OLDEST_XACT as OLDEST_APPL_ID from SYSIBMADM.SNAPDB"
LOG_AVAILABLE        LOG_USED            OLDEST_APPL_ID
-------------------- -------------------- --------------------
          3671186068            338637932                   22
  1 record(s) selected.

coe6p001:db2zwn 58> db2 "SELECT SUBSTR(APPL_NAME,1,10) AS APPL_NAME, APPL_STATUS FROM
SYSIBMADM.SNAPAPPL_INFO where AGENT_ID=22"
APPL_NAME  APPL_STATUS
---------- ----------------------
R3load     UOWWAIT
  1 record(s) selected.
```

8.2.5 Sorts

If you use R3load, sorts can occur, for example, during the export or during the INDEX creation phase of an importing process. The command shown in Example 8-13 provides information about active sorts, total sorts, and sort overflows, and therefore helps to tune DB2 parameters related to sorting.

Example 8-13 SQL: Sorts

```
coe6p001:db2zwn 70> db2 "SELECT ACTIVE_SORTS, TOTAL_SORTS, SORT_OVERFLOWS from
SYSIBMADM.SNAPDB"
ACTIVE_SORTS         TOTAL_SORTS          SORT_OVERFLOWS
------------------- -------------------- --------------------
                  26                   41                    0
  1 record(s) selected.
```

8.2.6 Utility heap size

Each DB2 LOAD process uses its own data buffer. This buffer is allocated from DB2's utility heap memory area. If the data buffer is not defined explicitly, DB2 calculates an intelligent default that is based on free space within the utility heap.

If multiple loads are running in parallel, you should monitor the utility heap to avoid heap full situations. You can do so by either using the DB2 memory tracker (`db2mtrk`) tool or the `db2top` tool (for more information see 8.1, "db2top " on page 206).

Without running any R3load process, an initial size of 64 KB is allocated.

Example 8-14 illustrates the output of db2mtrk.

Example 8-14 db2mtrk: Utility heap

```
coe6p001:db2zwn 1> db2mtrk -d
Tracking Memory on: 2008/06/04 at 14:41:12
Memory for database: ZWN
   utilh       pckcacheh   other       catcacheh   bph (1)     bph (S32K)
   64.0K       192.0K      192.0K      192.0K      315.8M      832.0K
   bph (S16K)  bph (S8K)   bph (S4K)   lockh       dbh         apph (487)
   576.0K      448.0K      384.0K      85.8M       21.3M       64.0K
   apph (486)  apph (485)  apph (484)  appshrh
   64.0K       64.0K       64.0K       192.0K
```

Example 8-15 illustrates the output of the `db2top` with the `-m` option.

Example 8-15 db2top –m

```
[-]14:42:48,refresh=2secs(0.059)                    Memory                    AIX,part=[1/1],DB2ZWN:ZWN
[d=Y,a=N,e=N,p=ALL]                                                                                 [qp=off]

                                        25%         50%         75%       100%
                    Memory hwm%   ---------------------------------------------------
                    Sort Heap%
                    Mem Skew%
                    Pool Skew%

    Memory                  Memory                Percent      Current         High Percent      Maximum # of
    Type        Level       Pool                    Total         Size    WaterMark     Max          Size Pool(s)
 ----------- ---------- -------------------- ------- ------------ ------------ ------- ------------ -------
    Instance    DB2ZWN      Monitor                0.07%       320.0K       320.0K  62.50%       512.0K       1
    Instance    DB2ZWN      FCMBP                  0.44%         1.9M         1.9M 206.67%       960.0K       1
    Instance    DB2ZWN      Other                  3.24%        14.3M        18.0M  13.44%       106.9M       1
    Database    ZWN         Applications           0.06%       256.0K       256.0K   0.52%        48.0M       4
    Database    ZWN         Database               4.80%        21.3M        21.3M   7.60%       280.5M       1
    Database    ZWN         Lock Mgr              19.33%        85.8M        85.8M  99.93%        85.8M       1
    Database    ZWN         Utility                0.01%        64.0K        64.0K   0.01%       781.2M       1
    Database    ZWN         Package Cache          0.04%       192.0K       192.0K   0.00%        20.0G       1
    Database    ZWN         Catalog Cache          0.04%       192.0K       192.0K   0.00%        20.0G       1
    Database    ZWN         Other                  0.04%       192.0K       192.0K   0.94%        20.0M       1
    Database    ZWN         BufferPool            71.63%       318.0M       318.0M   0.31%       100.0G       5
    Database    ZWN         Unknown[20]            0.04%       192.0K       192.0K   0.24%        78.1M       1
    Application ZWN         Applications           0.06%       256.0K       256.0K   0.52%        48.0M       4
    Application ZWN         Other                  0.20%       896.0K       896.0K   0.00%        80.0G       4
```

With active DB2 LOAD processes, memory for load data buffers is allocated from the utility heap. Example 8-16 illustrates the output of db2mtrk showing that the utility heap allocated is 335.3 MB.

Example 8-16 db2mtrk: Utility heap

```
coe6p001:db2zwn 1> db2mtrk -d
coe6p001:db2zwn 9> db2mtrk -d
Tracking Memory on: 2008/06/04 at 14:59:17
Memory for database: ZWN
   utilh       pckcacheh   other       catcacheh   bph (1)     bph (S32K)
   335.3M      384.0K      192.0K      320.0K      315.8M      832.0K
   bph (S16K)  bph (S8K)   bph (S4K)   lockh       dbh         apph (14)
   576.0K      448.0K      384.0K      85.8M       21.6M       128.0K
   apph (12)   apph (11)   apph (10)   apph (9)    appshrh
   64.0K       64.0K       64.0K       64.0K       192.0K
```

Monitoring using `db2top` with that `-m` option has the same result as that shown in Example 8-17.

Example 8-17 db2top –m

```
[-]14:59:40,refresh=2secs(1.524)                        Memory                        AIX,part=[1/1],DB2ZWN:ZWN
[d=Y,a=N,e=N,p=ALL]                                                                                      [qp=off]

                                          25%         50%         75%       100%
                        Memory hwm%   ---------------------------------------------------
                        Sort Heap%    ------
                        Mem Skew%
                        Pool Skew%

      Memory                Memory                Percent      Current         High Percent      Maximum # of
      Type        Level     Pool                    Total         Size    WaterMark     Max          Size Pool(s)
     ----------- ---------- -------------------- ------- ------------ ------------ ------- ------------ -------
      Instance    DB2ZWN    Monitor                 0.04%       320.0K       320.0K  62.50%       512.0K       1
      Instance    DB2ZWN    FCMBP                   0.10%       832.0K       832.0K  86.67%       960.0K       1
      Instance    DB2ZWN    Other                   1.86%        14.5M        14.5M  13.56%       106.9M       1
      Database    ZWN       Applications            0.05%       384.0K       384.0K   0.62%        60.0M       5
      Database    ZWN       Database                2.77%        21.6M        21.6M   7.71%       280.5M       1
      Database    ZWN       Lock Mgr               11.00%        85.8M        85.8M  99.93%        85.8M       1
      Database    ZWN       Utility                43.00%       335.3M       340.7M  42.92%       781.2M       1
      Database    ZWN       Package Cache           0.05%       384.0K       384.0K   0.00%        20.0G       1
      Database    ZWN       Catalog Cache           0.04%       320.0K       320.0K   0.00%        20.0G       1
      Database    ZWN       Other                   0.02%       192.0K       192.0K   0.94%        20.0M       1
      Database    ZWN       BufferPool             40.78%       318.0M       318.0M   0.31%       100.0G       5
      Database    ZWN       Unknown[20]             0.02%       192.0K       192.0K   0.24%        78.1M       1
      Application ZWN       Applications            0.05%       384.0K       384.0K   0.62%        60.0M       5
      Application ZWN       Other                   0.21%        1.6M         1.7M   0.00%       100.0G       5
```

8.2.7 Measuring table size

The statement shown in Example 8-18 reports the logical space allocated to the table. This is the amount of space that the table knows about. It is separated into the components for the basis table data (DATA_OBJECT_L_SIZE), long field data (LONG_OBJECT_L_SIZE), and LOB field data (LOB_OBJECT_L_SIZE).

Example 8-18 SQL: Statement to obtain table size

```
db2 "select TABNAME, DATA_OBJECT_L_SIZE, LONG_OBJECT_L_SIZE, LOB_OBJECT_L_SIZE,
DATA_OBJECT_L_SIZE+LONG_OBJECT_L_SIZE+LOB_OBJECT_L_SIZE AS L_SIZE from
SYSIBMADM.ADMINTABINFO where tabschema='SAP<SAPSID>' order by tabname"
```

8.3 NMON

NMON is a free tool to analyze performance of systems that are running on AIX and Linux. The following Web site contains a description on how to download and use NMON:

http://www.ibm.com/developerworks/aix/library/au-analyze_aix/

You can start NMON either in interactive mode or in data capture mode. While the first option is suitable for *ad hoc* monitoring, the capture option allows subsequent analysis of bottlenecks that may have been present throughout the migration process.

Example 8-19 shows NMON in interactive mode, displaying CPU utilization, I/O activity, and top processes.

Example 8-19 nmon –p –t –d

