IBM®

A High Performance On Demand Retail Data Warehouse Solution using DB2 for Linux

Authors: High Performance On Demand Solutions (HiPODS) team
Web address: ibm.com/websphere/developer/zones/hvws
Technical contact: Dawn Seymour
Management contact: Haider Rizvi

Date: October 14, 2004

Status: Final 1.0

Abstract: *This paper describes the results of a proof of concept (PoC) IBM completed for a major retailer with a large data warehouse. HiPODs demonstrated its capacity for creating a representative customer environment and proving how IBM's products meet customer requirements. The key requirements for this PoC were the database extract/transform/load (ETL) process, scalability and performance of DB2 queries when multidimensional clusters are implemented on the Linux platform, and database provisioning.*

Executive summary

In early 2004 IBM conducted a multiphased proof of concept (PoC) to demonstrate that a data warehouse for the retail industry could perform well on low cost commodity hardware running Linux™. The PoC consisted of three phases: I. Extract/transform/load (ETL) processing, II. Query processing and data growth, and III. Provisioning additional hardware capacity.

The objective of Phase I was to prove that daily ETL processes perform well with DB2® for Linux. Phase II demonstrated that typical retail warehouse end-user queries would show improved query performance by using IBM's patented multidimensional clusters (MDCs). Phase III implemented the IBM Tivoli® Provisioning Manager (TPM) to automate the process of provisioning additional servers and disk for the data warehouse. Phases II and III had the added benefit of demonstrating the near-linear scalability of DB2 for growth of database volume and system hardware capacity.

We established these success criteria for the PoC:

- Integration into an existing environment with minimal changes
- Process daily warehouse updates and inserts, especially for peak volume days
- Execute business queries against the data warehouse that take advantage of multidimensional clusters (MDCs)
- Provide near linear scalability
- Demonstrate provisioning capabilities

The major results of the PoC include:

- A daily merge rate of 22,471 rows per second was achieved.
- The extensive exploitation of MDCs resulted in better than linear data and processor scalability.
- The data redistribution when provisioning new nodes achieved 12-14 GB per hour for each new partition.

This paper describes the steps taken and results obtained for each phase of the PoC. We are very pleased with the improvements that this PoC yielded, as we were able to demonstrate that IBM's DB2 Data Warehouse solution is able to meet the needs of the most demanding businesses.

Note: Before using this information, read the information in "Notices" on the last page.

Contents

Figure list

Introduction

The Linux Retail Data Warehouse proof of concept (PoC) was conducted to demonstrate IBM's ability to host a large-scale data warehouse on clustered Linux servers. The intent of this paper is to describe how we set up the data warehouse and to share the excellent results.

IBM assembled the retail warehouse cluster at its High Performance On Demand Solutions (HiPODS) test facility at the IBM Silicon Valley Lab (SVL) in San Jose, California. Working with teams from across IBM -- DB2 Toronto Labs, Linux Integration Centers, the Business Integration Center of Competency and others -- IBM generated a considerable database based upon our experiences with large retail business customers.

During the course of the PoC, IBM was able to validate that the combination of the latest Linux operating system and database levels would provide a more stable environment with better database performance than previously experienced.

The first section of this document, O*verview*, introduces the PoC hardware and software configuration and describes in general how the data warehouse was set up. A high-level overview of the schema is presented as well as a description of how we obtained the data to populate the database.

The second section, *PoC tests and results*, describes the tests we conducted and the results obtained. The results show the ETL performance achieved during the course of the PoC and that we were able to obtain better than linear scaling for data growth and near linear scaling for processor growth. The results of multiuser scaling tests are also presented.

The last main section, *Hints and tips*, discusses lessons learned that we hope prove useful to others working in the large-scale Linux data warehouse environment.

The appendixes provide details about the project plan, the hardware and software configuration, data characteristics, the database generation tool, and the queries.

