IBM

e-Business Intelligence: Data Mart Solutions with DB2 for Linux on zSeries

Data population, data refresh, data mart joins with DB2 for Linux V7 federated databases

Intelligent data mart resource allocation with z/VM V4.2

Data access with Dimensional Insight OLAP tool

Viviane Anavi-Chaput
Raghu G Alvandar
Leticia Cruz, Andrea Harris
Jay Huie, Nin Lei
Merrilee Osterhoudt
Darren Swank, Mike Tebolt
Shrinivas B Vadavadagi
Tangy Ward

Redbooks

ibm.com/redbooks

International Technical Support Organization

e-Business Intelligence: Data Mart Solutions with DB2 for Linux on zSeries

February 2002

SG24-6294-00

Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 147.

First Edition (February 2002)

This edition applies to DB2 for z/OS Version 7 and DB2 for z/Linux Version 7 for use with the z/OS Version 1.2, z/Linux Version 2.4 kernel, and z/VM Version 4.2 Operating Systems.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Contents

Preface

This IBM Redbook explains the benefits of Business Intelligence solutions using DB2 for Linux on zSeries. We discuss several studies done at IBM zSeries Teraplex Integration Center to demonstrate how the latest features of the zSeries hardware and z/VM 4.2 operating environment extend the value proposition for consolidating data marts on DB2 for Linux for zSeries.

The book describes fast data transfer using HiperSocket connections, investigates scalability and performance issues in the context of data mart population, data mart refresh, data mart joins, intelligent resource management of zSeries under z/VM, and application deployment. We evaluate DB2 V7 latest utilities and DB2 federated database distributed queries.

The book also includes functional tests of OLAP tools such as Dimensional Insight.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Viviane Anavi-Chaput is a Senior IT Specialist for BI and CRM with the IBM International Technical Support Organization in Poughkeepsie, New York. She writes extensively, teaches worldwide, and presents at international conferences. Before joining the ITSO in 1999, Viviane was a Senior Data Management Consultant at IBM Europe, France. She was also an ITSO Specialist for DB2 at the San Jose Center during 1990-1994.

Raghu G Alvandar has been a Senior Software Engineer at IBM Global Services (India) Limited for 18 months. He has three years of experience in application design and development, and two years in database design and development.

Leticia Cruz has been a part of the IBM zSeries Teraplex Integration Center in Poughkeepsie, New York since 1996. She has 21 years of experience in data processing. She specializes in building DB2 databases for a VLDB environment. She has worked on various proofs-of-concept involving the world's largest corporations. She also teaches WLM for Business Intelligence offered to customers worldwide. She has extensive experience in system and application analysis, design and implementation.

Andrea Harris is an Advisory Software Engineer at the IBM zSeries Teraplex Integration Center in Poughkeepsie, New York. She has worked extensively in the Business Intelligence field since 1995, building and maintaining very large databases, testing new BI functions in DB2, executing performance measurements and analyzing data for customer proofs-of-concept. Andrea also presents the results of proofs-of-concept and performance studies to IBM's customers and marketing teams at conferences and at the IBM Briefing Center.

Jay Huie is a Software Engineer at the IBM Customer Enablement Center in Poughkeepsie, New York. He has worked for IBM for two years in system test and development positions. He has more than four years experience running Linux and has followed the culture even longer. His present responsibilities include working with various Linux distributors, Linux system support, and tool development. He holds a Bachelor of Science in Computer Engineering degree from Case Western Reserve University.

Nin Lei is a Senior Software Engineer at the IBM Global Business Intelligence Solutions Center. He is recognized as an expert in DB2 performance and database design. Nin's primary focus is on Very Large Data Bases (VLDB), covering areas of database and application design and performance analysis. He is also skilled in system analysis, systems management, object technology and distributed systems. He has served in many capacities, including consultant, architect, designer, and performance analyst.

Merrilee Osterhoudt joined IBM in 1983, starting her career in MVS design and development, coordinating new releases and management of Early Support Programs. From 1992-2000 she was a member of the S/390 Business Intelligence solutions team, and developed and implemented the worldwide technical marketing support strategy for data warehouse solutions. In 2001, Merrilee joined IBM's Teraplex Integration Center as the team lead for technical marketing collateral and communications.

Darren Swank is an Advisory Software Engineer at the IBM zSeries laboratory in Poughkeepsie, New York. He has 12 years of experience with DB2 and has spent the last 9 years specializing in data warehousing. He holds a Masters degree in Management Information Systems.

Mike Tebolt is a Senior Software Engineer at the IBM zSeries Teraplex Center in Poughkeepsie, NY. He has been with IBM for 20 years, much of that time as an MVS and DB2 System Programmer.

Shrinivas B. Vadavadagi is a Senior Software Engineer at IBM Global Services (India) Limited. He has three years of experience in application, database design and development. He has worked at IBM for a little over a year.

Tangy Ward is a Software Engineer at the IBM zSeries laboratory in Poughkeepsie, New York. She has been in IBM for 19 years. She started in MVS Support and spent 11 years in DB2 Systems Support. Since 1997, she has worked in the Poughkeepsie Customer Enablement Center using her DB2 and Project Management skills for Proof of Concepts, DB2 Solution Test and Parallel Sysplex/DB2 Datasharing Customer Education. She holds a Bachelor's degree from Pratt Institute in Data Systems Management.

Thanks to the following people for their contributions to this project:

Donna von Dehsen
IBM Linux on zSeries Development, Poughkeepsie, New York

Laurie Francisco
IBM DB2 Development, Toronto

James Goethals
IBM zSeries Networking, Raleigh, North Carolina

Charlene M. Hamiwka
Dimensional Insight, Inc.

Clark Josephs
IBM zSeries Teraplex Center, Poughkeepsie, New York

Michael MacIsaac
IBM International Technical Support Organization, Poughkeepsie, New York

Anand Mehta
Dimensional Insight, Inc.

Chris Panetta
IBM zSeries Performance, Poughkeepsie, New York

William G. White
IBM International Technical Support Organization, Poughkeepsie, New York

Thanks also to Terry Barthel, Alison Chandler and Alfred Schwab for their editorial assistance, and Ella Buslovich for her graphics assistance.

Notice

IBM trademarks

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to the address on page ii.

1

Introducing BI solutions using Linux on zSeries

This chapter introduces the Business Intelligence (BI) integrated solution on zSeries using DB2 for Linux on zSeries as the data mart server. It highlights the following:

- Challenges facing IT organizations and why they are looking at Linux on zSeries and S/390 as a compelling solution for leveraging assets and reducing costs
- Features of zSeries and z/VM operating systems that make the platform a good option for the consolidation of data mart servers and BI application servers
- Focus areas of the IBM Teraplex Center study.

In the redbook *eBusiness Intelligence: Leveraging DB2 for Linux on S/390*, SG24-5687, we discuss how the inherent strengths of S/390 and zSeries can be leveraged with DB2 on Linux to build a powerful infrastructure for end-to-end BI solutions. This book expands that discussion and focusses on the results of a study conducted by IBM's zSeries Teraplex Integration Center to demonstrate how the latest features of the zSeries hardware and z/VM 4.2 operating environment extend the value proposition for consolidating data marts on DB2 for Linux on zSeries.

1.1 The advantages of Linux on zSeries

The introduction of the Linux operating environment to the S/390 and zSeries platform has generated a great deal of interest among IT organizations with significant investments in the mainframe infrastructure. Many of these shops are also supporting numerous UNIX servers running various applications and middleware. At a time when IT is being challenged to reduce costs while continuing to meet the increasing demands for computing services, curbing the costs associated with the maintenance of numerous platforms and multiple server farms through server consolidation has become a key initiative. IBM has positioned Linux on its zSeries and S/390 servers as a compelling option for cost-conscious customers to consolidate multiple UNIX servers onto a single mainframe server. Consider the following factors:

- Attractive pricing of Linux-dedicated processors (with the hardware option, Integrated Facility for Linux (IFL))
- Attractive pricing of the Linux operating system from distributors
- OTC pricing model for IBM software running on Linux, such as DB2 UDB and Intelligent Miner Scoring, as well as non-IBM products
- Reduced systems management costs
- Ability to leverage the less expensive UNIX skill base
- Easy porting of applications developed in UNIX to the nonproprietary Linux operating system, often needing no more than a recompile
- Ability of applications developed on Linux to run on Linux on all platforms with a recompile

1.2 The growth of Business Intelligence

Over time, Business Intelligence applications have played an increasingly important role in the success of businesses. As a result, more and more business units are building data marts. As the successes of groups using data marts become apparent, the number of users who want access to individual data marts increases, as does the demand for more data and more sources of data to feed the marts. In addition, the trend toward globalization has put requirements on IT to provide uninterrupted availability of decision support databases and applications.

1.2.1 Increasing demands for data marts

Data warehouses and Operational Data Stores have usually been based on DB2 for OS/390 because most of the source data for these structures resides on S/390 production systems, and this proximity reduces the cost of moving large volumes. Data marts, however, are frequently implemented on UNIX servers. This trend is largely due to the following factors:

- The users of data marts often require rich SQL functionality to manipulate and dissect the data. DBMSs on UNIX have usually delivered more of these functions than the transaction-oriented DB2 for z/OS-OS/390.
- The growth of off-the-shelf BI applications has been almost exclusively focused on UNIX platforms.
- Data marts are usually funded by business units rather than IT. It is relatively easy to develop a compelling business case for the initial startup cost of a data mart on a small UNIX or NT server, while the policies of charge-back and resource allocation can add complexity to the zSeries-S/390 environment.
- Query workloads have traditionally been assigned a low priority when sharing resources with production systems. This has prompted business units to fund their own server solutions in order to insure that there are processing resources tuned and dedicated to their decision support applications.

1.2.2 High cost of multiple distributed data marts

The growth in the need for data marts has contributed greatly to the proliferation of UNIX servers. The cumulative effect of numerous business units acquiring independent data marts is the growth of large server farms that present IT with a whole new set of cost of ownership and manageability issues, most pointedly:

- The overall reduction in price performance as a result of a cumulative growth in latent capacity that cannot be utilized
- The infrastructure and systems management costs of loading and maintaining external marts
- The challenge of meeting the end-user demands for continuous availability of data marts with current data
- Complex systems management, increased costs in staffing and maintenance of a complex network infrastructure.

1.3 Consolidation of data marts on DB2 for Linux

Linux on zSeries can provide an excellent solution for consolidating multiple data marts on a single server to reduce cost and still deliver the same benefits of separate isolated servers - dedicated resources, and the isolation of applications and data. In addition to the expected cost savings associated with consolidation and the economies of IFL and OTC software pricing, zSeries hardware and the z/VM operating environment offer additional benefits to DB2 data marts on Linux. The next section will discuss how zSeries provides the capability to:

- Dedicate and/or share processing resources for marts and applications based on business priorities and get full utilization of available capacity.
- Reduce the time for loads and refreshes of marts to minimize down times and make current data more readily available.
- Deploy new data marts and BI applications faster.

1.3.1 Intelligent management of processing resources

Intelligent resource management is a key differentiation of Linux on zSeries and z/VM as a platform for consolidating multiple UNIX servers onto a single physical machine. This is the only hardware and software platform today that enables you to consolidate multiple data marts and BI application servers onto a single physical server, optimize resource utilization across all of the workloads and still provide the application and data isolation benefits of distributed servers.

Characteristically, data mart utilization fluctuates from hour to hour, day to day, week to week, or month to month, based on the business cycle and requirements of the end users. Typically, the average utilization of UNIX servers is 40-60% of the available capacity. When data marts are running on separate servers, each server must be configured to provide acceptable performance during high utilization periods, resulting in latent capacity, or “white space“, during lower utilization. There is no way to leverage this “white space”, by diverting underutilized capacity of one data mart server to other data marts that may be resource-constrained. The cumulative effect, is a large percentage of the total processing resources remains idle a great deal of the time, continually driving up the cost/user over time.

The zSeries platform is singularly equipped to provide intelligence resource sharing together with application and data isolation to multiple data marts consolidated on a single machine. This is achieved by creating virtual data mart servers using PR/SM to vertically partition the hardware resources into separate logical servers called LPARs, and/or z/VM Virtual Machine software support to further partition a logical server horizontally into multiple virtual servers.

Logical machine partitioning with PR/SM

The PR/SM facility of zSeries hardware (like S/390), allows the partitioning of the processors into logical partitions called LPARs, to create multiple logical servers on a single physical machine. These logical servers behave like physically independent servers, providing dedicated resources while maintaining the isolation and integrity of applications and data. LPARs can be defined with dedicated resources that are not shared with other logical servers, or they can be defined in sharing mode with options, like weighting and capping, that instruct the system how to control sharing between the logical servers. Each LPAR has the same isolation and integrity as if it were a standalone server.

You can consolidate multiple data marts by carving a single zSeries machine into several individual logical data mart servers, and then leverage the resource management capabilities of PR/SM to maximize the utilization of available capacity across those data marts to insure that each mart gets its optimal capacity (based on the available capacity of your configuration). By consolidating on the same server as the ODS, data warehouse and/or production systems, your marts can further benefit from the close proximity to the source systems and to each other.

z/VM virtual images

The VM operating system for S/390 provides you with the capability to further partition an LPAR into virtual machines that run guest operating systems like MVS. z/VM V4.2 for zSeries now supports Linux guest systems on IFL processors, and has One Time Charge licensing for running Linux workloads. In addition to weighting and capping of LPAR resources, z/VM provides the ability to assign percentages of the LPAR's resources to each Linux virtual image. This gives you one more level of partitioning of processing resources and a greater granularity of intelligent resource management across multiple data marts.

Figure 1-1 on page 6 is a representation of a zSeries server partitioned into multiple PR/SM LPARs.

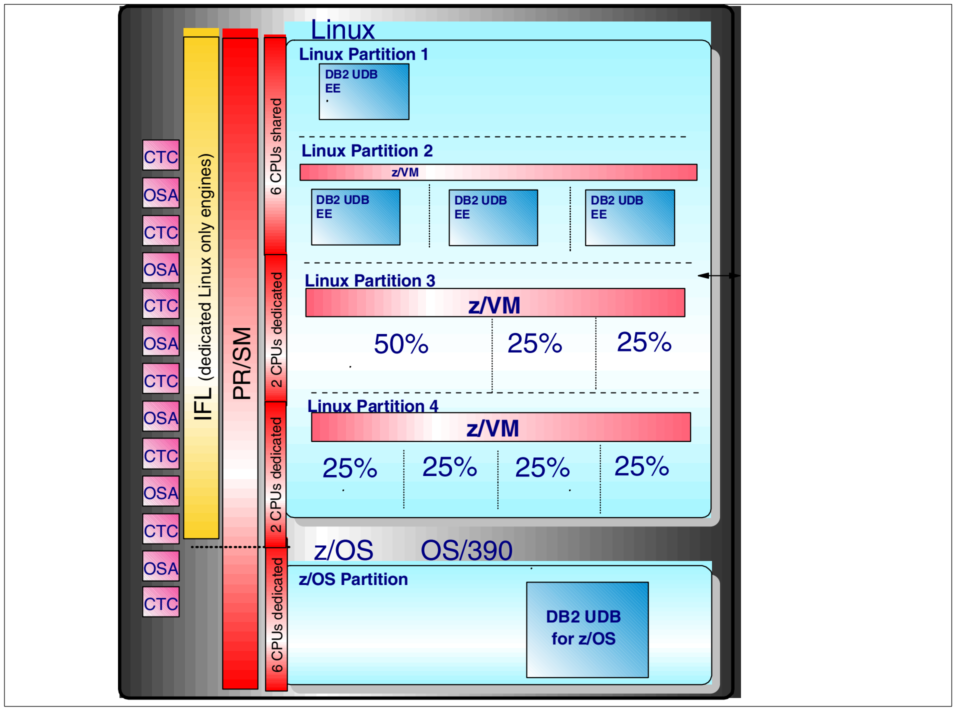


Figure 1-1 zSeries server partitioned into multiple PR/SM LPARs

Four partitions are running on a total of ten IFL processors dedicated to Linux and one partition has six processors dedicated to z/OS. Linux Partitions 2, 3 and 4 are each further partitioned under z/VM into multiple virtual machines running Linux guest systems. In this example, Linux Partition 1 and Linux Partition 2 are sharing six processors. Each of these LPARs will get a minimum of 50% of the capacity when running at full utilization, and each will have access to any excess capacity that becomes available if the other reduces its resource consumption. On the other hand, Linux Partition 3 and Linux Partition 4 each have dedicated processors, which neither LPAR can exceed, regardless of its processing needs. The z/OS Partition is allocated, and cannot exceed, six non-IFL processors. (Note that z/OS and Linux can not share processing resources across LPARs.)

Each of the data marts in Linux Partition 2 are equally sharing the resources available to the partition at any given time. When all four data marts are at full utilization, Mart 1 (in Linux Partition 1) will have three processors, and Marts 2, 3 and 4 (in Linux Partition 2) will have one processor each. If Mart 1 goes off line, Marts 2, 3 and 4 will have two processors each, if needed. If all but Mart 4 go off line, Mart 4 will have access to all six processors. All of this is managed dynamically by PR/SM and z/VM, with no manual intervention once the sharing options have been set up.

The z/VM virtual machines in Linux Partitions 3 and 4 have been defined using prioritization. These could be running data marts, application servers or Web servers, for example. By using priorities we have established how z/VM will distribute the resources available to each of these LPARs when they are at full utilization. When there is capacity freed up by any of the virtual images, it will be made available to the others.

The capability to share white space across multiple data mart servers, and intelligently manage resources based on business priorities, is a unique function of zSeries (and S/390) that gives you the flexibility to optimize the use of all of the available capacity 24 hours per day, 365 days per year. The Teraplex conducted a study of the efficiency of the resource management capabilities of zSeries PR/SM and z/VM in managing processing resources while processing needs fluctuate across several data marts. The details of that study are discussed later in this book.

1.3.2 Efficient data transfers

A large portion of time related to creating and maintaining a data mart is devoted to the movement of the data from one platform to another. Consolidating several data marts in Linux LPARs on the same physical platform as the data warehouse and ODS allows data transfers to the data marts to be done internally from LPAR-to-LPAR, without going out on the LAN. LPAR-to-LPAR direct data transfers avoid using the LAN and thus eliminate the overhead and risk associated with external network traffic. One approach to LPAR-to-LPAR communications on S/390 and zSeries servers is to use OSA-Express Gigabit adapters, dedicated or shared. Even more exciting is a feature, available only on zSeries models called HiperSockets.

What are HiperSockets?

HiperSockets provide very fast TCP/IP communication between consolidated Linux, z/VM and z/OS virtual servers on zSeries. The purpose of HiperSockets on zSeries servers is to provide a highly available, high-speed network connection among multiple combinations of virtual servers and LPARs when consolidating servers on zSeries. This totally integrated any-to-any TCP/IP network provides benefits not achievable by grouping multiple distributed servers around a zSeries interconnected by external networking technology. HiperSockets provide up to four internal LANs which act like TCP/IP networks within the zSeries. This integrated zSeries Licensed Internal Code (LIC) function, coupled with supporting operating system device drivers, establishes a higher level of network availability, security, simplicity, performance, and cost effectiveness than available when connecting single servers or LPARs together using an external TCP/IP network.

The HiperSockets function, also known as iQDIO or internal QDIO, is tightly integrated into the zSeries system to provide users with attachment to as many as four high-speed “logical” Local Area Networks (LANs) with minimal system and network overhead. HiperSockets eliminates the need to utilize I/O subsystem operations and the need to traverse an external network connection to communicate between LPARs in the same zSeries server. By consolidating multiple servers onto a single zSeries server, you can now connect the many virtual servers using HiperSockets. HiperSockets also offer many opportunities to cut costs by reducing network administration costs. Since HiperSockets do not use an external network, you can free up system and network resources, eliminating attachment costs while improving availability and performance. HiperSockets are easy to install, operate and maintain, and are completely transparent to applications.

Exploitation of HiperSockets by consolidated data marts

DB2 data marts running on Linux as virtual servers under z/VM and PR/SM can benefit from the exploitation of HiperSockets in the following ways:

- Data mart population: The trend has been for data marts to get larger and larger as business analysts get more sophisticated in using data. By reducing the time it takes to do initial loads of hundreds of gigabytes of data, new data marts can be brought on line faster, maximizing their value to the business sooner.
- Data mart refresh: Globalized companies are giving business users across many time zones access to data marts, putting greater pressure on IT to keep marts available around the clock. A dynamic business environment focused on the customer wants and needs is requiring data mart users to have data that is refreshed more and more frequently, some times several times per day. The data transfer speeds of HiperSockets can be leveraged to shorten the time required to refresh data marts.
- Accessing multiple data sources: many users do analysis that requires the merging of data from the data mart with multiple data sources outside of the mart, such as the data warehouse, other data marts and flat files. When you have DB2 data marts running on Linux on the same physical zSeries server as the DB2 for z/OS data warehouse, you can exploit HiperSockets to improve data transfer times for cross joins.

In the following chapters of this book we discuss the results of tests that we conducted at the Teraplex to study the data transfer rates between a DB2 data warehouse on z/OS and several data marts consolidated in LPARs running Linux guest systems under z/VM using HiperSockets versus doing the same data transfer to an external mart across a LAN.

1.3.3 Faster deployment of data marts

There are many reasons for bringing new data marts on line, such as:

- A company, or a business unit within a company, is in the early stages of implementing a business intelligence strategy and in the initial deployment phase.
- A new product, business unit or initiative has created a new set of data. Some marketing units will build a new data mart for each campaign and discard it when the campaign is finished.
- Changes in the business climate require actions to realize new opportunity or reverse loss of existing opportunity. New competitors, government regulations, and natural disasters, are examples of factors that can influence a company's performance in the marketplace.

In all cases, the sooner the data mart becomes available to analysts and decision makers, the sooner it can contribute to achieving the company's goals.

In the UNIX environment, the time to order, receive delivery and install new hardware and software for a new data mart can take weeks. In contrast, the physical addition of a new data mart in a Linux guest running under z/VM on a zSeries LPAR could take less than an hour!

z/VM

Under z/VM on an existing LPAR, a new virtual machine can be defined dynamically, in minutes, without any disruption to any other images or applications. The capacity that is assigned to a new data mart would be determined by your business priorities.

z/VM has a feature that enables you to replicate an existing Linux image, thereby dynamically creating a new Linux server that is ready for the installation of DB2 and/or application code.

PR/SM

You can configure a whole new LPAR faster than the time required to acquire a new server.

Capacity Upgrade on Demand

zSeries also offers a feature called Capacity Upgrade on Demand (CUD). CUD gives you the ability to dynamically enable spare processors that are already installed on a zSeries server, when a new workload needs them. CUD can be exploited to add capacity to handle new data marts or increased workload of existing data marts. You can also use CUD to activate additional capacity temporarily to meet an unsustained increase in demand, and then take the extra capacity off line when normal usage patterns return. This provides outstanding flexibility to the business intelligence infrastructure, making it responsive to fluctuating business needs while allowing you to effectivlly manage the cost to numerous levels of granularity.

HiperSockets

HiperSockets offers a fast path for transferring data between LPARs on a single zSeries server. This could improve the time required to do the initial load of a new data mart, further reducing the time it takes to get it into production.

Linux application development and deployment

The hardware neutrality of Linux makes it very attractive for the development and deployment of applications. You can develop an application on Linux on any supporting hardware platform and deploy it on Linux on any other hardware platform by doing a simple compile. There is no expensive porting effort that adds time and cost to moving applications into production from small test systems. The conversion of UNIX applications to Linux is also a relatively simple undertaking. Vendors, as well as IT shops that have in-house applications, can quickly port their UNIX code to Linux. This greatly facilitates a strategy for consolidating existing data marts onto DB2 UDB for Linux on zSeries.

Imagine the flexibility that an LPAR running Linux under z/VM, utilizing HiperSockets for loading the mart, can offer to a marketing organization that creates a temporary data mart for each campaign! It is conceivable that a small data mart could be defined and loaded in an hour!

1.4 zSeries Teraplex Integration Centers

IBM's Teraplex Integration Centers are in place to facilitate the implementation of Business Intelligence and related solutions on IBM hardware and software platforms. We have a highly skilled staff of database and systems experts who specialize in very large database and Business Intelligence infrastructure. The primary service offered by the Teraplex Centers is large-scale proofs of concept conducted at IBM facilities, either during the platform selection phase or the architecture design and validation phase of a very large database solution. When appropriate, the Teraplex Centers will conduct proofs-of-concept at customer locations. In addition to proofs-of-concept, Teraplex Center personnel offer remote and on-site consultation support such as customer briefings, solution design review and assurance, and data warehouse tuning recommendations, to assist customers in making the best platform selection and implementation decisions for their Business Intelligence solutions.

The Teraplex Centers often conduct independent studies that explore the extended capabilities of IBM technologies in supporting Business Intelligence solutions. The remainder of this book discusses several studies done at the zSeries Teraplex Integration Center to demonstrate how the latest features of the zSeries hardware and z/VM 4.2 operating environment extend the value proposition for consolidating data marts on DB2 for Linux on zSeries.

- Fast data transfer of HiperSockets in the context of:
  - Data mart population
  - Data mart refresh
  - Distributed data marts
- Intelligent resource management of zSeries PR/SM and z/VM
- Application deployment

1.5 Summary

zSeries and z/VM offer unique advantages for multiple data marts consolidated on a single zSeries server, such as:

- Attractive IFL hardware and z/VM OTC software pricing to lower costs of multiple data marts
- Intelligent resource management to maximize utilization of processing resources across multiple data marts
- Faster time to production
- Faster data transfers to help increase availability of marts and currency of data
- Increased flexibility

2

Test environment infrastructure

This chapter introduces the test environment infrastructure we built at the IBM zSeries Teraplex Center and Customer Enablement Center at Poughkeepsie, New York. We describe the following:

- Hardware/software configurations
- Network connectivity setup
- Linux on zSeries system setup
- Federated database setup

2.1 Hardware/software configurations

This section describes the hardware and software configurations.

Figure 2-1 gives an overview of the system configuration we used to run our test scenarios.

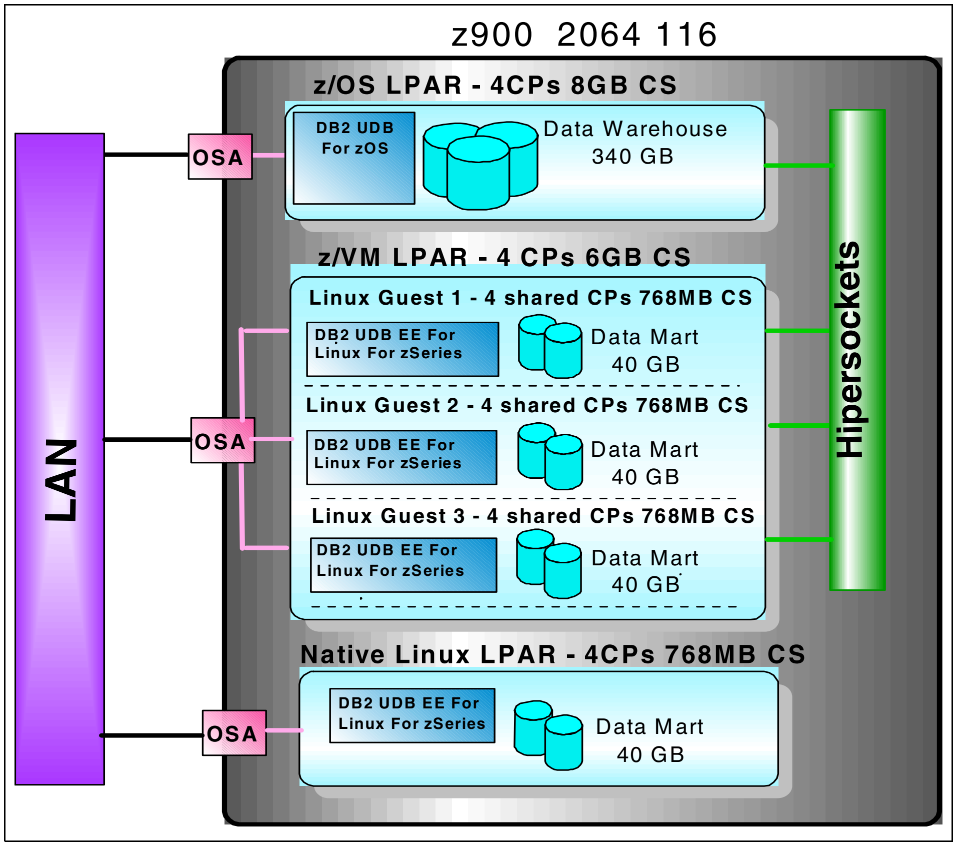

z900 2064 116
z/OS LPAR - 4CPs 8GB CS
DB2 UDB For zOS
Data Warehouse 340 GB
z/VM LPAR - 4 CPs 6GB CS
Linux Guest 1 - 4 shared CPs 768MB CS
DB2 UDB EE For Linux For zSeries
Data Mart 40 GB
Linux Guest 2 - 4 shared CPs 768MB CS
DB2 UDB EE For Linux For zSeries
Data Mart 40 GB
Linux Guest 3 - 4 shared CPs 768MB CS
DB2 UDB EE For Linux For zSeries
Data Mart 40 GB
Native Linux LPAR - 4CPs 768MB CS
DB2 UDB EE For Linux For zSeries
Data Mart 40 GB
LAN
OSA
OSA
OSA
Hipersockets

Figure 2-1 System configuration

The zSeries server (z900 2064 116 model) has three Logical Partitions (LPARs) running the following environments:

- A z/OS LPAR running z/OS V1.2 with a 340-GB data warehouse
- A z/VM LPAR running z/VM V4.2 with three Linux on zSeries V2.4 kernel guests, each with a 40 GB data mart
- A Native Linux LPAR running Linux on zSeries V2.4 kernel with a 40 GB data mart

All LPARs can be accessed by LAN using OSA Gigabit Ethernet connections.

The z/VM LPAR and z/OS LPAR additionally can be accessed internally in the zSeries using the HiperSocket (or iQDIO) connections.

The Native Linux LPAR is set up with only a LAN connection. This LPAR can be considered a data mart on a separate UNIX server, such as an RS/6000 or Sun server connected to the data warehouse on a zSeries server by LAN.

2.1.1 Configuration objectives

The objective of this configuration was to create two distinct data warehouse/data mart environments in which we could run a series of test cases to contrast the two.

One environment can be considered “consolidated”, where the z/OS-based data warehouse resides on the same physical machine as three Linux on zSeries-based data marts running in virtual machines under z/VM. Each of the Linux images appears to have its own four-CPU, 768MB machine with a full set of I/O paths and network adapters. In actuality, they are sharing the CPUs, real storage, I/O paths, and network card allocated to the z/VM LPAR.The network path between the Linux images and the z/OS image uses a HiperSocket “virtual LAN”, where the data transfer is memory-to-memory between LPARs, using no actual network cards or cables.

The other environment uses a Linux on zSeries-based data mart running natively in an LPAR, with dedicated CPUs, real memory, and network interface. The network path between this native Linux image and the z/OS data warehouse image is via separate OSA Gigabit cards connected to a real LAN. This configuration represents an environment where the z/OS data warehouse and Unix-based data mart are located on physically separate servers.

2.1.2 Hardware configuration

We used a zSeries server, model z900 2064 116, with the following attributes:

Central processors

The zSeries server has 16 central processors (CPs) of which we used 12: 4 CPs for each LPAR.

Memory

We used approximately 16 GB of central storage divided among the LPARs as shown in Figure 2-1 on page 14.

z/VM V4.2 can utilize greater than 2 GB of real storage and the virtual pages of a Linux guest can reside above the 2 GB boundary. However, for certain operations like I/O, where a virtual address must be translated to a real address, z/VM copies the affected page below the 2 GB boundary. When storage below the 2 GB boundary becomes constrained, paging results. Since Linux takes whatever storage is not allocated and uses it for file cache, it tends to use all of the storage available to it. Paging, then, is a concern for z/VM systems hosting Linux guests, and better performance may be achieved by allocating a smaller virtual machine size to Linux.

Initially, we defined the Linux guests with a virtual machine size of 2 GB, but then reduced it to 768 MB and actually saw better performance due to a decrease in z/VM paging.