```
nmon12aV=VolumeGroupsHost=coe6p001Refresh=2 secs16:10.30
 CPU-Utilisation-Small-View EntitledCPU=  4.00 UsedCPU=  1.940
Logical  CPUs              0----------25-----------50----------75----------100
CPU User%  Sys% Wait% Idle%|           |           |           |           |
  0  25.2  20.8  54.0   0.0|UUUUUUUUUUUUsssssssssWWWWWWWWWWWWWWWWWWWWWWWWWW>|
  1  40.1   9.9   5.4  44.6|UUUUUUUUUUUUUUUUUUUUUUsssWW                    >|
  2  30.2   9.9   5.9  54.0|UUUUUUUUUUUUUUUUsssWW                          >
  3  34.2  11.4   5.9  48.5|UUUUUUUUUUUUUUUUUUsssssWW                    >  |
  4  30.5  17.2   3.4  48.8|UUUUUUUUUUUUUUUUsssssssssW                       >
  5  29.7  10.4  56.4   3.5|UUUUUUUUUUUUUUUUssssssWWWWWWWWWWWWWWWWWWWWWWWWWW> |
  6  21.8   1.0  38.6  38.6|UUUUUUUUUUUWWWWWWWWWWWWWWWWWWWWW                 >
  7   2.5   2.5   0.0  95.0|Us                                               >
EntitleCapacity/VirtualCPU +-----------|-----------|-----------|-----------+
 EC  36.1   9.7   1.2   1.6|UUUUUUUUUUUUUUUUUUUUsssss-------------------------|
 VP  36.1   9.7   1.2   1.6|UUUUUUUUUUUUUUUUUUUUsssss-------------------------|
EC=  48.5%  VP=  48.5%     +--No Cap---|-----------|-----------100% VP=4 CPU+
 Disk-KBytes/second-(K=1024)
Disk     Busy  Read  Write 0----------25-----------50-----------75--------100
 Name          KB/s   KB/s |           |           |            |           |
hdisk1     0%      0      0|           |           |            |           |
hdisk0     2%      0     20|W
hdisk2    34%    175   1625|RRWWWWWWWWWWWWWWWWW                  >
hdisk3    34%    199   1611|RRWWWWWWWWWWWWWWWWW                  >
hdisk4    37%    207   1603|RRRWWWWWWWWWWWWWWWWW                 >
hdisk5    35%    167   1639|RRWWWWWWWWWWWWWWWW                      >
Totals           747   6497+-----------|-----------|-------------|----------+
 Top-Processes-(443) Mode=3  [1=Basic 2=CPU 3=Perf 4=Size 5=I/O 6=Cmds]
  PID     %CPU  Size   Res   Res   Res  Char RAM      Paging       Command
          Used    KB   Set  Text  Data   I/O  Use io other repage
 1908802  93.2   252M  252M    76  252M 9908M  1%   15    0    0 db2sysc
 2134106  32.5 12852 10756  2596  8160  1761M  0%    0    0    0 R3load
 1855690  22.7 11804  9708  2596  7112   585M  0%    0    1    0 R3load
  168038   5.5  1280  1152     0  1152     0   0%    0    1    0 kbiod
 1851408   2.4   512   512     0   512     0   0%    0    0    0 nfssync_kproc
   16392   2.4   640   448     0   448     0   0%    0    0    0 lrud
  172170   1.6   832   768     0   768     0   0%    0    0    0 rtcmd
  561178   0.9   448   384     0   384     0   0%    0    1    0 aioserver
 1949698   0.9  1748  1824   132  1692  4887   0%    0   28    0 db2fm
  315548   0.6   936   964    60   904     0   0%    0    0    0 xmwlm
  757996   0.5 11504 11576   276 11300  2469   0%    0    0    0 nmon12a_aix536
  135278   0.4   508   512     4   508     0   0%    0    0    0 syncd
  749728   0.3   448   384     0   384     0   0%    0    1    0 aioserver
  102490   0.3   448   384     0   384     0   0%    0    3    0 aioserver
  508158   0.2   448   384     0   384     0   0%    0    0    0 aioserver
 1146928   0.2   448   384     0   384     0   0%    0    2    0 aioserver
 1028342   0.2   448   384     0   384     0   0%    0    0    0 aioserver
  110690   0.2   448   384     0   384     0   0%    0    0    0 aioserver
 1458414   0.2   448   384     0   384     0   0%    0    1    0 aioserver
   24588   0.2   640   448     0   448     0   0%    0    0    0 psmd
  602158   0.2   448   384     0   384     0   0%    0    1    0 aioserver
 1417470   0.2   448   384     0   384     0   0%    0    0    0 aioserver
  651330   0.2   448   384     0   384     0   0%    0    0    0 aioserver
```

Figure 8-1 is based on NMON data that was captured during the import of tables to the target system.

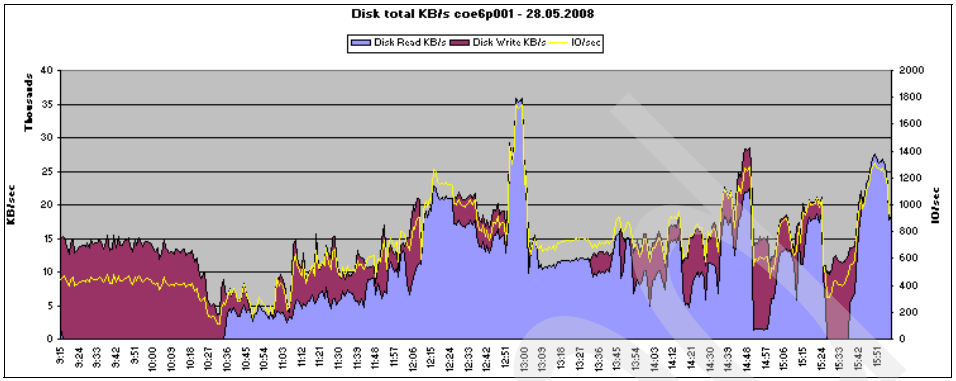


Figure 8-1 nmon monitoring: I/O

We can clearly see that until 10:30h, we only have write activity. After this point, the index creation and compression dictionary build starts.

When looking at the CPU utilization in Figure 8-2, we see that at the same point in time the “Wait%” is increasing. Our test system clearly is I/O-bound.

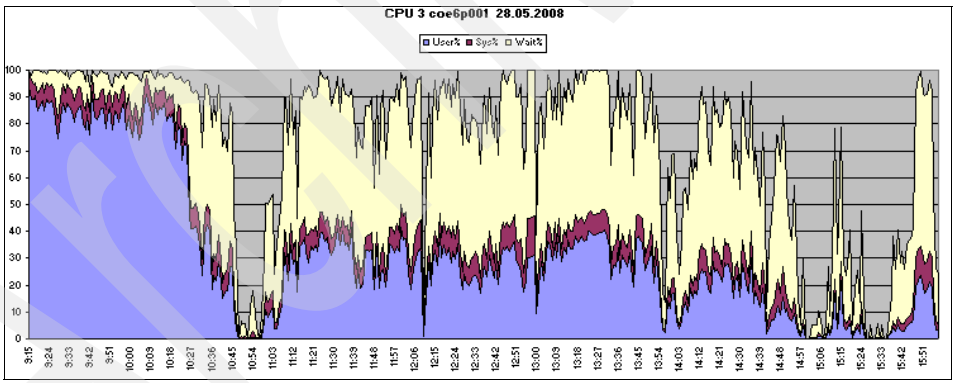


Figure 8-2 nmon monitoring: CPU

9

Considerations for migrating SAP NetWeaver BW-based systems

Heterogeneous system copies of SAP NetWeaver BW systems or systems based on SAP NetWeaver Business Warehouse (SAP NetWeaver BW) (such as SAP SEM®, SAP SCM) require special treatment.

While they are in principle also exported and imported using the same R3load method like non-BW systems, you must perform special preparation (source system) and post processing (target system) steps because SAP has implemented database-specific features for BW-based systems.

On DB2 for Linux, UNIX, and Windows, these special features are:

- Database partitioning feature (DPF): Hash partitioning
- Multi-dimensional clustering

In addition to this, there are differences in the structure of tables and number and structure of secondary indexes for each database platform.

This chapter briefly describes the process of migrating an SAP NetWeaver BW-based system.

A comprehensive description is contained within the white paper *Heterogeneous SAP NetWeaver Business Intelligence (BI) System Copy to IBM DB2 for Linux, UNIX and Windows*, written by Brigitte Bläser. The document is available in the SAP Community Network at:

https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/c0546f90-998b-2a10-0fae-989576a8cb39

Steps of a heterogeneous SAP NetWeaver BW system copy

Starting with SAP NetWeaver Business Warehouse 3.0B, three additional steps are defined for SAP NetWeaver Business Warehouse (SAP NetWeaver BW) system migrations:

- On the source system, you must execute report SMIGR_CREATE_DDL. This report compiles the target database platform Data Definition Language (DDL) for creating the database-specific SAP NetWeaver BW tables and indexes in the target database. R3load uses the DDL to create the tables and indexes correctly in the target database.
- On the source system, you must execute SAP_DROP_TMPTABLES.
- On the target system, after database import, you must execute report RS_BW_POST_MIGRATION, which performs a number of SAP NetWeaver BW-specific post migration tasks.

Figure 9-1 depicts the SAP NetWeaver BW migration procedure.

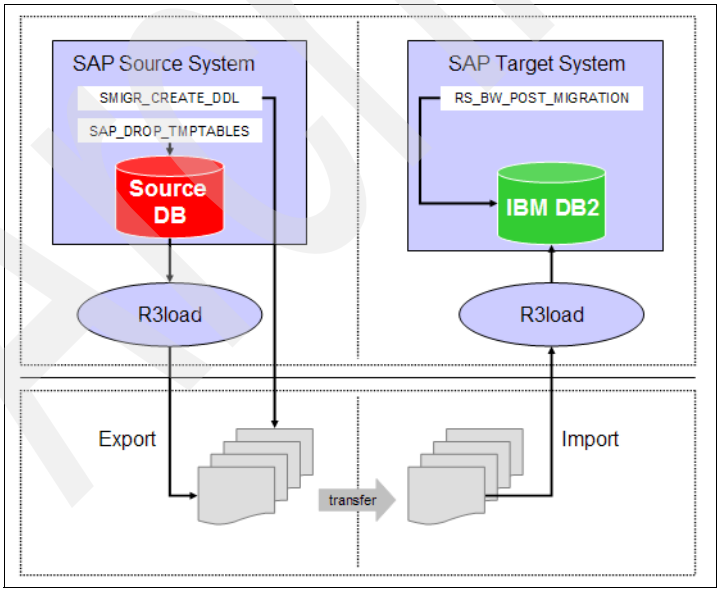


Figure 9-1 SAP NetWeaver BW migration procedure

9.1 Pre-migration tasks on the source system

Report SAP_DROP_TMPTABLES drops SAP NetWeaver BW temporary tables on views on the source system. Execute this report on the source system before exporting the database.

Report SMIGR_CREATE_DDL creates DDL files to support the creation of DB2-specific objects in the target database. Execute this report on the source system before exporting the database.

Figure 9-2 shows an example of the SMIGR_CREATE_DDL report.

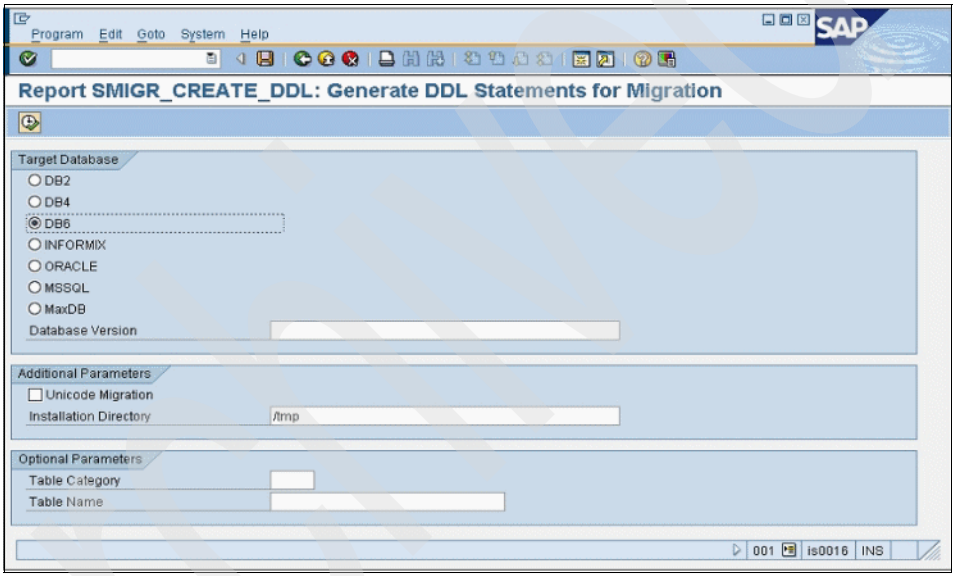


Figure 9-2 Report SMIGR_CREATE_DDL

The generated DDL files have the form shown in Example 9-1.

Example 9-1 DDL file

```
tab: <Table name>
sql: <DDL Statement(s) for table>
ind: <Index name>
sql: <DDL Statement(s) for index>
```

The files are created with the following naming convention:

```
<dataclass>.SQL
```

R3load can use these DDL files to create database objects that are based on the syntax provided in the file.

Figure 9-3 shows the input files that are used by R3load.

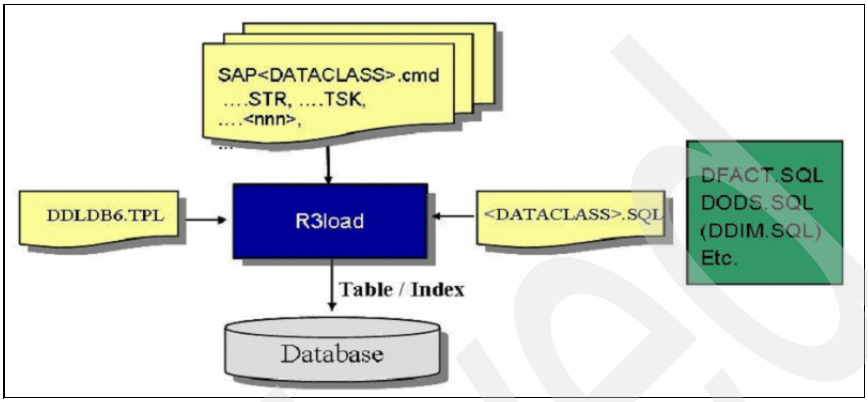


Figure 9-3 Input files to R3load for importing the target database

Example 9-2 shows the DDL that is generated for a standard E fact table of an InfoCube that does not use range partitioning.

Example 9-2 DDL file for an E fact table

```
tab: /BIC/EBFABCUBE5
sql: CREATE TABLE "/BIC/EBFABCUBE5"
("KEY_BFABCUBE5P" INTEGER DEFAULT 0 NOT NULL,
"KEY_BFABCUBE5T" INTEGER DEFAULT 0 NOT NULL,
"KEY_BFABCUBE5U" INTEGER DEFAULT 0 NOT NULL,
"KEY_BFABCUBE51" INTEGER DEFAULT 0 NOT NULL,
"KEY_BFABCUBE52" INTEGER DEFAULT 0 NOT NULL,
"/BIC/BCRMEM_QT" DECIMAL(000017,000003) DEFAULT 0 NOT NULL,
"/BIC/BCRMEM_VA" DECIMAL(000017,000002) DEFAULT 0 NOT NULL,
"/BIC/BINVCD_VA" DECIMAL(000017,000002) DEFAULT 0 NOT NULL,
"/BIC/BINVCD_QT" DECIMAL(000017,000003) DEFAULT 0 NOT NULL,
"/BIC/BRTNSVAL" DECIMAL(000017,000002) DEFAULT 0 NOT NULL,
"/BIC/BRTNSQTY" DECIMAL(000017,000003) DEFAULT 0 NOT NULL) IN "&location&"
INDEX IN "&locationI&"
LONG IN "&locationL&"
PARTITIONING KEY ( "KEY_BFABCUBE51"
, "KEY_BFABCUBE52"
, "KEY_BFABCUBE5T"
, "KEY_BFABCUBE5U"
)
USING
HASHING;
ALTER TABLE "/BIC/EBFABCUBE5" LOCKSIZE ROW;
```