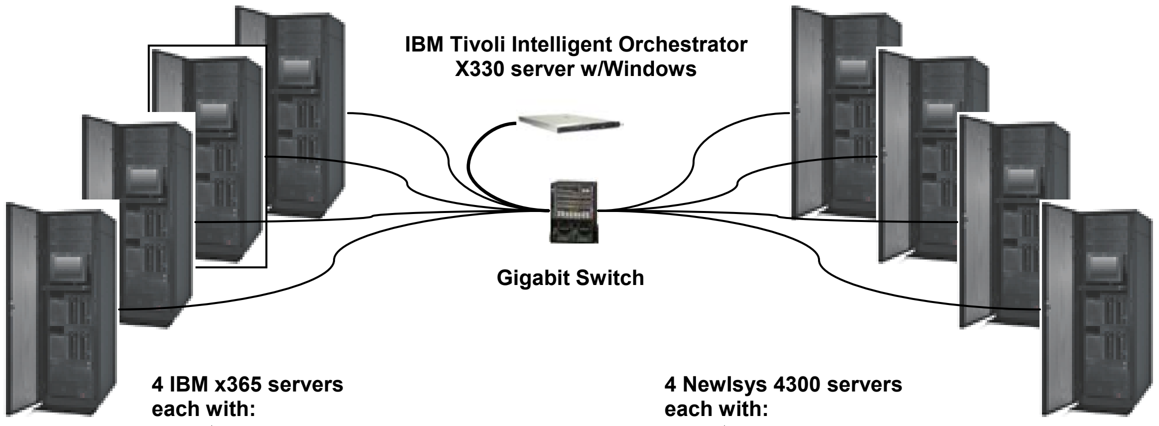
Overview

This section introduces the hardware and software configuration used for the PoC. It also describes the database configuration and how the data was generated.

PoC cluster configurations

We configured two separate Linux clusters (see Figure 1), one using IA32 processor technology and the 32-bit Red Hat Enterprise Linux (RHEL) AS v 3.0 operating system and the other using the AMD processor technology and the 64-bit RHEL v 3.0 operating system. Each server had the same amount of memory and disk resources.

Appendix A contains details about the server hardware, disk subsystem, and software.



4 IBM x365 servers
each with:
- **4 x Intel Xeon MP 2.80GHz**
- **16 GB PC2100 DDR Memory**
- **Integrated dual 10/100/1000 Ethernet**
- **5 x ServerRAID-6M u320 scsi controller 256MB cache**
- **8 x IBM EXP400 External**

4 Newlsys 4300 servers
each with:
- **4 x AMD Opteron 848 2.2GHz**
- **16 GB PC2700 DDR Memory**
- **Integrated dual 10/100/1000 Ethernet**
- **5 x ServerRAID-6M u320 scsi controller 256MB cache**
- **8 x IBM EXP400 External SCSI**

Figure 1. The retail data warehouse cluster

Database setup

We defined a single database instance for each cluster and one logical database partition per CPU. This resulted in 16 database partitions for each cluster. Each logical database partition had the same amount of disk and memory.

Large fact tables were spread across all 16 database partitions. Smaller tables were set up in a single partition.

An additional partition on the first server in the cluster was used for the catalog partition. The first data partition also performed the coordinator function for end-user queries.
Additional details on database setup are in *Hints and tips* and Appendixes C and D.

Data generation

In order to perform the PoC, we needed data to populate the tables. Based on previous experience, we understood that the characteristics of the data would be very important to the performance of the data warehouse as well as to our ability to produce meaningful query results.

To ensure that our PoC would represent a real-world environment, we started with DDL based on the data warehouse of one known retail customer. We altered the table definitions and naming conventions to protect the customer's intellectual capital. However, we kept characteristics of the data that affect space utilization, such as average column and row widths, as well as percentage of NULL values. In addition, we defined data characteristics that affected our ability to join tables during queries, for example, creating line items with valid product codes.