Network connections

We implemented the following:

- HiperSockets (no hardware required, included in the latest zSeries microcode level)
- 3 OSA-Express Gigabit cards
- 3 OSA EN/TR cards

HiperSockets

HiperSockets provides up to four integrated TCP/IP virtual LANs accessible among combinations of partitions and virtual servers within the z900. There is no network hardware required: This function is implemented in Licensed Internal Code, together with supporting operating system device drivers. The data transfer is handled much like a cross address space memory move using the memory bus. Because of the memory-to-memory nature of the data flow, HiperSockets provides several advantages over traditional network connections among servers:

- Cost: There are no external network components required for server-to-server connectivity. There are no networking boxes or cables to buy and maintain. Additionally, removing server-to-server traffic from the data center network potentially frees up bandwidth for client/server traffic.
- Security: Since the data transferred via HiperSockets is never put on a wire, there is no exposure to having it tapped.
- Performance: A memory-to-memory data transfer is not impacted by network component latency. Applications where server-to-server network performance is a factor in overall performance can see improvements by using HiperSockets.

HiperSockets connections can be customized to use one of four possible maximum frame sizes (MFS): 16K, 24K, 40K, 64K. TCP/IP for Linux will adjust the maximum transmission unit (MTU) based on the MFS. For our testing, we selected an MFS of 64K, due to the characteristics of our workload, which involves sending large files between servers. For a more interactive workload, where smaller packets of data are transmitted, a smaller MFS may be appropriate.

Use of a HiperSocket connection is transparent to an application. It looks like any other TCP/IP network link. On z/OS, there are HCD/IOCP definitions required to define the IQD CHPIDs and devices, as well as Communications Server definitions required to define the devices, links, and IP addresses. With Linux on zSeries, there are updates to /etc/modules.conf, /etc/chandev.conf, and network scripts. Refer to Appendix C, "Networking definitions" on page 109 for examples of the HCD, Communications Server, and Linux definitions used in this project.

For additional information on defining HiperSocket links refer to:

- *z/OS V1R2.0 Communications Server IP Configuration Guide,* SC31-8775
- *z/OS V1R2.0 Communications Server IP Configuration Reference,* SC31-8776
- *zSeries Hipersockets*, REDP0160

For Linux on zSeries, refer to the following Web site:

```
http://www10.software.ibm.com/developerworks/opensource/linux390/documentat
ion-2.4.shtml
```

In our testing, we used HiperSocket links for data transmission between the z/OS LPAR and the Linux guest images running in the z/VM LPAR. We also used HiperSocket links to transmit data between the Linux guests as shown in Figure 2-1 on page 14.

OSA-Express Gigabit

For comparison purposes, we set up a private LAN to transmit data between the Native Linux box and the z/OS and z/VM Linux guest images (refer to Figure 2-1 on page 14). Each LPAR had a dedicated OSA-Express Gigabit card providing connectivity to the private LAN. Refer to Appendix C, "Networking definitions" on page 109 for examples of the OSA definitions used in this project.

OSA EN/TR

Though not shown in Figure 2-1 on page 14, each LPAR also had an OSA EN/TR card. This was used to provide login connectivity from the IBM network.

Disk configuration

We used Enterprise Storage Systems (ESS) model 2105-F20 with a total disk capacity of 1.6 TB. The ESS was configured with 32 ESCON channels spread across 8 logical control units. The ESCON channels were shared among the LPARs using ESCON Multiple Image Facility (EMIF).

2.1.3 Software configuration

We used the following software products:

- z/OS V1R2
  - DB2 Universal Database (UDB) for z/OS V7.1
  - DB2 Performance Monitor for z/OS Version 7.1
  - DB2 High Performance Unload V1.1
  - Omegamon II for DB2 V510
- z/VM V4R2
- Linux on zSeries kernel V2.4.7
  - DB2 Universal Database Enterprise Edition (UDB EE) V7.2 for Linux on zSeries
  - SAR for Linux
  - Dimensional Insight product suite
  - perl 5.6.0

Several of the products used were pre-GA versions:

- z/OS V1R2
- z/VM V4R2
- Linux on zSeries kernel V2.4.7

As with all new software or new function such as HiperSockets, it is recommended to consult the latest Preventive Service Planning (PSP) information available.

2.2 Linux on zSeries system setup

The Linux on zSeries native and guest “system” volumes were configured similarly, with separate volumes allocated for root, swap, and the /usr/IBMdb2 and /home directories. This was done more for the convenience of having them on separate volumes, rather than due to the size requirements of these directories, so we could, for example, reformat the root volume and load a new root file system without affecting our DB2 code or /home directory.

For each Linux guest image, we did take advantage of VM minidisks to consolidate the swap and /usr/IBMdb2 virtual volumes on the same real DASD.

Each Linux image was given 17 volumes for DB2 objects: One for log files, and sixteen for tablespace containers. Additional volumes were used for the ftp of files unloaded from DB2 on z/OS, and for holding performance data collected during test runs.

2.2.1 Installation of Linux on zSeries

We used a prebuilt Linux on zSeries root file system, *t*ar'ed and stored on a separate ftp server at our location. This made our installation somewhat different than if we were using one of the common Linux on zSeries distributions. See A.2, "Installing Linux in a z/VM virtual machine" on page 95 for the steps we followed to install a Linux on zSeries image under z/VM.

2.2.2 z/VM Virtual Images setup

z/VM System Administration utility

The z/VM System Administration Facility, which comes with z/VM V4R2, provides tools which make it easy to configure and manage Linux images running under z/VM. These tools are based on some of the ease-of-use functions that were part of the S/390 Virtual Image Facility for Linux (VIF) product. For those installations using VIF, there is a VIFMIGR facility, allowing the migration of a VIF configuration to a z/VM system. For more information on the System Administration Facility, see the Web site `http://www.vm.ibm.com`, then click **Learn more about z/VM V4R2.0**, and then, under Reference Information, click **New System Administration Facility for z/VM**.

Whereas the System Administration Facility is intended for use on a non-tailored z/VM system, the z/VM system used in our tests had been tailored to be used by many different test groups. We therefore set up our Linux images "manually", editing z/VM directory statements and using appropriate utilities to update the z/VM configuration.

Linux on zSeries guest definitions

In our z/VM directory, we defined three Linux guests with the userids guest1, guest2 and guest3. Each user was given a virtual machine size of 768 MB and 4 CPUs.

Each user was given a 50 cylinder minidisk, on which was stored the Linux ramdisk image, obtained via ftp from a server which holds Linux on zSeries distributions at our location. See A.2, “Installing Linux in a z/VM virtual machine” on page 95 for the procedure used to ftp the Linux ramdisk image to z/VM, and IPL it.

Each user was given a full volume minidisk for its Linux root directory, defined with a virtual address of 7580. Since guest operating systems under VM use virtual addresses, each Linux guest can be set up to have an identical configuration, while actually using different hardware, helping to simplify administration.

Table 2-1 shows the base setup of volumes on our Linux images.

Table 2-1 Linux image base volume setup

Linux Image	Use	Real Addr	Virtual Addr	Volume Label
Native	/ (root)	E520		BILNRT
	swap	E521		BILNSW
	/usr/IBMdb2	E522		BILNDB
	/home	892A		BILNHM
Guest1	/ (root)	E735	7580	BILV1R
	swap	8926	7581	BILV1D
	/usr/IBMdb2	8926	7582	BILV1D
	/home	8927	7583	BILV1H
Guest2	/ (root)	E736	7580	BILV2R
	swap	8A26	7581	BILV2D
	/usr/IBMdb2	8A26	7582	BILV2D
	/home	8A27	7583	BILV2H
Guest3	/ (root)	E737	7580	BILV3R
	swap	8B26	7581	BILV3D
	/usr/IBMdb2	8B26	7582	BILV3D
	/home	8B27	7583	BILV3H

Note that the swap volume and volume containing /usr/IBMdb2 were located on the same physical volume for the Linux guests. The ability to carve real DASD into minidisks, and allocate only the space needed for a particular guest image, is an efficiency gained by running virtual machines under VM. However, in this project, since we were dealing with large amounts of data, we typically dedicated full volumes to the Linux guest images.

The remaining DASD we allocated to each Linux guest was done by dedicating entire 3390 volumes. Each guest was given 17 volumes for DB2 objects: One for log files, and sixteen for tablespace containers. Additional volumes were dedicated for the ftp of files unloaded from DB2 on z/OS, and for holding performance data collected during test runs.

The Linux guests shared an OSA-Express EN/TR card for client connectivity, and shared IQD (HiperSockets) CHPIDs for server-to-server connectivity within the zSeries. Although we had four IQD CHPIDs defined on our zSeries, only the one defined with the maximum frame size of 64KB was used in our tests. Each guest was dedicated three device numbers corresponding to the OSA card, and three device numbers corresponding to an IQD CHPID. To see the network definitions used, refer to Appendix C, “Networking definitions” on page 109.

To see an example of the VM directory entries for one of our Linux guest images, see A.1, “Sample z/VM directory entry” on page 93.

Cloning a Linux on zSeries image under z/VM

One of the efficiencies of running Linux under z/VM is the ability to quickly clone a Linux image. This allows a new Linux server to be set up, literally in minutes, as opposed to the longer lead time associated with obtaining and installing new hardware and software, in the case of a physically separate server. While cloning is especially easy using the System Administration Facility, with its PARTITION COPY command, it is also easily accomplished manually, using the z/VM DDR utility, as shown in A.3, “Cloning a Linux image on z/VM” on page 99.

Linux on zSeries under z/VM performance notes

Hints and tips regarding the performance of Linux under z/VM can be found on the Web at:

```
http://www.vm.ibm.com/perf/tips/linuxper.html
```

One of the items discussed is how a smaller virtual machine size could improve the performance of a Linux guest. As mentioned above in the hardware configuration section, we saw better performance when we decreased the size of our Linux virtual machines from 2 GB to 768 MB.

Another hint mentioned on the Web site is to disable the minidisk cache for minidisks used as Linux swap volumes. We did this by coding the MINIOPT NOMDC keywords in the z/VM directory for the minidisks used as swap volumes (see A.1, “Sample z/VM directory entry” on page 93).

2.3 Federated database setup

Many of our scenarios required submitting SQL that referenced tables in different databases, located on different operating system images. We set up federated databases to accomplish this.

A federated database system or federated system is a database management system (DBMS) that supports applications and users submitting SQL statements referencing two or more DBMSs or databases in a single statement. An example is a join between tables in two different DB2 databases. This type of statement is called a distributed request.

A DB2 federated system consists of a DB2 UDB instance, a database that will serve as the federated database, and one or more data sources. The federated database contains catalog entries identifying data sources and their characteristics. A data source consists of a DBMS and data. Applications connect to the federated database just like any other DB2 database. See Figure 2-2 for a visual representation of a federated database environment.

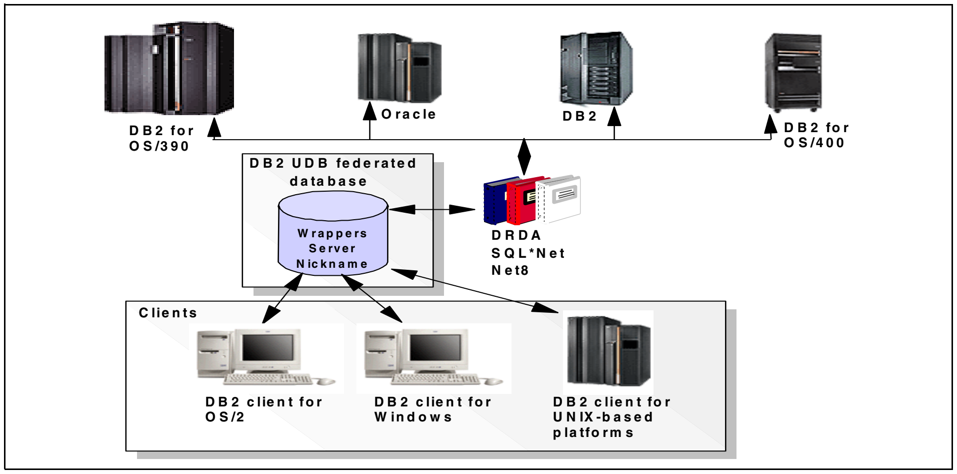


Figure 2-2 DB2 Federated Database configuration

A DB2 federated system provides location transparency for database objects. If information (in tables and views) is moved, references to that information (called nicknames) can be updated without any changes to applications that request the information. A DB2 federated system also provides compensation for DBMSs that do not support all of the DB2 SQL dialect, or certain optimization capabilities. Operations that cannot be performed under such a DBMS (such as recursive SQL) are run under DB2.

DB2 UDB EE V7.2 provides support for distributed requests across databases and DBMSs. You can, for example, perform a UNION operation between a DB2 table and an Oracle view. Supported DBMSs include DB2, members of the DB2 family (such as DB2 for zSeries and DB2 for AS/400) and Oracle.

In our configuration, where we wanted to use a single SQL statement to move rows from a table in DB2 on z/OS to a table in a data mart database on Linux, we added federated database definitions to the data mart database. So in our case, our data mart databases were federated databases.

We created nicknames in each data mart database on Linux, representing the tables on z/OS. Each nickname refers to a set of Server, Node, User Mapping, and Wrapper definitions, which tell DB2 on Linux where in the network the remote table is located, what protocol should be used to retrieve the data, and what user authority should be used to connect to the remote DB2.

Figure 2-3 illustrates how the nickname and supporting definitions of a federated database setup work. In this example, user db2inst1, who is connected to a local database on a Linux system, inserts rows into a local table using SQL with a nickname that points the target of the select statement to a table in DB2 on z/OS. In the example, the table referred to as TABLENICK in DB2 on Linux is actually TABLEZOS located in DB2 on z/OS.

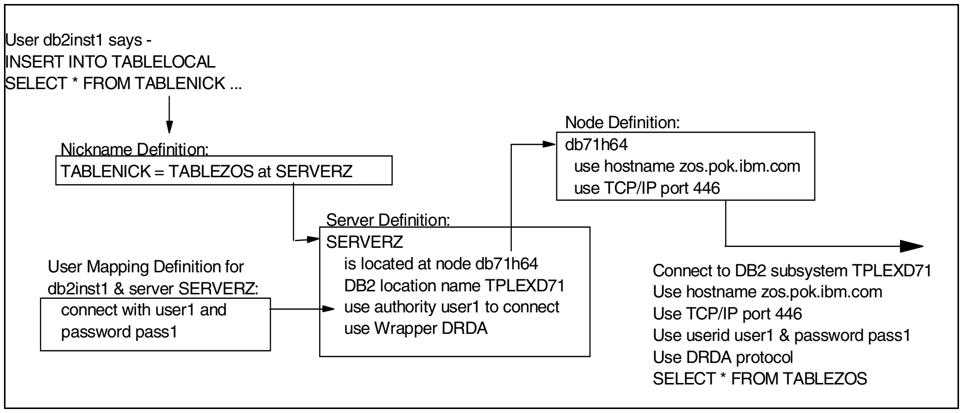

User db2inst1 says -
INSERT INTO TABLELOCAL
SELECT * FROM TABLENICK ...
Nickname Definition:
TABLENICK = TABLEZOS at SERVERZ
Node Definition:
db71h64
use hostname zos.pok.ibm.com
use TCP/IP port 446
Server Definition:
SERVERZ
is located at node db71h64
DB2 location name TPLEXD71
use authority user1 to connect
use Wrapper DRDA
User Mapping Definition for
db2inst1 & server SERVERZ:
connect with user1 and
password pass1
Connect to DB2 subsystem TPLEXD71
Use hostname zos.pok.ibm.com
Use TCP/IP port 446
Use userid user1 & password pass1
Use DRDA protocol
SELECT * FROM TABLEZOS

Figure 2-3 Federated database example

We also created federated database definitions allowing SQL to be run from DB2 on Linux joining rows from local tables with those from tables in data marts on other Linux images. Refer to Appendix D, “Federated database setup” on page 121 for the federated database definitions used in this project.

3

Test scenarios and measurement methodology

This chapter introduces the test scenarios and measurement methodology used to demonstrate the capabilities of zSeries as a platform to consolidate the enterprise data warehouse and all the functional data marts. This chapter includes a description of the following:

- Test scenarios
  - Data population
  - Data refresh
  - Distributed Join
  - z/VM resource allocation across Linux on zSeries guests
  - BI solutions for Linux on zSeries
- Measurement methodology
  - Data collection techniques
  - Measurement process
- Data model
  - Data warehouse data model
  - Data mart data model

3.1 Test scenarios

We designed the following test scenarios:

- Data population
- Data refresh
- Distributed data mart access
- z/VM resource allocation across Linux guests
- BI solutions for Linux on zSeries

3.1.1 Data population

Data population for our tests includes extracting data from the data warehouse on a z/OS LPAR, transferring the data to the data mart on a Linux system and populating the data mart.

Data population is an important issue in building a data mart. Even though typically a data mart is only populated once, large amounts of data may be manipulated between the data warehouse and the data mart, so it is important to understand the characteristics of different population techniques. We take a look at two different techniques.

- Utility based using unload, FTP, Load
- SQL based using insert/select and federated databases

Another important aspect of the data population from a server consolidation standpoint is the amount of traffic sent over the LAN to populate the data marts. Sending 100, 30 or even 15 GB over the LAN to populate just one data mart can cause significant delays to other network traffic. Using the zSeries platform to consolidate both the enterprise data warehouse and the functional data marts, it is possible to bypass the LAN, avoiding unnecessary delays to other applications that need to use the LAN. For more details on the data population test scenarios see the chapter on Data Population.

3.1.2 Data refresh

Data refresh is very similar to data population but involves smaller amounts of data. Data is refreshed or added to an existing data mart on a regular basis. This is typically on a monthly, weekly, or daily basis. For our tests the time frame is irrelevant. For our measurement we are executing a single refresh period, for example, one week's worth of data.

Similar to data population, we are extracting data from the data warehouse on a z/OS LPAR, transferring the data to the data mart on a Linux system and adding it to the data mart. The three main differences between population and refresh are:

- Refresh is done on an ongoing basis as opposed to a one-time population.
- Refresh deals with smaller amounts of data.
- Refresh adds data to an already populated data mart.

For the data refresh scenario we utilized the insert/select technique using the federated database technology. Even though the amount of data for data refresh is typically less than the data used for the initial population, it can be substantial.

It is important to limit the network traffic over the LAN, to avoid disrupting other applications using the same LAN resources. That is one of the benefits of consolidating the data marts on the same zSeries server as the data warehouse. Using HiperSockets, the data does not go out on the LAN.

We also demonstrated the scalability of the zSeries by refreshing multiple data marts concurrently. We refreshed one, then two, and finally three data marts concurrently. For more details on data refresh test scenarios see the Data Refresh chapter.

3.1.3 Distributed data mart access

We can join different data marts using a distributed request. A distributed request is the ability to access different databases in a single SQL statement. Using distributed requests we joined:

- A Linux on zSeries data mart to another Linux on zSeries data mart
- A Linux on zSeries data mart to a z/OS data warehouse

In both cases we used the built-in federated database function of DB2 for Linux on zSeries to accomplish this. We executed several SQL joins to query data across Linux data marts on the same zSeries server qualifying varying amounts of data. Additionally, we simulated a “power” user submitting a query from a Linux data mart, joining it with the z/OS data warehouse. This “power” query qualified millions of rows from the data warehouse that were joined with rows in the data mart and presented to the user. For more details on the distributed join test scenarios, see the Distributed Join chapter.

3.1.4 z/VM resource allocation across Linux guests

In a consolidated environment it is extremely important to be able to manage and prioritize the system resources. In our consolidated environment not all data marts are created equal. Some data marts may have different budgets for their IT processing needs, defining the amount of system resources allocated to it. z/VM can effectively utilize all system resources by assigning them when and where they are needed, across all the Linux on zSeries guests. Traditional separate isolated data marts do this with a high cost of wasted "white space" or under-utilized systems.

Our test scenarios demonstrate the intelligent resource allocation of z/VM across all the Linux guests in these three areas:

- Allocating resources based on individual business unit needs
- Reducing costs by utilizing "white space"
- Managing individual business unit fixed budgets

For more details on the test scenarios that demonstrate the capabilities of z/VM resource allocation across Linux guests, see Chapter 7, "z/VM resource allocation across Linux guests" on page 71.

3.1.5 BI solutions for Linux on zSeries

This test scenario demonstrates that IBM business partner tools (Dimensional Insight in particular) can and do run on Linux on zSeries. The Dimensional Insight suite of tools that we used for this test scenario is a Web-enabled OLAP solution that has been developed on Linux for eight years. We demonstrated through the "porting", installation, and use of this third party's suite of tools that BI products run effectively on Linux on zSeries. Some of the highlights of this scenario are:

- Migrating or "porting" to Linux on zSeries with a simple recompile
- Building a 20 million row sphere
- Demonstrating a high concurrency of users and throughput of reports

For more details on the test scenarios that demonstrate the ability of Linux on zSeries to support third parties' BI tools and applications, see Chapter 8, "BI solutions for Linux on zSeries" on page 81.

3.2 Measurement methodology

This section describes the measurement methodology we utilized for this study.

3.2.1 Data collection techniques

We used several monitors on z/OS, Linux and z/VM to do performance analysis of the test cases. During the measurement interval we used batch monitors on z/OS to capture key DB2 and RMF performance data. On the Linux system, we collected performance data with VMSTAT, IOSTAT, DB2BATCH and sar for Linux. On z/VM we collected CP monitor data using VMPRF. In addition to the batch monitors, we used various online monitors for real-time dynamic analysis.

Batch performance monitors for z/OS

We used the following performance tools on z/OS:

Resource Measurement Facility (RMF) I is used to capture and report the following key system performance information on z/OS for each measurement:

- CPU Utilization
- Channel Activity
- DASD Activity
- Paging Activity
- Workload Activity: Resource consumption (Processor, I/O, Storage)
- Virtual Storage Usage

DB2 Trace Facility is used to capture the following key DB2 performance information on z/OS for each measurement:

- Application-time, DB2-time, Wait-time for the Query Workload
- Buffer Pool Activity
- DB2 Sort Activity
- Parallelism Activity
- Locking Activity

Specific DB2 traces to be started on z/OS are:

- Accounting Classes: 1, 2, 3
- Statistics Classes: 1,3,4,5
- Monitor Classes: 1
- Performance Class: 31 IFCID (22,63,221,222,231)

DB2 Performance Monitor (DB2PM) is used to generate reports based on the DB2 trace information captured.

Online performance monitors for z/OS

We used the following online performance tools on z/OS:

- **RMF monitors II & III** to dynamically monitor the same metrics captured by the Batch RMF monitors
- **DB2PM Online** to dynamically monitor DB2 performance data
- **Omegamon for MVS and DB2** to dynamically monitor system and DB2 activity

Performance monitors for Linux on zSeries

We used the following performance tools on Linux on zSeries:

- **VMSTAT** to monitor the CPU and memory activity.
- **IOSTAT** for monitoring system input/output device loading by observing the time the devices are active in relation to their average transfer rates.
- **DB2BATCH** to gather information similar to what DB2PM on z/OS provides. Db2batch with perf_detail 4 was run to obtain elapsed and CPU times. Perf_detail 4 returns a snapshot for the database manager, the database, the application, and the statement (the latter is returned only if auto commit is off, and single statements, not blocks of statements, are being processed). The only time we turned off auto commit was for the insert select data refresh and data population tests.
- **sar** to gather selected cumulative activity parameters in the operating system, such as CPU utilization, I/O transfer rate, paging and network activity and load averages.

Performance monitors for z/VM

We used the following performance tools on z/VM:

- **VMPRF** to gather system performance reports
- **Real Time Monitor** to get a view of current system performance

3.2.2 Measurement process

The steps taken for each test case measurement are as follows:

1. Verify that the appropriate DB2 for z/OS traces are on.
2. Display/validate the hardware configuration on z/OS.
3. Display/validate buffer pool settings on z/OS.
4. Check whether all DB2 for z/OS index space and tablespace data sets are open.
5. Verify that the appropriate system and DB2 for z/OS settings are correct.

6. Verify that the appropriate RMF traces are on z/OS.
7. Validate that the appropriate monitors are started on z/OS.
8. Set the appropriate WLM Policy on z/OS.
9. Start vmstat, iostat, and sar on Linux systems and direct output to files.
10. Start collecting and writing CP monitor data on z/VM using VMPRF if running measurement on Linux guests.
11. Start the DB2 for z/OS statistics interval.
12. Start the workloads to be measured.
13. Wait a specific measurement interval or until the end of the workload.
14. End the DB2 for z/OS statistics interval.
15. Stop vmstat, iostat, and sar.
16. Save SMF data.
17. Process CP monitor data with VMPRF if measurement is on Linux guests.
18. Save VMPRF data if running the measurement on Linux guests.

Each measurement has a unique *runid* associated with it. If multiple measurements are run for one test case, they can be identified from the runid.

3.3 Data model

This section describes the data warehouse and data mart data models and puts them in context of a possible business scenario.

3.3.1 Data warehouse data model

The data warehouse on DB2 for z/OS is designed with one database containing eight partitioned table spaces. By definition, each partitioned table space contains one table. Figure 3-1 on page 32 depicts the data warehouse data model. To make the data model easy to understand we put it in the context of a manufacturer.

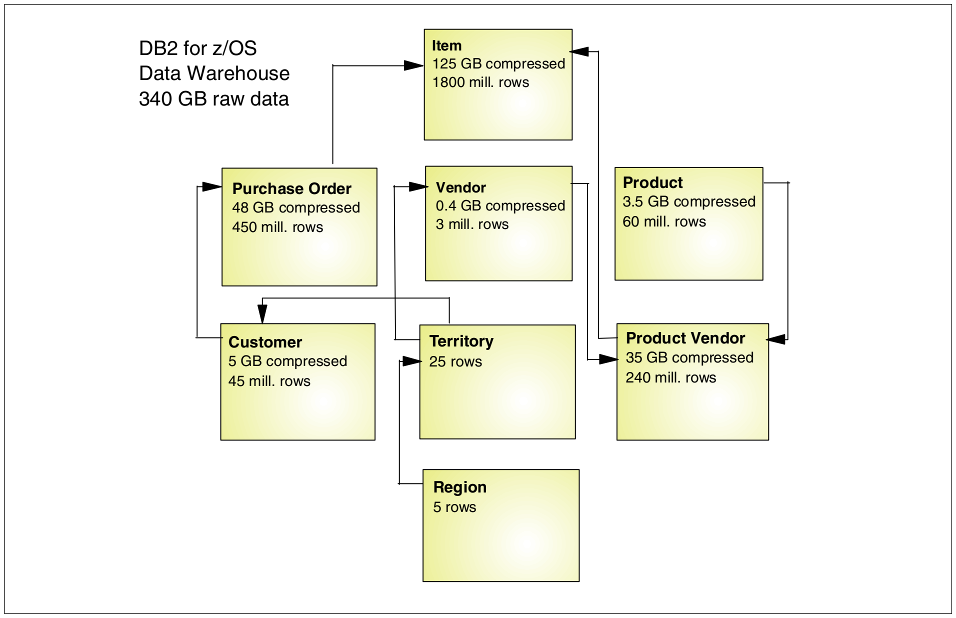


Figure 3-1 Data warehouse data model on DB2 for z/OS

3.3.2 Data mart data model

The four data marts on z/Linux have identical data models. For simplicity we created them the same as the data warehouse, only smaller. Each data mart is a subset of the data warehouse. Figure 3-2 on page 33 describes the data mart data model.

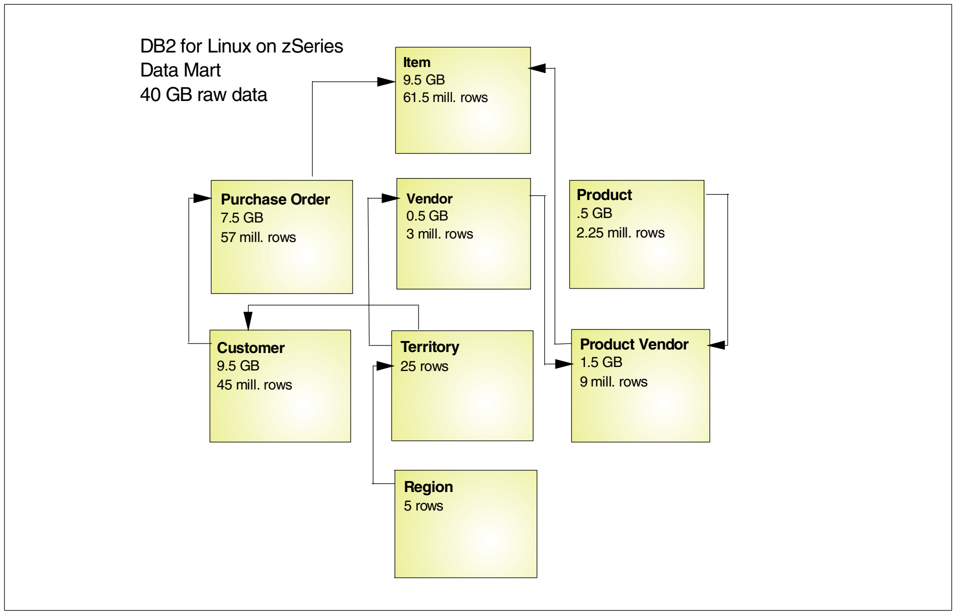


Figure 3-2 Data mart data model on DB2 for Linux on zSeries

Customer, vendor, territory, and region tables have exactly the same data as the data warehouse, and they are the same across all data marts. The number of rows and column widths are identical but the sizes of the tables are larger. This is because the z/OS data warehouse was able to take advantage of hardware compression, but the Linux on zSeries data marts do not have that function. Purchase Order, Item, Product, and Product Vendor tables are a subset of the data in the warehouse. All three data marts contain five years of data, but they contain different products.

This is not to imply that a typical manufacturer would necessarily model their business in this manner. Without having a real customer to work with, it is easy to demonstrate the concept of multiple data marts.

4

Data population

This chapter focuses on selected data population techniques between data warehouse and data marts consolidated on the same zSeries platform.

A large portion of time related to implementing and maintaining a data mart,is devoted to the movement of the data from one platform to another. The primary source of data for decision support systems is most often the data warehouse on zSeries.

Consolidating several data marts in Linux LPARs on the same physical platform as the data warehouse, brings the data marts into close proximity to the DB2 source systems. Now the transfer of data to the marts is done across LPARs, through memory, using HiperSockets or an OSA Express Gbit card. This eliminates the overhead of network traffic and reduces the time to load the marts.

These hardware features can be combined with software solutions for the rapid population of the data marts in Linux LPARs. DB2 data load/unload tools and utilities from IBM, such as the DB2 High Performance Unload and Extract-Transform-Move-Load (ETML) packaged tools, and DB2 UDB EE's support for distributed requests across DBMSs, take advantage of these internal data transfer mechanisms between LPARs to further reduce the time required to maintain the currency of data marts.

This adds up to being able to populate more data marts with more data in a shorter amount of time.

For the study we conducted at the zSeries Teraplex Integration Center, we focused on two different data population techniques:

- SQL-based

  The first technique was implemented using a Distributed SQL statement enabled by a federated database.
- DB2 utility-based technique

  This is a combination of DB2 High Performance Unload utility (HPU), File Transfer Protocol (FTP), and DB2 Load utility.

Two test scenarios were created for each of the data population techniques. We varied the type of connection between the z/OS LPAR, where the data warehouse resides, and the Linux LPARs, where data marts are built. Since the HiperSockets feature is only available on zSeries, running both techniques using the traditional LAN connection will provide a more comprehensive coverage for this study.

- HiperSockets

  Also known as IQDIO or internal QDIO, HiperSockets offers a significant value in server consolidation connecting many virtual servers. In our test environment, we used IQDIO to connect the three z/VM Linux guests to each other and to z/OS.
- OSA Express GBit Ethernet card

  To simulate a traditional LAN connection to a server on a network, the native Linux LPAR was connected to z/OS using the OSA Express GBit Ethernet card.

For each of the measurements. we collected the elapsed time, CP consumption and number of rows processed.

The test scenarios described above were designed in order to meet the following objectives:

- Demonstrate the behavior of the data population process when the z/OS data warehouse and Linux data marts are consolidated on the same zSeries server
- Contrast this behavior with the data population process where the z/OS data warehouse and Linux data marts are on physically separate servers.
- Compare the performance where it is relevant and show the pros and cons of each data population solution.

4.1 Data population techniques

Several data population techniques can be used to move data between a data warehouse and data marts. The data population process is also referred to as the Extract-Transform-Move-Load (ETML) process. We can categorize these techniques as follows:

- SQL-based
- Utility-based
- ETML tools

When dealing with multiplatform DB2 databases, one should always make sure that the CCSID value specified in the DB2 for z/OS is one that DB2 for Linux can convert. For example, in our DB2 for z/OS, we specified the following values:
SCCSID=37
MCCSID=65534
GCCSID=65534
MIXED=NO

4.1.1 SQL-based

Two SQL-based techniques for populating tables in a data mart are:

- A single distributed INSERT/SELECT SQL statement

  This requires that a federated database is set up on the DB2 for Linux data mart. Once this is done, an INSERT/SELECT statement can be issued from the Linux LPAR to populate the data mart.