```
ind: /BIC/EBFABCUBE5~O
ind: /BIC/EBFABCUBE5~O1
ind: /BIC/EBFABCUBE5~O3
ind: /BIC/EBFABCUBE5~P
sql: CREATE UNIQUE INDEX "/BIC/EBFABCUBE5~P" on "/BIC/EBFABCUBE5"
("KEY_BFABCUBE5T",
"KEY_BFABCUBE51",
"KEY_BFABCUBE52",
"KEY_BFABCUBE5U",
"KEY_BFABCUBE5P")
CLUSTER
ALLOW REVERSE SCANS;
```

For more details, see the white paper *Heterogeneous SAP NetWeaver Business Intelligence (BI) System Copy to IBM DB2 for Linux, UNIX, and Windows*.

9.2 Post-migration tasks on the target system

After successfully loading your target database, you must run report RS_BW_POST_MIGRATION in the target SAP system.

This report has the following variants:

- SAP&POSTMGR

  Used when the DB platform has not changed, for example, when performing a pure Unicode conversion.

- SAP&POSTMGRDB

  Used when the DB platform has changed. It includes a number of additional repair operations that are necessary because of the SAP NetWeaver BW implementation differences on the different database platforms.

For more details see the white paper *Heterogeneous SAP NetWeaver Business Intelligence (BI) System Copy to IBM DB2 for Linux, UNIX, and Windows*.

10

Unicode details

This chapter contains information about Unicode and its implementation in an SAP system environment. Implementing Unicode has some effects on the infrastructure sizing, and the conversion process may be a complex project. Although the details and influences during the technical migration are explained in this chapter, this is just a small part of the work that must be done.

In this chapter we explain some of the most important background information that influences the sizing for the data server as well as information about the SAP-specific Unicode implementation. Reading this chapter, you are able to understand that a Unicode conversion and a database migration to DB2 are based on the same technical concept. Therefore, a migration to DB2 can be easily integrated into a Unicode conversion project without significantly increasing its complexity.

10.1 Flow of Unicode conversion

In this section we provide a high-level overview of the Unicode conversion. From this overview, you can gain a clear understanding that the database conversion technicality is not only important, but also critical, as it determines the system downtime. In most cases the majority of the project runtime and efforts are spent on the application-related conversion tasks. For a more detailed description see the *Unicode Conversion Guides*.

A Unicode conversion consists of the following phases:

- PREPARATION phase
  - Set up the conversion project.
  - Check the prerequisites.
  - Prepare MDMP (SAP transactions SPUMG/SPUM4).
  - Enable customer developments (SAP transaction UCCHECK).
- CONVERSION phase
  - Prepare non-Unicode system (all table rows in the database need code page information):
    - Data volume reduction (archiving)
    - SPUMG scans

      Consistency check, tables without language information (build up vocabulary), tables with ambiguous language information (words with ambiguous language information are added to vocabulary), tables with language information, reprocess (checks if the code page information in the vocabulary is sufficient for the conversion), INDX analysis, INDX repair.
    - Nametab handling
  - System copy (based on R3load), consisting of database export with codepage conversion and database import with optional codepage conversion
  - Unicode system
    - Conversion completion (general tasks and application-specific tasks)
    - SUMG
- Post processing phase
  - Start Unicode system and do basic checks.
  - Verify data consistency.
  - Integration testing focused on language handling.

10.2 Introduction to code page and character notation

Unicode allows a consistent representation of multiple languages and their characters in an IT system.

Before using Unicode, SAP systems used the following methods to represent the characters of one or more languages:

- Standard single code page: All characters belong to the same code page.
- Blended code page: SAP-constructed code page with characters taken from different standard code pages. Use SAP report RSCPINST to check this.
- Multi-display/multi-processing (MDMP): Code page is chosen dynamically due to logon language. Different users can use different code pages, for example, English, German, and Japanese can be used in one system.

There are single and double byte code pages. Double byte code pages are mainly used for some Asian languages that have a lot more characters than other alphabets.

Figure 10-1 shows the code page ISO Latin-1 (also called ISO8859-1).

HEX DIGITS 1ST → 2ND ↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0			(SP) SP010000	0 ND100000	@ SM050000	P LP020000	` SD130000	p LP010000			(RSP) SP300000	° SM190000	À LA140000	Ð LD620000	à LA130000	ð LD630000
-1			! SP020000	1 ND010000	A LA020000	Q LQ020000	a LA010000	q LQ010000			¡ SP030000	± SA020000	Á LA120000	Ñ LN200000	á LA110000	ñ LN190000
-2			" SP040000	2 ND020000	B LB020000	R LR020000	b LB010000	r LR010000			¢ SC040000	² ND021000	Â LA160000	Ò LO140000	â LA150000	ò LO130000
-3			# SM010000	3 ND030000	C LC020000	S LS020000	c LC010000	s LS010000			£ SC020000	³ ND031000	Ã LA200000	Ó LO120000	ã LA190000	ó LO110000
-4			$ SC030000	4 ND040000	D LD020000	T LT020000	d LD010000	t LT010000			¤ SC010000	´ SD110000	Ä LA180000	Ô LO160000	ä LA170000	ô LO150000
-5			% SM020000	5 ND050000	E LE020000	U LU020000	e LE010000	u LU010000			¥ SC050000	µ SM170000	Å LA280000	Õ LO200000	å LA270000	õ LO190000
-6			& SM030000	6 ND060000	F LF020000	V LV020000	f LF010000	v LV010000			¦ SM050000	¶ SM250000	Æ LA520000	Ö LO180000	æ LA510000	ö LO170000
-7			' SP050000	7 ND070000	G LG020000	W LW020000	g LG010000	w LW010000			§ SM240000	· SD630000	Ç LC420000	× SA070000	ç LC410000	÷ SA060000
-8			(SP060000	8 ND080000	H LH020000	X LX020000	h LH010000	x LX010000			¨ SD170000	¸ SD410000	È LE140000	Ø LO620000	è LE130000	ø LO610000
-9			) SP070000	9 ND090000	I LI020000	Y LY020000	i LI010000	y LY010000			© SM520000	¹ ND011000	É LE120000	Ù LU140000	é LE110000	ù LU130000
-A			* SM040000	: SP180000	J LJ020000	Z LZ020000	j LJ010000	z LZ010000			ª SM210000	º SM200000	Ê LE160000	Ú LU120000	ê LE150000	ú LU110000
-B			+ SA010000	; SP140000	K LK020000	[SM060000	k LK010000	{ SM110000			« SP170000	» SP180000	Ë LE180000	Û LU160000	ë LE170000	û LU150000
-C			, SP080000	< SA030000	L LL020000	\ SM070000	l LL010000	| SM130000			¬ SM660000	¼ NF040000	Ì LI140000	Ü LU180000	ì LI130000	ü LU170000
-D			- SP100000	= SA040000	M LM020000	] SM080000	m LM010000	} SM140000			¯ (SHY) SP320000	½ NF010000	Í LI120000	Ý LY120000	í LI110000	ý LY110000
-E			. SP110000	> SA050000	N LN020000	^ SD150000	n LN010000	~ SD190000			® SM530000	¾ NF050000	Î LI160000	Þ LT640000	î LI150000	þ LT630000
-F			/ SP120000	? SP150000	O LO020000	_ SP090000	o LO010000				¯ SM150000	¿ SP160000	Ï LI180000	ß LS610000	ï LI170000	ÿ LY170000

Figure 10-1 Code page ISO Latin-1

This single byte code page can store 256 characters (from x'00' to x'FF' in a hexadecimal representation). The first 128 characters represent the 7-bit-ASCII. The upper part (x'80' to x'FF') contains special characters that are used in several European languages.

Having installed the code page ISO Latin-1 in an ERP system, users can use the languages Danish, Dutch, English, Finnish, French, German, Icelandic, Italian, Norwegian, Portuguese, Spanish, and Swedish.

If you want to use another language, you must install another code page, for example, ISO Latin-2 (ISO8859-2) for Polish. See Figure 10-2. Having installed this code page, you can use the languages Croatian, Czech, English, German, Hungarian, Polish, Slovakian, and Slovenian.

HEX DIGITS 1ST → 2ND ↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		► SM590000	(SP) SP010000	0 ND100000	@ SM050000	P LP020000	` SD130000	p LP010000	▒ SF140000	╚ SF020000	(RSP) SP300000	° SM190000	Ŕ LR120000	Đ LD020000	ŕ LR110000	đ LD010000
-1	☺ SS000000	◄ SM630000	! SP020000	1 ND010000	A LA020000	Q LQ020000	a LA010000	q LQ010000	▓ SF150000	╩ SF070000	Ą LA440000	ą LA430000	Á LA120000	Ń LN120000	á LA110000	ń LN110000
-2	☻ SS010000	↕ SM760000	" SP040000	2 ND020000	B LB020000	R LR020000	b LB010000	r LR010000	▌ SF160000	╠ SF060000	˘ SD230000	˛ SD430000	Â LA160000	Ň LN220000	â LA150000	ň LN210000
-3	♥ SS020000	!! SP330000	# SM010000	3 ND030000	C LC020000	S LS020000	c LC010000	s LS010000	▐ SF110000	╬ SF080000	Ł LL620000	ł LL610000	Ă LA240000	Ó LO120000	ă LA230000	ó LO110000
-4	♦ SS030000	¶ SM250000	$ SC030000	4 ND040000	D LD020000	T LT020000	d LD010000	t LT010000	┤ SF090000	═ SF100000	¤ SC010000	´ SD110000	Ä LA180000	Ô LO160000	ä LA170000	ô LO150000
-5	♣ SS040000	§ SM240000	% SM020000	5 ND050000	E LE020000	U LU020000	e LE010000	u LU010000	╝ SF040000	╪ SF050000	Ľ LL220000	ľ LL210000	Ĺ LL120000	Ő LO260000	Í LL110000	ő LO250000
-6	♠ SS050000	▬ SM700000	& SM030000	6 ND060000	F LF020000	V LV020000	f LF010000	v LV010000	╒ SF010000	◣ SF570000	Ś LS120000	ś LS110000	Ć LC120000	Ö LO180000	ć LC110000	ö LO170000
-7	• SM570000	↨ SM770000	' SP050000	7 ND070000	G LG020000	W LW020000	g LG010000	w LW010000	◙ SF610000	▬ SF600000	§ SM240000	ˇ SD210000	Ç LC420000	× SA070000	ç LC410000	÷ SA060000
-8	◘ SM570001	↑ SM320000	(SP060000	8 ND080000	H LH020000	X LX020000	h LH010000	x LX010000	© SM520000	╙ SF360000	¨ SD170000	¸ SD410000	Č LC220000	Ř LR220000	č LC210000	ř LR210000
-9	○ SM750000	↓ SM330000	) SP070000	9 ND090000	I LI020000	Y LY020000	i LI010000	y LY010000	╗ SF230000	╓ SF390000	Š LS220000	š LS210000	É LE120000	Ů LU280000	é LE110000	ů LU270000
-A	◙ SM750002	→ SM310000	* SM040000	: SP130000	J LJ020000	Z LZ020000	j LJ010000	z LZ010000	╖ SF240000	╨ SF400000	Ş LS420000	ş LS410000	Ę LE440000	Ú LU120000	ę LE430000	ú LU110000
-B	♂ SM280000	← SM300000	+ SA010000	; SP140000	K LK020000	[SM060000	k LK010000	{ SM110000	╕ SF250000	╤ SF410000	Ť LT220000	ť LT210000	Ë LE180000	Ű LU260000	ë LE170000	ű LU250000
-C	♀ SM290000	∟ SA420000	, SP080000	< SA030000	L LL020000	\ SM070000	l LL010000	| SM130000	╜ SF260000	╞ SF420000	Ź LZ120000	ź LZ110000	Ě LE220000	Ü LU180000	ě LE210000	ü LU170000
-D	♪ SM030000	↔ SM780000	- SP100000	= SA040000	M LM020000	] SM080000	m LM010000	} SM140000	¢ SC040000	═ SF430000	(SHY) SP320000	″ SD250000	Í LI120000	Ý LY120000	í LI110000	ý LY110000
-E	♫ SM910000	▲ SM600000	. SP110000	> SA050000	N LN020000	^ SD150000	n LN010000	~ SD190000	¥ SC050000	╬ SF440000	Ž LZ220000	ž LZ210000	Î LI160000	Ţ LT420000	î LI150000	ţ LT410000
-F	☼ SM690000	▼ SV040000	/ SP120000	? SP150000	O LO020000	_ SP090000	o LO010000		╗ SF030000	® SM530000	Ż LZ300000	ż LZ290000	Ď LD220000	ß LS610000	ď LD210000	˙ SD290000

Figure 10-2 Code page ISO Latin-2

The trick to Unicode is that there is only one code page that comprises the characters of all languages. Especially when exchanging data between different business partners with different IT systems and perhaps different languages, there may be trouble concerning the accuracy of the data if you are not using Unicode. These problems do not appear in a pure Unicode system landscape.

For more information about the use of languages and code pages in SAP applications, see *SAP Note 73606*. The note also contains attached files with useful information about the relationship of languages and code pages.

For more information about Unicode conversions of SAP products, see *SAP Note 548016*. This note contains references to other SAP Notes that are related to Unicode.

The Unicode character notation is similar to the concept described above. But with over 100,000 characters defined, the tables are divided into Unicode character code charts that group the characters by scripts. All the defined characters and their notations can be found at:

http://www.unicode.org/charts

A Unicode character is represented by the following form:

U+<Code>

This is also called a code point, because it describes a certain *point* (character) in the Unicode code page.

The Unicode standard range that is relevant for SAP systems spans characters from *U+0000 to U+FFFF* and addresses 65,535 characters.

Table 10-1 shows a few Unicode characters, codes, and code charts.