We then worked with a third party firm to create a program that generated data with these characteristics. The resulting tool (dbgg[1]) was used to generate the data required to run our tests. It was used to generate the data required to bulk-up the data warehouse and to generate the data required for daily ETL processing.

dbgg generates data for two main fact tables -- orders and order_items -- as well as several dimension tables. These tables are described in further detail in Appendix C.

ETL process

A critical function in any data warehouse is the ability to populate the warehouse and keep it up to date. The source of the latest data is often obtained from the production OLTP systems. In this retail data warehouse, the initial population (bulk-up) and daily updates were performed using SQL to transform and load the warehouse data.

The ETL (extract/transform/load) process used for the PoC consisted of four basic steps:
1. Generate input data
2. Load the input data from flat files into the load tables
3. Using Merge for daily processing (Insert for bulk-up) populate the Staging warehouse tables
4. Using a denormalizing sub-select and Merge (or Insert for bulk-up), populate the data warehouse tables

[1] dbgg is a tool offered with services from Gradient Systems. For more information, see Appendix D.

Logical Data Warehouse Load Process

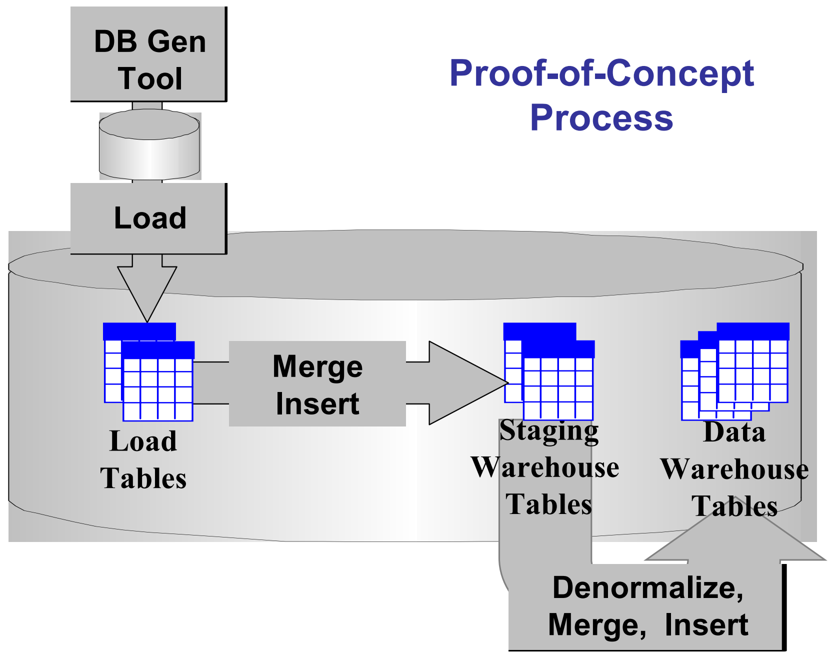


Figure 2. Logical data warehouse ETL process

Figure 2 shows the steps and the multiple sets of tables. Each set represents a level of denormalization, and each set would be appropriate for a different user community. For example, the staging warehouse tables could be used for auditing purposes, while business analysts would more likely use the final data warehouse tables that are further denormalized (order rows now containing summary of line item data, and line items now containing some order data).

PoC tests and results

This section describes the tests and the results achieved for each phase.

Phase I. ETL

The purpose of the ETL phase was to prove that daily ETL processing would perform well with DB2 for Linux in this clustered environment.

Populate the data warehouse

The steps to populate (bulk-up) the data warehouse were completed on both the IA32 and AMD clusters (four servers in each). Because the database is initially empty, the ETL process uses INSERT statements, not MERGE statements.

Figure 3 shows the main tables that required data generation and the number of rows that were generated at ¼ the full data size, and the number of rows generated for full scale.