- A program with Select/Insert statements

  A program can be written to extract data from the data warehouse and insert it in the data mart. The program issues a Select statement to read rows from the source table or tables and inserts them into the target table.

We chose to run our data population SQL-based test cases using the distributed INSERT/SELECT SQL technique. This method is efficient and easy to implement. Setting up a federated database on each of the DB2 for Linux LPARs helped us prepare for other test cases in this study such as distributed joins and data refresh.

You may also refer to redbook *e-Business Intelligence: Leveraging DB2 for Linux on S/390,* SG24-5687 for descriptions of other SQL techniques for data population.

4.1.2 Utility-based

The utility-based data population technique is a combination of DB2 utility, file transfer protocol and DB2 Load utility. The following are a few examples of such combinations:

- DB2 Export and Import utilities can move data between DB2 tables. The DB2 Export utility moves data from source DB2 tables into flat files. You can then use those files to Import or Load that data into the target DB2 table.
- The DB2 Load utility moves data from flat files into target DB2 tables. DB2 Load moves data much faster than DB2 Import utility when large amounts of data are involved. Data unloaded with DB2 Export utility can also be loaded with the DB2 Load utility.
- The Autoloader utility splits large amounts of data and loads the split data into different partitions of a partitioned database.
- DB2 DataPropagator (DPROP) is a component of DB2 that allows automatic copying of updated tables into other DB2 tables.
- The DB2 DSNTIAUL utility can read data from a DB2 for z/OS table and put it in a sequential file in the format requested by the DB2 Load utility. You can then FTP this file to the data mart site and load it using the DB2 Load utility.
- The DB2 Unload utility unloads data from a DB2 for z/OS table. This high performing utility became available with DB2 V7. The unloaded data can then be FTPed to the data mart site and loaded there using the DB2 Load utility.
- The DB2 cross loader function is also new in V7. It is part of the DB2 Load utility and allows you to read data from one DB2 platform and load it on another DB2 platform.
- The DB2 High Performance Unload (HPU) is a high-speed DB2 utility that unloads data from DB2 for z/OS. This utility supports a large array of options including complex SQL and joins. Here, too, you can FTP the unloaded data to the data mart and load it there using the DB2 Load utility.

The utility-based data population techniques are also described in redbook *e-Business Intelligence: Leveraging DB2 for Linux on S/390,* SG24-5687.

For our DB2 utility-based technique test cases, we chose HPU to unload data from DB2 for z/OS tables. HPU is a high-speed DB2 utility which can unload DB2 tables from either table spaces or an image copy. It can also unload multiple tables and partitions at the same time. It also has the capability to create the output files in a delimited format (which proves to be more efficient to transmit and load to a table). And most importantly, HPU supports complex SQL such as correlated subqueries and joins.

4.1.3 ETML tools

There are several Extract-Transform-Move-Load (ETML) packaged tools available in the market.

The DB2 Data Warehouse Manager and Data Warehouse Center (DWC) are examples of ETML tools which can be used to move data from operational databases into a data warehouse or a data mart.

There are several other third-party vendor ETML packaged tools that are not mentioned or described in this book.

4.2 SQL-based distributed INSERT/SELECT SQL

Data population using a distributed INSERT/SELECT SQL statement is very efficient and simple to implement. This SQL-based technique is enabled by the federated database on DB2 for Linux.

This test case involves populating the 40 GB data mart on a Linux LPAR using a distributed SQL to select rows from data warehouse tables on z/OS and insert the selected rows into a table in the data mart. A total of eight distributed SQLs were created to populate the data mart, one SQL for each table. Refer to 3.3, “Data model” on page 31 for detailed information on both data warehouse and data mart tables.

All the data marts were set up as a federated database to support this test scenario as well as the other test cases for this study. The implementation details of the federated database setup are in Appendix D, “Federated database setup” on page 121.

In order to optimize the use of available resources, all of the eight SQLs have to run at the same time. To implement this, we wrote a script to start, run the SQLs, and stop the performance monitors when all the SQLs complete.

There are two variations of this SQL-based data population scenario, as shown in Figure 4-1 on page 40, one populating the z/VM Linux Guest1 using HiperSockets connections, and the other populating the data mart on the Native Linux server using LAN connections.

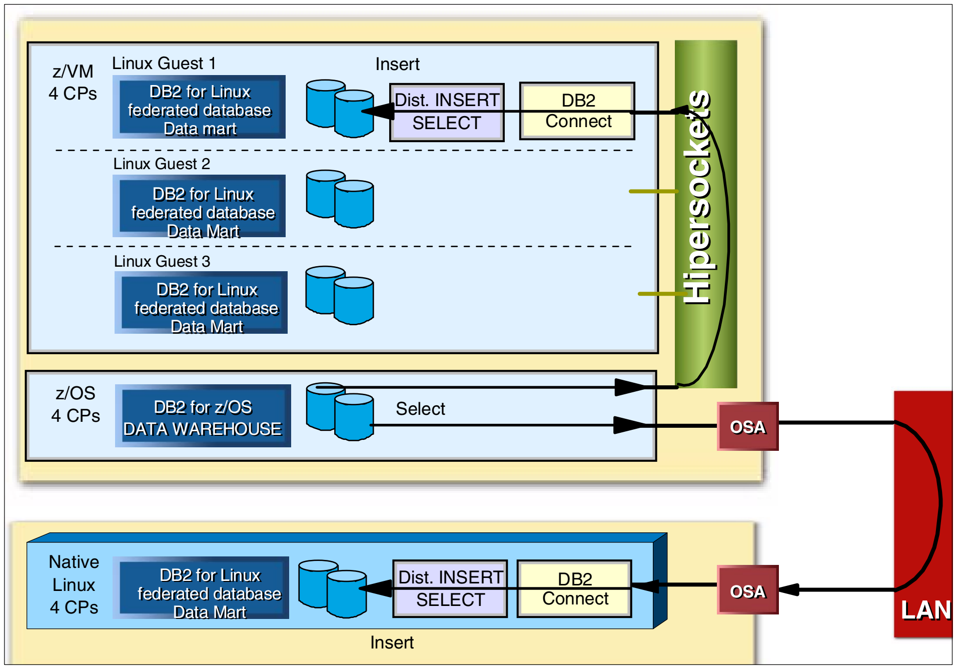

z/VM
4 CPs
Linux Guest 1
DB2 for Linux federated database Data mart
Insert
Dist. INSERT SELECT
DB2 Connect
Linux Guest 2
DB2 for Linux federated database Data Mart
Linux Guest 3
DB2 for Linux federated database Data Mart
Hipersockets
z/OS
4 CPs
DB2 for z/OS DATA WAREHOUSE
Select
OSA
Native Linux 4 CPs
DB2 for Linux federated database Data Mart
Dist. INSERT SELECT
DB2 Connect
Insert
OSA
LAN

Figure 4-1 Data population distributed INSERT/SELECT SQL scenario - LAN and HiperSockets connections

Each server—z/VM, z/OS and Linux—has four processors each. The three Linux Guests share four processors under z/VM.

4.2.1 SQL performance

Eight SQLs were written to populate the 40 GB data mart on DB2 for Linux. The “Select” query in each SQL varies according to how much and from where data needs to be extracted from the z/OS data warehouse table.

For some tables, a simple SELECT * query is used to extract all the data needed to populate the corresponding tables in the data mart, such as the Region, Territory, Vendor, and Customer. This makes all these tables the same for all data marts and the data warehouse.

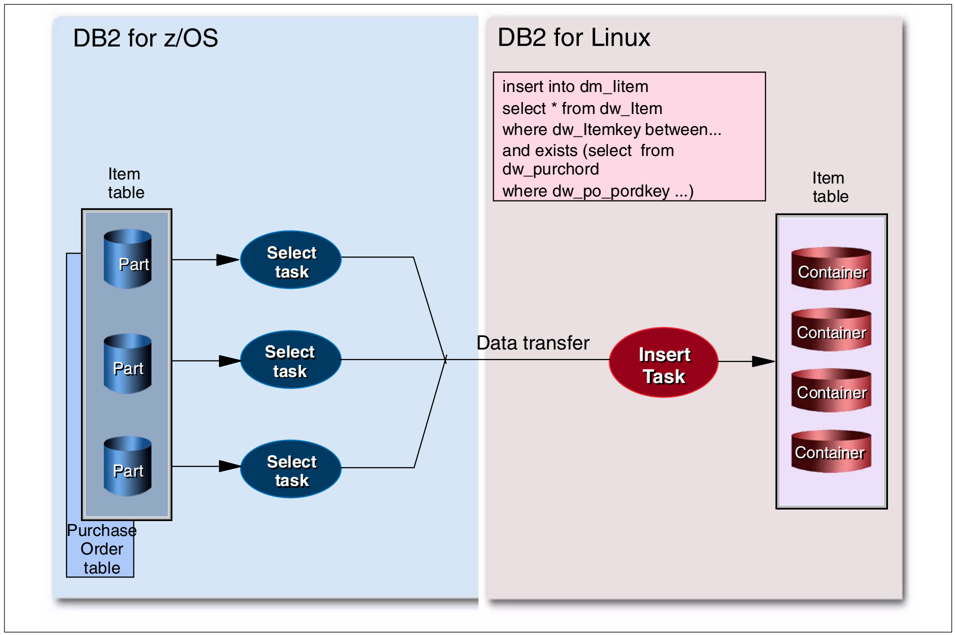
We used a predicate in the SELECT * query to qualify only a range of product numbers from the z/OS data warehouse into the Product and Product Vendor tables in the data mart. The predicate values are different for each data mart that is populated because each represents a different set of products..

We used a correlated subquery to extract data from the z/OS data warehouse in the SELECT query for the Item and Purchase Order tables.

Although the SELECT query varies according to which table in the data mart needs to be populated, the data flow for all the distributed SQLs is the same.

Figure 4-2 shows the data flow of the distributed INSERT/SELECT SQL we used to populate the Item table.

.



```
insert into dm_litem
select * from dw_Item
where dw_Itemkey between...
and exists (select  from
dw_purchord
where dw_po_pordkey ...)
```

Figure 4-2 Distributed INSERT/SELECT SQL data flow

This SQL selects all the items ordered from a group of manufacturers within a five-year period. To extract the rows from the z/OS data warehouse and insert the qualified rows into the Item table in the data mart, a distributed INSERT/SELECT SQL is issued from DB2 for Linux. The actual SQL is shown in E.1, “Distributed SQL INSERT/SELECT” on page 127.

The SELECT query, as seen in Figure 4-2, creates parallel tasks on DB2 for z/OS because the Item table is defined in a partitioned table space. These tasks represent the concurrent reads on the table, which means a much faster scan of data on z/OS. On the other hand, the data transfer and insert of the qualified rows in DB2 for Linux table are sequential although the Item table is defined in multiple containers.

Access path

When the distributed SQL is issued from DB2 for Linux, the SELECT query is executed first. Since all the tables in the z/OS data warehouse were created in a partitioned table space, DB2 optimizer can choose a maximum degree of parallelism equal to the number of partitions on each table.

For our test scenario, we populated all the tables in the data mart at the same time. Since we wanted to maximize CP utilization without consuming an excessive amount of virtual storage while executing this measurement, we capped the maximum degree of parallelism at 20. To implement this, we specified PARAMDEG = 20 in the DB2 for z/OS ZPARM.

When a distributed INSERT/SELECT SQL is issued on DB2 for Linux, it goes through a two-phase optimization process: First on DB2 for Linux and second on DB2 for z/OS. Optimization in Linux is rule-based, while it is cost-based on z/OS.

When dealing with this type of distributed query, we have to be aware of the possibility of change in access path during the second phase of optimization. For example, in our distributed INSERT/SELECT SQL, we used a BETWEEN predicate to filter the rows to be selected. DB2 for Linux optimized this query by changing the predicate to a GREATER THAN OR EQUAL TO and LESS THAN OR EQUAL TO predicate. Using this new predicate, DB2 for z/OS reoptimized this query and chose a table space scan access path. If DB2 for Linux optimizer did not change the original predicate of the query, DB2 for z/OS would have chosen an index-only access path, which is more efficient. In order to force an indexed access path, we updated the catalog statistics on DB2 for z/OS.

Index-only access is better when selecting a subset of rows from a large table, such as the queries we used for the Item, Purchase Order, Product, and Product Vendor tables. However, when selecting all the rows in a table, table space scan access gives better performance. The queries for the Customer, Vendor, Territory, and Region use a tablespace scan.

Locking

We used table-level locking for all the tables on DB2 for Linux for the duration of the data population measurement. This is appropriate for this scenario because we have a single INSERT/SELECT SQL for each table, and concurrency with user access is not an issue since this is an initial load on each table.

If we used row-level locking for example, it would have required too much virtual storage to record each lock and some overhead to manage all of the locks for each table. For example, 61.5 million locks, one for each row, would have been needed for the Item table alone. One table lock is more efficient in a data population scenario when the whole table is loaded for the first time.

Logging

On DB2 for Linux we created each table with the "not logged initially" option. When activated it will turn off logging for the unit of work (UOW). We do not think it is necessary to do logging while populating the tables for the first time. To enable this option, we included an ALTER TABLE TABLE tblname ACTIVATE NOT LOGGED INITIALLY statement just before the INSERT/SELECT SQL statement. To execute both of these statements as one UOW, we used the `-c off` option in the **`db2batch`** command. This way, the commits between each statement were turned off.

By turning off the log option in this data population scenario, we prevented DB2 for Linux from writing 40 GB of data twice, once on the log file and once on the table. With the "not logged initially" option, data is written only on the table. This avoids flooding the logs, wasting considerable disk space, and slowing down the whole insert process. A full image copy is taken at the end of the load to make the table recoverable. In case of failure, the insert is restarted from the beginning.

4.2.2 Workload performance

We run one distributed INSERT/SELECT SQL statement per table on DB2 for Linux to populate each table of the data mart as summarized in Table 4-1.

Table 4-1 Data population using distributed SQL INSERT/SELECT workload

	# of SELECT statements on z/OS	**# of INSERT statements on Linux**
Item	1	1
Purchase Order	1	1
Customer	1	1
Product	1	1
Vendor	1	1
ProductVendor	1	1
Region	1	1
Territory	1	1
Total workload	8	8
Concurrency level	8	8

To take advantage of I/O and CPU cycles, we populated all 8 tables (40 GB data) by running the 8 distributed INSERT/SELECT SQL statements concurrently. E.1, “Distributed SQL INSERT/SELECT” on page 127 shows the INSERT-SELECT data population scripts we used to load each table of the data mart.

Figure 4-3 illustrates the execution of the data population workload.

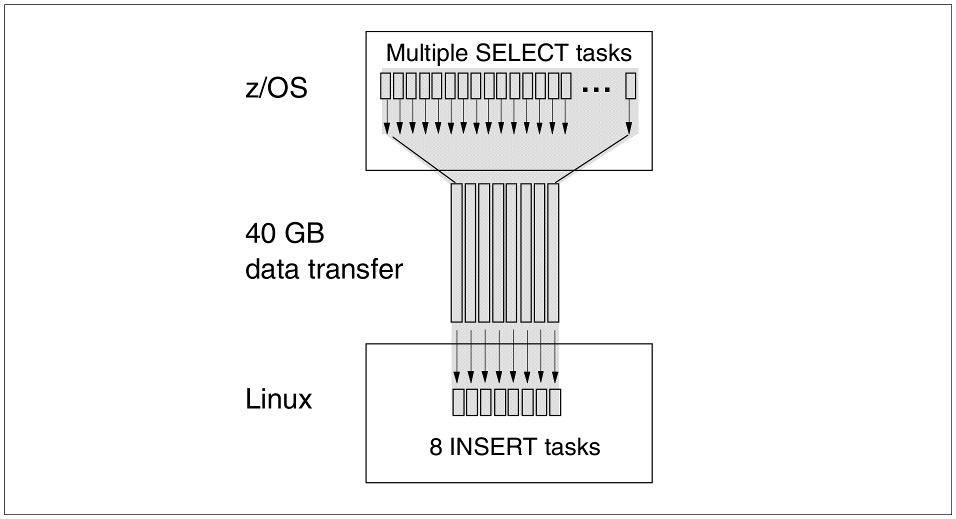


Figure 4-3 Data population using distributed SQL INSERT/SELECT workload execution

Allowing each SELECT to further parallelize on z/OS maximizes the CPU utilization on z/OS, which in turn maximizes the network throughput. As the 8 concurrent SELECT tasks parallelize on DB2 for z/OS, the data is retrieved much faster, thus producing more data to be sent at once over the network. This results in each one of the INSERT tasks on Linux having a full queue of data to be inserted into their respective table. As a result, the Linux CPs are used effectively, since we have at least as many tasks as we do CPs.

Figure 4-4 on page 45 summarizes the results of data population using the distributed SQL INSERT/SELECT technique. The charts show the elapsed time (in seconds) and throughput rate (rows processed per second) when the workload was run with LAN and HiperSockets connections.

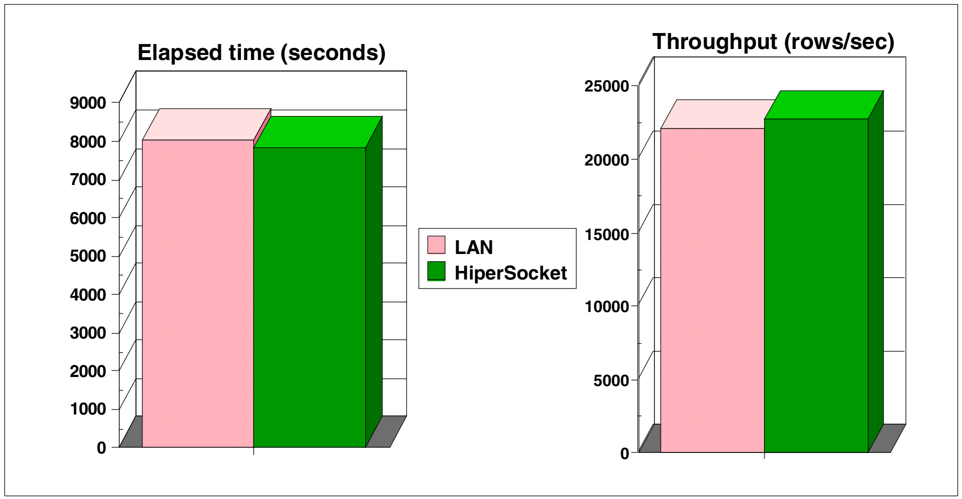


Figure 4-4 INSERT/SELECT elapsed time and throughput - LAN vs HiperSockets

On the average, it took slightly more than 2 hours to populate the 40 GB datamart. It took 134 minutes to extract data from the z/OS data warehouse and insert 177,842,562 rows into the datamart using LAN connection, and 130 minutes using HiperSockets. The throughput rate was 22,106 and 22,725 rows per second on the LAN and HiperSockets, respectively.

Data transfer using HiperSockets had a 3% improvement impact on this particular process. There was no significant difference in elapsed time and throughput when using LAN and HiperSockets because the network speed was not the constraining factor in this process. Most of the time was spent on DB2 on z/OS and DB2 on Linux.

4.3 Utilities - HPU/FTP/LOAD

Data population with HPU/FTP/LOAD is a batch process that can be executed in three steps. The advantage of this solution is that it can be broken down into smaller steps, each step being conducted at different times depending upon the availability of the system resources. For example, we can FTP data overnight when the network traffic is light, or when there is less network traffic.

DB2 High Performance Unload (HPU) can also do multiple unloads from image copy data to eliminate interference with DB2 production databases. HPU spawns multiple processes to read a particular tablespace in parallel.

The three steps involved in the HPU/FTP/LOAD scenario are shown in Figure 4-5. They are:

- Unload of data from the data warehouse on z/OS to sequential files
- Transfer of the sequential files from z/OS to the data marts in Linux
- Load of the sequential files into the data mart tables in Linux

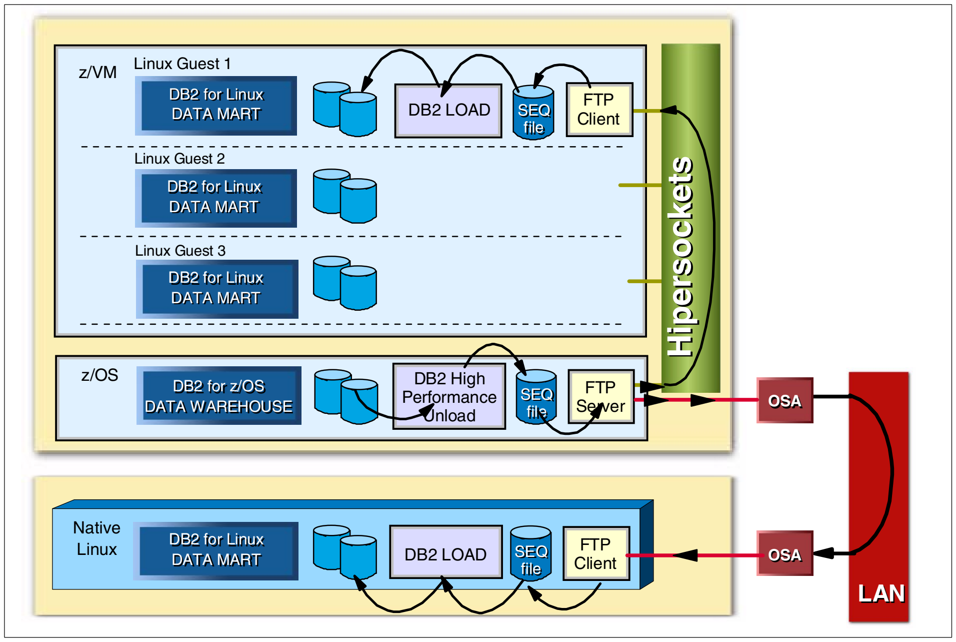


Figure 4-5 Data population HPU/FTP/LOAD scenario - LAN and HiperSockets connections

For this test case scenario, several HPU, FTP and DB2 Load utility jobs were created to enable multiple stream processing.

The number of HPU jobs created for each table varies according to the amount of data extracted from the data warehouse. For example, one HPU job is sufficient for smaller tables such as Region, Territory, Product, Vendor and Product Vendor. But for the larger ones, we have to create about 8 to 16 jobs, depending on the table, in order to split the output into smaller sequential files. The total size for the Customer, Purchase Order and Item table is close to 9 GB each. The SQL we used for the HPU is very similar to those we used for the distributed SQL test case scenario.

Item is the largest table in the data warehouse. We unloaded close to 9 GB of data from this table, transferred these files to Linux and loaded them to the Item table in the Linux data mart. Figure 4-6 on page 47 shows these processes.

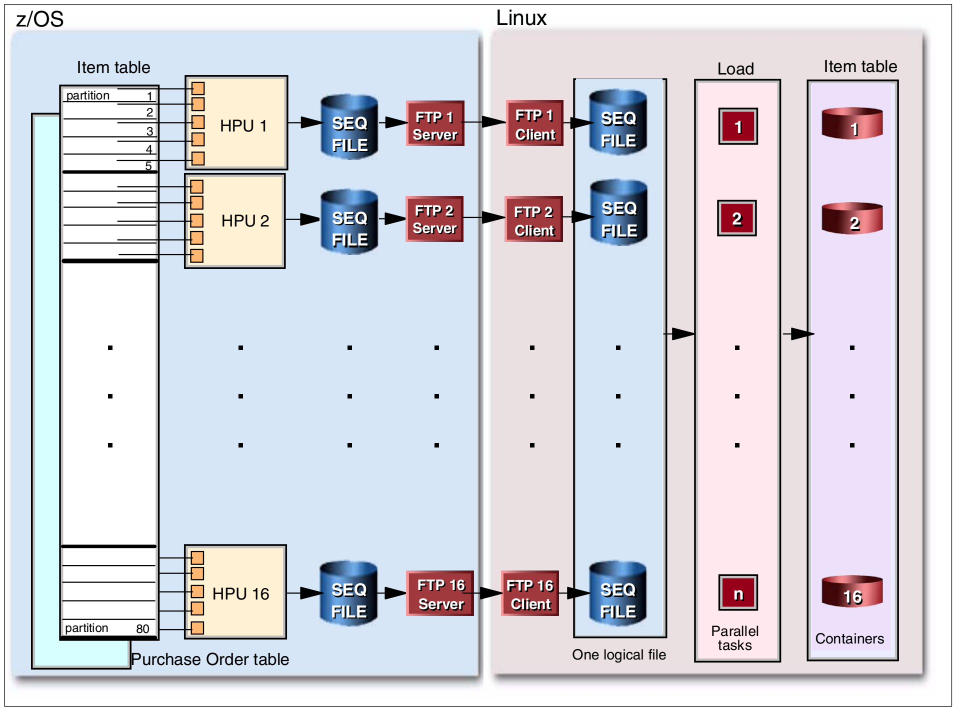


Figure 4-6 HPU/FTP/LOAD data flow for Item table

4.3.1 HPU

HPU is a utility that unloads data from a DB2 table to a sequential output file. It can unload multiple tables from the same tablespace and output each of them into separate files. It enables the user to specify a different format for these output files. We chose the delimited format for our test case for efficiency in FTP and Load, and ease in setting up the LOAD script.

HPU supports complex queries and subqueries to read data. It exploits the direct use of VSAM's buffering capability, which lets an entire cylinder to be read in a single I/O.

One of the reasons we used HPU over DB2 Unload and DSNTIAUL is because of its ability to handle complex queries and joins to unload data. For example, to populate the Item table, we unloaded five years of data for selected items from the data warehouse tables. Purchase order date (PODATE) is not a column in the Item table. So to extract the needed rows from the Item table, we have to read the Purchase Order table first to verify that the date of purchase for the order is within five years. If it is, then we scan the Item table until we find the matching Purchase Order (PO). If the same PO exists in the Item table, a record is written to the output file. Otherwise, this record is ignored. This cycle continues until all the rows in the PO table is read.

We created 16 HPU jobs to unload the data from the Item table. Each of these scans five partitions of the Item and Purchase Order tables. We chose to split the unload jobs so that we could have sequential output files of more manageable size.

Refer to E.2.1, “DB2 HPU jobs” on page 129 to see the HPU job utility statements we used.

4.3.2 FTP

The second step in this utility-based HPU/FTP/LOAD scenario is the transfer of the delimited sequential files from z/OS to Linux.

When all the HPU jobs are complete, the sequential files are tranferred from z/OS to the data mart site in Linux, using FTP. The ASCII option is used to translate the EBCDIC data into ASCII format required by the Linux platform. The FTP server runs on z/OS and the FTP client runs on the Linux site. The sequential files were transmitted to a Linux site twice with a variation on the type of connections:

- FTP to Linux Guest1 on z/VM using a HiperSockets connection
- FTP to Native Linux box using a LAN connection

Refer to E.2.2, “Data transfer through LAN connection” on page 132 to see the FTP commands we used in this scenario.

4.3.3 Load

The third and last step for this test case scenario is the load of the sequential files into the DB2 for Linux tables.

The DB2 Load utility on Linux reads all the input files for a table sequentially. It treats all of these as one logical file and loads the data into the table. For example, it treats all 16 sequential files as one logical input for the Item table. The DB2 utility on Linux creates several parallel tasks to load this into the 16 containers of the Item table. There is no 1 to 1 relationship between the Load tasks and the containers. Refer to E.2.3, “DB2 Load utility jobs” on page 132 to see the DB2 Load jobs we used in this scenario.

4.3.4 Workload performance

We populated the 8 tables of each Linux data mart (Native and Guest) as summarized in Table 4-2.

Table 4-2 HPU/FTP/LOAD workload summary for each Linux data mart

	HPU jobs	**FTP commands**	**DB2 Load scripts**
Item	16	16	1
Purchase Order	8	8	1
Customer	8	8	1
Product	1	1	1
Vendor	1	1	1
Product Vendor	1	1	1
Region	1	1	1
Territory	1	1	1
Total	37 HPU jobs	37 FTP commands	8 Load scripts

We ran a total of 37 HPU jobs to extract 40 GB of data from the z/OS data warehouse. Concurrency level for this step is 4, i.e., we ran 4 HPU jobs at a time, since the SQLs for Item and Purchase Order are very CPU intensive. It took 148 minutes to extract all this data from the data warehouse.

The 37 delimited sequential output files from the HPU jobs were all FTP’ed to Linux. All of these files were transmitted concurrently. This process only took 7 minutes and 5 seconds with HiperSockets, and about 11 minutes with a LAN connection.

We ran 8 DB2 Load utility jobs to populate each of the data marts. At any given time, there were 4 Load utility jobs running on the Linux system. We automated this step by using a script which initially submits 4 Load utility jobs, then resubmits another DB2 Load job whenever one ends. This way, we are sure that we are optimizing the use of the Linux CPs.

Figure 4-7 on page 50 shows the elapsed times for each step in the HPU/FTP/LOAD data population technique and the FTP throughput for the two test case scenarios, using HiperSockets and LAN connections. In both scenarios, only the FTP elapsed times and throughput is affected by the type of connection, as shown in the chart on the left.

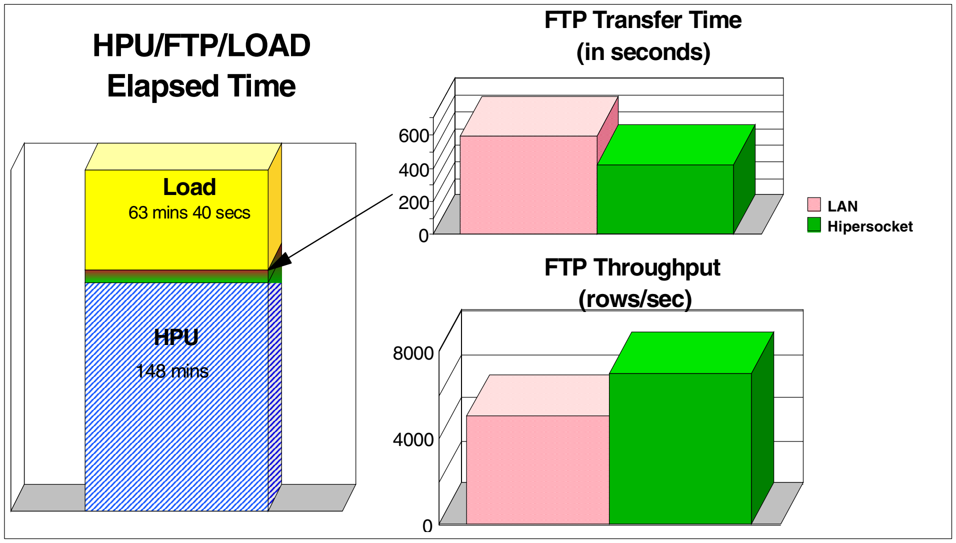


Figure 4-7 HPU/FTP/LOAD workload elapsed times

The two charts on the right compare the FTP workload behavior over the LAN and HiperSockets connections. The comparison shows that transfering 40 GB of data into a Linux data mart over the HiperSockets has shorter elapsed time and higher throughput. The HiperSockets connection is efficient because it is a memory-to-memory data transfer and uses 64K max frame size for each data transfer cycle, unlike the LAN connection, where network traffic slows down data transfer.

The FTP workload consisted of 37 parallel network transfers of the entire 40 GB of data, with source and destination files spread cross the Shark subsystem. With network throughput as a constraint, use of HiperSockets gave a significant elapsed time savings compared to the use of the LAN.

4.4 HPU/FTP/LOAD vs distributed SQL INSERT/SELECT

Figure 4-8 compares the elapsed time and throughput to populate a 40 GB data mart using two different techniques: the distributed SQL INSERT/SELECT versus HPU/FTP/LOAD.