Table 10-1 Unicode characters, codes, and code charts

Character	Unicode-code	Character code chart
a	U+0061	Latin-1
ä	U+00E4	Latin-1
Б	U+0411	Cyrillic
ش	U+0634	Arabic
ঋ	U+0987	Bengali

10.3 Unicode character encoding

There are different options to encode the Unicode character set. SAP systems mainly use two encoding schemes or Unicode transformation formats (UTF):

- UTF-8
- UTF-16

The different encoding schemes have different effects. For example, UTF-8 inherits variables length of the stored data while the actual UTF-16 representation depends on the Endianess of the operating system.

While UTF-16 is used in the SAP application server, SAP databases' management systems use either UTF-8 (or derivatives) or UTF-16. IBM DB2 for Linux, UNIX, and Windows uses UTF-8.

Both encoding schemes comprise the same Unicode character set and are therefore equivalent. A transformation between these schemes can be done by an algorithm without using any conversion tables. Therefore, this transformation is very fast.

We now investigate how UTF-16 and UTF-8 encode a Unicode character code in the range that is used by SAP.

UTF-16 has a very straightforward implementation for encoding, using exactly one byte pair for the character range used by SAP (U+0000 to U+FFFF). Table 10-2 shows the bit representation for UTF-16.

Table 10-2 UTF-16 bit distribution (one byte pair)

Unicode character code (bit-representation)	UTF-16 (big endian bit representation)	UTF-16 (little endian bit representation)
yyyyyyyy xxxxxxxx	yyyyyyyy xxxxxxxx	xxxxxxxx yyyyyyyy

UTF-8 has an encoding scheme with a variable length. For the character range used by SAP, this results in a length of one to three bytes. Table 10-3 illustrates the encoding.

Table 10-3 UTF-8 bit distribution without surrogate characters

Unicode character code (bit-representation)	1 byte	2 byte	3 byte
00000000 0xxxxxxx (U+0000 - U+007F)	0xxxxxxx		
00000yyy yyxxxxxx (U+0080 - U+07FF)	110yyyyy	10xxxxxx	
zzzzyyyy yyxxxxxx (U+0800 - U+FFFF)	1110zzzz	10yyyyyy	10xxxxxx

If we combine the information from Table 10-1 on page 230 with the encoding schemes shown in Table 10-2 on page 231 and Table 10-3 on page 231, we get the representation of characters versus encoding schemes shown in Table 10-4.

Table 10-4 Unicode-code transformation to encoding schemes UTF-16/UTF-8

Character	Unicode-code	UTF-16 (Big Endian)	UTF-16 (Little Endian)	UTF-8
A	U+0061	00 61	61 00	61
Ä	U+00E4	00 E4	E4 00	C3 A4
Б	U+0411	04 11	11 04	D0 91
ش	U+0634	06 34	34 06	D8 B4
ঋ	U+0987	09 87	87 09	E0 A6 87

Table 10-5 summarizes the encoding scheme attributes and the usage by the different databases.

Table 10-5 Unicode encoding: Usage by databases

Encoding	Description/attributes	Used by database
CESU-8	Similar to UTF-8	Oracle® MAX DB (8.0)
UTF-8	Variable length: 1 character = 1–4 bytes Platform-independent byte order (Endian) Used for XML	DB2 for LUW (DB6)
UTF-16	Fixed length: 1 character = 2 bytes Platform-dependent byte order (Little/Big Endian)	DB2/400 (DB4) SAP DB 7.0 SQL Server®

For the DB2 database the following is valid:

- SQL input parameters that have Unicode data type are converted from UTF-16 to UTF-8 by the database client library (that is, on the application server) before they are sent to the database server.
- Data that is fetched from the database is converted from UTF-8 to UTF-16 after being received by the application server.
- Data that is loaded by R3load into the database is sent in UTF-16 format and converted to UTF-8 on the database server.
- The collating sequence IDENTITY_16BIT on the DB server ensures that database and SAP application servers collate data the same way despite the different Unicode representation.

For more information see *SAP Note 991481*.

10.4 Big and little endian

The conversion of the database to Unicode is performed by the system copy tool R3load during the export phase. During this phase, the target code page must be defined, which is the Unicode code page. Due to the endianness (little/big endian) of the target system, it can be 4102 or 4103.

For more information about the endianess of target systems, see *SAP Note 552464*.

Table 10-6 provides an overview of the code pages required for different types of operating systems.

Table 10-6 SAP Unicode code page, endianness, and processor architecture

SAP Unicode code page	Processor architecture (operating system)
4103 (little endian)	Alpha Intel® X86 and clones X86_64 Itanium® (Windows and Linux) Solaris_X86_64
4102 (big endian)	IBM 390 AS/400® PowerPC® (AIX + Linux on Power) SPARC (Solaris) PA-RISC (HP-UX) Itanium (HP-UX)

If you are using multiple servers for export or import by applying the distribution monitor (see 3.5, “Distribution monitor: DistMon” on page 30), the code page must match the endianness of each server where import processes are started. Using different processor architectures might especially lead to a situation where the target code page is different for different servers.

During the import, the data is converted to the database code page.

10.5 Hardware requirements

When converting a system to Unicode, there are additional hardware requirements regarding CPU, memory, and storage. Table 10-7 lists the additional hardware requirements for Unicode. The data is based on *SAP note 1139642 Hardware Requirements in Unicode Systems* and our own observations.

Table 10-7 Additional hardware requirements for Unicode

Memory	+40...50%. Application servers are using UTF-16.
CPU	+10...30%, depending on transaction mix (MDMP, single code page, single byte, or double byte). Also depending on processor type
Storage (database)	UTF-8 / CESU-8: -10% … +10%. UTF-16: +20% .. +60%.
Network	No significant change.

Shrinking of the required storage may appear because of database reorg effects when using code pages that encode to mostly 1-byte Unicode characters. In this case the growth related to the Unicode conversion is very small. The reorg effects can lead to a database that is smaller on the target side.

A more detailed discussion about this topic can be found in the following link:

http://service.sap.com/unicode@sap

The values in Table 10-7 are only average values. For an individual customer installation, performance tests with appropriate monitoring might be required to learn more about the hardware requirements.

10.6 SAP tables, languages, code pages

The main point in a Unicode conversion is the knowledge of the right code page of every character string that is stored inside an SAP-owned database table.

If a system is running on a single code page system, this is not very complex. But as soon as a system is based on more than one code page, the assignment of the correct code page to every character string can become very difficult. Analyzing such systems might be complex and time consuming.

In this section we discuss how language information is stored in SAP tables.

Language information

Generally, you can divide SAP tables into the following groups with respect to language information:

- Language-dependent tables
- Language-independent tables

Language-dependent tables have a field that contains a language-key (for example, SPRAS). Figure 10-3 shows that table EKKO has a field SPRAS, which represents a language key.

Transp. Table EKKO Active
Short Text Purchasing Document Header

Attributes | Delivery and Maintenance | Fields | Entry help/check | Currency/Quantity Fields

Srch Help | Predefined Type | 1 / 78

Field	Key	Initi...	Data element	Data T...	Length	Deci...	Short Text
MANDT	☑	☑	MANDT	CLNT	3	0	Client
EBELN	☑	☑	EBELN	CHAR	10	0	Purchasing Document Number
.INCLUDE	☐	☐	EKK00ATA	STRU	0	0	Purchasing Document Header: Data Part
BUKRS	☐	☐	BUKRS	CHAR	4	0	Company Code
BSTYP	☐	☐	EBSTYP	CHAR	1	0	Purchasing Document Category
BSART	☐	☐	ESART	CHAR	4	0	Purchasing Document Type
BSAKZ	☐	☐	BSAKZ	CHAR	1	0	Control indicator for purchasing document type
LOEKZ	☐	☐	ELOEK	CHAR	1	0	Deletion indicator in purchasing document
STATU	☐	☐	ESTAK	CHAR	1	0	Status of purchasing document
AEDAT	☐	☐	ERDAT	DATS	8	0	Date on which the record was created
ERNAM	☐	☐	ERNAM	CHAR	12	0	Name of Person who Created the Object
PINCR	☐	☐	PINCR	NUMC	5	0	Item number interval
LPONR	☐	☐	LPONR	NUMC	5	0	Last item number
LIFNR	☐	☐	ELIFN	CHAR	10	0	Vendor's account number
SPRAS	☐	☐	SPRAS	LANG	1	0	Language Key
ZTERM	☐	☐	DZTERM	CHAR	4	0	Terms of payment key
ZBD1T	☐	☐	DZBDET	DEC	3	0	Cash (prompt payment) discount days
ZBD2T	☐	☐	DZBDET	DEC	3	0	Cash (prompt payment) discount days
ZBD3T	☐	☐	DZBDET	DEC	3	0	Cash (prompt payment) discount days
ZBD1P	☐	☐	DZBD1P	DEC	5	3	Cash discount percentage 1
ZBD2P	☐	☐	DZBD2P	DEC	5	3	Cash Discount Percentage 2
EKORG	☐	☐	EKORG	CHAR	4	0	Purchasing Organization
EKGRP	☐	☐	BKGRP	CHAR	3	0	Purchasing group

Figure 10-3 Table EKKO with language key SPRAS

The data type LANG marks a field that is acting as a language key. If we look into an SAP ERP 2004 IDES system, we find the information listed in Table 10-8.

Table 10-8 Language key in SAP tables

Number of tables owned by SAP	47,303
Tables with language key	22,628 (48%)
Tables without language key	24,675 (52%)
Different field names for language key in SAP-tables	700

Most used names	Fieldname	Used
	LANGU	7442
	SPRAS	7308
	SPRSL	1670
	LANGUAGE	940
	DDLANGUAGE	364
	TXTSP	298
	TDSPRAS	278
	SPRACHE	272
	MASTERLANG	268
	LANGU_CREA	218
	LANG	203
	LTXSP	160
	MASTER_LG	100

For tables that contain a language key, the relationship between language and code page is defined. Therefore, you can easily identify the correct code page information. These tables are *no problem candidates* during a Unicode conversion.

Tables that have no language key can be divided into two groups:

- Tables that do not have any text in local languages (for example, because they are for system internal use and are not changed by the customer).
- Tables that do have texts in languages that belong to different code pages. These are the *real problem candidates* that make the most effort in a Unicode conversion.

This is because you must assign a code page to these strings by automatic, semi-automatic, or manual methods. With these assignments a vocabulary is built up, which is used by R3load during the Unicode conversion step (normally when exporting the database).

11

Setup for a Unicode migration

This chapter contains a description of a customer Unicode conversion project, including the enablement of DB2 row compression. We carefully selected an example that contains many concepts described in this book. The setup is a good example for the overall goal of balancing all available resources by using the appropriate DB2 optimizations and the correct distribution of the components to different servers. The migration was done using the migration monitor. This means that the SAPInst tool was not used to run the export and import automatically, but paused during this phase.

11.1 Architecture overview

This Unicode migration setup uses three servers and four MigMon instances. Two hosts are used for the database export. They both use local storage to dump the export data to disk. Export packages are distributed among two groups. One group (host B) comprises the big tables. The other group (host A) comprises all remaining tables (rest tables). The big tables were split out by the Package Splitter into separate packages. Each export is controlled by a MigMon instance (MigMon A on host A and MigMon B on bost B) with its own configuration. Figure 11-1 shows an architectural overview of the setup.

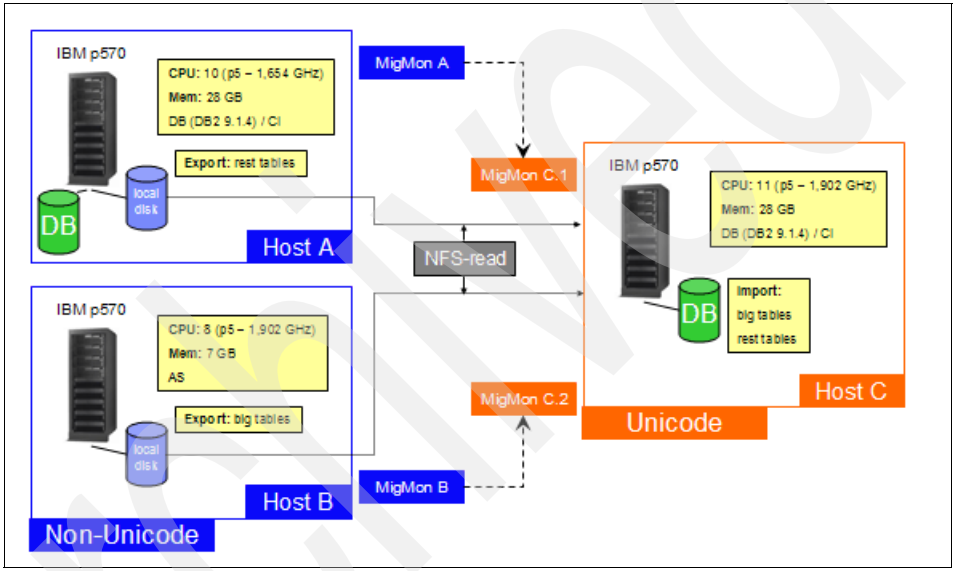


Figure 11-1 Unicode migration setup

On the import side there is one host (host C). The export disks of host A and host B are mounted using NFS (for reading) on host C. The import is controlled by two MigMon instances (MigMon C.1 and MigMon C.2).

The group *big tables* on host B contains a subset of tables that is exported with the unsorted unload option. During the import the tables from the group *big tables* are compressed during the load process.

11.2 MigMon configuration for host A

Host A is used for the export of the group named *rest tables*. Figure 11-2 shows the export_monitor_cmd.properties file for MigMon A.