Table	¼ scale rows	Full scale rows
CUSTOMERS	10,000,000	40,000,000
LOAD_ORDER_ITEMS	181,500,000	726,000,000
LOAD_ORDERS	90,000,000	360,000,000
PRODUCTS	40,000	40,000
SELLERS	25,000	100,000
STORE_GROUP_SELLERS	25,000	100,000
STORE_GROUPS	1,000	1,000

Figure 3. Table sizes in number of rows

Figure 4 shows the comparative times of these activities for each of the two main fact tables. Time shown represents the full scale data (approximately 360 million rows for ORDERS and 726 million rows for ORDER_ITEMS) using DB2 V8.2.

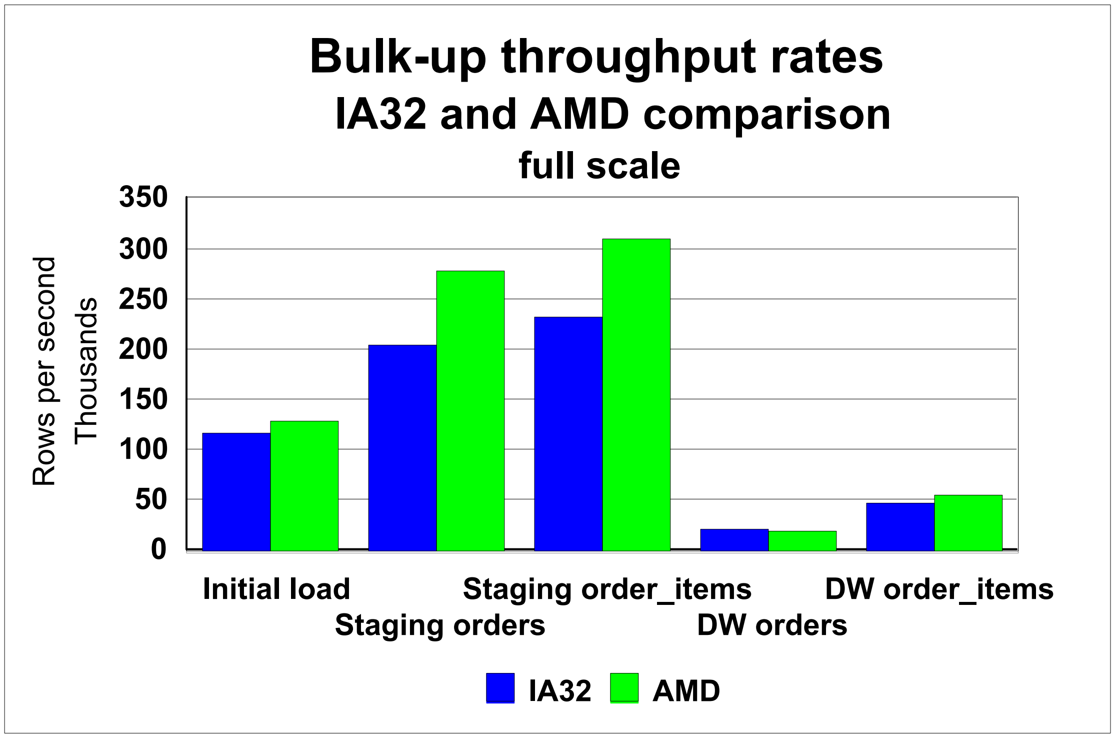


Figure 4. Bulk-up throughput rates

The initial load testing was done using the DB2 LOAD utility, loading data from a flat file into a partitioned table. The numbers reflected here include the loading of all tables required for the test.

The staging tests consisted of an INSERT SQL from the initial LOAD table into the STAGING warehouse table. These tables were partitioned on the same column (so collocation was achieved). The staging tables also used DB2's MDC, clustering the data on region_id.

The DW tests consisted of an INSERT SQL that denormalized the data from the staging tables into the DW warehouse tables. Again, the tables are partitioned on the same column. However, the DW tables are organized using MDC columns of market_group and order_day.