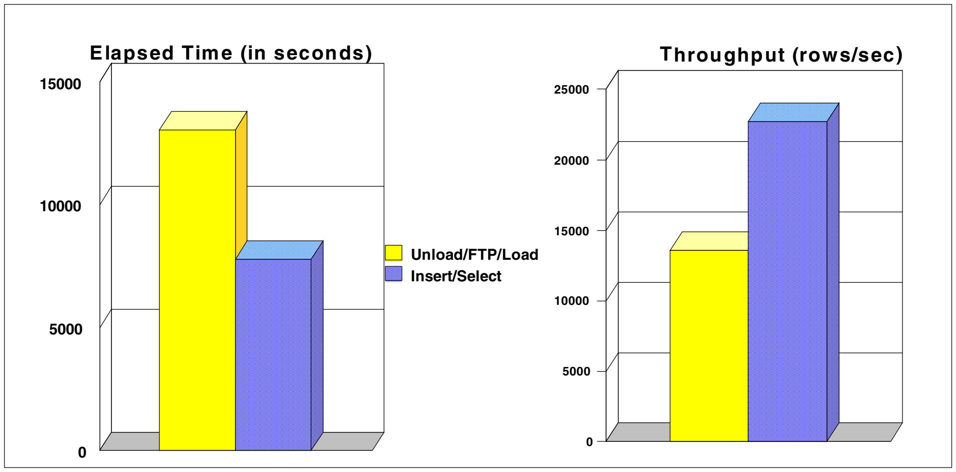


Figure 4-8 HPU/FTP/LOAD vs distributed SQL INSERT/SELECT

This chart tells us that the distributed SQL INSERT/SELECT technique is the more efficient and faster way to populate a data mart. It only took a little bit more than two hours to populate the 40 GB data mart using this method, which is extracting data from a z/OS warehouse and inserting it into a Linux data mart at the rate of 22,725 rows per second.

On the other hand, it took more than three and a half hours to execute a similar process using the HPU/FTP/LOAD technique.

Although the distributed SQL INSERT/SELECT technique is faster and simpler to implement, some users may consider using the HPU/FTP/LOAD because of the following reasons:

- If the process needs to be broken down into 3 separate processes and each process is run at separate times depending on the availability of resources.
- If they want to unload from image copies to avoid contention with online work.

5

Data refresh

This chapter describes data refresh from the data warehouse to the data marts consolidated on the same zSeries server, and the ability to concurrently refresh multiple data marts in order to shorten maintenance windows.

Over time, business users tend to do increasingly more sophisticated analysis on data and the demand for more data grows on a continual basis. The success of initial decision support systems spawns requirements for new business intelligence applications deployed across more functions and departments. This results in the growth of the number of data marts and the amount of data needed. Along with the growth in data and the number of data marts, comes increased demand for 24x7 access to the data marts. This growth presents IT with the challenge of keeping data marts current with decreased time for the data transformation and movement processes to complete. This challenge is complicated even further where there are large numbers of distributed data mart servers.

A large portion of time related to implementing and maintaining a data mart is devoted to the movement of the data from one platform to another. The primary source of data for decision support systems is most often the operational system on zSeries. Recognizing this, solution architects have designed data warehouse, ODS, and data mart infrastructures on DB2 for z/OS, so that they can leverage the proximity of the data to maximize the performance of moving large volumes of data into the enterprise warehouse, ODS, and data marts. Outboard data marts (i.e. data marts on separate servers) have not been able to capitalize on this proximity to the warehouse, and suffer the cost of moving data across networks.

Consolidating several data marts on Linux guests under z/VM on the same physical platform as the data warehouse and ODS, brings the data marts into close proximity to the DB2 source systems. Now the transfer of data to the data marts is done between LPARs, eliminating the overhead of network traffic.

In addition, HiperSockets enable communication between LPARs using TCP/IP over high speed internal data transfer pipes. Data will be moved from one LPAR to another, through memory, creating high internal network bandwidth.

To meet the increased demand for round the clock access to the data marts, maintenance windows need to be as short as possible. The ability to concurrently refresh data marts can shorten maintenance windows. More data marts can be refreshed with more data in a shorter amount of time.

Our test cases measure elapsed time, CPU consumption and throughput. The objective is to demonstrate the scalability capabilities of multiple concurrent refreshes against data marts consolidated on the same zSeries server.

We have three scenarios where we refresh:

- One data mart
- Two data marts concurrently
- Three data marts concurrently

5.1 Data refresh techniques

Data refresh techniques are similar to data population techniques. To refresh data in the data marts, you can use the same SQL-based techniques, utilities and ETML tools as described in 4.1, “Data population techniques” on page 37.

Data refresh is a repetitive maintenance activity executed daily, weekly, or monthly, sometimes concurrently with application workloads running against the same data mart. Depending on the batch window availability you have in your shop, you can design the data refresh process as a batch process or as an online process moving only changed or new data to the data mart. Piped designs and parallel or concurrent executions are performance enhancers for the data refresh process. For more explanations on the data refresh design alternatives, refer to *Business Intelligence Architecture on S/390 - Presentation Guide,* SC24-5641.

5.2 DB2 distributed SQL statement - INSERT/SELECT

For our data refresh scenarios we used distributed INSERT/SELECT SQL statements. This is an online process executing dynamic SQL statements. The advantage of this solution is that it is efficient and simple to implement.

In these scenarios we refresh a total of 2 million rows (290 MB) per data mart. The rows are selected from the Item and Purchase Order tables in the z/OS data warehouse and inserted into the Item and Purchase Order tables in the Linux data marts on z/VM. Each data mart is refreshed with a different set of data since each mart contains different products.

We ran the following three scenarios:

- Refresh one data mart on z/VM Linux Guest1.
- Concurrently refresh two data marts on z/VM Linux Guest1 and Guest2.
- Concurrently refresh three data marts on z/VM Linux Guest1, Guest2, and Guest3.

Figure 5-1 on page 56 depicts the concurrent refresh of the three data marts.

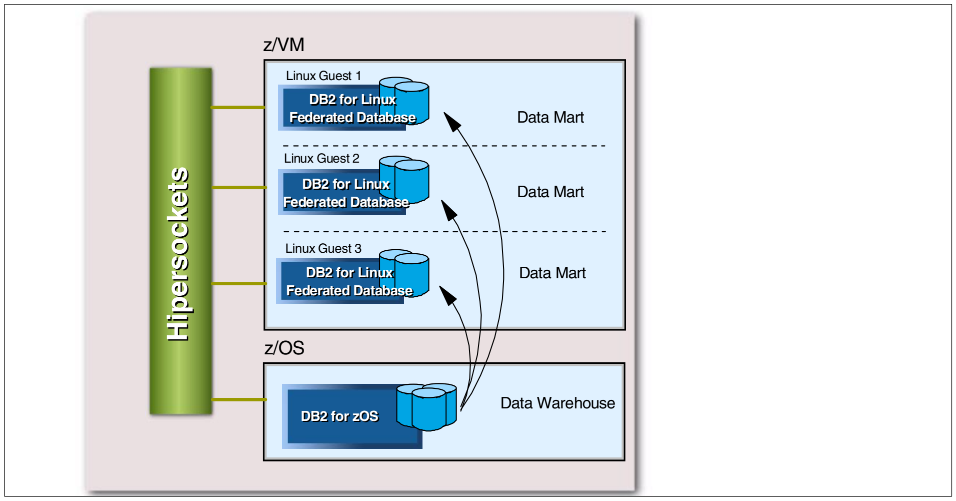


Figure 5-1 Data refresh scenarios - concurrent refresh of three data marts

A pictorial view of the concurrent refresh of two data marts would look similar, but with just two arrows from the data warehouse to Guest1 and Guest2. And the refresh of one data mart would have just one arrow from the data warehouse to Guest1.

The three Linux Guests on z/VM shared four processors and z/OS also had four processors. We used HiperSocket connections to move data from the data warehouse to the data marts. All DB2 for Linux data marts are set up as federated databases. The INSERT/SELECT SQL statement is initiated in the DB2 for Linux environment.

5.2.1 SQL performance

The INSERT/SELECT statements select rows from the data warehouse Item and Purchase Order tables and insert those rows into the data mart Item and Purchase Order tables. Refer to Appendix F, “Data refresh scripts” on page 133 to see the scripts we used for data refresh scenarios.

We wanted to accelerate the insert process by using multiple inserts into each table, so we designed the refresh process using 8 SQL statements to refresh the Item table and 8 SQL statements to refresh the Purchase Order table. Figure 5-2 on page 57 shows the Item refresh example where each INSERT/SELECT statement reads from 10 partitions of the data warehouse Item table on z/OS and inserts into the data mart Item table on Linux Guest1.

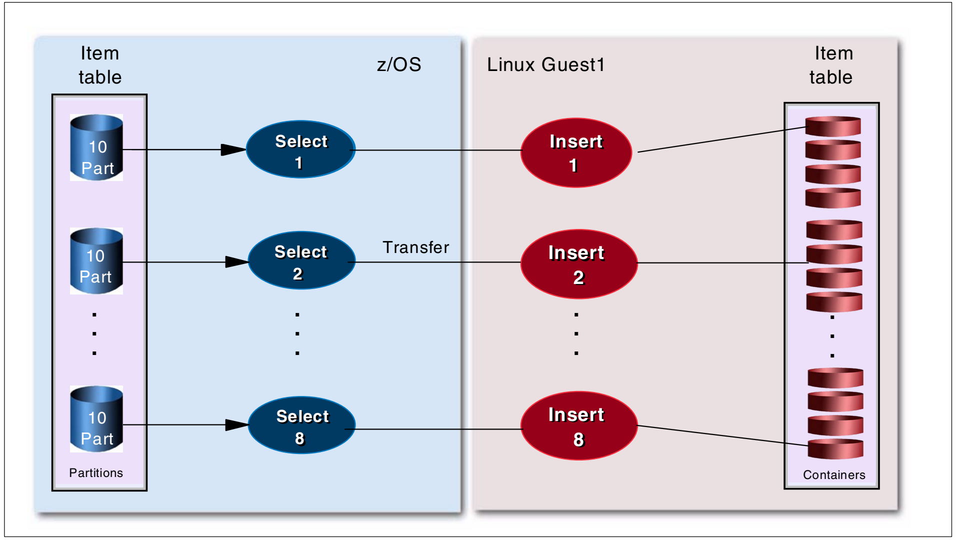


Figure 5-2 Data refresh data flow

Access path

The SELECT portion of the statement executes on z/OS. For the data refresh scenario we manually create parallel executions by executing multiple SQL statements to refresh one table. Thus, we have one SELECT task on z/OS matching one INSERT task on Linux.

We encountered the same optimization issues as in data population, whereby DB2 for Linux changed the BETWEEN predicate into a GREATER THAN OR EQUAL TO and LESS THAN OR EQUAL TO predicate, and the optimizer on DB2 for z/OS then wanted to do a tablespace scan. See “Access path” on page 42 for details.

Locking

We used row level locking on DB2 for Linux for data refresh scenarios.

In data population scenarios we used table level locking because we had only one INSERT/SELECT per table.

In data refresh scenarios we use row level locking because we have multiple INSERT/SELECT statements running against each table. If we used table level locking, the first INSERT/SELECT against the table would run holding a lock on the entire table, and all the other INSERT/SELECT statements would time out because they wouldn’t be able to get the lock.

The high number of locks generated made us change the following settings to avoid lock escalations and deadlocks:

- On z/OS we received lock escalation messages on the Item and Purchase Order tables for package distserv sql3cd07 as follows:

  ```
  DSNI031I  % DSNILKES - LOCK ESCALATION HAS OCCURRED 498
  FOR
    RESOURCE NAME = LINBIDB.TSITEM
    LOCK STATE =  S
    PLAN NAME : PACKAGE NAME = DISTSERV : SQLC3D07
    STATEMENT NUMBER = 000006
    CORRELATION-ID = db2agent (NA
    CONNECTION-ID = SERVER
    LUW-ID = C0A88C74.0148.2BD417213208
    THREAD-INFO = MIKET    : pelinuxl           : miket            :
  ```

  Package SQLC3D07 is bound by default with isolation level RR. We rebound it with isolation CS to resolve lock escalations.

- On the Linux guests we were getting deadlocks while inserting into Item and Purchase Order tables. We received the following message:

  ```
  SQL0911N The current transaction has been rolled back because of a deadlock
  or timeout. Reason code "<2>".
  ```

  To resolve the deadlock issue we changed the following on DB2 for Linux:

  – Maxlocks changed from 50 to 60 in the database configuration file
  – Locklist changed from 1000 to 20000 in the database configuration file
  – DB2_RR_TO_RS profile variable turned “on” using the **db2set** command.

Logging

Logging was on in data refresh scenarios.

5.2.2 Workload performance

We ran three separate tests to demonstrate data refresh scalability.

In the first test we refresh just the data mart on Linux Guest1. We run a total of 16 distributed INSERT/SELECT statements (8 to refresh the Item table and 8 to refresh the Purchase Order table), with a concurrency level of 3, as shown in Table 5-1 on page 59.

Table 5-1   *Data refresh workload against Guest1*

	# of SELECT statements on z/OS	**Total # of INSERT statements on Linux Guest 1**
Item	8	8
Purchase Order	8	8
Total workload	16 statements	16 statements
Concurrency level	3	3

For the second test we concurrently refresh the data marts on Guest1 and Guest2. For *each* data mart we use the same number of SQL statements and concurrency level as we used in the first test; i.e., 16 distributed INSERT/SELECT statements (8 to refresh the Item table and 8 to refresh the Purchase Order table) and a concurrency level of 3. Table 5-2 depicts the total number of SELECT and INSERT statements and the total concurrency level overall.

Table 5-2   *Data refresh workload against Guest1 and Guest2*

	# of SELECT statements on z/OS	**Total # of INSERT statements on Linux Guests 1 and 2**
Item	16	16
Purchase Order	16	16
Total workload	32 statements	32 statements
Concurrency level	6	6

In the third test, we concurrently refresh the data marts on Guest1, Guest2 and Guest3. Again, for *each* data mart we use the same number of SQL statements and concurrency level as we used in the first test; i.e., 16 distributed INSERT/SELECT statements (8 to refresh the Item table and 8 to refresh the Purchase Order table) and a concurrency level of 3. Table 5-3 depicts the total number of select and insert statements and the total concurrency level overall.

Table 5-3   *Data refresh workload against Guest1, Guest2 and Guest3*

	# of SELECT statements on z/OS	**Total # of INSERT statements on Linux Guests 1, 2, and 3**
Item	24	24

	# of SELECT statements on z/OS	**Total # of INSERT statements on Linux Guests 1, 2, and 3**
Purchase Order	24	24
Total workload	48 statements	48 statements
Concurrency level	9	9

Figure 5-3 shows the elapsed time for refreshing just one, then two concurrent, and then three concurrent data marts.

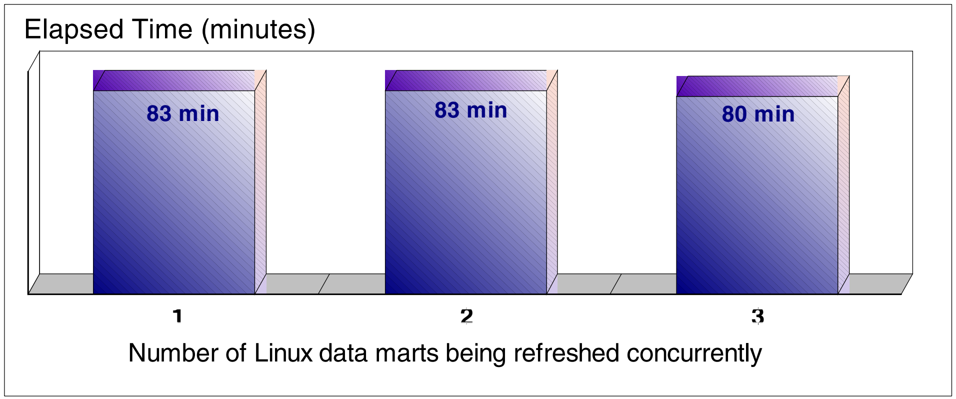


Figure 5-3 Concurrent data refresh on Guest1, Guest2, and Guest3 - elapsed time

Not only are concurrent data refreshes of multiple Linux data marts possible, but there is no elongation in elapsed time as more data marts are concurrently refreshed. This strategy of refreshing multiple concurrent data marts can be utilized in order to shorten maintenance windows.

Figure 5-4 on page 61 shows the CPU utilization on z/OS as 1, 2, then 3 data marts are concurrently refreshed.

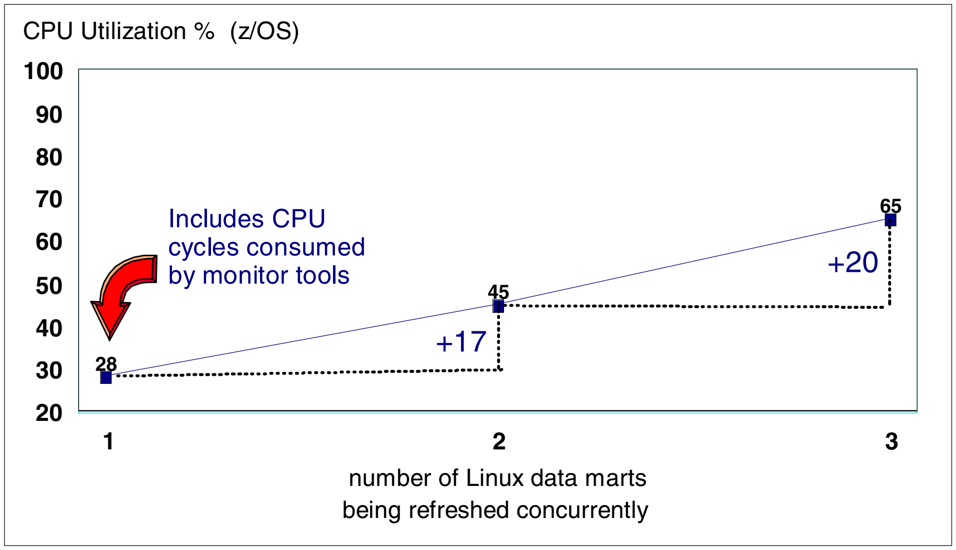


Figure 5-4 Concurrent data refresh on Guest1, Guest2, Guest3 - CPU usage on z/OS

CPU utilization increases in a linear fashion. There is very little difference in the CPU utilization delta going from 1 to 2 concurrent mart refreshes compared to going from 2 to 3 concurrent mart refreshes. This is because the CPU utilization for just one mart refresh includes the initial CPU overhead of performance monitoring tools that were utilized in all three test scenarios.

Figure 5-5 on page 62 shows the CPU utilization on each of the data marts in these scalability tests. The processing for 3 inserts per data mart barely uses any CPU; the CPU utilization is just 1% on each data mart. There is plenty of CPU resource available for any other applications to use.

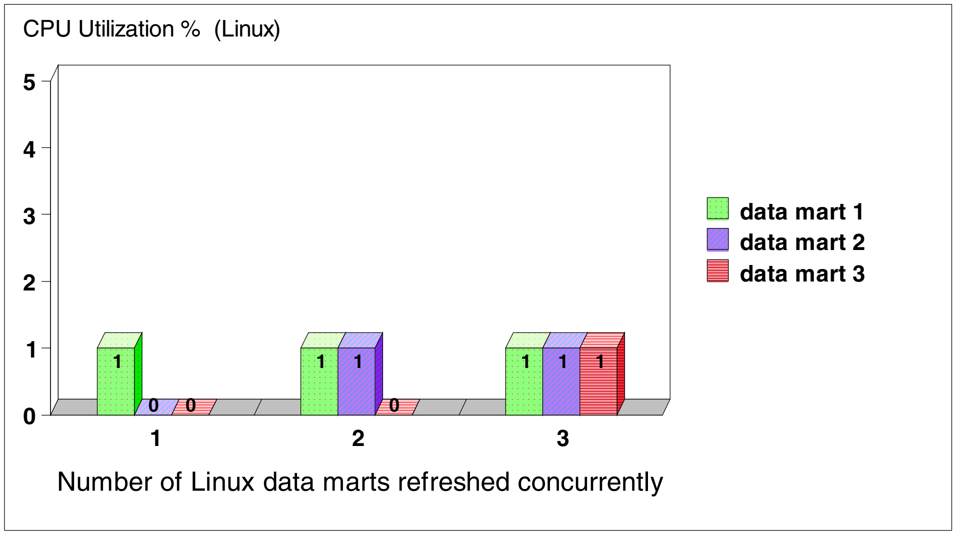


Figure 5-5 Concurrent data refresh on Guest1, Guest2, Guest3 - CPU usage on Linux

5.3 Summary

By consolidating data marts on the same zSeries server as the data warehouse, you can take advantage of their proximity and avoid the overhead of network traffic by moving data internally between LPARs using HiperSockets. In addition, you can concurrently refresh multiple data marts, thereby shortening your maintenance window.

6

Distributed data mart access

This chapter focuses on how the distributed requests supported by DB2 federated database systems provide transparent SQL access and join capabilities across multiple data marts in a single SQL statement.

With DB2 federated systems you can:

- Use a single SQL statement to access and join tables located across multiple data sources without needing to know the source location
- View your enterprise data as if it were local.

We demonstrate the capabilities of DB2 distibuted requests using a set of 3 queries extracting up to 3 million rows from the Linux Guest2 data mart and joining them locally with data from the Linux Guest1 data mart. We highlight the amount of time required to complete the entire execution.

Elapsed time is the key metric in this scenario. The objectives are to show:

- How federated database functions can simplify the process of accessing data at distributed data mart locations to address important business questions or decisions
- How well the DB2 optimizer performs when retrieving or accessing and manipulating large amounts of data across data marts

6.1 Value for Business Intelligence

Performing distributed SQL Select statements using DB2 federated databases is a simple and effective way to query and analyze data in different data marts. The following business examples show how federated databases can help address important business questions or decisions.

Insurance policy example

Figure 6-1 illustrates the insurance policy example.

Consider a business which has an automotive insurance data mart and a homeowner insurance data mart that reside in different DB2 database locations.

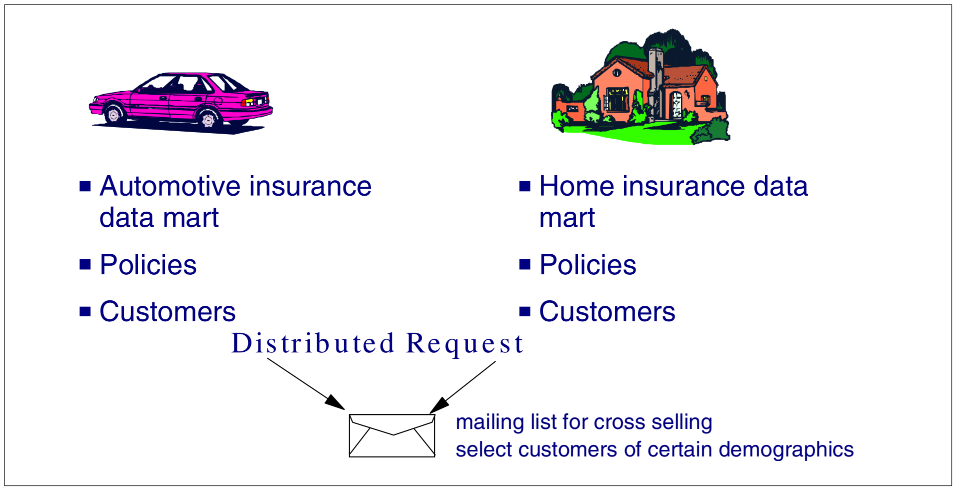

Automotive insurance data mart
Policies
Customers
Home insurance data mart
Policies
Customers
Distributed Request
mailing list for cross selling
select customers of certain demographics

Figure 6-1 Distributed request - insurance policy example

There is policy and customer information contained in both marts. Using federated database systems, a distributed request can be performed selecting and qualifying rows from these two marts that could generate a demographic listing of customers that would be candidates for cross-selling their insurance products.

Retail example

Figure 6-2 on page 65 illustrates a retail example.

Consider a business which has a sales data mart and an inventory data mart in different DB2 database locations.

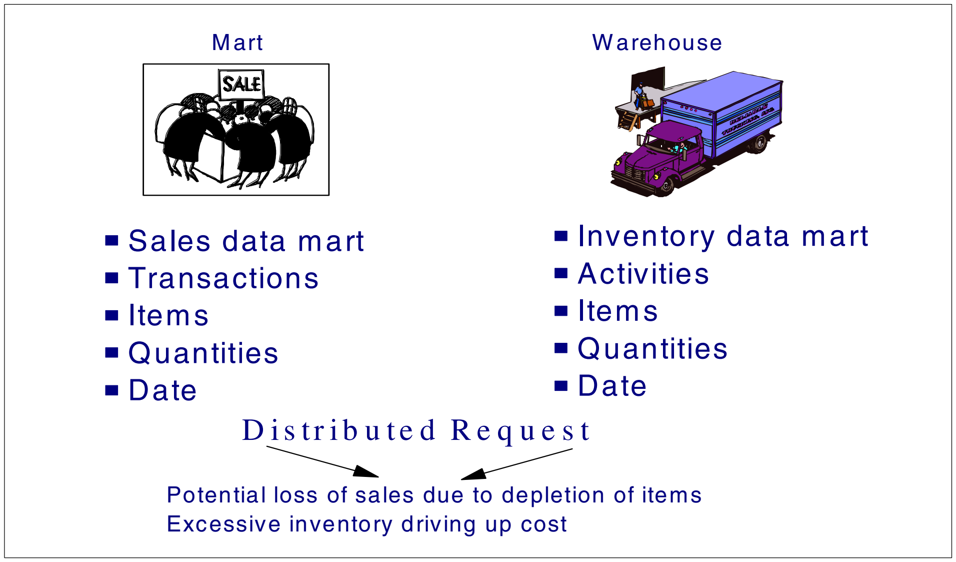


Figure 6-2 Distributed request - retail scenario

There are Item, Quantity and Date information in both marts. The sales mart also maintains transaction information while the inventory mart maintains stock balances. Using the distributed request facilities of DB2, the following business decisions/actions can be taken in response to transaction activity for any given day in a timely manner. See Table 6-1.

Table 6-1 Transaction activity and business decisions/actions

Sales	**Inventory**	**Decision/action**
0	0	Loss of sales; restock inventory
High	Low	Restock inventory
Low	High	Excessive inventory; put items on sale

The inventory database maintains the finished goods as well as the in-process inventory quantity. This database includes the expected finished date for the in-process items.

There are times when the stock on hand is not enough to cover an immediate sale. Having access to the inventory database will enable sales to query the quantity and expected finished date for a certain item or items which are still in-process, giving sales the flexibility to sell not only the finished goods but also the in-process items.

6.2 Distributed request scenarios

A user may submit a query from a Linux data mart, combining it with data in a z/OS data warehouse. This query can qualify millions of rows from the data warehouse on z/OS and join them with rows in the data mart to present back to the user.

Figure 6-3 shows the distributed request tests we ran in our test environment.

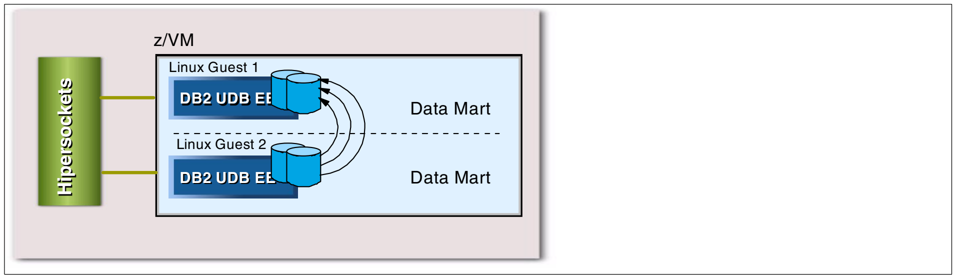


Figure 6-3 Distributed request scenarios

The DB2 federated system in Guest1 connects to DB2 for Linux in Guest2 and reads from 200,000 to 3 million rows. The results are presented locally in the DB2 for Linux data mart on Guest1.

Guest1 and Guest2 share four processors under z/VM. Data is transferred using HiperSocket connections. Joining data marts on the same server allows high-speed distributed joins at internal memory speeds.

6.2.1 SQL preformance

We ran three queries using DB2 distributed requests to test the ease of implementation and the execution performances.

Figure 6-4 on page 67 shows the design of the distributed request queries we tested.

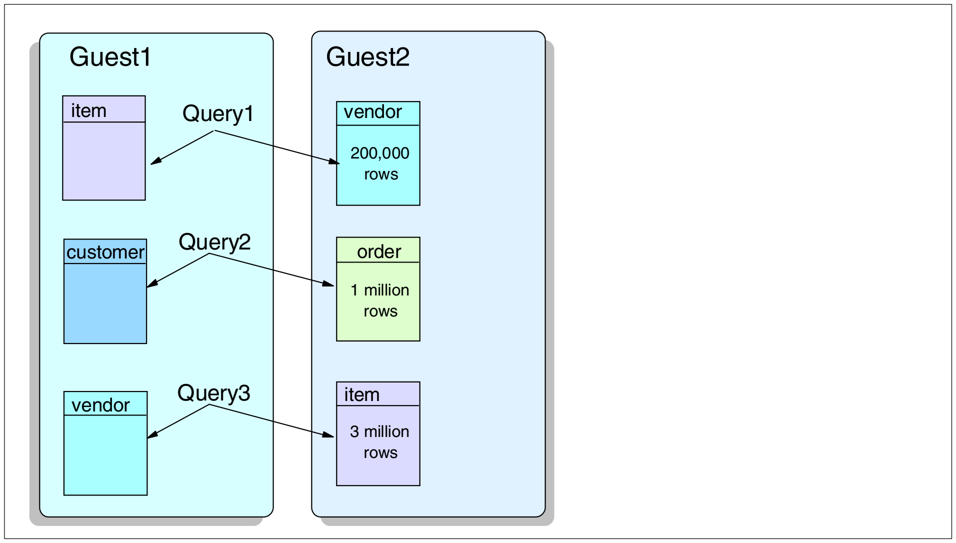


Figure 6-4 Distributed request queries

- Query1 is a distributed request extracting 200,000 rows from Guest2.

  This query lists all of the items shipped in a given week from vendors where the account balance is less than $1000. We are using vendors' account balances here to find out those vendors that we are not buying too many things from (since the balance is less than $1000). We then use this information to consolidate the number of vendors by buying those items from different vendors.

- Query2 is a distributed request extracting 1 million rows from Guest2.

  This query lists all of the customers from a certain territory whose account balances are less than $5,000.00.

  We use the account balances here to determine the customers that are not spending much money with our company. Then we plan a marketing campaign to make those customers aware of the comprehensive set of products that our company provides.

- Query3 is a distributed request extracting 3 million rows from Guest2.

  This query lists all the items ordered from a vendor by product line. The output from this query will be sent to the customers to provide them with the summary of items ordered from a particular vendor for each product line.

Refer to Appendix G, “Distributed request scripts” on page 135 to see the SQL distributed requests we tested.

We did no special tuning and used the default DB2 settings for those tests. Figure 6-5 shows the elapsed times of the three queries and how well the DB2 optimizer is tuned to handle distributed requests.

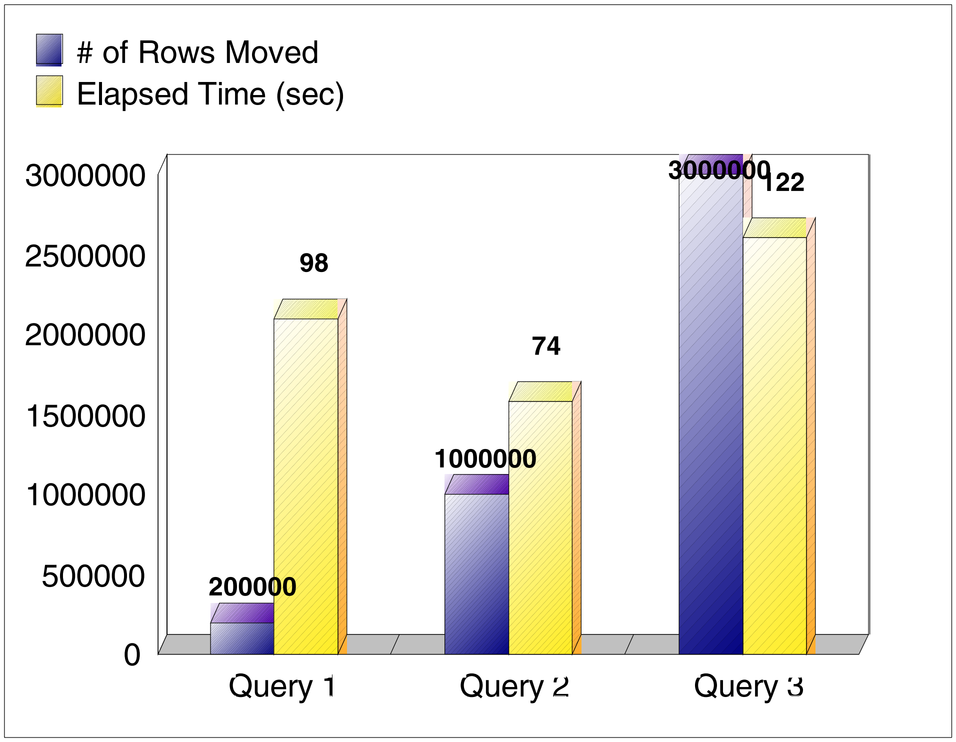


Figure 6-5 Distriubuted request queries - throughput and elapsed times

Data access and transfer varied from 200,000 rows to 3 million rows and took an insignificant time to complete. An impressive total of 4.2 million rows were accessed in less than 5 minutes of elapsed time!