```
server
net
exportDirs=/data
installDir=/install_E04/migmon_rest_logs
orderBy=ORDERBY.TXT
ddlFile=/data/DB/DDLDB6.TPL
monitorTimeout=30
r3loadExe=/install_E04/migmon_rest/R3load
tskFiles=yes
extFiles=no
dataCodepage=4102
loadArgs=-stop_on_error
jobNum=2
netExchangeDir=/data/exchange
trace=all
```

Figure 11-2 export_monitor_cmd.properties for MigMon A on host A

The most interesting parameters for this setup are highlighted in bold and italic fonts and are:

- orderBy

  The `ORDERBY.TXT` file contains a list of package names that determines the sequence of the export.

- ddlFile

  This parameter states the path to the standard DDL template file that should be used for the export.

- dataCodepage

  Depending on the endianness of the target hardware, choose either 4102 (big endian) or 4103 (little endian).

The parameter r3loadExe states the path to the executable that is taken when invoking R3load. Using this parameter you can use a R3load executable different from the one in the kernel directory. The parameter NetExchangeDir defines the path to the MigMon signal files (*.SGN), which are written when the export is finished successfully and gives the starting point to the importing process.

The orderBy file `ORDERBY.TXT` is listed in Figure 11-3.

```
[REST_TABLES]
jobNum=8
0038_REPOSRC
0039_REPOTEXT
0040_DYNPSOURCE
0041_INDTEXT
0042_BSAS
0043_AUFM
0044_EKBE
0045_BSAD
0046_COBK
0047_S999
0048_CDHDR
0049_CKMI1
0050_COEPB
0051_RESB
0052_NAST
0053_ZBCWK2
0054_ZCORMM1292_SUI_1
0055_CE4WWOC
0056_S501
0057_VBAK
0058_VBKD
0059_ZPCOCOST
0060_SWELOG
0061_ZDS027
0062_JEST
0063_CMFP
0064_MKPF
0065_VFKN
0066_BSE_CLR
0067_LIKP
0068_VRKPA
0069_ACCTCR
0070_AFRU
0071_LSHR
0072_LTAP
0073_VBRK
0074_AFWI
0075_AUSP
0076_EKPO
0077_S027
0078_S905
0079_S903
0080_CIFBALSEL
0081_VFSI
0082_SWELTS
0083_SOOD
0084_M_MEBE
0085_SOCS
0086_VAKPA
0087_TPALOG
0088_ZDS028
0089_ESLL
0090_ZMSEGGP1_LOG
0091_MCKALKW
0092_ZBCWK1
0093_EDIDS
0094_GLIDXA
0095_VBUK
0096_S026
0097_SOOS
0098_MONI
0099_COEPL
0100_DOKCLU
0101_BSAK
0102_VBSS
0103_S034
0104_RSEG
```

Figure 11-3 orderBy file for MigMon A on host A

Besides the list of packages there is also a parameter jobNum that defines the number of parallel R3load processes to use. Together with the jobNum parameter of the export_monitor_cmd.properties file, it defines in sum 10 parallel R3load processes for MigMon A on host A.

To start MigMon the script export_monitor.sh is used. To use a different dbdb6slib.o file (fitting the R3load) and to incorporate Unicode conversion-specific environment variables, this script can be adapted, as shown in Figure 11-4.

```
#!/bin/sh
#
# Export Monitor startup on UNIX
#
export JAVA_HOME=/usr/java14_64
export DIR_LIBRARY=/install_E04/migmon_rest
export I18N_NAMETAB_TIMESTAMPS=IGNORE
# Make sure prerequisite dbms_type environment variable is set
if [ -z "$dbms_type" ]; then
  echo "The dbms_type environment variable is not defined."
  echo "This environment variable is needed to run this program."
  exit 1
fi
.
.
.
set -x
"$JAVA_HOME"/bin/java -showversion -Xmx1024m -cp
    "$MIG_HOME/..:$MIG_HOME/migmon.jar:$MIG_HOME/activation.jar:$MIG_HOME/mail.jar" com.sap.inst.lib.app.SecureStartup "-
    Xmx1024m" com.sap.inst.migmon.exp.ExportMonitor -dbType $dbms_type "$@"
```

location of dbdb6slib

necessary for unicode-conversion

Figure 11-4 Adapted export_monitor.sh for MigMon A on host A

If you also put in the environment variable `JAVA_HOME` you do not have to set it in your command-line session.

11.3 MigMon configuration for host B

Host B is working as an application server for the source system. It is used to export the tables in the group big tables. These are the 37 biggest tables. The first 14 packages are exported with the unsorted unload feature. All 37 tables are compressed during the import.

Figure 11-5 shows the export_monitor_cmd.properties file for MigMon B.

```
server
net
exportDirs=/data
installDir=/install_E04/migmon_biggest_logs
ddlFile=/data/DB/DDLDB6.TPL
ddlMap=DDLMAP.TXT
orderBy=ORDER.TXT
monitorTimeout=30
r3loadExe=/install_E04/migmon_biggest/R3load
tskFiles=yes
extFiles=no
dataCodepage=4102
loadArgs=-stop_on_error
jobNum=1
netExchangeDir=/data/exchange
trace=all
```

Figure 11-5 export_monitor_cmd.properties file for MigMon B on host B

Since there are packages (tables) that are exported unsorted while others are exported sorted, you must define two different DDL template files. This request is fulfilled by the use of a `ddlMap` file, as shown in Figure 11-6.

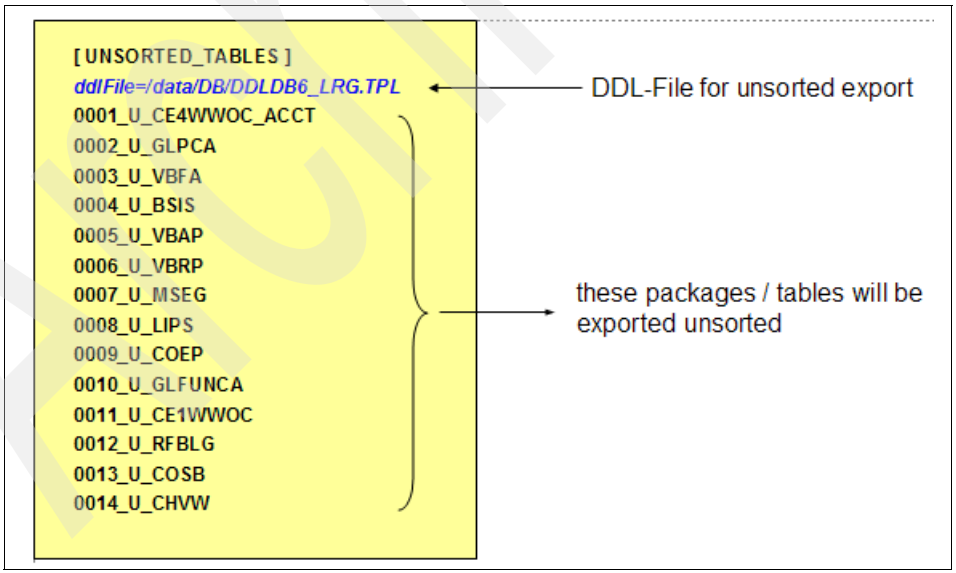


Figure 11-6 ddlMap file for unsorted export

In the `ddlMap` file you can map packages to a specific DDL template file. The `DDLDB6_LRG.TPL` file is missing the `ORDER_BY_PKEY` keyword and therefore uses the unsorted unload feature.

All other packages handled by MigMon B use the standard Data Definition Language (DDL) file from the `export_monitor_cmd.properties` file shown in Figure 11-5 on page 242.

The order of the export is defined by the orderBy file `ORDER.TXT`, as shown in Figure 11-7. The package names are listed in a two-column fashion only for display purposes. In fact, it is a file with one package name on each row.

```
[FIRST]
jobNum=17
0001_U_CE4VWVOC_ACCT
0002_U_GLPCA
0003_U_VBFA
0004_U_BSIS
0005_U_VBAP
0006_U_VBRP
0007_U_MSEG
0008_U_LIPS
0009_U_COEP
0010_U_GLFUNCA
0011_U_CE1VWVOC
0012_U_RFBLG
0013_U_COSB
0014_U_CHVW
0015_VBEP
0016_VFKONV
0017_VBUP
0018_COSP
0019_VBPA
0020_VAPMA
0021_CDCLS
0022_CE3VWVOC
0023_BSIM
0024_SRRELROLES
0025_VRPMA
0026_KOCLU
0027_IDOCREL
0028_STXH
0029_BKPF
0030_EDI40
0031_MARI
0032_E071K
0033_VLPMA
0034_STXL
0035_COSS
0036_VBFS
0037_ZMSEG
```

Figure 11-7 orderBy file for MigMon B on host B

The parameter jobNum defines that there will be 17 parallel R3load jobs. Together with the jobNum of the `export_monitor_cmd.properties` file, which is 1, there will be 18 R3load jobs in total on host B.

> **Note:** Do not set the jobNum parameter to 0. This does not mean that no load process is started. In fact, an R3load process for each package defined is started at once.

11.4 MigMon configuration for host C

Host C serves as database and central instance for the target system. It is the only import host. To import the two groups from host A and host B, two MigMon

instances are configured on host C (MigMon C.1 and MigMon C.2). As shown in Figure 11-1 on page 238, MigMon C.2 interacts with the exporting MigMon B and handles the packages that contain the large tables. MigMon C.1 is coupled to MigMon A and handles the remaining packages.

11.4.1 Configuration for MigMon C.1

The file system for the export of host A is mounted using NFS to the mount point `/exportdb_hostA` on host C. The `import_monitor_cmd.properties` file for MigMon C.1 looks similar to the one shown in Figure 11-8.

The packages are imported with the standard DDL file `DDLDB6.TPL`. Instead of the `dataCodepage` (like on the export part), the `dbCodepage` must be set to the Unicode code page. The loadArgs are set so that the DB2 Fast LOAD API is used whenever applicable. There will be 15 parallel R3load jobs for this import group (jobNum).

```
importDirs=/exportdb_hostA
installDir=/install_E04/migmon_rest_logs
orderBy=name
ddlFile=/exportdb_hostA/DB/DDLDB6.TPL
monitorTimeout=30
r3loadExe=/install_E04/migmon_rest/R3load
tskFiles=yes
extFiles=no
dbCodepage=4102
loadArgs=-stop_on_error -loadprocedure fast LOAD -nolog -c 300
jobNum=15
exchangeDir=/exportdb_hostA/exchange
trace=all
```

Figure 11-8 import_monitor_cmd.properties file for MigMon C.1 on host C

11.4.2 Configuration for MigMon C.2

The MigMon C.2 imports the packages that were exported by MigMon B on host B. This is the group of the 37 biggest tables. They are compressed during the import. The appropriate `import_monitor_cmd.properties` file looks similar to the one shown in Figure 11-9.

```
importDirs=/exportdb_hostB
installDir=/install_E04/migmon_biggest_logs
orderBy=name
ddlFile=/exportdb_hostB/DB/DDLDB6_COMPRESSED.TPL
monitorTimeout=30
r3loadExe=/install_E04/migmon_biggest/R3load
tskFiles=yes
extFiles=no
dbCodepage=4102
loadArgs=-stop_on_error -loadprocedure fast LOADCOMPRESS -nolog -c 300
jobNum=15
exchangeDir=/exportdb_hostB/exchange
trace=all
```

Figure 11-9 import_monitor_cmd.properties file for MigMon C.2 on host C

Since all the packages are handled in the same way, there is no need for multiple DDL file templates. The template file used (`DDLDB6_COMPRESSED.TPL`) is a copy of the standard DDL file with one modification. There has been added a `COMPRESS YES` to the `CREATE TABLE` template.

The file system for the export of host B is mounted using NFS to the mount point `/exportdb_hostB` on host C.

R3load is invoked with the loadprocedure fast LOADCOMPRESS parameter. This parameter causes R3load to create a compression dictionary and to compress the imported data during LOAD.

Figure 11-10 shows the script (import_monitor.sh) that is used to invoke MigMon C.2.