Daily ETL performance

On a daily basis, the main fact tables need to be updated to reflect the most recent activity from the production systems. The dbgg tool can be used to generate data for the daily ETL processing by specifying the number of rows to generate and the percentage of rows that would reflect updates to the existing data.

When these daily files were generated, the ETL process was followed to update the warehouse tables using a MERGE statement. This enables the processing of both INSERT and UPDATE activity within a single SQL statement. In order to accommodate the possibility of multiple updates for a single order, the MERGE statement also uses the ROW_NUMBER function in a sub-select to produce unique input rows for the UPDATE and INSERT. In our case, this function is used to select only the most recent rows for processing.

The elapsed time of the MERGE statement for the ORDERS and ORDER_ITEMS tables from a temporary LOAD table into the staging warehouse was the primary measure of ETL performance for this PoC.

Each region was run as a separate MERGE job. The amount of data to be inserted was roughly .13% of the number of rows already loaded for the region. For example, 535 million rows already exist for region 1 and an additional 674 thousand rows are merged in. As you can see in Figure 5, the region with the largest number of rows (region 1) is obtaining the best merge rate (shown in rows merged per second). All of the daily ETL testing was done against the full database size on the IA32 16 processor cluster.

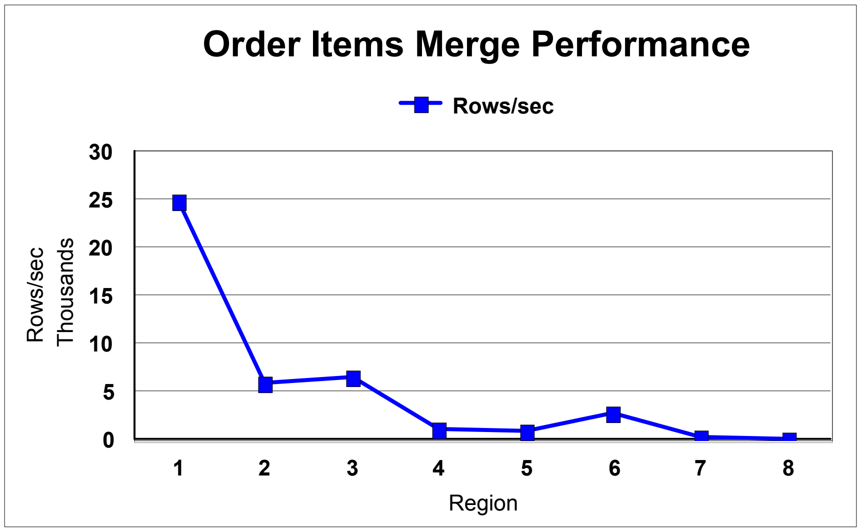


Figure 5. Order items merge performance

The results are well within the performance requirements set out by our retail customers.

Phase II. Query testing

The purpose of Phase II was to show how, through the use of MDC tables, DB2 can execute queries against a large data warehouse better than our competition.

We generated data for one quarter the size of the original production tables so that we could run queries with less data and thus show how well DB2 scales when data volumes grow, and the servers' resources remain the same. With this type of scaling, one would expect that queries would take approximately one quarter of the time that they would at full scale data.

After obtaining from our retail customer the general business questions that the data warehouse is used to answer, we wrote seven queries based on the questions. We also obtained three additional queries from the customer's production environment. We modified these queries to match our altered DDL. Appendix E contains the details regarding the queries and their characteristics.

Bulk-up of the data was done for all tables on both clusters (32-bit and 64-bit) at 1/4 size. Queries were run and measured. Then bulk-up to full scale data was completed and queries were run again on both systems. The warehouse fact tables that were being queried were organized by market_group and order_day. These dimensions are extremely relevant to the types of queries that our retail customer executes against their production data.

Analysis of the query workload showed that DB2 was utilizing the benefits of multidimensional clustering where appropriate. This meant that for some of the queries, a substantial amount of data could be ignored, saving time and system resources. See Figure 20 in Appendix E for a synopsis of which access methods were used and approximation of data retrieved.