Note also that the majority of the elapsed time was spent in query processing, not data movement. Therefore, as in the case of Query 2, other factors influenced the elapsed time of the query, such as SQL complexity or amount of aggregation.

6.3 Summary

Business decisions requiring timely access to critical data are facilitated by using DB2 distributed requests which access multiple data marts in a single SQL statement. The SQL programming is simple, the DB2 optimizer automatically takes care of tuning issues, and no special user skills are required to have well performing SQL statements.

By consolidating data marts on the same z/VM server you can easily access data from different data marts and transfer large amount of data internally, on the same zSeries, thus avoiding external network traffic.

7

z/VM resource allocation across Linux guests

This chapter focuses on allocating system resources based on workload requirements among the data marts on the Linux guests under z/VM. We explore how z/VM can be used to allocate system resources in the following three areas:

- Allocating resources based on individual business unit needs

  Businesses typically run many different applications, and it is important to be able to efficiently manage and allocate system resources among the different application servers based upon the needs of the business. Multiple applications and workloads can be consolidated onto Linux images running under a single z/VM server while maintaining complete application isolation and data integrity, and ensuring that each will receive the capacity it requires.

- Reducing costs by utilizing “white space”

  Individual applications have unique processor utilization requirements based on the business cycle and functional requirements of the business users. Utilization levels will fluctuate up and down throughout the business day, at different times of the week and even at different times of the business cycle. For example, a payroll application generally requires more system resources for end of the month processing while an OLAP application may have high resource requirements every day during first shift. During periods when system resource requirements are low and the CPU is underutilized, there is idle machine time. This idle or wasted time is called “white space”. When data marts are running on separate servers, each server must be configured to provide acceptable performance during high utilization periods, and as a result, there is “white space” on each server during periods of lower utilization. Data marts on separate servers are unable to share their unused capacity. When one data mart server is experiencing high utilization and performance degradation, it can not make use of any unused capacity from another server. When a large percentage of the total processing resources remains idle a great deal of the time, it drives up the cost/user over time. By consolidating data marts across multiple Linux guests under a single z/VM server, the unused capacity of the data marts that are experiencing a lull in activity, can be exploited by data marts that are experiencing high utilization. This results in more efficient use of system resources and reduced cost of ownership.

- Managing individual business units’ fixed budgets

  There are instances where individual business units are on a fixed budget, and they need to have a limit placed on the amount of system resources they can consume to prevent budget overruns. With z/VM, you can place a cap on the maximum amount of system resources that can be obtained by each of the VM guests.

Our test cases measure CPU consumption on each Linux guest and the aggregate z/VM processor consumption. The objectives of the test cases are to:

- Ensure workloads across the z/VM Linux guests can run concurrently in an efficient and manageable environment, and to demonstrate the resource management capabilities of distributing the CPU cycles in a saturated environment.
- Prove efficient use of total system resources by allowing one of the Linux guests under z/VM to utilize any additional resources that are available (white space) when the other Linux guests do not need all of their allocated resources.
- Demonstrate the ability to cap the amount of processing resource consumed by a given z/VM Linux guest.

7.1 Allocating resources

A virtual machine's share represents the percentage of system resources to which the virtual machine is entitled and can be specified using a z/VM SHARE directory statement or SET SHARE command.

In our scenarios we use the z/VM SET SHARE command to allocate the desired system resources to each Linux guest.

There are two types of shares that can be set: absolute share and relative share.

7.1.1 Absolute share

An absolute share allocates to a virtual machine an absolute percentage of all the available system resources (processor, real storage, paging I/O). VM reserves 1% of available resources for virtual machines with relative shares. So, if the sum of all the absolute shares assigned exceeds 99%, share values are reduced to a normalized value. Assigning an absolute share can be used to guarantee the availability of a certain percentage of processing time, storage capacity, and paging capacity to a virtual machine with high requirements for any of these resources.

7.1.2 Relative share

A relative share allocates to a virtual machine a portion of the system resources, which remain after allocating resources to those machines with absolute shares. A virtual machine with a relative share receives system resources in proportion to the relative share values assigned other virtual machines. Assigning relative shares to virtual machines can be used to guarantee their relative priority in using system resources. By default, each virtual machine is assigned a relative share of 100.

7.1.3 Maximum share

In addition to the normal share settings discussed above, a second value, representing a maximum share, can be set for a virtual machine. The maximum share limits a virtual machine from using more than a specified amount of processor resources. For example, a virtual machine can be given a normal absolute share of 50% and a maximum absolute share of 75%. Additional keywords can be used to specify the type of limit to be applied to a specified normal or maximum share:

- NOLIMIT

  The user is not limited from consuming processor resources. This can be specified for a normal share, and it is the default.
- LIMITHARD

  The user does not get more than the maximum share.
- LIMITSOFT

  The user does not get more than the maximum share, unless no other user can use the available resources.

Refer to *z/VM 4.2 Performance*, SC24-5999, for further information on z/VM performance and setting shares. This manual can be accessed via the Internet at:

```
http://www.vm.ibm.com/pubs/pdf/vm420bas.html.
```

7.2 Addressing individual business needs

In this scenario, we demonstrate allocating resources based upon the varying needs of each data mart. On each of the three Linux guests we ran a workload capable of driving the CPU to 100%. We issued the following commands on z/VM to allocate a certain share of the CPU resources to each Linux guest based on business needs:

```
SET SHARE Guest1 ABS 50% NOLIMIT
SET SHARE Guest2 ABS 25% NOLIMIT
SET SHARE Guest3 ABS 25% NOLIMIT
```

The workload running on Guest1 is considered the higher priority workload, so we set a minimum share of 50% of the CPU to Guest1. The workloads running on Guest2 and Guest3 are lower priority, so we set both Guest2 and Guest3 with a minimum share of 25% CPU. We specified NOLIMIT on each of the guest's share of resources.

Figure 7-1 on page 75 illustrates z/VM resource allocations.

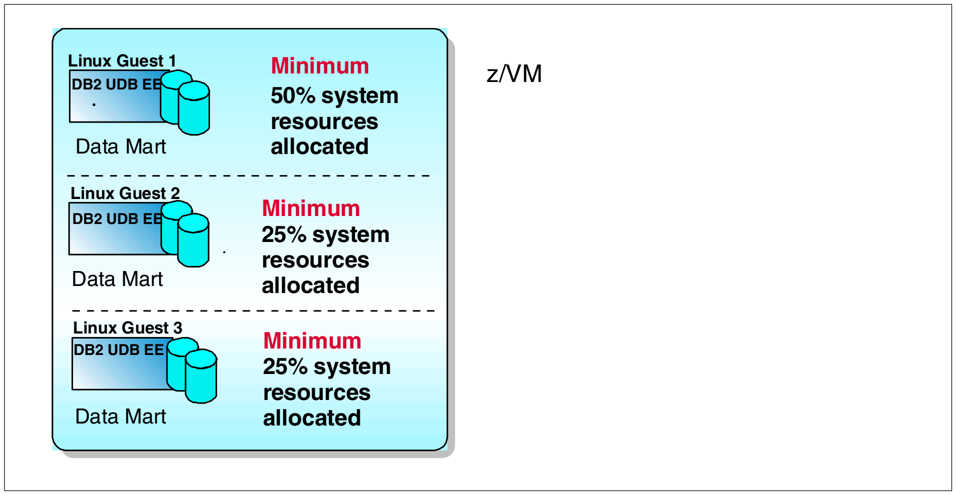


Figure 7-1 z/VM resource allocation - NOLIMIT

For each Linux guest, Figure 7-2 compares the CPU % allocated via the SET SHARE command, the actual CPU % consumed, and the CPU % the workload was capable of driving.

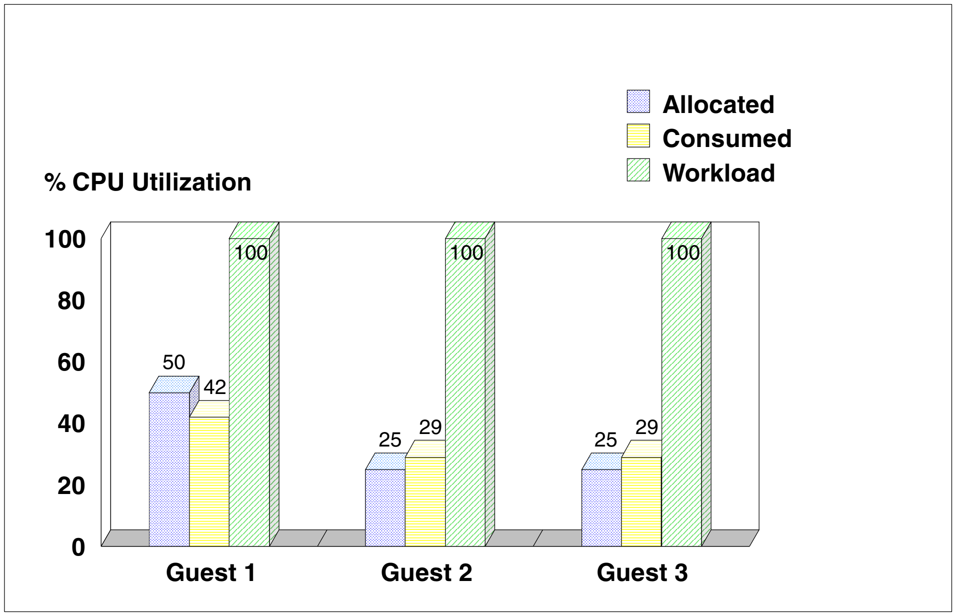


Figure 7-2 CPU utilization for Guest1, Guest2, and Guest3 - no limit cap

The actual CPU utilization for each of the Linux guests closely matches what was allocated via the SET SHARE command. The higher priority workload on Guest1 obtains more CPU than Guest2 and Guest3. The workloads across the Linux guests can run concurrently in an efficient and manageable environment, and z/VM is capable of distributing the CPU in a saturated environment based on business needs.

7.3 Reducing cost by utilizing white space

In this scenario we demonstrate the efficient use of total system resources when data marts are consolidated across multiple Linux guests under a single z/VM server. A single Linux guest is allowed to utilize resources that are not in use by the other Linux guests.

On Guest1, we ran a workload that was capable of driving the CPU to 100%. On Guest2 and Guest3, we ran workloads which only drove each of their CPUs to 7%. We issued the same z/VM SET SHARE commands as in the previous test case to allocate a certain share of CPU resource to each Linux guest based on business needs:

```
SET SHARE Guest1 ABS 50% NOLIMIT
SET SHARE Guest2 ABS 25% NOLIMIT
SET SHARE Guest3 ABS 25% NOLIMIT
```

The workload running on Guest1 is considered the higher priority workload, so we set a minimum share of 50% of the CPU to Guest1. The workloads running on Guest2 and Guest3 are lower priority, so we set both Guest2 and Guest3 with a minimum share of 25% CPU. We specified NOLIMIT on each of the guest's share of resource. Refer to Figure 7-1 on page 75 for an illustration of resource allocations.

Figure 7-3 on page 77 compares, for each Linux guest, the CPU % allocated via the SET SHARE command, the actual CPU % consumed, and the CPU % the workload was capable of driving.

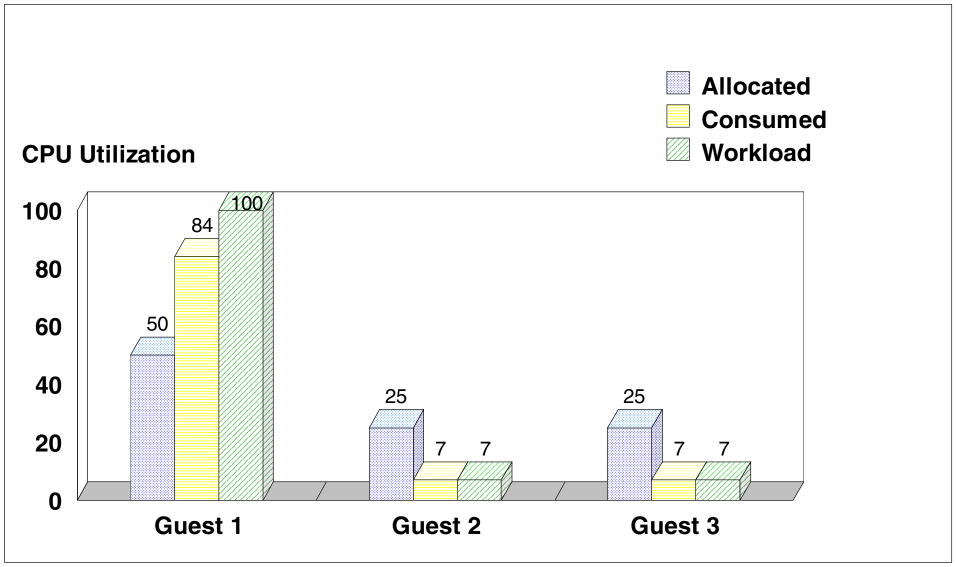


Figure 7-3 CPU utilization for Guest1, Guest2, and Guest3

Because each of the workloads running on Guest2 and Guest3 is utilizing only 7% of the CPU, there is spare CPU capacity that can be utilized by the CPU intensive workload executing on Guest1. Since we specified NOLIMIT on the SET SHARE command, Guest1 is able to take advantage of this white space and utilizes the CPU cycles not being used by Guest2 and Guest3. This proves again the ability of z/VM to allocate resources based on business needs, and also the benefit of consolidating data marts on a single server. By allocating capacity when and where it's needed, white space and the cost of ownership are reduced.

7.4 Managing fixed budgets

In this scenario we demonstrate how z/VM can be used to manage individual business units' fixed budgets. We give two examples:

- Single Linux guest
- Multiple Linux guests

7.4.1 Single Linux guest

In the first example, we look at just a single Linux guest. Guest1 and Guest3 are idle; there are no workloads running on them. On Guest2 we ran a workload which is capable of driving the CPU to 100% utilization. We issued the following command on z/VM to allocate a minimum of 25% of the CPU to Guest2, but also allow Guest2 to take advantage of any idle CPU cycles (NOLIMIT keyword):

```
SET SHARE Guest2 ABS 25% NOLIMIT
```

We then reran the same workload on Guest2, but issued the following command on z/VM so that Guest2 would not be allowed to have more than 25% of the CPU (LIMITHARD keyword):

```
SET SHARE Guest2 ABS 25% LIMITHARD
```

Figure 7-4 compares the cap placed on the CPU % allocated via the SET SHARE command, the actual CPU % consumed, and the CPU % the workload was capable of driving when Guest2 is allowed make use of white space versus when a cap is placed on the amount of CPU it can obtain.

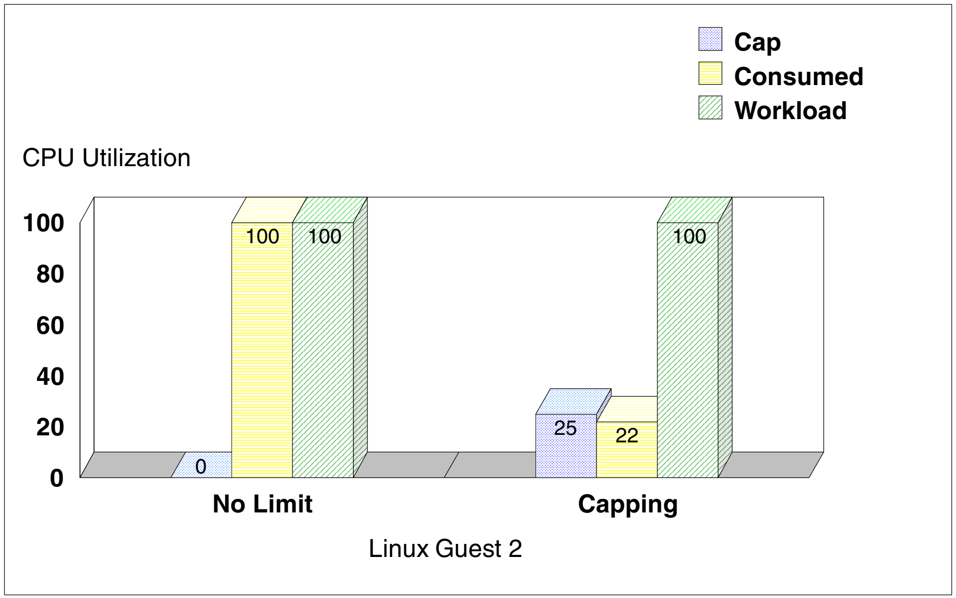


Figure 7-4 CPU utilization for Linux Guest2

On the left side of the chart, we see that with the NOLIMIT keyword, Guest2 is able to utilize the idle CPU cycles and take up all 100% of the available CPU.

On the right side of the chart, we see that although Guest2's workload is capable of utilizing all 100% of the CPU and although there are idle cycles available, Guest2 didn't get more than 25% of the CPU resource, because it is capped at 25% (abs 25% limithard).

7.4.2 Multiple Linux guests

In our second example, we are now running workloads on all three Linux guests. On Guest1, we ran a workload that was capable of driving the CPU to 100%. On Guest2 and Guest3, we ran workloads which only drove each of their CPUs to 7%.

We issued the following commands on z/VM to allocate a certain share of the CPU resources to each Linux guest based on the business needs:

```
SET SHARE Guest1 ABS 50% LIMITHARD
SET SHARE Guest2 ABS 25% NOLIMIT
SET SHARE Guest3 ABS 25% NOLIMIT
```

The workload running on Guest1 is considered the higher priority workload, so we set a minimum share of 50% of the CPU to Guest1. The workloads running on Guest2 and Guest3 are lower priority, so we set both Guest2 and Guest3 with a minimum share of 25% of the CPU. But, in this case, Guest1 belongs to a business unit that is on a strict budget, and they can't afford to pay for more than 50% of the CPU, so we specified the LIMITHARD keyword to cap the maximum amount of available CPU for Guest1 to 50%.   Guest2 and Guest3 have bigger budgets, so they are not limited (NOLIMIT keyword). Figure 7-5 illustrates the resource allocation.

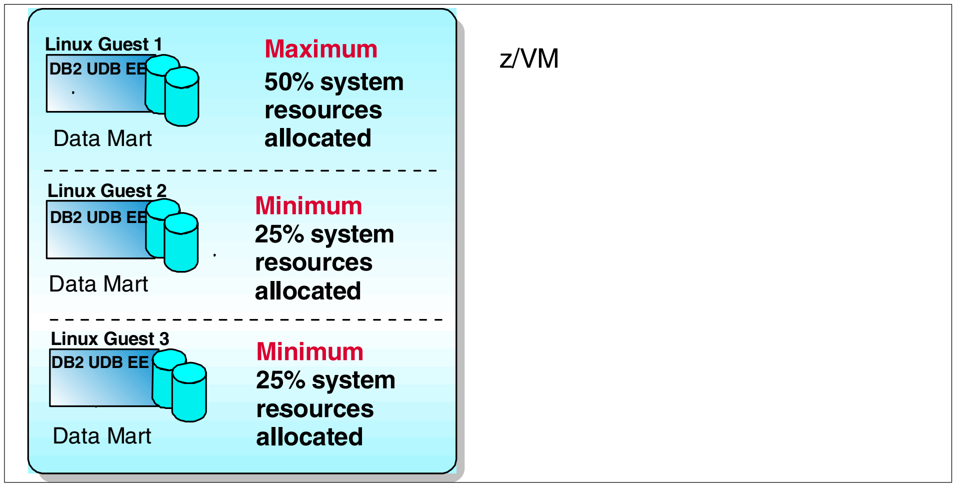


Figure 7-5   z/VM resource allocation - limithard

Figure 7-6 compares, for each Linux guest, the CPU % allocated via the SET SHARE command, the actual CPU % consumed, and the CPU % the workload was capable of driving.

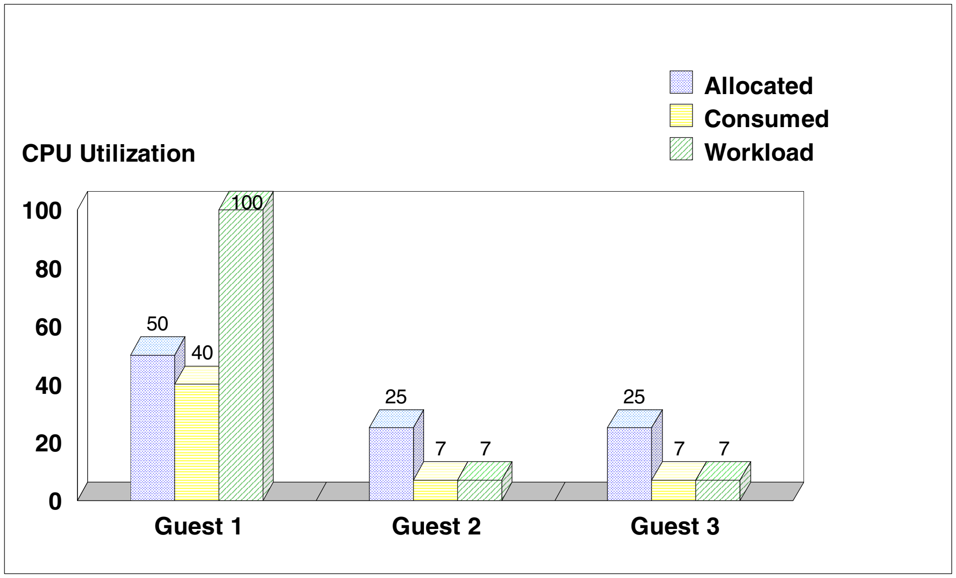


Figure 7-6 CPU utilization for Guest1, Guest2, and Guest3 - maximum share cap

Here we see that, although Guest2 and Guest3 were each allocated a minimum share of 25% of the CPU with no limit on their maximum share, their workloads are only consuming 7% of the CPU, leaving idle cycles or white space. Guest1, whose workload could potentially drive the CPU to 100%, does not take up more than 50% of the CPU. Guest1 is unable to take advantage of the idle cycles, because of the cap placed on the maximum amount of CPU it can obtain (LIMITHARD keyword). This demonstrates the ability of z/VM to allocate and limit resources based on a business unit's budget constraints.

7.5 Summary

By consolidating data marts across multiple Linux guests under a single z/VM server, you are able to take advantage of z/VM's intelligent resource allocation. With z/VM, you have the flexibility of allocating different amounts of system resources to each Linux guest based upon business needs. You can manage individual business units' fixed budgets and prevent budget overruns by limiting resources available to a Linux guest. And z/VM can allocate resources across the guests, when and where they are needed, resulting in a reduction in white space and total cost of ownership.

8

BI solutions for Linux on zSeries

This chapter provides an overview of the Dimensional Insight business intelligence tool using Linux on zSeries. We demonstrate that a BI solution developed for Linux on the Intel platform can be ported to Linux on zSeries. We then examine how effectively the Dimensional Insight solution would run on a Linux guest under z/VM with an average workload.

This chapter describes the following:

- Dimensional Insight products overview
  - DI-Atlantis
  - DI-Integrator
  - DI-Builder
  - DI-ProDiver
  - DI-WebDiver
  - DI-ReportDiver
  - DI-DiveLine
- Dimensional Insight test scenario
  - Workload setup
  - Workload performance

8.1 Dimensional Insight products overview

The Dimensional Insight family of products provides a comprehensive BI solution that allows information to be retrieved throughout the enterprise with speed, flexibility, and accessibility. Dimensional Insight databases are referred to as a “sphere” of data. Users who wish to query information from this database do “dives” into the data space. Dives can be done from any starting point, and users can navigate to any other item for on-the-fly ad-hoc analysis without the need for complicated SQL scripts of predefined access paths.

Dives can also be predetermined for routine analysis of data. These reports can be generated and saved for other users to access, allowing them to view a prescribed subset of data without the need to spend time processing the data themselves.

Figure 8-1 shows the interactions of the Dimensional Insight products.

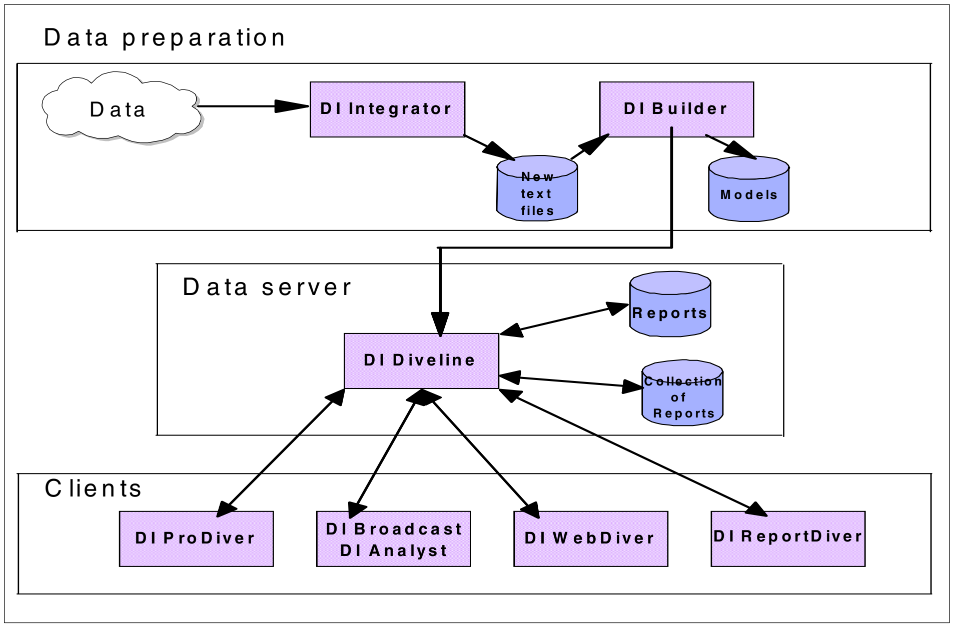


Figure 8-1 Dimensional Insight component layout

For the latest information on Dimensional Insight tools and solutions direct your browser to:

```
http://www.dimins.com
```

The following sections give an abstract of the DI-Atlantis suite of tools.

8.1.1 DI-Atlantis

DI-Atlantis is the backbone of the Dimensional Insight solution. This suite of tools enables organizations to replace the traditional complicated reporting and data access methods of the past (SQL queries, RPG programming, etc.) with a much more integrated and streamlined solution for their BI needs. The DI-Atlantis tools make reports and data accessible to end users directly in a convenient and easy-to-use manner, eliminating the need for Information Services personnel to assist end users directly with data retrieval.

The DI-Atlantis suite of products is comprised of several modules which as a whole handle the extraction, transformation, processing and access to user data. Data is first translated into the Dimensional Insight sphere within which dives are done. The DI-Integrator and DI-Builder *packages* are responsible for creating the data sphere which is referred to as a *model*. Access to the model is then done using one of the many DI-Clients, which connect to the DI-DiveLine server that retrieves data from the model. Dimensional Insight's DI-Diver, DI-ProDiver, DI-WebDiver, and DI-ReportDiver are all client interface options which allow end user access to the data models through the network.

DI-Integrator

The Integrator module of DI-Atlantis acts as a preprocessor of the input data and can accept inputs from various sources, which it then combines to form flat files that are imported into the DI-Builder module. The DI-Integrator is controlled by a description file and specifies the source of the data, the operations to be performed upon it, and where to put the resulting output. Some operations the Integrator can perform are joins, table lookup, row selection and column selection. These and other functions are specified in a text file written in the Data Integrator's Object Language.

DI-Builder

The DI-Builder module accepts processed data from the DI-Integrator package and transforms this data into a model, or sphere, to be analyzed using one of Dimensional Insight's many clients. The model uses a patented data structure that the Dimensional Insight tools are optimized to use. The builder transforms the data by processing it into a more optimum structure. Most of this processing allows the DI-Builder to optimize the data layout, thereby reducing much of the burden of performing runtime analysis from the server and clients. This allows Dimensional Insight queries to be much more responsive and efficient in their access of the data.

DI-ProDiver

DI-ProDiver is a Windows-based program providing a full-featured graphical interface. It allows for creating, viewing, analyzing data and generating reports which can be made viewable to other Dimensional Insight users. DI-ProDiver can also be used to create models, from either text files, or an ODBC source. DI-ProDiver works with the DI-DiveLine server to perform data analysis and display. The data retrieved can be viewed and printed in many formats, including tabular format or more graphical representations.

DI-WebDiver

DI-WebDiver is a Java applet served through a Web server and accessed through a Java-capable browser. DI-WebDiver is similar in functionality to DI-ProDiver in that it allows end users to access models stored on a remote server. Users are given the freedom of accessing their data through the World Wide Web with the same intuitive interface as DI-ProDiver.

DI-ReportDiver

DI-ReportDiver is also a Java applet much like DI-WebDiver. However, DI-ReportDiver is designed to view only predefined reports. It contains the same diving engine as the other Diver Clients, but this engine is used only to display reports that were generated with ProDiver and WebDiver clients. Users access DiveBooks, which are collections of reports. These DiveBooks define how DI-ReportDiver should display the latest available data for the viewer. DI-ReportDiver supports a slim client who can display up-to-date information using a simplified GUI.

DI-DiveLine

DI-DiveLine is the data server software which enables the Dimensional Insight Business Intelligence solution to function as a cohesive whole. DI-DiveLine allows you to have a centrally controllable solution for BI data, which means there is no need to update other locations when new models or Reports are created. Likewise, neither do multiple refreshes need to be done when new data is added. As each client connects and queries the server, it will automatically access the latest data available. DI-DiveLine allows you to centrally store and control access to models, reports and specific views, making the necessary information readily available to those who need it, and protected from those who don't.

8.2 Porting and installing Dimensional Insight on Linux

In order to install the DI-Atlantis suite of tools on zSeries, we needed to build the Dimensional Insight tools which would run on Linux on zSeries.

First, we compiled the standard Dimensional Insight code with no prior modifications for our tests and produced the necessary distribution files. The Linux systems that we installed had all the necessary development tools and libraries. Most, if not all, of these tools are standard on almost all Linux installations regardless of platform or distribution because many Linux tools are distributed in source form.

Then, we began the build process and encountered a problem only when faced with the question of byte ordering. zSeries machines are big-endian and that information is needed to be set in the Dimensional Insight build configuration files with a standard C preprocessor define statement.

Once the configuration choice had been selected and the proper files modified, the build process proceeded without any further complication. It took only a short amount of time to build all the tools necessary to complete the DI-Atlantis suite. This demonstrates that for well-written code, a “port” to Linux is often just as simple as a recompile, provided the code has been written with platform considerations in mind, or is not dependent on hardware specifics.

These criteria are fairly common in the UNIX world. Code is often written to the POSIX standard, and programs which adhere to this programming specification meet with little challenge when being ported to a new platform that supports POSIX. This is especially the case with the cross-platform nature of Linux. Less than 1% of the stock Linux Kernel needed to be modified to allow it to run on zSeries hardware. This leaves very little possibility for inconsistencies in behavior between hardware platforms that are running an instance of the same Linux Kernel version that has been compiled for their hardware considerations.

From this point, the installation proceeded according to the Dimensional Insight documentation, except for a slight change in the inetd configuration. Since we used xinetd (a next generation replacement for inetd) on our Linux systems, we had to alter the installation slightly to account for this discrepancy. This was a minimal change and the procedure has been reported back to Dimensional Insight to be included in their Linux installation instructions.

Figure 8-2 on page 86 shows our Dimensional Insight setup.

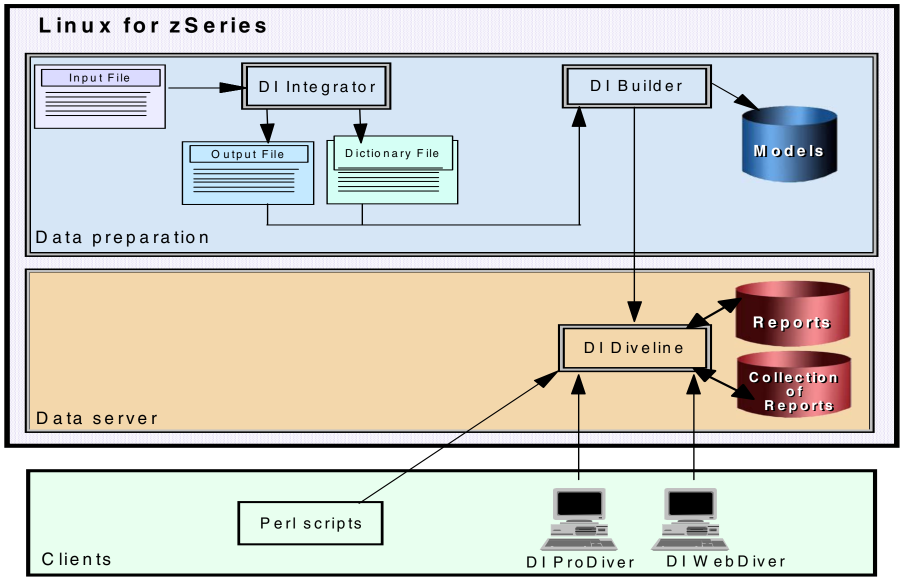


Figure 8-2 Dimensional Insight product setup

Because all Dimensional Insight clients use the same clear text protocol for communication, each instance of a Perl script that connects to the DI-DiveLine server represents a client connection.