```
#!/bin/sh

#
# Import Monitor startup on UNIX
#

export JAVA_HOME=/usr/java14_64
export DB6LOAD_COMPRESSION_THRESHOLD=200000
export DIR_LIBRARY=/install_E04/migmon_biggest
.
.
.
# Set default value for MIG_HOME environment variable
if [ -z "$MIG_HOME" ]; then
  MIG_HOME=.
fi

set -x

"$JAVA_HOME"/bin/java -showversion -Xmx1024m -cp
     "$MIG_HOME/.:$MIG_HOME/migmon.jar:$MIG_HOME/activation.jar:$MIG_HOME/mail.jar" com.sap.inst.lib.app.SecureStartup "-
     Xmx1024m" com.sap.inst.migmon.imp.ImportMonitor -dbType $dbms_type "$@"
```

number or rows to build up compression dictionary

location of dbdb6slib

Figure 11-10 Script import_monitor.sh for MigMon C.2 on host C

The changes that were made to the standard script are shown in blue. The environment parameter `DB6LOAD_COMPRESSION_THRESHOLD` is used by R3load to define the number of rows that should be used to build up the compression dictionary by REORG. This value should be about 1% of the numbers of rows of the biggest table to be imported.

The environment variable DIR_LIBRARY is pointing to the path where the dbdb6slib.o that should be used is located.

11.5 Results

Using this setup, the results shown in Table 11-1 have been reached.

Table 11-1 Achieved results of sample setup

DB size (before compression)	~ 1 TB
Export/import time	11.5 hrs
DB storage savings	50 %
DB size (after compression)	~ 500 GB

Looking into the details, the typical tables determined the overall runtime of the migration. An interesting result is the runtime of the table RFBLG that took over

11 hours to export and only 15 minutes to import, so this table would be a good candidate for table splitting.

11.6 Time join diagram for MigMon B + C.2

This part contains the large tables that determine the overall runtime of the conversion. The typical tables like EDI40, CDCLS, or KOCLU are running very long. Further optimization for these tables is not necessary, as the table RFBLG was running during the entire migration. Due to the setup, the creation of the views is started prior to the end of the import of the table RFBLG. Using such a setup can be critical, and you must check whether the creation of all views has been done successfully. Otherwise, you must repeat the execution of package SAPVIEW.

Figure 11-11 shows a part of the time join diagram for MigMon B + C.2.

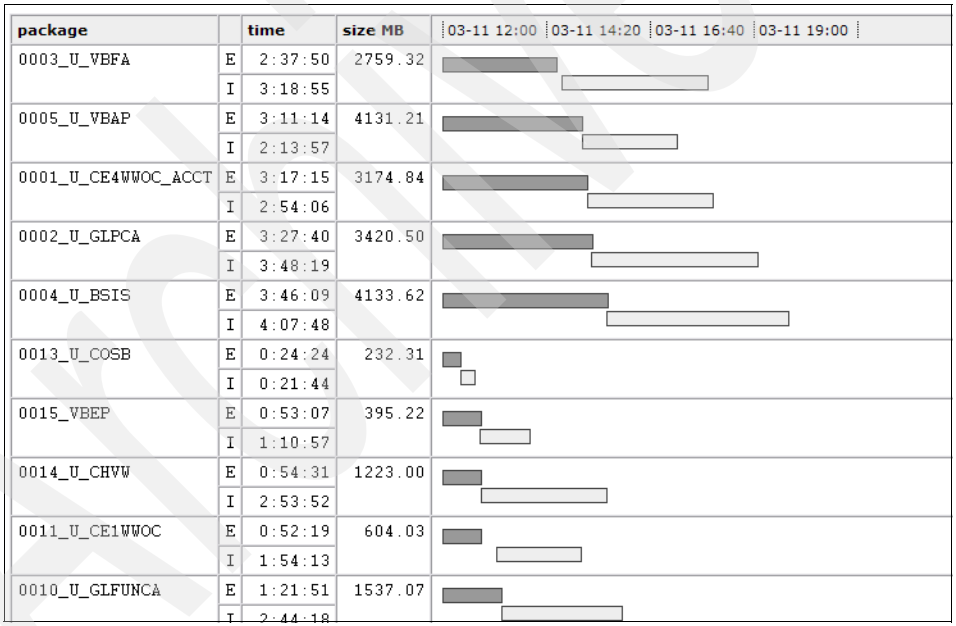


Figure 11-11 Time join diagram for MigMon B + C.2

11.7 Time join diagram for MigMon A + C.1

Figure 11-12 shows a part of the time join diagram for MigMon A + C.1. It contains the export and import times for the group *rest tables*, which defined the subset of tables not contained in the group *big tables*.

package		time	size MB
0039_REPOTEXT	E	0:08:53	45.80
	I	0:12:26	
0040_DYNPSOURCE	E	0:20:45	241.15
	I	0:17:07	
0041_INDTEXT	E	0:04:57	11.45
	I	0:06:21	
0043_AUFM	E	0:19:17	415.29
	I	0:20:24	
0044_EKBE	E	0:23:18	320.71
	I	0:18:27	
0042_BSAS	E	0:27:40	348.67
	I	0:16:54	
0045_BSAD	E	0:31:00	301.90
	I	0:13:28	
0038_REPOSRC	E	3:09:22	3503.40
	I	0:58:16	
SAPAPPL0_002	E	0:10:35	152.52
	I	0:07:30	
SAPAPPL0_001	E	0:01:01	4.00
	I	0:02:23	

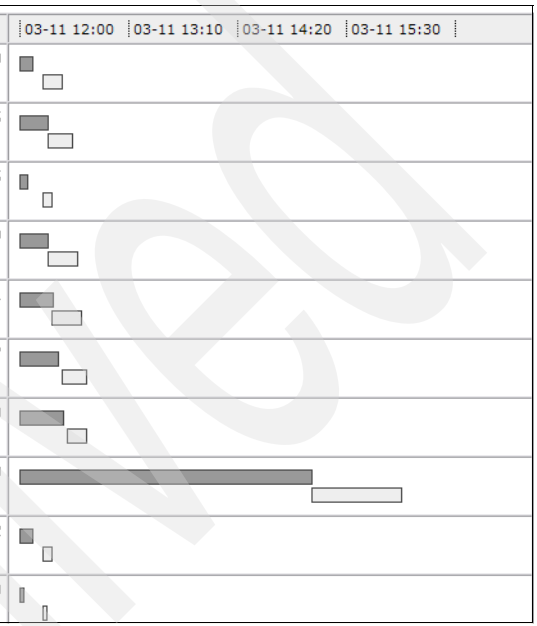


Figure 11-12 Time join diagram for MigMon A + C.1

11.8 Compression rates

Figure 11-13 shows the compression rate per table that was achieved during the import.

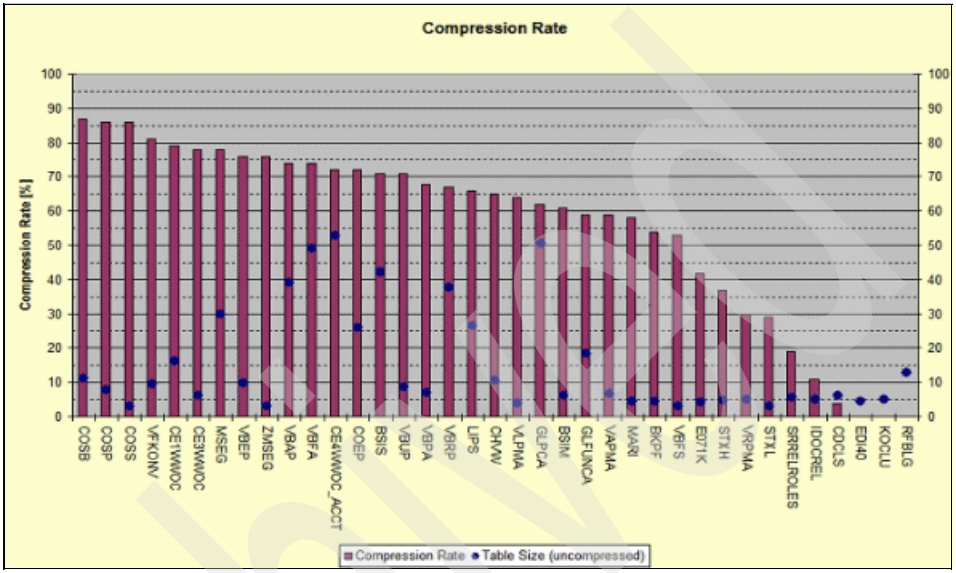


Figure 11-13 Compression rate and size of uncompressed tables

The tables are sorted by descending compression rate. Additionally, the uncompressed size of each table is displayed. The cloud of points that show the table size has a rather random distribution. This points to the fact that there is only a weak correlation between the compression rate and the size of a table. The compression rate depends much more on the structure of a table. For example, the CO-related tables all have structures that support a high compression rate. Since these are often very large tables in customer installations that make intensive use of the controlling features of SAP, those customers especially have the opportunity to save an extraordinarily large amount of storage.

The last four tables are table clusters. The compression rate for this type of table is usually less optimal.

A

Background information

This appendix provides additional background information about the topics discussed in this book.

Test environment

We used different hardware environments for the tests. We performed our tests on two dedicated test environments, which are described in detail in this section. Other tests were performed in real customer environments or data is derived from real migrations.

Be aware that the test results are heavily influenced by the underlying hardware. The capacity of the different hardware components such as CPU, memory, storage, and network is an important factor that influences performance. If the components are not well balanced this can also affect performance. For example, an insufficient amount of spindles or storage adapters can reduce I/O bandwidth. Therefore, the tests can only provide an indication how certain techniques and selected options affect performance. They show a trend, but you may see different effects in your own hardware environment.

IBM System p with network-attached storage

This hardware environment comprises the following components:

- Server: System p, Model 9133-55A, 8 way, PowerPC_POWER5, 1648 MHz, 64 GB memory, AIX 5.3
- Storage: NetApp® Model 3700
- Gigabit switch

For the tests, we used one LPAR (DLPAR1) in uncapped CPU sharing mode. Therefore, all eight CPUs were available during the tests.

The storage system was connected to the server using a switch. Two Gigabit Ethernet Adapters were used on the storage side and the server side to connect to the switch. The table spaces of the databases that were used for the export and import of the data were striped across four LUNs, as follows:

- /vol/vol5/lun0 Lun0 Adapter en0 156.268 GB online AIXGroup: 0
- /vol/vol5/lun1 Lun1 Adapter en1 156.268 GB online AIXGroup: 1
- /vol/vol5/lun2 Lun2 Adapter en0 156.268 GB online AIXGroup: 2
- /vol/vol5/lun3 Lun3 Adapter en1 156.268 GB online AIXGroup: 3

During the tests we monitored the CPU utilization on both the IBM System p server and the NetApp storage to check for any hardware bottlenecks that could potentially affect the scalability tests. We also monitored the disk utilization of the NetApp storage and data transfer rate of the two Ethernet adapters of the System p server. Both adapters achieved roughly identical transfer rates during the export/import tests.

Figure A-1 illustrates the System p environment.

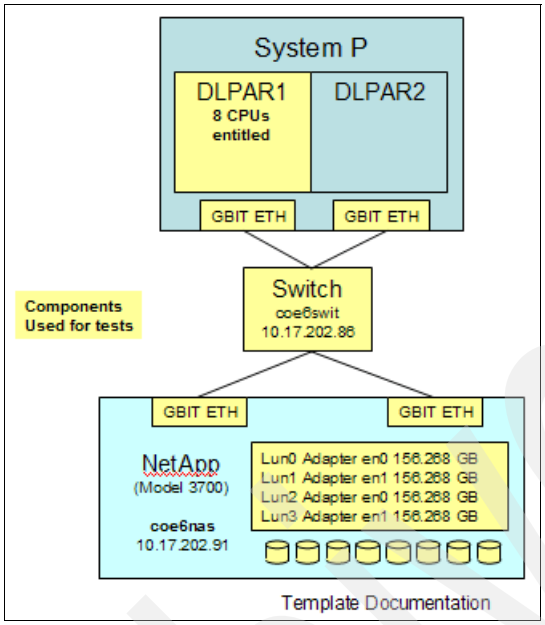


Figure A-1 IBM System p with network attached storage

IBM System x with local attached storage

This hardware environment comprises the following components:

- Server: System x®, Model 3400, 2 x Quad Core Intel Xeon® Processor E5430, 2,66Ghz, 8 GB memory
- Storage: local S-ATA Disks, 200 GB, 7200 RPM
- Novell® SLES 10 operating system, kernel 2.6.16.27-0.9-smp

The disk setup for the System x machine was straightforward. One S-ATA disk was used for swap, the root file system, and hosting the export directories. The database was installed on a RAID0 array built out of 3 x 200 GB S-ATA disks attached to the internal IBM ServeRAID controller. The disk partitioning layout was the following:

- /dev/sda1 /
- /dev/sda1 swap
- /dev/sdb1 /db2

During the tests we monitored the CPU utilization on the server using NMON for Linux to check for any hardware bottlenecks that could potentially effect the

scalability tests. We also monitored the disk utilization and transfer rates on the internal disks. This system mainly was used to make investigations during the import performance tests having a disk bottleneck. This gives a better indication of the tuning effects of DB2 and R3load as the goal mainly is to minimize disk I/O by avoiding any DB2 heap overflows.

For the comparison between local import and import using an application server, the AIX LPAR described above was used as the application server to import with R3load. On the Linux database server, two databases have been installed:

- The BN6 database, which was a non-Unicode database
- The BN7 database, which was an identical copy but created as a Unicode database

The DB2 version used for the import tests was DB2 V9.5.

Figure 11-14 illustrates the System x environment.

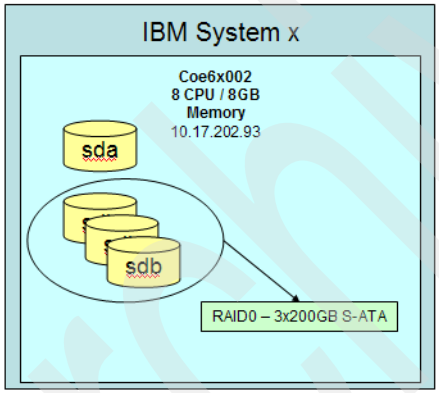


Figure 11-14 BM System x with local attached storage

DB2 row compression based on R3load 6.40

This section provides additional information about the compression feature discussed in 7.18.4, “DB2 row compression based on R3load 6.40” on page 157.

Script for compressed import for a single table

You can use the script shown in Example A-1 (using a c-shell) to do a compression of a table during import of data by R3load. The sample size of the number of rows used for the creation of the compression dictionary is maintained in the script, so it can be handled individually for every table.

Example: A-1 Script for the import of table <table-name> with a 1% sample