The installation procedures used for these products are discussed in Appendix H, “Dimensional Insight - Atlantis Installation” on page 137.

8.3 Setting up the Dimensional Insight workload

It was decided that 100 active concurrent users was a reasonable amount of load for the test system. This number was selected based on the experience of the Dimensional Insight team with typical customer deployment configurations of their tools. We did not design our tests or our system configuration to maximize this number, nor did we attempt to measure optimal scalability in any empirical manner. Instead, we tried to focus on a typical middle-to-high-end range of load for the DI-DiveLine product and test the viability of running this as a guest under z/VM as a function of the stability of the setup and product.

For our testing, we did some Perl scripting around a Perl script provided by the vendor. Dimensional Insight's dives work on a clear text-based protocol, much like POP or HTTP. A graphical client sends commands in this language to the DI-DiveLine server, which performs the appropriate actions based on the commands issued. The results are sent back to the client performing the dive and can then be displayed in whatever manner the client and the user wish.

This separation of processing allows us to simulate many concurrent client requests simply by reproducing the clear text dialogue between the server and the client. The DI-ProDiver client can be used to produce reports that are saved on the server to be viewed by a client later. The DI-ProDiver GUI can also save transcripts of its actions and produce log files of the session. We used these log files as input to our Perl script in order to duplicate the actions a typical user would perform when doing dives.

Twelve different sets of typical user actions were generated and used as input to our testing. Some of these dives were simple dives only a few levels deep and some were much more complex. Reports were one of the following forms:

- Multi-crosstab: One vertical dimension, one horizontal dimension with more than one summary column
- Multi-level: Levels 2 and 5 deep are used, data joined from multiple dimensions
- Multi-tab: More than one vertical dimension, either 2 or 3 levels
- Multi-tab Multi-crosstab: Multi-tab first and then multi-crosstab, only 2 levels deep

For each of the 4 forms (6 types total) of input, we created 12 user actions, which retrieved either 1 million or 10 million rows. A distribution was determined for the 12 user actions that we thought would adequately simulate a typical daily workload. A 100-entry table was created in which each activity appeared a number of times according to the distribution value we assigned to it.

Table 8-1 describes the 12 user actions, their selected distributions, and their approximate completion time.

Table 8-1 User actions with distribution and minimum completion time

User action No.	User action description	Distribution (%)	Minimum completion time
1	Multi-level, 2 level 1 M rows output	15%	10 sec
2	Multi-tab 2 summary column, 1 M rows output	15%	12 sec
3	Multi-level, 2 level 10 M rows output	10%	20 sec
4	Multi-level, 5 level, 1 M rows output	10%	15 sec
5	Multi-level, 5 level, 10 M rows output	15%	21 sec
6	Multi-tab 2 summary column, 10 M rows output	15%	19 sec
7	Multi-tab 3 summary column, 1 M rows output	10%	22 sec
8	Multi-tab 3 summary column, 10 M rows output	3%	34 sec
9	Multi-cross tab, 1M rows output	5%	26 sec
10	Multi-cross tab, 10 M rows	2%	42 sec
11	Multi-tab Multicross tab, 1 M rows output	3%	53 sec
12	Multi-tab Multicross tab, 10 M rows output	2%	60 sec

Each user action was designed to simulate a user connection, which would include a recorded “wait time” of a few seconds. Thus the Dimensional Insight script would send the appropriate data, and after the response from the server, would then wait the recorded amount of time. If the server took longer during the test to send the data, then the overall time of the script would be longer; otherwise, the script would wait the predefined number of seconds.

An emulated client would make a random decision from this 100-entry table as to which test entry to run, simulating a distribution of work that would be typical of the daily activities performed by Dimensional Insight customers in their day-to-day business environment. For example, many employees might need to access a report on sales data for their work, so it was determined that this was a function which was more common than an employee who needed data for the whole year.

The Perl script was designed to moderate the number of concurrent connections seen by the server. Since the workload of the clients is light due to their lack of graphical data representation, all the connections could be run and controlled from one central system acting as multiple clients. Again, because of the separation between the DI-DiveLine server and the clients using the clear text protocol, the server performs the same amount of work whether or not the client chooses to display the results.

8.4 Workload performance

Figure 8-3 describes the methodology we used to generate the workload.

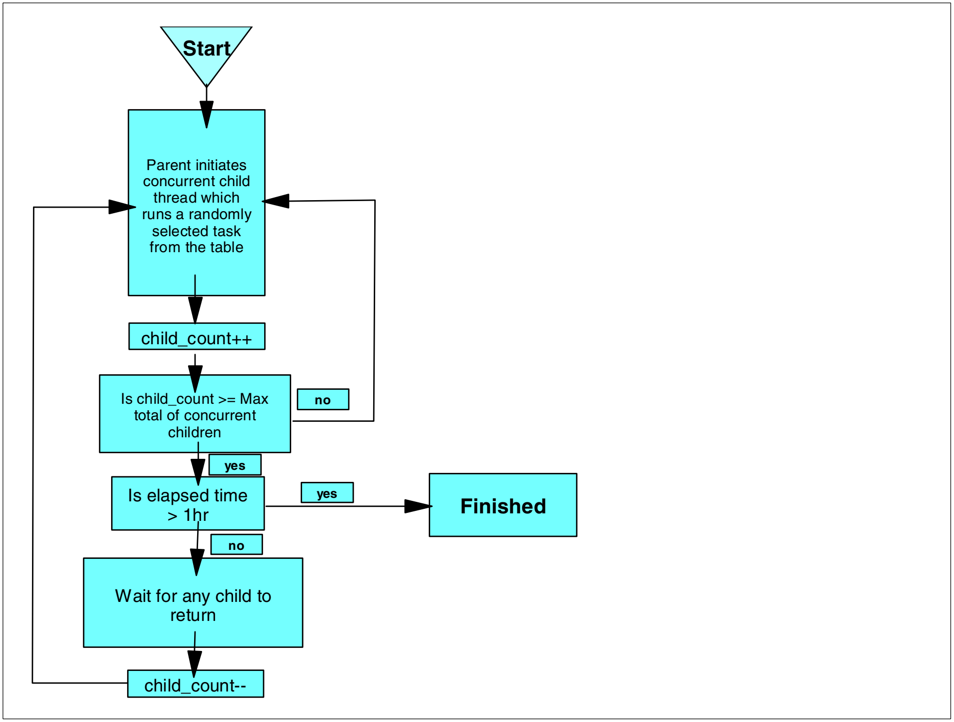

Start
Parent initiates concurrent child thread which runs a randomly selected task from the table
child_count++
Is child_count >= Max total of concurrent children
no
yes
Is elapsed time > 1hr
yes
Finished
no
Wait for any child to return
child_count--

Figure 8-3 Methodology to generate the workload

Users were allowed to connect repeatedly over a 1-hour period for all of the tests. The rate of connections was determined by the number of active concurrent connections that were allowed, the time each dive took to complete, and the particular workload selection of the Perl scripts (which was determined by random selection).

We ran all tests with the same 100-entry table representing the predetermined distribution of work, while varying the number of simulated active concurrent clients, keeping in mind that each run's behavior was randomly determined. Table 8-2 describes the throughput achieved by concurrent active users.

Table 8-2 Active users and throughput

Active concurrent users	**Approximate total reports accessed in 1 hour**
60	14000
75	17500
100	24000

We measured the overall throughput with 60, 75, and 100 concurrent active users. The total throughput from each test was measured by a counter in the Perl script which was incremented after every simulated dive was started. Because the actual overall work done and the order of selections were completely random for each measurement, this provided us with approximate throughput measurements for each 1-hour period.

For example, measurements included dives that were started but not completed before the end of the hour time frame. At the completion of the hour of stress, the script was abruptly terminated. Each run was made against the Linux guest server which was running under z/VM and sharing 4 processors with the other two images, which were mostly idle. Because of the reliance on a random distribution of reports versus a predefined order, it is virtually impossible to draw a performance-based conclusion from these results.

Figure 8-4 on page 91 shows the projected throughput achievement of 100 active users.

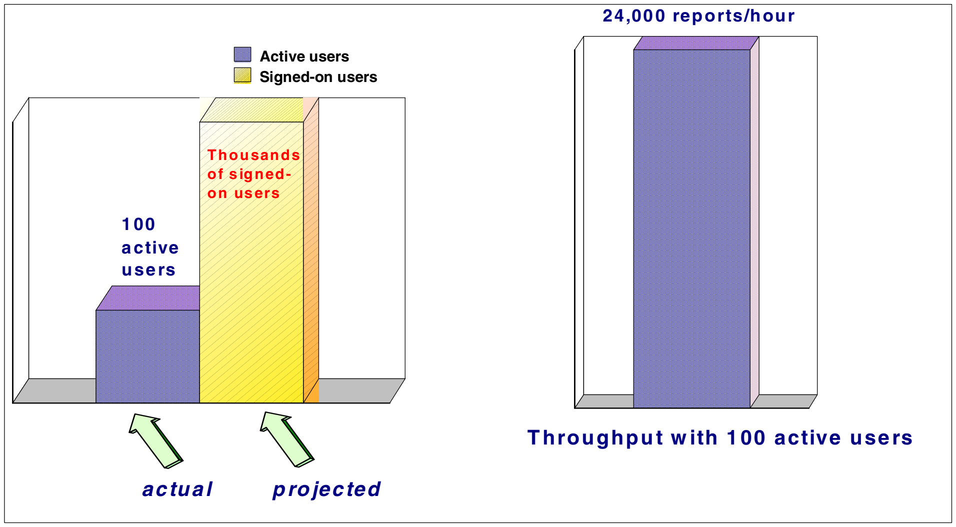


Figure 8-4 100 users projected throughput achievement

While not a performance-based measurement, the test shows that the DI-DiveLine server was quite effective at supporting a large number of active concurrent users on Linux on zSeries. Those users are connected to the server and are actively performing queries. In real life, it is uncommon for users to be so active in their queries. A typical user might make a few queries a minute and the connection might sit idle from the server's perspective, while the end user analyzes the data just retrieved. This is commonly referred to as “think time”.

Our test scenario simulated a lower number of signed-on users (100) than would typically be logged on in a real-world scenario. Because each signed-on user was also an active user, our test cases generated more processing work in a given amount of time than is typically seen in a production environment. We assume that this situation translates into thousands of signed-on users in a typical production environment.

Linux on zSeries was quite capable of handling the workload with little or no tuning of the environment. The fact that this solution was more than capable of supporting a high rate of active user connections is an indication that Linux on zSeries is a viable business platform on which one can rapidly deploy a stable and robust BI solution.

In spite of the approximate number of active user connections over a 1-hour period, this test is a useful demonstration of the stability of a BI application that was ported from the Linux on Intel platform to the Linux on zSeries platform with a simple recompile.

A

Linux guests on z/VM

A.1 Sample z/VM directory entry

The following z/VM directory entry is for our guest1 Linux on zSeries image running under z/VM:

```
USER GUEST1 999999 768M 768M G
 INCLUDE CMSUSER
 MACHINE ESA 8
 CPU 00 BASE
 CPU 01
 CPU 02
 CPU 03
* 191 A Disk - 50 CYL
 MDISK 191  3390   41   50 V62M01 MR  ALL WRITE MULTI
* Linux Root
 MDISK 7580 3390   DEVNO E735   MWV ALL ALL   ALL
* Swap - No Minidisk Cache
 MDISK 7581 3390    1 5000 BILV1D MR  ALL WRITE MULTI
 MINIOPT NOMDC
* DB2 code at /usr/IBMdb2. 5000 CYL leaves lots of room for other things
 MDISK 7582 3390 5001 5000 BILV1D MR  ALL WRITE MULTI
* Linux Home Directory
 DEDICATE 7583 8927
*                   Log, Tablespace & FTP VOLUMES
 DEDICATE 7584 980B
 DEDICATE 7585 980C
 DEDICATE 7586 990B
 DEDICATE 7587 990C
 DEDICATE 7588 9A0B
 DEDICATE 7589 9A0C
 DEDICATE 758A 9B0B
 DEDICATE 758B 9B0C
 DEDICATE 758C 9C0B
 DEDICATE 758D 9C0C
 DEDICATE 758E 9D0B
 DEDICATE 758F 9D0C
 DEDICATE 7590 9E0B
 DEDICATE 7591 9E0C
 DEDICATE 7592 9F0B
 DEDICATE 7593 9F0C
 DEDICATE 7594 9811
 DEDICATE 7595 9911
 DEDICATE 7596 9A11
 DEDICATE 7597 9B11
 DEDICATE 7598 9C11
 DEDICATE 7599 9D11
 DEDICATE 759A 9E11
 DEDICATE 759B 9F11
*                   PERF DATA VOLUME
 DEDICATE 759C 8B2A

*                  Dimensional Insight Vols
 DEDICATE 75A9 980D
 DEDICATE 75AA 990D
 DEDICATE 75AB 9A0D
*                   Fast Ethernet
 DEDICATE 1084 1084
 DEDICATE 1085 1085
*                   OSA-Express Gigabit
 DEDICATE 1F00 1F00
 DEDICATE 1F01 1F01
 DEDICATE 1F02 1F02
*                   IQD (Hipersocket) 16K
 DEDICATE 2000 2000
 DEDICATE 2001 2001
 DEDICATE 2002 2002
 DEDICATE 2003 2003
*                   IQD (Hipersocket) 24K
 DEDICATE 2100 2100
 DEDICATE 2101 2101
 DEDICATE 2102 2102
 DEDICATE 2103 2103
*                   IQD (Hipersocket) 40K
 DEDICATE 2300 2300
 DEDICATE 2301 2301
 DEDICATE 2302 2302
 DEDICATE 2303 2303
*                   IQD (Hipersocket) 64K
 DEDICATE 2400 2400
 DEDICATE 2401 2401
 DEDICATE 2402 2402
 DEDICATE 2403 2403
*----------------------------------------------------------
```

A.2 Installing Linux in a z/VM virtual machine

We installed the guest1 image on z/VM LPAR with following steps:
Note that we used a prebuilt root file system containing the Linux on zSeries 2.4 kernel, *tar*'d and stored on a local ftp server.

1. Logon to the zVM system with userid guest1.
2. Get the files to bring up the ramdisk image from server :

```
set msg on
set emsg on
set impcp on
term chardel off
vmlink tcpip
ftp 12.17.8.223 (and enter userid & password)
cd /pub/31/esa24_12
ascii
  locsite fix 80
  get parmfile linux.parm (replace
  binary
  locsite fix 80
  get initrd initrd.txt (replace
  get ESA24_12_smp_vm.img vmlinux.txt (replace
  quit
```

3. Load the ramdisk files into the VM reader and IPL:

```
close rdr
purge rdr all
spool punch * rdr
punch vmlinux txt a (noh
punch linux parm a (noh
punch initrd txt a (noh
ch rdr all keep nohold
i 00c
```

4. Respond to the ramdisk network setup dialog:

```
Welcome to Linux for S/390
Is your machine connected to a network (Yes/No) ? yes
```

```
Select the type of your network device
1) for lcs osa token ring
2) for lcs osa ethernet
3) for qdio osa ethernet
4) for channel to channel and escon channel connections
5) for IUCV
6) for CLAW
Enter your choice (1-6): 2

Please type in the channel device configuration options, e.g. to select
relative port 1 on device 0xfd00 you should enter:
lcs0,0xfd00,0xfd01,0,1
lcs parameter:
lcs0,0x1084,0x1085,0,1

Please enter your IP address:
9.7.22.1
Please enter the net mask:
255.255.255.0
Please enter the broadcast address:  [9.7.22.255]
9.7.22.255
Please enter the net address:  [9.7.22.0]
9.7.22.0
Please enter the gateway address:  [9.12.22.1]
9.7.22.1
Please enter the IP address of the DNS server:
9.7.16.2
Please enter your host name:
pelv6211
Please enter the DNS search domain:
dom.pok.ibm.com
```

5. Login and get the root tar file from the server:

   a. Login using root, pwdroot:

   ```
   insmod dasd_mod dasd=7580
   ```

   b. Format the dasd to be used for root:

   ```
   dasdfmt -d cdl -b 4096 -f /dev/dasd/7580/device
   ```

   c. Add a partition:

   ```
   fdasd /dev/dasd/7580/device
   n (to add a new partition)
   enter (enter twice to use the whole volume)
   w (to write & exit)
   ```

   d. Label the dasd:

   ```
   fdasd -l BILV1R /dev/dasd/7580/device
   ```

```
yes (to confirm)
w  (to write & exit)
```

e. Make a file system and mount the device:

```
mke2fs -b 4096 /dev/dasd/7580/part1
mount -t ext2 /dev/dasd/7580/part1 /mnt
cd /mnt
```

f. Get the tar root directory:

```
ftp 12.27.18.28   (and enter userid & password)
    cd /pub/31/esa24_12
    binary
    get initfs_big.tgz
    quit
    tar xzfBp initfs_big.tgz
```

6. Customize the new Linux image:

   a. Customize the new file system by copying some configuration files from the ramdisk filesystem to the corresponding places of the new file system:

```
cp /etc/resolv.conf /mnt/etc/resolv.conf
cp -r /etc/sysconfig/* /mnt/etc/sysconfig
cp /etc/modules.conf /mnt/etc
cp /etc/chandev.conf /mnt/etc
```

   b. Customize /mnt/boot/parmfile:

```
dasd=7580-7583    root=/dev/dasd/7580/part1 noinitrd
```

   c. Customize /mnt/etc/fstab:

```
/dev/dasd/7580/part1   /        ext2   defaults,errors=remount-ro 0 1
none                  /proc     proc   defaults  0   0
```

   d. Go into /mnt/boot and run zipl in order to enable Linux to be IPL-ed from the new disk:

```
zipl -t . -i image-2.4.7-0tape-dasd -p parmfile
```

7. Shutdown the ramdisk image:

```
umount /mnt
halt
```

8. Then from the guest1 ID on VM, IPL the new Linux image:

```
    i 7580
```

9. Set up the other DASD:

   a. Set up 7581 as a swap partition:

```
dasdfmt -d cdl -b 4096 -f /dev/dasd/7581/device
    fdasd /dev/dasd/7581/device
    n (to add a new partition)
    enter (enter twice to use the whole volume)
    w (to write & exit)
```

b. Change the label to BILV1S:

```
  fdasd -l BILV1S /dev/dasd/7581/device
  y (to confirm)
  w (to write & exit)
```

c. Make a file system:

```
  mke2fs -b 4096 /dev/dasd/7581/part1
```

d. Format for swap

```
  mkswap /dev/dasd/7581/part1
```

e. You should see messages like these:

```
  mkswap: warning: truncating swap area to 2097144kB
  Setting up swapspace version 1, size = 2147471360 bytes
```

f. Activate the swap space:

```
swapon /dev/dasd/7581/part1
```

g. Format 7582 and mount at /usr/IBMdb2:

```
dasdfmt -d cdl -b 4096 -f /dev/dasd/7582/device
fdasd /dev/dasd/7582/devic
  n (to add a new partition)
  enter (enter twice to use the whole volume)
  w (to write & exit)
```

h. Change the label to BILV1D:

```
  fdasd -l BILV1D /dev/dasd/7582/device
  y (to confirm)
  w (to write & exit)
```

i. Make a file system and mount it:

```
mke2fs -b 4096 /dev/dasd/7582/part1
    cd /usr
    mkdir IBMdb2
    mount /dev/dasd/7582/part1 /usr/IBMdb2
```

j. Format 7583 and mount at /home:

```
dasdfmt -d cdl -b 4096 -f /dev/dasd/7583/device
  fdasd /dev/dasd/7583/device
  n (to add a new partition)
  enter (enter twice to use the whole volume)
  w (to write & exit)
```

k. Change the label to BILV1H:

```
  fdasd -l BILV1H /dev/dasd/7583/device
  y (to confirm)
  w (to write & exit)
```

l. Make a file system:

```
mke2fs -b 4096 /dev/dasd/7583/part1
```

m. Then do the mount

```
mount /dev/dasd/7583/part1 /home
```

10. Update /etc/fstab :

```
/dev/dasd/7580/part1   /            ext2    defaults,errors=remount-ro 0 1
/dev/dasd/7581/part1   swap     swap    defaults    0       0
/dev/dasd/7582/part1   /usr/IBMdb2  ext2    defaults,errors=remount-ro 0 1
/dev/dasd/7583/part1   /home        ext2    defaults,errors=remount-ro 0 1
none                   /proc        proc    defaults    0       0
```

A.3 Cloning a Linux image on z/VM

We cloned the Linux image known as guest1 to create a new image known as guest3.

We wished to copy the following objects from the guest1 system:

Table A-1 Objects copied from guest1

Linux Partition	VM Type	Virtual Addr	Real Addr	Cloned Real Addr
root	full pack minidisk	7580	E735	E737
swap	minidisk	7581	8926	8B26
/usr/IBMdb2	minidisk	7582	8926	8B26
/home	dedicated volume	7583	8927	8B27

Here is the procedure we used:

1. Identify the volumes to be used for the guest3 image. The real addresses we used are shown in Table A-1 under Cloned Real Addr.
2. Make the dasd to contain the cloned root directory accessable to the guest1 image. Some changes are required to the cloned root directory for things like IP address and host name. In order to change the cloned root directory from guest1, we added the E737 address to the VM directory entry for guest1, and gave it the virtual address 75D0, which we added to guest1's parmfile. Then we ran zipl on guest1.
3. Shut down guest1 and logoff the userid from VM.

4. From a VM ID, use the DDR utility to copy the three real addresses to their corresponding targets. We used three IDs and ran the DDRs concurrently. Here are the statements we used to run one of the DDRs from the ID MIKET:

```
ATTACH  E735  MIKET  E735
ATTACH  E737  MIKET  E737
DDR
SYSPRINT  CONS
INPUT   E735  3390
OUTPUT  E737  3390
COPY    ALL
```

5. Run ICKDSF to reformat the target volume of each DDR to give it a new volume serial label. Here are the ICKDSF statements we used:

```
ICKDSF
 CONSOLE
 CONSOLE
 REFORMAT UNIT(E737) VOLID(BILV3R) NVFY
 REPLY U
 END
```

6. Logon the guest1 ID and IPL the guest1 image. Mount the volume containing the cloned root directory. Update the files that require customization, and run zipl. In our case, we modified the following files:

```
/etc/chandev.conf
/etc/fstab
/etc/sysconfig/network
/etc/sysconfig/network-scripts/ifcfg-*
/boot/parmfile
```

7. Unmount the cloned root directory volume from guest1. Detach the cloned root directory volume from the guest1 VM Id.
8. Update the VM directory with entries for guest3, using the cloned volumes.
9. Logon guest3 to VM and IPL Linux.

B

DB2 for Linux scripts to create the data mart

DB2 for Linux scripts to create the data mart involve the following:

- Creating the database
- Creating the buffer pools
- Creating the table spaces
- Creating the tables

B.1 Creating the database

We used the following commands to create four databases:

```
create db N1 on/mnt/BIL00A
create db guest1 on/mnt/BIL0010A
create db guest2 on/mnt/BIL20A
create db guest3 on/mnt/BIL30A
```

B.2 Creating the buffer pools

We created the bufferpool for the database as follows. The bufferpool created is nearly 500 MB, which is reasonable for a Linux having virtual storage of 2 GB.

```
connect to N1;
create bufferpool BP1 size 30000  pagesize 16k;
connect reset;
terminate;
```

B.3 Creating table spaces

Each database has eight table spaces defined in it. We give here the scripts to create the eight table spaces.

```
CONNECT TO N1;

CREATE TABLESPACE TSITEM
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSITEM' 70943,
   FILE '/MNT/BIL010/TSITEM' 70943,
   FILE '/MNT/BIL109/TSITEM' 70943,
   FILE '/MNT/BIL110/TSITEM' 70943,
   FILE '/MNT/BIL209/TSITEM' 70943,
   FILE '/MNT/BIL210/TSITEM' 70943,
   FILE '/MNT/BIL309/TSITEM' 70943,
   FILE '/MNT/BIL310/TSITEM' 70943,
   FILE '/MNT/BIL409/TSITEM' 70943,
   FILE '/MNT/BIL410/TSITEM' 70943,
   FILE '/MNT/BIL509/TSITEM' 70943,
   FILE '/MNT/BIL510/TSITEM' 70943,
   FILE '/MNT/BIL609/TSITEM' 70943,
   FILE '/MNT/BIL610/TSITEM' 70943,
   FILE '/MNT/BIL709/TSITEM' 70943,
   FILE '/MNT/BIL710/TSITEM' 70943)
EXTENTSIZE 32
PREFETCHSIZE 512
BUFFERPOOL BP1;

CREATE TABLESPACE TSPORD
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSPORD' 66355,
   FILE '/MNT/BIL010/TSPORD' 66355,
   FILE '/MNT/BIL109/TSPORD' 66355,
   FILE '/MNT/BIL110/TSPORD' 66355,
   FILE '/MNT/BIL209/TSPORD' 66355,
   FILE '/MNT/BIL210/TSPORD' 66355,
   FILE '/MNT/BIL309/TSPORD' 66355,
   FILE '/MNT/BIL310/TSPORD' 66355,
   FILE '/MNT/BIL409/TSPORD' 66355,
   FILE '/MNT/BIL410/TSPORD' 66355,
   FILE '/MNT/BIL509/TSPORD' 66355,
   FILE '/MNT/BIL510/TSPORD' 66355,
   FILE '/MNT/BIL609/TSPORD' 66355,
   FILE '/MNT/BIL610/TSPORD' 66355,
   FILE '/MNT/BIL709/TSPORD' 66355,
   FILE '/MNT/BIL710/TSPORD' 66355)
EXTENTSIZE 32
PREFETCHSIZE 512
BUFFERPOOL BP1;

CREATE TABLESPACE TSCUST
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSCUST' 81347,
   FILE '/MNT/BIL010/TSCUST' 81347,
   FILE '/MNT/BIL109/TSCUST' 81347,
   FILE '/MNT/BIL110/TSCUST' 81347,
   FILE '/MNT/BIL209/TSCUST' 81347,
   FILE '/MNT/BIL210/TSCUST' 81347,
   FILE '/MNT/BIL309/TSCUST' 81347,
   FILE '/MNT/BIL310/TSCUST' 81347,
   FILE '/MNT/BIL409/TSCUST' 81347,
   FILE '/MNT/BIL410/TSCUST' 81347,
   FILE '/MNT/BIL509/TSCUST' 81347,
   FILE '/MNT/BIL510/TSCUST' 81347,
   FILE '/MNT/BIL609/TSCUST' 81347,
   FILE '/MNT/BIL610/TSCUST' 81347,
   FILE '/MNT/BIL709/TSCUST' 81347,
   FILE '/MNT/BIL710/TSCUST' 81347)
EXTENTSIZE 32
PREFETCHSIZE 512
BUFFERPOOL BP1;

CREATE TABLESPACE TSVENDOR
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSVENDOR' 4751,
   FILE '/MNT/BIL010/TSVENDOR' 4751,
   FILE '/MNT/BIL109/TSVENDOR' 4751,
   FILE '/MNT/BIL110/TSVENDOR' 4751,
   FILE '/MNT/BIL209/TSVENDOR' 4751,
   FILE '/MNT/BIL210/TSVENDOR' 4751,
   FILE '/MNT/BIL309/TSVENDOR' 4751,
   FILE '/MNT/BIL310/TSVENDOR' 4751,
   FILE '/MNT/BIL409/TSVENDOR' 4751,
   FILE '/MNT/BIL410/TSVENDOR' 4751,
   FILE '/MNT/BIL509/TSVENDOR' 4751,
   FILE '/MNT/BIL510/TSVENDOR' 4751,
   FILE '/MNT/BIL609/TSVENDOR' 4751,
   FILE '/MNT/BIL610/TSVENDOR' 4751,
   FILE '/MNT/BIL709/TSVENDOR' 4751,
   FILE '/MNT/BIL710/TSVENDOR' 4751)
EXTENTSIZE 32
PREFETCHSIZE 512
BUFFERPOOL BP1;

CREATE TABLESPACE TSPRVEND
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSPRVEND' 15729,
   FILE '/MNT/BIL010/TSPRVEND' 15729,
   FILE '/MNT/BIL109/TSPRVEND' 15729,
   FILE '/MNT/BIL110/TSPRVEND' 15729,
   FILE '/MNT/BIL209/TSPRVEND' 15729,
   FILE '/MNT/BIL210/TSPRVEND' 15729,
   FILE '/MNT/BIL309/TSPRVEND' 15729,
   FILE '/MNT/BIL310/TSPRVEND' 15729,
   FILE '/MNT/BIL409/TSPRVEND' 15729,
   FILE '/MNT/BIL410/TSPRVEND' 15729,
   FILE '/MNT/BIL509/TSPRVEND' 15729,
   FILE '/MNT/BIL510/TSPRVEND' 15729,
   FILE '/MNT/BIL609/TSPRVEND' 15729,
   FILE '/MNT/BIL610/TSPRVEND' 15729,
   FILE '/MNT/BIL709/TSPRVEND' 15729,
   FILE '/MNT/BIL710/TSPRVEND' 15729)
EXTENTSIZE 32
PREFETCHSIZE 512
BUFFERPOOL BP1;

CREATE TABLESPACE TSPROD
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSPROD' 3031,
   FILE '/MNT/BIL010/TSPROD' 3031,
   FILE '/MNT/BIL109/TSPROD' 3031,
   FILE '/MNT/BIL110/TSPROD' 3031,
   FILE '/MNT/BIL209/TSPROD' 3031,
   FILE '/MNT/BIL210/TSPROD' 3031,
   FILE '/MNT/BIL309/TSPROD' 3031,
   FILE '/MNT/BIL310/TSPROD' 3031,
   FILE '/MNT/BIL409/TSPROD' 3031,
   FILE '/MNT/BIL410/TSPROD' 3031,
   FILE '/MNT/BIL509/TSPROD' 3031,
   FILE '/MNT/BIL510/TSPROD' 3031,
   FILE '/MNT/BIL609/TSPROD' 3031,
   FILE '/MNT/BIL610/TSPROD' 3031,
   FILE '/MNT/BIL709/TSPROD' 3031,
   FILE '/MNT/BIL710/TSPROD' 3031)
EXTENTSIZE 32
PREFETCHSIZE 512
BUFFERPOOL BP1;

CREATE TABLESPACE TSREGION
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSREGION' 100)
EXTENTSIZE 8
PREFETCHSIZE 8
BUFFERPOOL BP1;

CREATE TABLESPACE TSTERR
PAGESIZE 16384
MANAGED BY DATABASE
USING  (FILE '/MNT/BIL009/TSTERR' 100)
EXTENTSIZE 8
PREFETCHSIZE 8
BUFFERPOOL BP1;

CONNECT RESET;
TERMINATE;
```

B.4 Creating the tables

Each table space contains one table.