```
setenv TMPDIR <install-directory>
setenv DB6LOAD_COMPRESSION_THRESHOLD <1% sample-size>
cd <install-directory>
db2 "drop table <sap-schema>.<table-name>"
db2 "commit"
/usr/sap/<SID>/SYS/exe/run/R3load -i <table-name>.cmd          \
                            -dbcodepage MDMP                   \
                            -k <migration key>                 \
                            -l <table-name>.logc               \
                            -loadprocedure fast LOADCOMPRESS   \
                            -c 10000
mv <table-name>.logc <table-name>.logc_<1% sample-size>
```

The variable parts of the script (*<variable>*) must be substituted by values that fit to your special configuration.

Migration monitor properties file for compressed import

The ability to use the compressed import for a subset of tables needs the use of multiple `DDLDB6.TPL` files. The mapping of the DDL template files to the packages is defined in a mapping file, which is configured in the `import_monitor_cmd.properties` file.

Example A-2 relies on the following conditions:

- The directory where the export is stored is named */exportdir.* Normally, you will find the subdirectories DATA and DB here. If you use a NetWeaver system that supports JAVA-migration you will find the subdirectory ABAP or JAVA, or both.
- The import installation directory is named `/mig_imp`. Here, for example, the cmd-files, TSK files, and log files are stored.
- The mapping of DDL-template files to packages is done in the file `/mig_imp/ddlmap_file`.

Example: A-2 import_monitor_cmd.properties file with DDL file mapping

```
#
# Import Monitor options
#
# Common options
#
# List of import directories, separator on Windows ; on UNIX :
importDirs=/exportdir
# Installation directory
installDir=/mig_imp
# Package order: name | size | file with package names
orderBy=size
# DDL control file, default is DDL<DB_TYPE>.TPL
ddlFile=/exportdir/DB/DDLDB6.TPL
# File with mapping between DDL files and package names
ddlMap=/mig_imp/ddlmap_file
# Monitor timeout in seconds
monitorTimeout=30
#
# R3load options
#
# Optional path of R3load executable
r3loadExe=
# Generation of task files: yes | no
tskFiles=yes
# Inclusion of extent files: yes | no
extFiles=yes
# DB code page for the target database
dbCodepage=1100
# Migration key
migrationKey=<migration key>
# R3load omit value, can contain only 'DTPIV' letters
omit=
# Additional R3load arguments for TASK phase
taskArgs=
# Additional R3load arguments for LOAD phase
loadArgs=-loadprocedure fast LOADCOMPRESS -merge_bck -nolog
# Number of parallel import jobs
jobNum=5
#
# Exchange options
#
# Exchange directory
exchangeDir=
#
# Socket options
#
#socket
```

```
# Local hostname, default is 127.0.0.1
localhost=
# Server port number
port=
#
# Trace option
#
# Trace level
trace=all
#
# E-mail options
#
# SMTP server
mailServer=
# "From" email address
mailFrom=
# "To" email address
mailTo=
```

DDL template mapping file

Example A-3 shows the structure of a DDL mapping file. The packages (in this case, they are all single tables) following the DDL file definition for compressed import will be compressed during the import. All other packages will use the uncompressed import using the standard DDL file from the properties file shown in Example A-6 on page 260.

Example: A-3 DDL mapping file

```
# DDL-mapping file
# normal import
ddlFile = /exportdir/DB/DDLDB6.TPL
# package names
# compressed import
ddlFile = /exportdir/DB/DDLDB6_C.TPL
# package and table names
ACCTIT
ANLC
ANLP
BKPF
BPEJ
BPEP
BSAK
BSAS
BSE_CLR
BSIS
COBK
COEP
FMIFIIT
MSEG
```

DDL file for uncompressed import

In addition to the standard table classes, some customer-defined table classes (beginning with ZZ) and their mapping to the table spaces (data, index, long) are shown in the *loc:* section in Example A-4.

Example: A-4 DDL template file DDLDB6.TPL for uncompressed import

```
drpind: DROP INDEX &ind_name&
crevie: CREATE VIEW &view_name&
( /{ &fld_name& /-, /} )
AS &query&
drpvie: DROP VIEW &view_name&
trcdat: TRUNCATE TABLE &tab_name&
deldat: DELETE FROM &tab_name& &where&
negtab: LICHECK MLICHECK
negdat: LICHECK MLICHECK
negind: LICHECK~O MLICHECK~O LICHECK^O MLICHECK^O
negvie:
# table storage parameters
loc: APPL0 PSAPSTABD PSAPSTABI PSAPSTABD
APPL1 PSAPBTABD PSAPBTABI PSAPBTABD
APPL2 PSAPPOOLD PSAPPOOLI PSAPPOOLD
CLUST PSAPCLUD PSAPCLUI PSAPCLUD
POOL PSAPPOOLD PSAPPOOLI PSAPPOOLD
SDIC PSAPDDICD PSAPDDICI PSAPDDICD
SDOCU PSAPDOCUD PSAPDOCUI PSAPDOCUD
SLDEF PSAPEL620D PSAPEL620I PSAPEL620D
SLEXC PSAPEL620D PSAPEL620I PSAPEL620D
SLOAD PSAPLOADD PSAPLOADI PSAPLOADD
SPROT PSAPPROTD PSAPPROTI PSAPPROTD
SSDEF PSAPES620D PSAPES620I PSAPES620D
SSEXC PSAPES620D PSAPES620I PSAPES620D
SSRC PSAPSOURCED PSAPSOURCEI PSAPSOURCED
TEMP PSAPPROTD PSAPPROTI PSAPPROTD
USER PSAPUSER1D PSAPUSER1I PSAPUSER1D
USER1 PSAPUSER1D PSAPUSER1I PSAPUSER1D
ZZ001 PSAPBTAB2D PSAPBTAB2I PSAPBTAB2D
ZZ002 PSAPBTAB3D PSAPBTAB3I PSAPBTAB3D
ZZ003 PSAPTMP2D PSAPTMP2I PSAPTMP2D
ZZ004 PSAPTMP3D PSAPTMP3I PSAPTMP3D
ZZ005 PSAPFCABPD PSAPFCABPI PSAPFCABPD
ZZ006 PSAPBSAKD PSAPBSAKI PSAPBSAKD
ZZ007 PSAPBSADD PSAPBSADI PSAPBSADD
ZZ008 PSAPBSISD PSAPBSISI PSAPBSISD
ZZ009 PSAPAPQDD PSAPAPQDI PSAPAPQDD
ZZ010 PSAPBKPFD PSAPBKPFI PSAPBKPFD
ZZ011 PSAPZZGLD PSAPZZGLI PSAPZZGLD

# To addapt table space names for server consolidation
# e.g. PSAPBTABD -> <SAPSID>#BTABD
# Use the following parameter:
modify_tbs_names: 1
```

DDL file for compressed import

Example A-5 shows the DDL file for compressed import.

Example: A-5 DDL template file DDLDB6_C.TPL for compressed import

```
prikey: AFTER_LOAD ORDER_BY_PKEY
seckey: AFTER_LOAD
cretab: CREATE TABLE &tab_name&
        ( /{ &fld_name& &fld_desc& /-, /} )
        IN &location&
        INDEX IN &locationI&
        LONG IN &locationL&
    COMPRESS YES ;
        ALTER TABLE &tab_name& VOLATILE
drptab: DROP TABLE &tab_name&
crepky: CREATE UNIQUE INDEX &pri_key&
        ON &tab_name&
        ( /{ &key_fld& /-, /} )
        ALLOW REVERSE SCANS ;
        ALTER TABLE &tab_name&
        ADD CONSTRAINT &pri_key&
        PRIMARY KEY ( &key_fld& /-, )
drppky: ALTER TABLE &tab_name& DROP PRIMARY KEY ;
        DROP INDEX &pri_key&
creind: CREATE &unique& INDEX &ind_name&
        ON &tab_name&
        ( /{ &fld_name& /-, /} )
        ALLOW REVERSE SCANS
drpind: DROP INDEX &ind_name&
crevie: CREATE VIEW &view_name&
        ( /{ &fld_name& /-, /} )
        AS &query&
drpvie: DROP VIEW &view_name&
trcdat: TRUNCATE TABLE &tab_name&
deldat: DELETE FROM &tab_name& &where&
negtab: LICHECK MLICHECK
negdat: LICHECK MLICHECK
negind: LICHECK~O MLICHECK~O LICHECK^O MLICHECK^O
negvie:
# table storage parameters
loc: APPL0 PSAPSTABD                    PSAPSTABI                    PSAPSTABD
     APPL1 PSAPBTABD                    PSAPBTABI                    PSAPBTABD
     APPL2 PSAPPOOLD                    PSAPPOOLI                    PSAPPOOLD
     CLUST PSAPCLUD                     PSAPCLUI                     PSAPCLUD
     POOL  PSAPPOOLD                    PSAPPOOLI                    PSAPPOOLD
     SDIC  PSAPDDICD                    PSAPDDICI                    PSAPDDICD
     SDOCU PSAPDOCUD                    PSAPDOCUI                    PSAPDOCUD
     SLDEF PSAPEL620D                   PSAPEL620I                   PSAPEL620D
     SLEXC PSAPEL620D                   PSAPEL620I                   PSAPEL620D
     SLOAD PSAPLOADD                    PSAPLOADI                    PSAPLOADD
     SPROT PSAPPROTD                    PSAPPROTI                    PSAPPROTD
     SSDEF PSAPES620D                   PSAPES620I                   PSAPES620D
     SSEXC PSAPES620D                   PSAPES620I                   PSAPES620D
     SSRC  PSAPSOURCED                  PSAPSOURCEI                  PSAPSOURCED
     TEMP  PSAPPROTD                    PSAPPROTI                    PSAPPROTD
     USER  PSAPUSER1D                   PSAPUSER1I                   PSAPUSER1D
     USER1 PSAPUSER1D                   PSAPUSER1I                   PSAPUSER1D
     ZZ001 PSAPBTAB2D                   PSAPBTAB2I                   PSAPBTAB2D
     ZZ002 PSAPBTAB3D                   PSAPBTAB3I                   PSAPBTAB3D
     ZZ003 PSAPTMP2D                    PSAPTMP2I                    PSAPTMP2D
     ZZ004 PSAPTMP3D                    PSAPTMP3I                    PSAPTMP3D
     ZZ005 PSAPFCABPD                   PSAPFCABPI                   PSAPFCABPD
     ZZ006 PSAPBSAKD                    PSAPBSAKI                    PSAPBSAKD
     ZZ007 PSAPBSADD                    PSAPBSADI                    PSAPBSADD
```

```
     ZZ008 PSAPBSISD                    PSAPBSISI                    PSAPBSISD
     ZZ009 PSAPAPQDD                    PSAPAPQDI                    PSAPAPQDD
     ZZ010 PSAPBKPFD                    PSAPBKPFI                    PSAPBKPFD
     ZZ011 PSAPZZGLD                    PSAPZZGLI                    PSAPZZGLD

# To addapt table space names for server consolidation
# e.g. PSAPBTABD -> <SAPSID>#BTABD
# Use the following parameter:
modify_tbs_names: 1
```

DB2 row compression based on R3load 7.00

This section provides additional information about the compression feature discussed in 7.18.5, “DB2 row compression based on R3load 7.00” on page 163.

Migration monitor: import_monitor_cmd.properties

In this section you can find the import monitor properties files used for our compression tests.

Uncompressed load

Example A-6 shows the import monitor properties file for uncompressed load and subsequent compression triggered by R3load.

Example: A-6 import monitor properties file for uncompressed load

```
#
# Import Monitor options
#
#
# Common options
#
# List of import directories, separator on Windows ; on UNIX :
importDirs=/export/hjm3/nuc_sort/EXP
# Installation directory
installDir=/import/hjm3/nuc_sort/
# Package order: name | size | file with package names
orderBy=size
# DDL control file, default is DDL<DB_TYPE>.TPL
ddlFile=/export/hjm3/nuc_sort/EXP/DDLDB6.TPL
# File with mapping between DDL files and package names
ddlMap=
# Monitor timeout in seconds
monitorTimeout=30
#
# R3load options
#
# Optional path of R3load executable
r3loadExe=
# Generation of task files: yes | no
tskFiles=yes
# Inclusion of extent files: yes | no
extFiles=no
# DB code page for the target database
dbCodepage=1100
```

```
# Migration key
migrationKey=
# R3load omit value, can contain only 'DTPIV' letters
omit=
# Additional R3load arguments for TASK phase
taskArgs=
# Additional R3load arguments for LOAD phase
loadArgs=-loadprocedure fast LOAD
# Number of parallel import jobs
jobNum=10
#
# Exchange options
#
# Exchange directory
exchangeDir=/export/hjm3/nuc_sort
#
# Socket options
#
#socket
# Local hostname, default is 127.0.0.1
localhost=
# Server port number
port=
#
# Trace option
#
# Trace level
trace=all
#
# E-mail options
#
# SMTP server
mailServer=
# "From" email address
mailFrom=
# "To" email address
mailTo=
```

Compressed load with automatic dictionary creation

Example A-7 shows the import monitor properties file for compressed load with automatic dictionary creation (ADC).

Example: A-7 Import monitor properties file for compressed load with ADC

```
#
# Import Monitor options
#
#
# Common options
#
# List of import directories, separator on Windows ; on UNIX :
importDirs=/export/hjm3/nuc_sort/EXP
# Installation directory
installDir=/import/hjm3/nuc_sort/
# Package order: name | size | file with package names
orderBy=size
# DDL control file, default is DDL<DB_TYPE>.TPL
ddlFile=/export/hjm3/nuc_sort/EXP/DDLDB6.TPL
# File with mapping between DDL files and package names
ddlMap=
# Monitor timeout in seconds
monitorTimeout=30
#
```