We show here the scripts for creating the tables.

```
Connect to N1;

CREATE TABLE ITEM ( I_PORDKEY INT NOT NULL,
         I_PRODKEY INT NOT NULL,
         I_VENDKEY INT NOT NULL,
         I_LINENUMBER INT NOT NULL,
         I_QUANTITY REAL NOT NULL,
         I_SUBTOTAL REAL NOT NULL,
         I_DISCOUNT REAL NOT NULL,
         I_TAX REAL NOT NULL,
         I_RETURNFLAG CHAR(1) NOT NULL,
         I_STATUS CHAR(1) NOT NULL,
         I_SHIPDATE DATE NOT NULL,
         I_COMMITDATE DATE NOT NULL,
         I_RECEIPTDATE DATE NOT NULL,
         I_SHIPINSTRUCT CHAR(25) NOT NULL,
         I_SHIPMETHOD CHAR(10) NOT NULL,
         I_COMMENT VARCHAR(44) NOT NULL)
   IN  TSITEM;

CREATE TABLE CUSTOMER (C_CUSTKEY INT NOT NULL,
         C_NAME VARCHAR(25) NOT NULL,
         C_ADDRESS VARCHAR(40) NOT NULL,
         C_TERRKEY INTEGER NOT NULL,
         C_PHONE CHAR(15) NOT NULL,
         C_ACCTBAL REAL NOT NULL,
         C_MKTSEG CHAR(10) NOT NULL,
         C_COMMENT VARCHAR(117) NOT NULL)
   IN  TSCUST;

CREATE TABLE PURCHORD (PO_PORDKEY INT NOT NULL,
         PO_CUSTKEY INT NOT NULL,
         PO_STATUS CHAR(1) NOT NULL,
         PO_TOTALPRICE REAL NOT NULL,
         PO_PODATE DATE NOT NULL,
         PO_PORDPRIORITY CHAR(15) NOT NULL,
         PO_CLERK CHAR(15) NOT NULL,
         PO_SHIPPRIORITY INT NOT NULL,
         PO_COMMENT VARCHAR(79) NOT NULL)
   IN TSPORD;

CREATE TABLE VENDOR (V_VENDKEY INT NOT NULL,
         V_NAME CHAR(25) NOT NULL,
         V_ADDRESS VARCHAR(40) NOT NULL,
         V_TERRKEY INTEGER NOT NULL,
         V_PHONE CHAR(15) NOT NULL,
         V_ACCTBAL REAL NOT NULL,
         V_COMMENT VARCHAR(101) NOT NULL)
   IN TSVENDOR;

CREATE TABLE TERRITORY (T_TERRKEY INTEGER NOT NULL,
         T_NAME      CHAR(25) NOT NULL,
         T_REGIONKEY INTEGER NOT NULL,
         T_COMMENT   VARCHAR(152) NOT NULL)
   IN  TSTERR;

CREATE TABLE REGION (R_REGIONKEY INTEGER NOT NULL,
         R_NAME      CHAR(25) NOT NULL,
         R_COMMENT   VARCHAR(152) NOT NULL)
   IN  TSREGION;


CREATE TABLE PRODUCT (P_PRODKEY INT NOT NULL,
         P_NAME VARCHAR(55) NOT NULL,
         P_MFGR CHAR(25) NOT NULL,
         P_MODEL CHAR(10) NOT NULL,
         P_TYPE VARCHAR(25) NOT NULL,
         P_SIZE INT NOT NULL,
         P_CONTAINER CHAR(10) NOT NULL,
         P_RETAILPRICE FLOAT NOT NULL,
         P_COMMENT  VARCHAR(23) NOT NULL)
   in TSPROD;

CREATE TABLE PRODVEND ( PV_PRODKEY INT NOT NULL,
         PV_VENDKEY INT NOT NULL,
         PV_AVAILQTY INT NOT NULL,
         PV_VENDCOST FLOAT NOT NULL,
         PV_COMMENT VARCHAR(199) NOT NULL)
   in TSPRVEND;

connect reset;
terminate;
```

C

Networking definitions

This appendix contains the networking definitions used in this project. It includes:

- OSA-Express Gigabit definitions
- HiperSocket definitions
- General TCP/IP settings
- HiperSocket customization using Hardware Configuration Dialog (HCD)

C.1 OSA-Express Gigabit definitions

OSA-Express Gigabit resources:

Table C-1 OSA-Express Gigabit resources

LPAR IMAGE	CHPID	DEVICE #	IP ADDRESS
P06 (z/OS)	0B	1860-1863	192.168.140.106
L16 Native Linux	10	1900-1903	192.168.140.116
V62 Guest1 Linux	98	1F00-1F03	192.168.140.191
V62 Guest2 Linux	98	1F04-1F07	192.168.140.192
V62 Guest3 Linux	98	1F08-1F0B	192.168.140.193

C.1.1 OSA Express GBit definitions for z/OS

These are the OSA-Express Gbit definitions used on z/OS:

SYS1.VTAMLST (MPCTRL06) TRLE Definition

```
TRLP060B TRLE  LNCTL=MPC,                                                X
               READ=1860,                                                X
               WRITE=1861,                                               X
               DATAPATH=(1862),                                          X
               PORTNAME=GB0BP0L0,                                        X
               MPCLEVEL=QDIO
```

TCPIP.PROFILE

```
DEVICE GB0BP0L0 MPCIPA NONROUTER
LINK P06GB#1 IPAQENET GB0BP0L0

HOME
   9.7.22.74       PELP01
   192.168.140.106 P06GB#1
   10.1.1.106      LIQDCE5
   11.1.1.106      LIQDCE6
   12.1.1.106      LIQDCE7
   13.1.1.106      LIQDCF3

GATEWAY
192.168.140 =         P06GB#1   1500 0

START GB0BP0L0
```

C.1.2 OSA Express GBit definitions for Linux on zSeries

We include the definitions for Native Linux and the definitions for the Linux Virtual Images.

C.1.2.1 Native Linux

We did the following:

- Add to /etc/modules.conf:

```
alias eth2 qeth
```

- Add to /etc/chandev.conf:

```
add_parms,0x10,0x1900,0x1902,portname:GB10P0
qeth2,0x1900,0x1901,0x1902,0,0,0
```

- Add file /etc/sysconfig/network-scripts/ifcfg-eth2:

```
DEVICE=eth2
USERCTL=no
```

```
BOOTPROTO=yes
BROADCAST=192.168.140.255
NETWORK=192.168.140.0
NETMASK=255.255.255.0
IPADDR=192.168.140.116
```

C.1.2.2 Linux Virtual Images

We did the following

- Add to /etc/modules.conf:

```
alias eth1 qeth
```

- Add to /etc/chandev.conf:

```
add_parms,0x10,0x1f00,0x1f02,portname:GB98P0 (Use 0x1f04,0x1f06 for guest2,
0x1f08,0x1f0a for guest3)
qeth1,0x1f00,0x1f01,0x1f02,0,0,0  (Use 1f04-1f06 for guest2, 1f08-1f0a for
guest3)
```

- Add file /etc/sysconfig/network-scripts/ifcfg-eth1:

```
DEVICE=eth1
USERCTL=no
BOOTPROTO=yes
BROADCAST=192.168.140.255
NETWORK=192.168.140.0
NETMASK=255.255.255.0
IPADDR=192.168.140.191    (Use 192 for guest2, 193 for guest3)
```

C.2 HiperSockets definitions

We defined four IQD CHPIDs, shared between the z/OS and z/VM LPARs. Each IQD CHPID was assigned a different maximum frame size (MFS) as shown in Table C-2:

IQD CHPID	Max Frame Size
E5	16KB
E6	24KB
E7	40KB
F3	64KB

Table C-2 IQD CHPID and MFS

The z/OS V1R2.0 CS: IP Configuration Guide has this discussion of MFS, how it is defined using HCD or IOCP, and how it relates to the TCP/IP Maximum Transmission Unit:

> iQDIO Maximum Frame Size: The iQDIO hardware supports four different frame sizes referred to as the iQDIO MFS (Maximum Frame Size). Using HCD (or IOCP), the iQDIO MFS is configured on the IQD CHPID using the 'OS=' parameter. All LPARs communicating over the same IQD CHPID will then use the same IQD MFS. The MFS affects the largest packet that TCP/IP can transmit.

Table C-3depicts the four possible TCP/IP MTU sizes resulting from the iQDIO frame sizes:

OS=value	iQDIO Frame Size	TPC/IP MTU Size
00 (default)	16	8
40	24	16
80	40	32
C0	64	56

Table C-3 TCP/IP MTU sizes resulting from iQDIO frame sizes

Our test cases involved the transfer of large amounts of data: A DB2 query returning a large answer set, or the ftp of many large files. Consequently, we chose to use IQD CHPID F3, with a max frame size of 64KB, in our tests.

Each operating system image sharing an IQD CHPID was assigned an IP address corresponding to that CHPID, as shown in the table below. To transfer data using HiperSockets over CHPID F3, an IP address of 13.1.1.xxx was specified. For example, for a user on the Linux Guest1 image to ftp a file from z/OS using HiperSockets with a 64KB MFS, the user would ftp to address 13.1.1.106.

Table C-4 shows the HiperSocket (iQDIO) resources.

LPAR IMAGE	CHPID	DEVICE #	IP Address
P06 (z/OS)	E5	2000,32	10.1.1.106
	E6	2100,32	11.1.1.106
	E7	2300,32	12.1.1.106
	F3	2400,32	13.1.1.106
V62 Guest1 Linux	E5	2000-2003	10.1.1.191
	E6	2100-2103	11.1.1.191
	E7	2300-2303	12.1.1.191
	F3	2400-2403	13.1.1.191
V62 Guest2 Linux	E5	2004-2007	10.1.1.192
	E6	2104-2017	11.1.1.192
	E7	2304-2307	12.1.1.192
	F3	2404-2407	13.1.1.192
V62 Guest3 Linux	E5	2008-200B	10.1.1.193
	E6	2108-210B	11.1.1.193
	E7	2308-230B	12.1.1.193
	F3	2408-240B	13.1.1.193

Table C-4 HiperSocket (iQDIO) resources

C.2.1 IQDIO definitions for z/OS

We defined the following:

SYS1.VTAMLST (ATCSTR xx)

```
VTAMSIM=IQDIO,
```

TCPIP.PROFILE

```
; IQDIO DEVICE FOR CHPID E5
;
DEVICE IUTIQDE5 MPCIPA AUTORESTART
LINK LIQDCE5 IPAQIDIO IUTIQDE5
;
; IQDIO DEVICE FOR CHPID E6
;
DEVICE IUTIQDE6 MPCIPA AUTORESTART
LINK LIQDCE6 IPAQIDIO IUTIQDE6
;
; IQDIO DEVICE FOR CHPID E7
;
DEVICE IUTIQDE7 MPCIPA AUTORESTART
LINK LIQDCE7 IPAQIDIO IUTIQDE7
;
; IQDIO DEVICE FOR CHPID F3
;
DEVICE IUTIQDF3 MPCIPA AUTORESTART
LINK LIQDCF3 IPAQIDIO IUTIQDF3


HOME
   9.7.22.74       PELP01
   192.168.140.106 P06GB#1
   10.1.1.106      LIQDCE5
   11.1.1.106      LIQDCE6
   12.1.1.106      LIQDCE7
   13.1.1.106      LIQDCF3

GATEWAY
   10          =         LIQDCE5    8000      0
   11          =         LIQDCE6   16000      0
   12          =         LIQDCE7   32000      0
   13          =         LIQDCF3   56000      0

START IUTIQDE5
START IUTIQDE6
START IUTIQDE7
START IUTIQDF3
```

C.2.2 IQDIO Definitions for Linux on zSeries virtual images

We made the following updates:

- Add to /etc/modules.conf:

```
alias hsi2 qeth
alias hsi3 qeth
alias hsi4 qeth
alias hsi5 qeth
```

- Add to /etc/chandev.conf:

```
add_model,0x10,0x1731,5,0x1732,5,0,0
qeth2,0x2000,0x2001,0x2002,0,0,0  (Use 2004-2006 for guest2, 2008-200a for
guest3)
qeth3,0x2100,0x2101,0x2102,0,0,0  (Use 2104-2106 for guest2, 2108-210a for
guest3)
qeth4,0x2300,0x2301,0x2302,0,0,0  (Use 2304-2306 for guest2, 2308-230a for
guest3)
qeth5,0x2400,0x2401,0x2402,0,0,0  (Use 2404-2406 for guest2, 2408-240a for
guest3)
```

- Add file /etc/sysconfig/network-scripts/ifcfg-hsi2:

```
DEVICE=hsi2
USERCTL=no
ONBOOT=yes
BOOTPROTO=none
BROADCAST=10.1.1.255
NETWORK=10.1.1.0
NETMASK=255.255.255.0
IPADDR=10.1.1.191    (Use 192 for guest2, 193 for guest3)
```

- Similarly, add /etc/sysconfig/network-scripts/ifcfg-hsi3, ifcfg-hsi4, ifcfg-hsi5 with appropriate DEVICE, BROADCAST, NETWORK, and IPADDR parms. The IPADDR for each device is:

```
hsi2 - 10.1.1.191 (192 for guest2, 193 for guest3)
hsi3 - 11.1.1.191 (192 for guest2, 193 for guest3)
hsi4 - 12.1.1.191 (192 for guest2, 193 for guest3)
hsi5 - 13.1.1.191 (192 for guest2, 193 for guest3)
```

C.3 TCP/IP settings

We describe the TCP/IP settings on z/OS and Linux on zSeries.

C.3.1 TCP/IP settings on z/OS

The following statement is in the TCPIP.PROFILE on z/OS:

```
TCPCONFIG        UNRESTRICTLOWPORTS
     TCPSENDBFRSIZE 64K
     TCPRCVBUFRSIZE 64K
```

C.3.2 TCPIP settings on Linux on zSeries

We added the following statements to /etc/rc.d/rc.local on the Linux images:

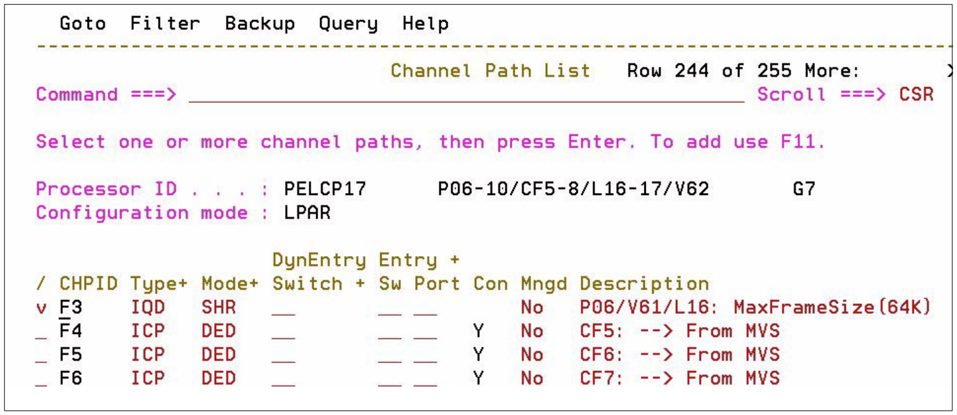
```
echo 8388608 > /proc/sys/net/core/wmem_max
echo 8388608 > /proc/sys/net/core/rmem_max
echo 65536 > /proc/sys/net/core/wmem_default
echo 65536 > /proc/sys/net/core/rmem_default
```

C.4 Customizing HiperSockets with HCD

Using Hardware Configuration Dialog (HCD) screens, we customized HiperSockets for iqdio chpid, control unit, and device definitions as follows.

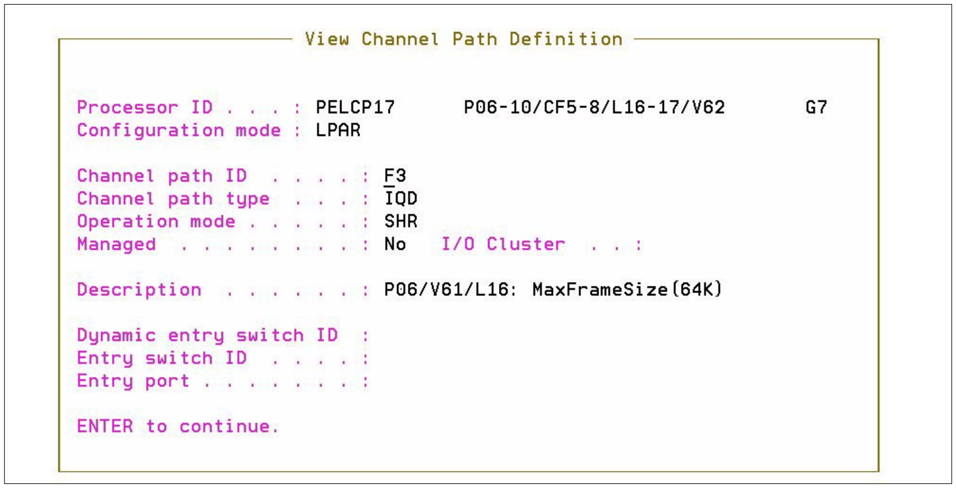
C.4.1 CHPID definitions

In the CHPID list, select the CHPID F3 (Figure C-1) to view the definition (Figure C-2 on page 117).



```
  Goto  Filter  Backup  Query  Help
------------------------------------------------------------------------------
                             Channel Path List    Row 244 of 255 More:       >
Command ===> ______________________________________________ Scroll ===> CSR

Select one or more channel paths, then press Enter. To add use F11.

Processor ID . . . : PELCP17      P06-10/CF5-8/L16-17/V62       G7
Configuration mode : LPAR

                    DynEntry Entry +
/ CHPID Type+ Mode+ Switch + Sw Port Con Mngd Description
v F3    IQD   SHR   __       __ __       No   P06/V61/L16: MaxFrameSize(64K)
_ F4    ICP   DED   __       __ __   Y   No   CF5: --> From MVS
_ F5    ICP   DED   __       __ __   Y   No   CF6: --> From MVS
_ F6    ICP   DED   __       __ __   Y   No   CF7: --> From MVS
```

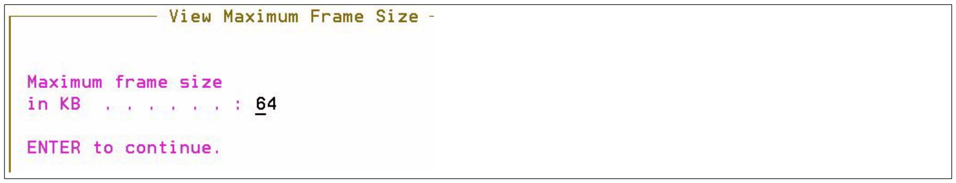
Figure C-1 Channel path list



```
 ---------------------- View Channel Path Definition ----------------------

  Processor ID . . . : PELCP17      P06-10/CF5-8/L16-17/V62       G7
  Configuration mode : LPAR

  Channel path ID  . . . . . : F3
  Channel path type  . . . . : IQD
  Operation mode . . . . . . : SHR
  Managed  . . . . . . . . . : No   I/O Cluster  . . :

  Description  . . . . . . . : P06/V61/L16:  MaxFrameSize(64K)

  Dynamic entry switch ID  :
  Entry switch ID  . . . . :
  Entry port . . . . . . . :

  ENTER to continue.
```

Figure C-2   Definition of CHPID F3

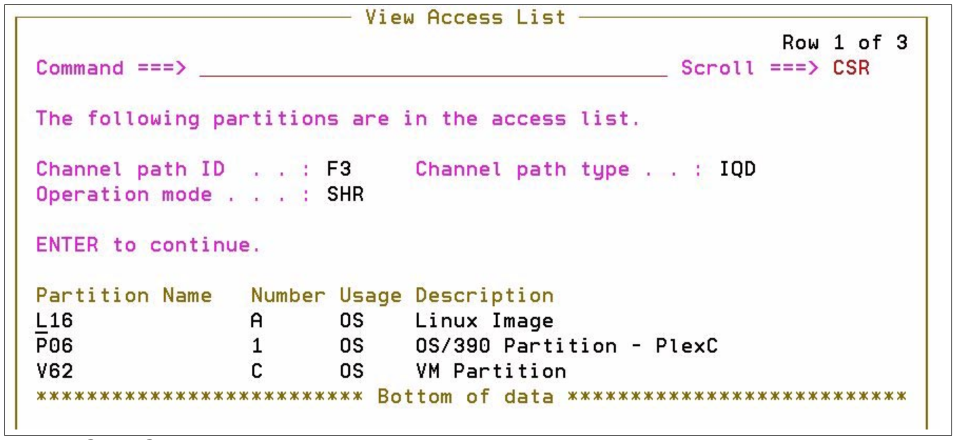
Press Enter to view the Maximum Frame Size (Figure C-3).



```
 ------------- View Maximum Frame Size -

  Maximum frame size
  in KB  . . . . . . . : 64

  ENTER to continue.
```

Figure C-3   Maximum Frame Size (MFS)

Define which LPARs have access to CHPID F3 (Figure C-4).

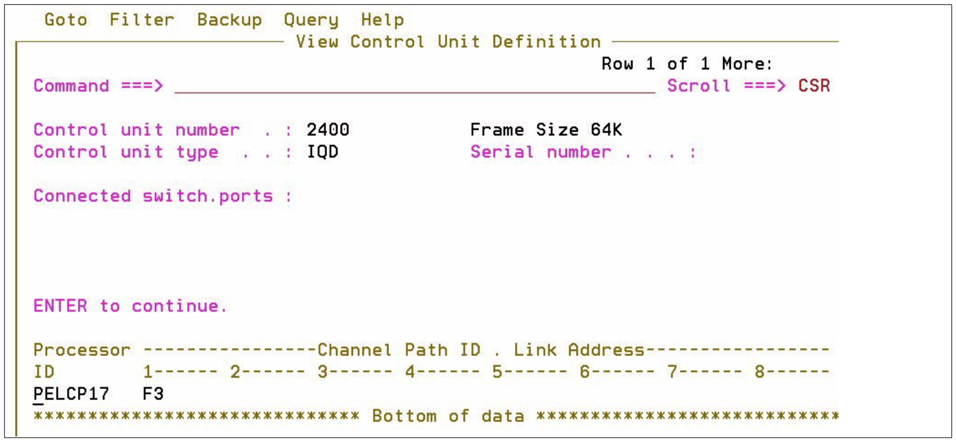


```
 ------------------------------ View Access List ------------------------------
                                                                    Row 1 of 3
 Command ===> ____________________________________________ Scroll ===> CSR

 The following partitions are in the access list.

 Channel path ID  . . : F3     Channel path type . . : IQD
 Operation mode . . . : SHR

 ENTER to continue.

 Partition Name    Number Usage Description
 L16               A      OS    Linux Image
 P06               1      OS    OS/390 Partition - PlexC
 V62               C      OS    VM Partition
 ************************** Bottom of data ***************************
```

Figure C-4   CHPID F3 access list

C.4.2 Control unit definition

Figure C-5 shows the control unit (CU) definition.

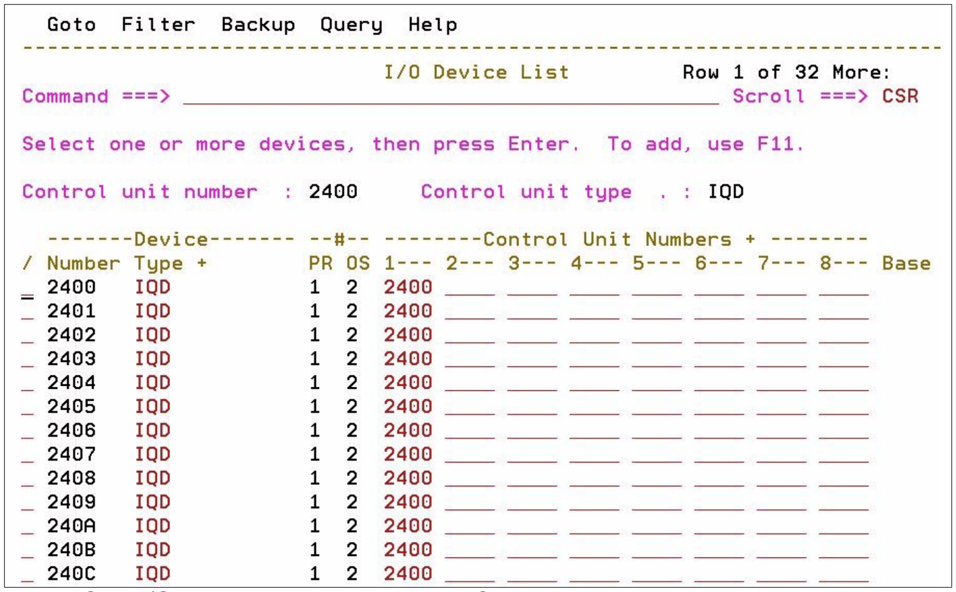


```
  Goto  Filter  Backup  Query  Help
 -------------------------- View Control Unit Definition ---------------------
                                                    Row 1 of 1 More:
 Command ===> ____________________________________________ Scroll ===> CSR

 Control unit number  . : 2400          Frame Size 64K
 Control unit type  . . : IQD           Serial number . . . :

 Connected switch.ports :




 ENTER to continue.

 Processor ----------------Channel Path ID . Link Address-----------------
 ID        1------ 2------ 3------ 4------ 5------ 6------ 7------ 8------
 PELCP17   F3
 ****************************** Bottom of data ****************************
```

Figure C-5 Control Unit definition

C.4.3 Device definition

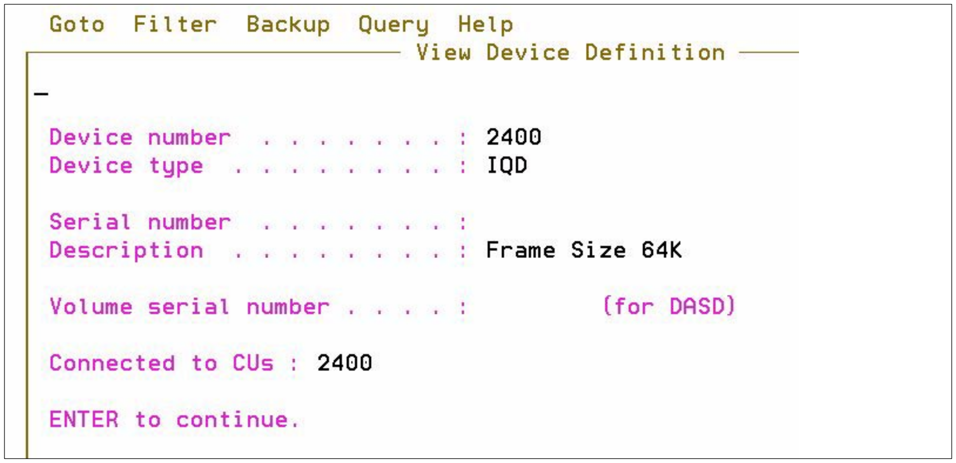
Figure C-6 shows the devices connected to the control unit.



```
  Goto  Filter  Backup  Query  Help
 ----------------------------------------------------------------------------
                              I/O Device List          Row 1 of 32 More:
 Command ===> ___________________________________________ Scroll ===> CSR

 Select one or more devices, then press Enter.  To add, use F11.

 Control unit number  : 2400     Control unit type  . : IQD

   -------Device------- --#-- --------Control Unit Numbers + --------
 / Number Type +        PR OS 1--- 2--- 3--- 4--- 5--- 6--- 7--- 8--- Base
 _ 2400   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2401   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2402   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2403   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2404   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2405   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2406   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2407   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2408   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 2409   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 240A   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 240B   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
 _ 240C   IQD           1  2  2400 ____ ____ ____ ____ ____ ____ ____
```

Figure C-6 I/O device list connected to the CU

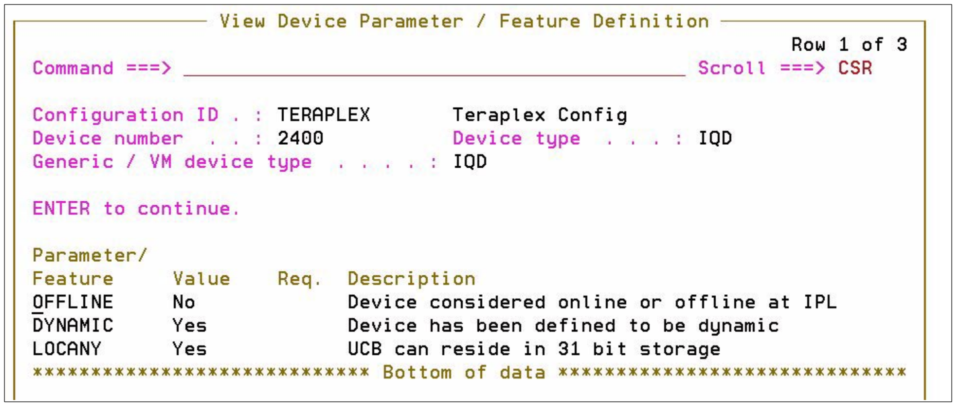
Figure C-7 shows the hardware device definition for IQD.



```
 Goto  Filter  Backup  Query  Help
                      View Device Definition

 Device number  . . . . . . . . : 2400
 Device type  . . . . . . . . . : IQD

 Serial number  . . . . . . . . :
 Description  . . . . . . . . . : Frame Size 64K

 Volume serial number . . . . :         (for DASD)

 Connected to CUs : 2400

 ENTER to continue.
```

Figure C-7 Hw device definition for IQD

Figure C-8 shows the z/OS device definition for IQD.



```
              View Device Parameter / Feature Definition
                                                          Row 1 of 3
Command ===> ____________________________________________ Scroll ===> CSR

Configuration ID . : TERAPLEX       Teraplex Config
Device number  . . : 2400           Device type  . . . : IQD
Generic / VM device type  . . . . : IQD

ENTER to continue.

Parameter/
Feature     Value    Req.  Description
OFFLINE     No             Device considered online or offline at IPL
DYNAMIC     Yes            Device has been defined to be dynamic
LOCANY      Yes            UCB can reside in 31 bit storage
***************************** Bottom of data ******************************
```

Figure C-8 z/OS device definition for IQD

D

Federated database setup

This appendix shows the following federated database setups:

- In the data mart database on the Native Linux LPAR, to access tables in DB2 on z/OS (for data population and data refresh)
- In the data mart database on the Guest1 Linux image, to access tables in DB2 on z/OS (for data population and data refresh)
- In the data mart database on the Guest1 Linux image, to access tables in the data mart database on the Guest2 Linux image (for distributed requests)

D.1 Native Linux federated database definitions

- Connect to the native1 (Native Linux data mart) database with user ID db2inst1

```
db2 => connect to native1

Database Connection Information
 Database server        = DB2/LINUX390 7.2.2
 SQL authorization ID   = DB2INST1
 Local database alias   = NATIVE1
```

- Create node definition
  - IP address of z/OS is 192.168.140.106 (via OSA Gbit)

– Port used by DB2 for z/OS is 471

```
db2 => catalog tcpip node db71gb remote 192.168.140.106 server 471
DB20000I  The SQL command completed successfully.
```

- Create a wrapper

```
db2 => create wrapper drda
DB20000I  The SQL command completed successfully.
```

- Create a server definition for DB2 on z/OS
  – The DB2 Location Name of the DB2 for z/OS subsystem is TPLEXD71

```
db2 => create server db71 type DB2/390 VERSION 7.1 WRAPPER DRDA
AUTHORIZATION MIKET PASSWORD XXX OPTIONS (NODE'db71gb',DBNAME'TPLEXD71')
DB20000I  The SQL command completed successfully
```

- Create a user mapping to connect to DB2 on z/OS with userid MIKET

```
db2 => CREATE USER MAPPING FOR db2inst1 SERVER db71 OPTIONS ( REMOTE_AUTHID
'MIKET', REMOTE_PASSWORD 'XXX' )
DB20000I  The SQL command completed successfully.
```

- Create nicknames for the tables that are in DB2 on z/OS

```
db2 => CREATE NICKNAME db71item FOR db71.lsjcruz.item
DB20000I  The SQL command completed successfully.

db2 => create nickname db71purchord for db71.lsjcruz.purchord
DB20000I  The SQL command completed successfully.

db2 => create nickname db71cust for db71.lsjcruz.customer
DB20000I  The SQL command completed successfully.

db2 => create nickname db71territory for db71.lsjcruz.territory
DB20000I  The SQL command completed successfully.

db2 => create nickname db71region for db71.lsjcruz.region
DB20000I  The SQL command completed successfully.

db2 => create nickname db71product for db71.lsjcruz.product
DB20000I  The SQL command completed successfully.

db2 => create nickname db71prodvend for db71.lsjcruz.prodvend
DB20000I  The SQL command completed successfully.

db2 => create nickname db71vendor for db71.lsjcruz.vendor
DB20000I  The SQL command completed successfully.
```

D.2 Guest1 Linux federated database definitions

WE describe the definitions to access tables in DB2 on z/OS and DB2 on Linux Guest2.