```
# R3load options
#
# Optional path of R3load executable
r3loadExe=
# Generation of task files: yes | no
tskFiles=yes
# Inclusion of extent files: yes | no
extFiles=no
# DB code page for the target database
dbCodepage=1100
# Migration key
migrationKey=
# R3load omit value, can contain only 'DTPIV' letters
omit=
# Additional R3load arguments for TASK phase
taskArgs=
# Additional R3load arguments for LOAD phase
loadArgs=-loadprocedure fast LOAD:COMPRESS_ALL
# Number of parallel import jobs
jobNum=10
#
# Exchange options
#
# Exchange directory
exchangeDir=/export/hjm3/nuc_sort
#
# Socket options
#
#socket
# Local hostname, default is 127.0.0.1
localhost=
# Server port number
port=
#
# Trace option
#
# Trace level
trace=all
#
# E-mail options
#
# SMTP server
mailServer=
# "From" email address
mailFrom=
# "To" email address
mailTo=
```

Compressed load based on R3load sampling

In the following you can find the import monitor properties files used for our tests with sample compression.

Phase 1

Example A-8 shows the phase 1 import monitor properties file for compressed load based on R3load sampling.

Example: A-8 Phase 1 import monitor properties file

```
#
# Import Monitor options
#
#
# Common options
#
# List of import directories, separator on Windows ; on UNIX :
importDirs=/export/hjm3/nuc_sort/EXP
# Installation directory
installDir=/import/hjm3/nuc_sort/
# Package order: name | size | file with package names
orderBy=size
# DDL control file, default is DDL<DB_TYPE>.TPL
ddlFile=/export/hjm3/nuc_sort/EXP/DDLDB6.TPL
# File with mapping between DDL files and package names
ddlMap=
# Monitor timeout in seconds
monitorTimeout=30
#
# R3load options
#
# Optional path of R3load executable
r3loadExe=
# Generation of task files: yes | no
tskFiles=yes
# Inclusion of extent files: yes | no
extFiles=no
# DB code page for the target database
dbCodepage=1100
# Migration key
migrationKey=
# R3load omit value, can contain only 'DTPIV' letters
omit=
# Additional R3load arguments for TASK phase
taskArgs=
# Additional R3load arguments for LOAD phase
loadArgs=-loadprocedure fast LOAD:SAMPLED_FULL_COMPRESS_ALL
# Number of parallel import jobs
jobNum=10
#
# Exchange options
#
# Exchange directory
exchangeDir=/export/hjm3/nuc_sort
#
# Socket options
#
#socket
# Local hostname, default is 127.0.0.1
localhost=
# Server port number
port=
#
# Trace option
#
# Trace level
trace=all
#
# E-mail options
```

```
#
# SMTP server
mailServer=
# "From" email address
mailFrom=
# "To" email address
mailTo=
```

Phase 2

Example A-9 shows the phase 2 import monitor properties file for compressed load based on R3load sampling.

Example: A-9 Phase 2 import monitor properties file

```
#
# Import Monitor options
#
#
# Common options
#
# List of import directories, separator on Windows ; on UNIX :
importDirs=/export/hjm3/nuc_sort/EXP
# Installation directory
installDir=/import/hjm3/nuc_sort/
# Package order: name | size | file with package names
orderBy=size
# DDL control file, default is DDL<DB_TYPE>.TPL
ddlFile=/export/hjm3/nuc_sort/EXP/DDLDB6.TPL
# File with mapping between DDL files and package names
ddlMap=
# Monitor timeout in seconds
monitorTimeout=30
#
# R3load options
#
# Optional path of R3load executable
r3loadExe=
# Generation of task files: yes | no
tskFiles=yes
# Inclusion of extent files: yes | no
extFiles=no
# DB code page for the target database
dbCodepage=1100
# Migration key
migrationKey=
# R3load omit value, can contain only 'DTPIV' letters
omit=
# Additional R3load arguments for TASK phase
taskArgs=
# Additional R3load arguments for LOAD phase
loadArgs=-loadprocedure fast LOAD
# Number of parallel import jobs
jobNum=10
#
# Exchange options
#
# Exchange directory
exchangeDir=/export/hjm3/nuc_sort
#
# Socket options
#
#socket
# Local hostname, default is 127.0.0.1
localhost=
```

```
# Server port number
port=
#
# Trace option
#
# Trace level
trace=all
#
# E-mail options
#
# SMTP server
mailServer=
# "From" email address
mailFrom=
# "To" email address
mailTo=
```

Definitions

To know how the numbers were calculated here are the formulas for the compression-factor and the compression-ratio:

Comp-factor (CF) $= \frac{Bytes(uncomp.)}{Bytes(comp.)}$ Eq. 1

Comp-Ratio (CR) $= \left(1 - \frac{Bytes(comp)}{Bytes(uncomp)}\right) * 100 \quad [\,\%\,]$ Eq. 2

and with concern of Eq. 1 we get CR as a function of CF:

CR = f(CF) $= \left(1 - \frac{1}{CF}\right) * 100$ Eq. 3

and by transforming Eq. 3 we get CF as a function of CR as:

CF = f(CR) $= \frac{100}{100 - CR}$ Eq. 4

STMM script

Example A-10 shows the STMM script.

Example: A-10 STMM script

```
Simple Script for parsing the db2diag.log for configuration changes by STMM:
#!/bin/bash
# Author:       Thomas Rech
# Usage:        stmmparse.sh
# Function:     Parses the db2diag.log for STMM change events and bufferpool
#               changes and extracts the essential information
#               of timestamp, changed paramter, old value and new value
#               in a file for usage with a spreadsheet.
#               Just a simple script, no error handling or advanced technologies
#Check if running with the right user account => <SID>adm or db2<SID>
if [ $USER != db2${DB2INSTANCE:3} -a $USER != ${DB2INSTANCE:3}adm ] ; then echo "Falscher User" ;
exit 1 ; else echo Parsing db2diag.log.....; fi
#Create copies of already existing file - but just once
if [ -f stmm.txt ] ; then chmod 777 stmm.txt; mv stmm.txt stmm.txt.bak ; else echo
stmm_essential_info_${DB2INSTANCE:3}_$HOST > stmm.txt; fi
#Parse db2diag.log and dump output to stmm.txt
#You may enhance the db2diag parse options e.g. Limit the search to a certain timeframe
db2diag -g EDUNAME:=db2stmm,FUNCTION\!:=stmmLog,DATA\!:=Package -fmt "%ts %{changeevent} %{msg}" >>
stmm.txt
#I know ..... But it is the easiest way to have access for both users
chmod 777 stmm.txt
#Just some formatting of the results to have a readable format for both spreadsheets (use tab as
seperator) and you
perl -i -p -e 's/To: \n/To:/g' stmm.txt
perl -i -p -e 's/To:          /To: /g' stmm.txt
perl -i -p -e 's/  / /g' stmm.txt
perl -i -p -e 's/  To:/ To:/g' stmm.txt
perl -i -p -e 's/<automatic>//g' stmm.txt
perl -i -p -e 's/ Altering bufferpool //g' stmm.txt
perl -i -p -e 's/STMM CFG DB BN6: //g' stmm.txt
perl -i -p -e 's/"IBMDEFAULTBP"/ "IBMDEFAULTBP"/g' stmm.txt
perl -i -p -e 's/ /     /g' stmm.txt
perl -i -p -e 's/"IBMDEFAULTBP"/"IBMDEFAULTBP  "/g' stmm.txt
perl -i -p -e 's/"Pckcachesz"/"Pckcachesz    "/g' stmm.txt
perl -i -p -e 's/"Sortheap"/"Sortheap      "/g' stmm.txt
perl -i -p -e 's/"Locklist"/"Locklist      "/g' stmm.txt
perl -i -p -e 's/"Maxlocks"/"Maxlocks      "/g' stmm.txt
perl -i -p -e 's/"//g' stmm.txt
echo Parsing of db2diag.log completed
echo File stmm.txt generated
```

Example 11-1 shows the STMM script output.

Example 11-1 STMM script output

```
2008-08-04-17.21.53.006533      IBMDEFAULTBP    From:   1000            To:     1500
2008-08-04-17.41.53.342496      IBMDEFAULTBP    From:   1500            To:     2250
2008-08-04-18.01.55.917397      IBMDEFAULTBP    From:   2250            To:     3375
2008-08-04-20.54.25.843259      IBMDEFAULTBP    From:   3375            To:     5062
2008-08-05-11.04.47.829275      Pckcachesz      From:   512             To:     5000
2008-08-05-11.04.47.837954      Sortheap        From:   195             To:     99
2008-08-05-11.07.47.864784      Sortheap        From:   99              To:     85
2008-08-05-11.08.17.871749      Sortheap        From:   85              To:     81
2008-08-05-11.31.48.114916      Sheapthres_shr  From:   1000            To:     1500
2008-08-05-11.31.48.148738      Sortheap        From:   81              To:     103
2008-08-05-11.41.48.101436      Sheapthres_shr  From:   1500            To:     2250
2008-08-05-11.41.48.117911      Sortheap        From:   103             To:     131
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbooks publication.

Publications

These publications are also relevant as further information sources:

- Bürckel, Nils; Davidenkoff, Alexander; Werner, Detlef: *Unicode in SAP-Systemen*, Galileo Press, ISBN(13) 978-3-89842-832-3
- Phipps, Toby: *Unicode Technical Report #26*

  http://www.unicode.org/reports/tr26
- *Unicode SAP Systems*

  http://service.sap.com/unicode@sap
- *Distribution Monitor User's Guide*, attachment of SAP Note 855772
- *Migration Monitor User's Guide*, attachment of SAP Note 855772 (DISTMON.SAR)
- *System Copy for SAP Systems Based on SAP NetWeaver 7.0 SR2 ABAP*, Document Version 1.1 - 06/22/2007
- *Package Splitter User's Guide*, in SPLIT.SAR

  service.sap.com/swdc → Support Packages and Patches → Additional Components → System Copy Tools → System Copy Tools 7.00 → #OS independent
- System Copy & Migration Optimization

  https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/8091fedf-5346-2a10-e4a0-a3cafe860199
- SAP Notes

  http://service.sap.com/notes
  - 0046272 - Implement new data class in technical settings
  - 0073606 - Supported Languages and Code pages
  - 0136702 - DB6: Move tables to other DB2 table spaces
  - 0362325 - DB6: Table conversion using DB6CONV
  - 0447519 - Kernel patches for code pages, languages and locales
  - 0515968 - DB6: Creating data classes and table spaces in DBA cockpit

- 0548016 - Conversion to Unicode
- 0552464 - What is Big Endian / Little Endian? What Endian do I have?
- 0738858 - R3load for Unicode conversion
- 0784118 - System Copy Java Tools
- 0855772 - Distribution Monitor
- 0857081 - Unicode conversion: downtime estimate
- 0867193 - ABAP and kernel patches for CU&UC in 46C
- 0928729 - Combined Upgrade & Unicode Conversion (CU&UC) FAQ
- 0936441 - Oracle settings for R3load based system copy
- 0952514 - Using the table splitting feature
- 0954268 - Optimization of Export: Unsorted unloading
- 0959698 - Twin Upgrade & Unicode Conversion (TU&UC) FAQ
- 0960280 - R3load Export Tuning for Split Tables with Oracle
- 0970518 - Hom./Het. System Copy SAP NetWeaver 2004s SR2
- 0971646 - 6.40 Patch Collection Hom./Het. System Copy ABAP
- 0978244 - CU&UC and NameTab errors for pools or clusters
- 0980067 - DB6: Using DB2 9 Row Compression
- 0989116 - Distribution Monitor: Troubleshooting Guide
- 0991481 - Database is using UTF-8 and not UTF-16?
- 1001383 - Distribution Monitor - Known Problems and fixes
- 1012025 - MDMP to Unicode conversion: Restart of R3load export
- 1040674 - R3ta in combination with unsorted unload
- 1043380 - Efficient table splitting for Oracle databases
- 1058437 - DB6: R3load options for compact installation with DB2 9

How to get IBM Redbooks publications

You can search for, view, or download Redbooks publications, IBM Redpapers publications, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Abbreviations and acronyms

CESU	Compatibility Encoding Scheme For UTF-16
DDL	Data Definition Language
DistMon	Distribution Monitor
ECC	Enterprise Central Component
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
MDMP	Multi-Display Multi-Processing
MigMon	Migration Monitor
NW	NetWeaver
UTF	Unicode Transformation Format

Index

Symbols

A

B

C

D

E

F

G

H

I

Q

R

S

T

U

V

W

X

Z

DB2 Optimization Techniques for SAP Database Migration and Unicode Conversion

Advanced migration techniques

Optimization strategies

Best practices

SAP migrations are a standard process nowadays. This IBM Redbooks publication describes optimization strategies and best practices for migrating SAP systems toward IBM DB2 for Linux, UNIX, and Windows, as well as for performing Unicode conversions. We discuss DB2-specific recommendations and best practices.

This book is intended for experienced SAP migration consultants involved in operating system and database migrations or Unicode conversions, choosing IBM DB2 as a target database platform. It addresses the advanced SAP migration techniques, considerations for database layout and tuning, and unique DB2 capabilities such as compressing the target database while loading data.

This book provides in-depth information about the best practices and recommendations for the source system database export, the advanced migration techniques, database layout and configuration, database import recommendations, SAP NetWeaver Business Warehouse, as well as background information about Unicode. We summarize our recommendations in one chapter that can be used as a quick reference for experienced migration consultants.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7774-00 ISBN 073843308X