D.2.1 Definitions to access tables in DB2 on z/OS

- Connect to the guest1 (Guest1 Linux data mart) database with userID db2inst1

```
db2 => connect to guest1

Database Connection Information
 Database server        = DB2/LINUX390 7.2.2
 SQL authorization ID   = DB2INST1
 Local database alias   = GUEST1
```

- Create node definition
  - IP address of z/OS is 13.1.1.106 (via HiperSockets)
  - Port used by DB2 for z/OS is 471

```
db2 => catalog tcpip node db71h64 remote 13.1.1.106 server 471
DB20000I  The SQL command completed successfully.
```

- Create a wrapper

```
db2 => create wrapper drda
DB20000I  The SQL command completed successfully.
```

- Create a server definition for DB2 on z/OS
  - The DB2 Location Name of the DB2 for z/OS subsystem is TPLEXD71

```
db2 => create server db71 type DB2/390 VERSION 7.1 WRAPPER DRDA
AUTHORIZATION MIKET PASSWORD XXX OPTIONS (NODE'db71h64',DBNAME'TPLEXD71')
DB20000I  The SQL command completed successfully.
```

- Create a user mapping to connect to DB2 on z/OS with userid MIKET

```
db2 => CREATE USER MAPPING FOR db2inst1 SERVER db71 OPTIONS ( REMOTE_AUTHID
'MIKET', REMOTE_PASSWORD 'XXX' )
DB20000I  The SQL command completed successfully.
```

- Create nicknames for the tables that are in DB2 on z/OS

```
db2 => CREATE NICKNAME db71item FOR db71.lsjcruz.item
DB20000I  The SQL command completed successfully.

db2 => create nickname db71purchord for db71.lsjcruz.purchord
DB20000I  The SQL command completed successfully.

db2 => create nickname db71cust for db71.lsjcruz.customer
```

```
DB20000I  The SQL command completed successfully.

db2 => create nickname db71territory for db71.lsjcruz.territory
DB20000I  The SQL command completed successfully.

db2 => create nickname db71region for db71.lsjcruz.region
DB20000I  The SQL command completed successfully.

db2 => create nickname db71product for db71.lsjcruz.product
DB20000I  The SQL command completed successfully.

db2 => create nickname db71prodvend for db71.lsjcruz.prodvend
DB20000I  The SQL command completed successfully.

db2 => create nickname db71vendor for db71.lsjcruz.vendor
DB20000I  The SQL command completed successfully.
```

D.2.2 Definitions to access tables in DB2 on Guest2

- Create node definition
  - IP address of Guest2 is 13.1.1.192 (via HiperSockets)
  - Port used by DB2 on Guest2 is 50000

```
catalog tcpip node g2h64 remote 13.1.1.192 server 50000
DB20000I  The SQL command completed successfully.
```

- Create a server definition for DB2 on Guest2
  - The data mart database on the Guest2 image is called GUEST2

```
CREATE SERVER g2srv TYPE DB2/UDB VERSION 7.2.2 WRAPPER DRDA AUTHORIZATION
DB2INST1 PASSWORD XXX OPTIONS  (NODE 'g2h64', DBNAME 'GUEST2')
DB20000I  The SQL command completed successfully.
```

- Create a user mapping to connect to DB2 on Guest2 with user ID db2inst1

```
CREATE USER MAPPING FOR db2inst1 server g2srv options (remote_authid
'db2inst1',remote_password 'xxx')
DB20000I  The SQL command completed successfully.
```

- Create nicknames for the tables that are in data mart database Guest2 on the Guest2 Linux image

```
db2 => create nickname g2purchord for g2srv.db2inst1.purchord
DB20000I  The SQL command completed successfully.

db2 => create nickname g2cust for g2srv.db2inst1.customer
DB20000I  The SQL command completed successfully.
```

```
db2 => create nickname g2territory for g2srv.db2inst1.territory
DB20000I  The SQL command completed successfully.

db2 => create nickname g2region for g2srv.db2inst1.region
DB20000I  The SQL command completed successfully.

db2 => create nickname g2product for g2srv.db2inst1.product
DB20000I  The SQL command completed successfully.

db2 => create nickname g2prodvend for g2srv.db2inst1.prodvend
DB20000I  The SQL command completed successfully.

db2 => create nickname g2vendor for g2srv.db2inst1.vendor
DB20000I  The SQL command completed successfully.

db2 => create nickname g2item for g2srv.db2inst1.item
DB20000I  The SQL command completed successfully.
```

E

Data population scripts

This appendix includes the SQLs, jobs and scripts we used for the data population test scenarios.

E.1 Distributed SQL INSERT/SELECT

The following are the SQLs we used to populate each table on the data mart. These SQLs were issued on DB2 for Linux. We used one SQL for each table in the data mart. Each of these SQLs extracts data from the table or tables on z/OS and inserts the selected data into the table in the data mart.

Item table

```
insert into db2inst1.Item
select * from db71Item I
where l_prodkey between 7500001 and 9750000
and exists (select 1 from db71purchord o
where I.l_pordkey = o.po_pordkey
and o_podate between '1992-01-01' and '1997-12-31')
```

Product table

```
insert into db2inst1.product
select * from db71product
where  p_prodkey between  7500001 and 9750000
```

Product Vendor table

```
insert into db2inst1.prodvend
select * from db71prodvend
where  pv_prodkey between  7500001 and 9750000
```

Region table

```
insert into db2inst1.region
select * from db71region
```

Vendor table

```
insert into db2inst1.vendor
select * from db71vendor
```

Territory table

```
insert into db2inst1.territory
select * from db71territory
```

Customer table

```
insert into db2inst1.customer
select * from db71cust
```

Purchase Order table

```
insert into db2inst1.purchord
select * from db71purchord ORDERTBL
where po_podate between '1992-01-01' and '1997-12-31'
and Exists
(select *from db71item
where (l_prodkey > 7500000 and l_prodkey <= 9750000)
and  l_pordkey = ORDERTBL.o_pordkey)
```

E.2 HPU-FTP-LOAD scripts

There are three steps involved in this data population technique. They are unloading data from zOS using DB2 High Performance Unload(HPU), transferring the data to Linux using File Transfer Proctocal(FTP) and loading the data to the tables in the data mart using DB2 UDB EE Load Utility.

The jobs, scripts and SQLs we used for the testcase scenarios are included in this section.

E.2.1 DB2 HPU jobs

The job control cards (JCL) we used to unload the data from the z/OS data warehouse tables are the same, except for the UNLOAD statement, specified in the //SYSIN DD card, the utilid in the PARM= on EXEC statement and the unldddn1 DD for the output dataset. Here's an example of the HPU JCL.

```
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB71,HPLIN1A'
//STEPLIB  DD  DSN=DB2710.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2710.SDSNLOAD,DISP=SHR
//         DD  DSN=DB2710.HINZ110.SINZLINK,DISP=SHR
//*
//SYSIN    DD  *
UNLOAD TABLESPACE database.tablespace
SELECT ......
OUTDDN (UNLDDN1)
FORMAT DELIMITED SEP ',' DELIM '"'
/*
//SYSPRINT DD  SYSOUT=*
//*
//UNLDDN1  DD  DSN=OUTPUTDATASET,
//    DISP=(,CATLG),SPACE=(CYL,(1800,100),RLSE)
```

This HPU job uses the unload statements to determine which data to extract from which table or tables. The records in the output sequential file are in delimited format, using a comma (,) to separate the fields in a record and double quotes (") to frame character, varchar, graphic, and vargraphic columns.

The unload statements for the Item and Purchase Order tables included a correlated subquery and join, so HPU has to extract the rows from the tables by executing the SQL statement.

For tables such as the Product, Vendor, Product Vendor, Region, Territory and Customer tables, we specified partitions to be unloaded using the PART parameter. In this case, the HPU job read the data directly from the VSAM files. Similar unload statements were used for the Region, Vendor and Territory tables. For these, however, we specified "ALL" in the PART parameter because we were unloading all the rows in the table.

The SQL statements for the Item and PurchaseOrder tables:

Unload statements

We used the following unload statements:

Item table

This is one of the SQLs we used to extract data from the Item table. This scans through the Purchase Order (index access only) table to verify that the Item was ordered within the last five years. The matching row on the Item table is then selected from the Item table.

```
UNLOAD TABLESPACE LINBIDB.TSITEM
SELECT * FROM LSJCRUZ.ITEM, LSJCRUZ.PurchOrd
WHERE I_PORDKEY = PO_PORDKEY
AND I_PORDKEY <= 112500000
AND (I_PRODKEY > 7500000 AND I_PRODKEY <= 9750000)
AND PO_PODATE BETWEEN '1992-01-01' AND '1997-12-31'
FOR FETCH ONLY;
```

As mentioned earlier, 16 jobs were created to extract the data from the Item table. The SQLs on each of these jobs are similar to the above example. The only difference between them is the value on the PO_PORDKEY <= predicate. This column in the Item table is what we used to specify which 5 partitions to unload from the Item table.

Purchase Order table

Here's an example of the unload statement and SQL we used to unload 5 years of data for certain products from the Purchase Order table.

```
UNLOAD TABLESPACE LINBIDB.TSPORD
SELECT * FROM LSJCRUZ.PURCHORD PORDTBL
WHERE PO_PORDATE BETWEEN '1992-01-01' AND '1997-12-31'
AND   (PO_PORDKEY BETWEEN 225000000 AND 450000000)
AND EXISTS
  (SELECT * FROM LSJCRUZ.ITEM
   WHERE (I_PRODKEY BETWEEN 7500000 AND 9750000)
AND  I_PORDKEY = PORDTBL.PO_PORDKEY)
FOR FETCH ONLY;
```

There were 8 HPU jobs created to extract the data from the Purchase Order table in the z/OS data warehouse. The Unload and SQL statements are similar except for the values in the PO_PORDKEY. We used this column to specify the range we want to unload for each HPU job.

Customer table

We created 8 HPU jobs to extract all of the data from the z/OS Customer table. Below is an example of the Unload statement we used to implement this.

```
UNLOAD TABLESPACE LINBIDB.TSCUST
PART(1,2,3,4,5,6,7,8,9,10)
SELECT * FROM LSJCRUZ.CUSTOMER
FOR FETCH ONLY;
```

The only difference between the 8 Unload statements is the partition numbers specified in the PART parameter. This parameter specifies from which partitions to extract the data.

Product table

Only one HPU job is created to extract the data we need to load into the Product table in the data mart.

```
UNLOAD TABLESPACE LINBIDB.TSPROD
PART(11,12,13)
SELECT *
FROM LSJCRUZ.PRODUCT
FOR FETCH ONLY;
```

ProdVend table

Similar to the Product table, we used only one HPU job to unload all the data from the ProdVend table on the z/OS data warehouse to load into the data mart.

```
UNLOAD TABLESPACE LINBIDB.TSPRVEND
PART(11,12,13)
SELECT * FROM LSJCRUZ.PRODVEND
FOR FETCH ONLY;
```

Region table

Here’s the Unload statement to extract all the rows from the Region table:

```
UNLOAD TABLESPACE LINBIDB.TSREGION
PART(ALL)
SELECT * FROM LSJCRUZ.REGION
FOR FETCH ONLY;
```

Vendor table

To unload all the rows from the Vendor table, we used the following:

```
UNLOAD TABLESPACE LINBIDB.TSVENDOR
PART(ALL)
SELECT * FROM LSJCRUZ.VENDOR
FOR FETCH ONLY;
```

Territory table

Here’s the unload statement we used to extract all the rows from the Territory table:

```
UNLOAD TABLESPACE LINBIDB.TSTERR
PART(ALL)
SELECT * FROM LSJCRUZ.TERRITORY
FOR FETCH ONLY;
```

E.2.2 Data transfer through LAN connection

We transmitted 37 delimited sequential files from the data warehouse server (z/OS) to Native Linux. We used a script to transmit all of these files at the same time. We used an FTP command for each of the files similar to the following:

```
ncftpget -a -u miket -p xxxxxx -Z 192.168.140.116 /mnt/BIL10C
\\'LINBIRAW.HPUNLOAD.CUSTOMER.P1T10.ANY\\' &>~/ncftp.out1 && mv
/mnt/BIL10C/\\'LINBIRAW.HPUNLOAD.CUSTOMER.P1T10.ANY\\'
/mnt/BIL10C/hpu_cust1.data &> ~/mv.out1
```

The above command transfers a part of the Customer data file (‘LINBIRAW.HPUNLOAD.CUSTOMER.P1T10.ANY) into specific mount positions on the Linux system. The command specifies the user ID, password, and the IP address of the Linux system. It also renames the file to hpu_cust1.data as it moves the data to mnt/BIL10C. This command also captures the errors during file transfer and writes the error messages in an output file called mv.out.

E.2.3 DB2 Load utility jobs

Eight (8) load scripts were used to populate the tables in the data mart. We created one script to load each table. Each of these scripts had the same LOAD parameter. The only difference between these scripts is the input, specified after the FROM keyword and the table name.

```
connect to guest1;
load from
/mnt/BIL10C/hpu_cust1.data,
/mnt/BIL013/hpu_cust2.data,
/mnt/BIL713/hpu_cust3.data,
/mnt/BIL707/hpu_cust4.data,
/mnt/BILFT7/hpu_cust5.data,
/mnt/BILFTB/hpu_cust6.data,
/mnt/BIL70C/hpu_cust7.data,
/mnt/BIL613/hpu_cust8.data
of del modified by coldel,
replace into customer statistics yes;
connect reset;
terminate
```

This load script first connects to the database on Linux, guest1. Using the specified input files, which are in delimited format, it replaces the rows in the customer table. It collects catalog statistics as it loads the data. It disconnects from the database when the load completes.

F

Data refresh scripts

This appendix describes the distributed SQL statements we used for data refresh scenarios.

We used 8 distributed INSERT/SELECT SQL statements similar to the following to refresh the Item table. Each SQL had a different range for i_pordkey (purchase order key). In addition, the SQLs for refreshing each data mart used a different range for i_prodkey (product key).

```
Insert into db2inst1.item
select * from db71item i
where   i_pordkey <= 225000000
and i_prodkey between 15000001 and 17250000
and exists (select 1 from db71purchord po
where i.i_pordkey = po.po_pordkey
and po.po_porddate between '1998-01-01' and '1998-02-07');
```

We used 8 distributed INSERT/SELECT SQL statements similar to the following to refresh the Purchase Order table. Each SQL had a different range for po_pordkey (purchase ordey key). In addition, the SQLs for refreshing each data mart used a different range for i_prodkey (product key).

```
insert into db2inst1.purchord
select * from db71purchord PO
where po_porddate between '1998-01-01' and '1998-02-07'
and po_pordkey <= 225000000
and   Exists
(select * from db71item
where (i_prodkey > 15000000 and i_prodkey <= 17250000)
and i_pordkey = PO.po_pordkey);
```

G

Distributed request scripts

Here we give the three queries we used for the DB2 distributed request scenarios that join data from multiple data marts.

- Query1 reads 200,000 rows from Guest2

```
select i_pordkey, i_prodkey, i_vendkey, i_linenumber, i_quantity,
i_subtotal, i_discount, i_tax, i_returnflag, i_status, i_shipdate,
i_commitdate, i_receiptdate, i_shipinstruct, i_shipmethod, i_comment
from g2item where i_shipdate between '1-1-1995' and '1-7-1995' and
i_vendkey in  (select v_vendkey from vendor  where v_acctbal < 999.90);
```

- Query2 reads 1 million rows from Guest2

```
select c_custkey, c_name, c_address, c_terrkey, c_phone, c_acctbal,
c_mktseg, c_comment from purchord, g2cust where po_custkey = c_custkey and
po_pordkey < 2 and c_terrkey = 5 and c_acctbal < 5000;
```

- Query3 reads 3 million rows from Guest2

```
select i_pordkey,  i_prodkey, i_vendkey, i_linenumber, i_quantity,
i_subtotal, i_discount, i_tax, i_returnflag, i_status, i_shipdate,
i_commitdate, i_receiptdate, i_shipinstruct, i_shipmethod, i_comment
from item, g2vendor where i_vendkey = v_vendkey and i_prodkey < 7500002;
```

H

Dimensional Insight - Atlantis Installation

H.1 DI-Atlantis installation

To install the DI-Atlantis package, follow these steps:

1. Unpack DI_Atlantis-pkg.tar:

```
[diuser@biserverbiserver DI_Atlantis]$ ls -l
/home/diuser/unpack-files/DI_Atlantis/DI_Atlantis-pkg.tar -rw-rw-r--    1
diuser   diuser   63354880 Oct  2 21:07
/home/diuser/unpack-files/DI_Atlantis/DI_Atlantis-pkg.tar
[diuser@biserver DI_Atlantis]$ tar -xf
/home/diuser/unpack-files/DI_Atlantis/DI_Atlantis-pkg.tar
[diuser@biserver DI_Atlantis]$ ls
DI-Demos       MapData      ReleaseNotes  Utilities  shared
Documentation  OldVersions  Software      bin
```

2. Copy files to /home/DI_Atlantis/bin:

```
[diuser@biserver DI_Atlantis]$ cd bin
[diuser@biserver bin]$ cp ../Software/integ-2.1.5b-linux integ
[diuser@biserver bin]$ cp ../Software/builder-3.0.17 builder
[diuser@biserver bin]$ ls
README  builder  integ
```

3. Add /home/DI_Atlantis/bin to PATH:

```
diuser@biserver DI_Atlantis]$ export PATH=$PATH:/home/DI_Atlantis/bin
```

H.1.1 Installing DI-DiveLine

To install DI-DiveLine, follow these steps:

1. Unpack the tar file with the DI-DiveLine package:

```
[diuser@biserver DI_Atlantis]$ ls Software/di-diveline-4.3.8-linux.tar
Software/di-diveline-4.3.8-linux.tar
[diuser@biserver DI_Atlantis]$ tar -xf Software/di-diveline-4.3.8-linux.tar
[diuser@biserver DI_Atlantis]$ ls
DI-Demos       MapData      ReleaseNotes  Utilities  diveline
Documentation  OldVersions  Software      bin        shared
[diuser@biserver DI_Atlantis]$ ls diveline
INSTALL  README  bin  cgi-bin  docs  html  install-di-diveline  samples
webdir
```

2. Add /home/DI_Atlantis/bin to PATH:

```
[diuser@biserver DI_Atlantis]$ export PATH=$PATH:/home/DI_Atlantis/bin
[diuser@biserver DI_Atlantis]$ cd diveline
[diuser@biserver diveline]$ ls
INSTALL  README  bin  cgi-bin  docs  html  install-di-diveline  samples
webdir
[diuser@biserver diveline]$ ls
INSTALL  README  bin  cgi-bin  docs  html  install-di-diveline  samples
webdir
[diuser@biserver diveline]$ ls -l
total 40
-rw-rw-r--    1 diuser   diuser       3955 May  2  2000 INSTALL
-rw-rw-r--    1 diuser   diuser        735 Sep 28 05:57 README
drwxrwxr-x    2 diuser   diuser       4096 Oct  2 21:15 bin
drwxrwxr-x    2 diuser   diuser       4096 Oct  2 21:15 cgi-bin
drwxrwxr-x    2 diuser   diuser       4096 Oct  2 21:15 docs
drwxrwxr-x    2 diuser   diuser       4096 Oct  2 21:15 html
-rw-rw-r--    1 diuser   diuser       4407 Aug 17 00:14 install-di-diveline
drwxrwxr-x    2 diuser   diuser       4096 Oct  2 21:15 samples
drwxrwxr-x    2 diuser   diuser       4096 Oct  2 21:15 webdir
```

3. Install DI-DiveLine:

```
[diuser@biserver diveline]$ chmod 775 install-di-diveline
[diuser@biserver diveline]$ ./install-di-diveline
Welcome to the DI-DiveLine installation script.
This install script will first ask you a series of questions.
At the end, it'll ask you if you want to go ahead and do the install.
Nothing will be modified until after that point.
Defaults for questions will appear in square brackets [].  Hitting
carriage return will automatically choose those defaults.  If you
don't know what to do, trust the defaults.
```

```
To abort the installation, hit Control-C.
Verifying current directory...
Found installation files.
Where should DI-DiveLine store configuration files and temporary files?
[/usr/local/lib/diveline]  /home/DI_Atlantis/diveline
DI-DiveLine can run at 3 security levels:
  Level 0:  No security checking (all users have access to all models
            and divebooks in the models directory).
  Level 1:  Security checking based on web login.
            If a model is not listed in the config file, then ALL users
            have access to it.
  Level 2:  Security checking based on web login.
            If a model is not listed in the acl file, then NO users
            have access to it.
What security level should DI-DiveLine run at? [0]
Install sample model and divebook? [y]

OK.  Go ahead and install? [y]
Integer "security_level": 0
0
Max Users: 20
0
Authentication Type: own
0
Debug Level: 0
0
0
0
0
[diuser@biserver diveline]$
```

4. As root, add the following line to /etc/services to specify which port which DI-DiveLine will listen to:

```
di-diveline     2130/tcp                        # DI-DiveLine
```

5. Add file /etc/xinetd.d/di-diveline with contents to specify the di-diveline executable which runs at the di-diveline port, and the dataroot directory which DI-DiveLine uses for its configuration:

```
service di-diveline
{
        flags           = REUSE
        socket_type     = stream
        wait            = no
        user            = root
        server          = /home/DI_Atlantis/diveline/bin/di-diveline
        log_on_failure  += USERID
        disable         = no
        server_args     = -dataroot /home/DI_Atlantis/diveline
}
```

6. As root, edit /etc/xinetd.conf and change or set the “instances” variable to a value greater than the maximum number of concurrent connections allowed.
7. Restart the xinetd daemon, so that the DI-DiveLine program will listen at the specified port in /etc/services file. Also set the instances number to how many users you want to allow to connect concurrently in the /etc/xinted.conf file:

```
[root]% killall -USR1 xinetd
[root]% exit
```

8. Log back in as the non-admin user and connect to DI-DiveLine through a telnet session to test the connection:

```
[diuser@biserver diveline]$ telnet 13.2.22.191 2130
Trying 13.2.22.191...
Connected to bilin.pok.ibm.com (13.2.22.191).
Escape character is '^]'.
version
diuser1
4.3 (8)
0
file_list
1
demo-divebook.dbk
1
demo-sales.mdl
1
demo-dairy/
0
quit
Connection closed by foreign host.
[diuser@biserver diveline]$
```

9. Add an administrator and a test user:

```
[diuser@biserver diveline]$ export DIDATA_DIR=/home/DI_Atlantis/diveline
[diuserdiuser@biserver diveline]$ cd bin
[diuser@biserver bin]$ ./dicfg add user -user admin -password admin
-admin_flag TRUE
0
[diuser@biserver bin]$ ./dicfg add user -user test -password test
-admin_flag FALSE 0
```

H.2 Installing Web components

DI has two Web components:

- DI-WebDiver
- DI-ReportDiver

H.2.1 Installing DI-WebDiver & DI-ReportDiver

To install the components, follow these steps:

1. Unpack the WebDiver and ReportDiver packages:

```
[diuser@biserver DI_Atlantis]$ tar -xf Software/webdiver-4.3.5.tar
[diuser@biserver DI_Atlantis]$ tar -xf Software/reportdiver-4.3.5.tar
[diuser@biserver DI_Atlantis]$ ls
DI-Demos       MapData      ReleaseNotes  Utilities  diveline     shared
Documentation  OldVersions  Software      bin        reportdiver  webdiver
[diuser@biserver DI_Atlantis]$ pwd
/home/DI_Atlantis
```

2. Add the html directory to the webserver's path:

```
This involves either setting up an aliases in your webserver's
configururation file or moving the html directory provided by DI to a path
accessable to the server. For our test we simply moved
/home/DI_Atlantis/webdiver/html to /var/www/webdiver where our apache
server would then distribute the DI index.html file which provides a link
to the java DI-WebDiver.
[diuser@biserver DI_Atlantis]$ mv ~diuser/DI_Atlantic/webdiver/html
/var/vvv/html/webdiver
And the same proceedure for ReportDiver;
[diuser@biserver DI_Atlantis]$ cp -R ~diuser/DI_Atlantic/reportdiver/html
/var/vvv/html/reportdiverls
```

H.3 Creating data extracts

At the time of writing, DI-Atlantis does not support ODBC connectivity to relational databases on Linux. Only ASCII text files are supported at this time.

If data resides on relational DBMS, you need to extract this data and put it into ASCII text files. In our test scenarios we did not investigate data extraction issues. We used sample data provided by Dimensional Insight in flat files.

H.4 Building models

We used the sample models and sample data provided with the DI-Demos package. We followed these steps:

1. Unpack DI-Demos:

```
[diuser@biserver data]$ pwd
/home/DI_Atlantis/diveline/data
[diuser@biserver data]$ mkdir DI_Demos
[diuser@biserver data]$ cd !$
```

```
cd DI_Demos
[diuser@biserver DI_Demos]$ tar -xf
/home/diuser/unpack-files/DI_Demo-dist.tar
```

Alternatively, if these are unpacked in a different directory, then a link can be made to that directory in the /home/DI_Atlantis/diveline/data/DI_Demos directory.

2. Build a sample model:

   Simple test models can be built using sample data provided in the DI-Demos package.

   The sample DI-Demos package includes data files (source data and dictionary files), integrator scripts, a builder description file, and a shell script to run through these. This will produce a model, dist.mdl, which is a simple sales/distribution data model:

```
[diuser@biserver DI_Benchmark]$ ls
/home/diuser/unpack-files/DI_Benchmark/DI_Benchmark.tar
/home/diuser/unpack-files/DI_Benchmark/DI_Benchmark.tar
[diuser@biserver DI_Benchmark]$ tar -xf
/home/diuser/unpack-files/DI_Benchmark/D
I_Benchmark.tar
[diuser@biserver DI_Benchmark]$ ls
Data  Logs  Models  Programs  Temp
[diuser@biserver DI_Benchmark]$ cd Programs/
[diuser@biserver Programs]$ ls -l
total 28
-rw-rw-r--    1 diuser   diuser        84 Sep 30 10:02 README
-rw-rw-r--    1 diuser   diuser      1671 Sep 30 09:26 customer.int
-rw-rw-r--    1 diuser   diuser       432 Sep 30 09:26 dist.dsc
-rw-rw-r--    1 diuser   diuser      2419 Sep 30 09:26 dist.int
-rwxrwxr-x    1 diuser   diuser       497 Sep 30 09:26 makedist
-rw-rw-r--    1 diuser   diuser       414 Sep 30 09:26 makedist.bat
-rw-rw-r--    1 diuser   diuser      2381 Sep 30 09:26 product.int

[diuser@biserver DI_Benchmark]$ ls -R
.:
DI_Benchmark.tar  Data  Logs  Models  Programs  Temp ./Data:
README       company.dic  product.dic      salesperson.dic  supplier.dic
company.dat  product.dat  salesperson.dat  supplier.dat ./Logs:
README ./Models: README ./Programs:
README  customer.int  dist.dsc  dist.int  makedist  makedist.bat
product.int ./Temp: README
Run with only the first 1000 rows generated for dist.txt
[diuser@biserver Programs]$ ./makedist -first 1000
DI-Atlantis Data Integrator v2.1 (5)
Copyright (C) 1991-2001 by Dimensional Insight, Inc
Processing customer.int...
Starting task customer...
```

```
30000 records written to file ../Temp/customer.txt.
DI-Atlantis Data Integrator v2.1 (5)
Copyright (C) 1991-2001 by Dimensional Insight, Inc
Processing product.int...
Starting task product...
3671 records written to file ../Temp/product.txt.
DI-Atlantis Data Integrator v2.1 (5)
Copyright (C) 1991-2001 by Dimensional Insight, Inc
Processing dist.int...
Starting task Distrib...
1000 records written to file ../Temp/dist.txt.
Building ../Models/dist.mdl from dist.dsc
Tue Oct  2 22:06:44 2001 Starting phase 1...
Tue Oct  2 22:06:44 2001 Starting phase 2...
Input information:
Num input records = 1000
Key information:
Key Company, num values = 3, max clump = (399)"Omega Brands"
Key Ship Month, num values = 12, max clump = (99)"November"
Key Customer Premise, num values = 2, max clump = (792)"off"
Key Customer, num values = 982, max clump = (2)"Allen Teri Pope"
Key Salesperson, num values = 143, max clump = (16)"House Account"
Key Supplier, num values = 132, max clump = (197)"DISCONTINUED (059)"
Key Product Class, num values = 6, max clump = (496)"WINE"
Key Brand, num values = 384, max clump = (25)"DEKUYPER"
Key Product, num values = 856, max clump = (5)"FARNESE TREBBIANO V - 1.5 LT
(01259)"
Tue Oct  2 22:06:44 2001 Starting phase 3...
Tue Oct  2 22:06:44 2001 Starting phase 4...
Detail information:
Dtl row size = 28, bucket size = 1000

Tue Oct  2 22:06:44 2001 Starting phase 5...
Tue Oct  2 22:06:44 2001 Starting phase 6...
Tue Oct  2 22:06:44 2001 Phase 6 -- 100% done
Tue Oct  2 22:06:44 2001 Build done
[diuser@biserver Programs]$
```

3. Link the Models directory back to the DiveLine data tree

```
[diuser@biserver Programs]$ cd /home/DI_Atlantis/diveline/data/
[diuser@biserver data]$ ln -s /home/DI_Benchmark/Models Benchmark
```

4. Using ProDiver, do File->Open, Benchmark/dist.mdl. Then reports can be created, and saved as marker files.

H.5 Creating reports

Using the DI-Diver, open the models that you created by File->Open. When you open the DI-Diver for the first time, the application will prompt for the DataServer IP address. Give IP address of the DI-Integrator data server. Select one of the Dives from the console window.

Now, from the model file, select the interrelated dimensions that you want to be included in the report. Once you have selected your dimensions, save the report through File->Save Marker. Save the report to the DI-DiveLine server as a .MRK file. These reports can be accessed by the clients connected through the LAN through DI-ProDiver.

To create reports that have to be accessed via a browser, a user has to follow the same steps as described for the DI-WebDiver. Such reports can be accessed either through a browser or a DI-ReportDiver.

H.6 Users accessing reports

Users can access the reports either through DI-WebDiver or DI-ReportDiver. For example, we accessed the reports by pointing the browser to this URL :

```
http://biserver/webdiver/index.html
http://biserver/reportdiver/index.html
```

Users can also view the reports through DI-Diver by connecting to the DiveLine data server.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 146.

- *Business Intelligence Architecture on S/390 - Presentation Guide*, SG24-5641
- *e-Business Intelligence: Leveraging DB2 for Linux on S/390,* SG24-5687

Other resources

These publications are also relevant as further information sources:

- *z/OS V1R2.0 Communications Server IP Configuration Guide,* SC31-8775
- *z/OS V1R2.0 Communications Server IP Configuration Reference,* SC31-8776
- *zSeries Hipersockets*, REDP0160

Referenced Web sites

These Web sites are also relevant as further information sources:

- Linux for zSeries home page

  `www-1.ibm.com/servers/eserver/zseries/os/linux/`
- `Linux for zSeries systems`

  `http://www10.software.ibm.com/developerworks/opensource/linux390/documentation-2.4.shtml`
- `VM systems`

  `http://www.vm.ibm.com`
- `z/VM Linux guest performance`

  `http://www.vm.ibm.com/perf/tips/linuxper.html`

- z/OS homepage

  `http://www.ibm.com/servers/eserver/zseries/zos`
- IBM Mart homepage

  `www-1.ibm.com/servers/eserver/zseries/zmart/`
- DB2 UDB homepage

  `www.ibm.com/software/data/db2/udb/index.html`
- DB2 UDB download site

  `www.ibm.com/software/data/db2/udb/downloads.html`
- QMF homepage and download site

  `www.rocketsoftware.com/qmf/index.html`
- IBM Mart Installation Assistance

  `www-1.ibm.com/servers/eserver/zseries/zmart/contact_us.html`
- Download sites for Linux on zSeries Distributors

  `www-1.ibm.com/servers/eserver/zseries/os/linux/dist.html`
- IBM Linux homepage

  `www.ibm.com/linux/`
- Linux Online

  `linux.org/`

How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

ibm.com/redbooks

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Special notices

Index

IBM Redbooks

e-Business Intelligence: Data Mart Solutions with DB2 for Linux on zSeries

(0.2”spine)
0.17”<->0.473”
90<->249 pages

e-Business Intelligence: Data Mart Solutions with DB2 for Linux on zSeries

Data population, data refresh, data mart joins with DB2 for Linux V7 federated databases

Intelligent data mart resource allocation with z/VM V4.2

Data access with Dimensional Insight OLAP tool

This IBM Redbook explains the benefits of Business Intelligence solutions using DB2 for Linux on zSeries. We discuss several studies done at IBM zSeries Teraplex Integration Center to demonstrate how the latest features of the zSeries hardware and z/VM 4.2 operating environment extend the value proposition for consolidating data marts on DB2 for Linux for zSeries.

The book describes fast data transfer using HiperSocket connections, investigate scalability and performance issues in the context of data mart population, data mart refresh, and cross joins, intelligent resource management of zSeries under z/VM, and application deployment. We evaluate DB2 V7 latest utilities and DB2 federated database distributed queries.

The book also includes functional tests of ISV tools such as Dimensional Insight.

SG24-6294-00 ISBN 0738423890