The elapsed time of these queries, run sequentially, is the basis for comparison between all of our scaling points. Additional query tests were executed by running variants of the original queries as part of a multistream test.

In short, the query testing showed:

- Appropriate use of MDC block indexes
- Appropriate use of data partition elimination
- Better than linear scaling for data growth
- Near or better than linear scaling for processor growth

Query test results

Queries were run at each scale level before and after the following changes to the database. Results reported reflect the best results obtained at each level.

- Add indexes on (ORDER_NUMBER, MARKET_GROUP, ORDER_DAY) on dw_orders and dw_order_items tables so that DB2 optimizer can recognize the correlation on these three columns between the two tables.
- Add an index on (CUSTOMER_NUMBER) on dw_orders table. Q4 time improves 72% (for example, from 912 seconds to 168 seconds).
- Partitioning customers table. Q4 time improves from 168 sec to 9 sec; Q11 improves from 85 seconds to 2 seconds.
- Transforming year() function to constants. For example, year(order_day) >=1998 is transformed to order_day >= date('1998-01-01'). Q1, Q2, and Q8 improve. For example, Q1 improves from 320 seconds to 110 seconds.

Data scaling tests

Figure 6 shows the expected elapsed time for each query at linear scaling from 1/4 (turquoise) to full data (blue) on the IA32 cluster. The full scale line indicates that elapsed times were actually better than linear scaling (green).

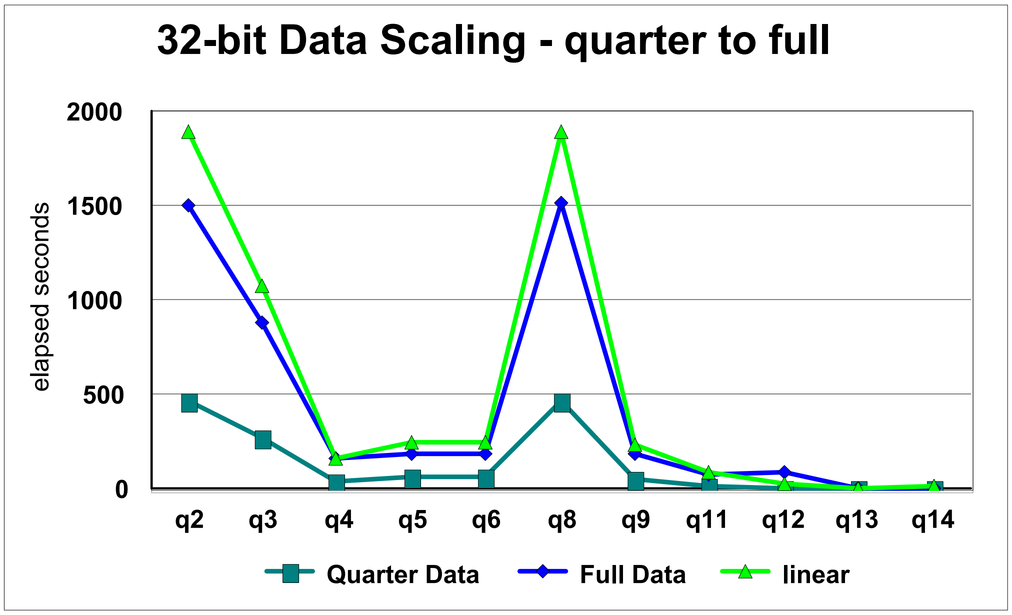


Figure 6. 32-bit scaling -- quarter to full

Figure 7 shows the expected elapsed time for each query at linear scaling from 1/4 (turquoise) to full data (blue) on the AMD cluster. The full scale line indicates that elapsed times were actually better than linear scaling (green). Better than linear scaling was achieved. By using MDCs, DB2 has to examine far fewer rows than would be necessary if the database had to do table scans.

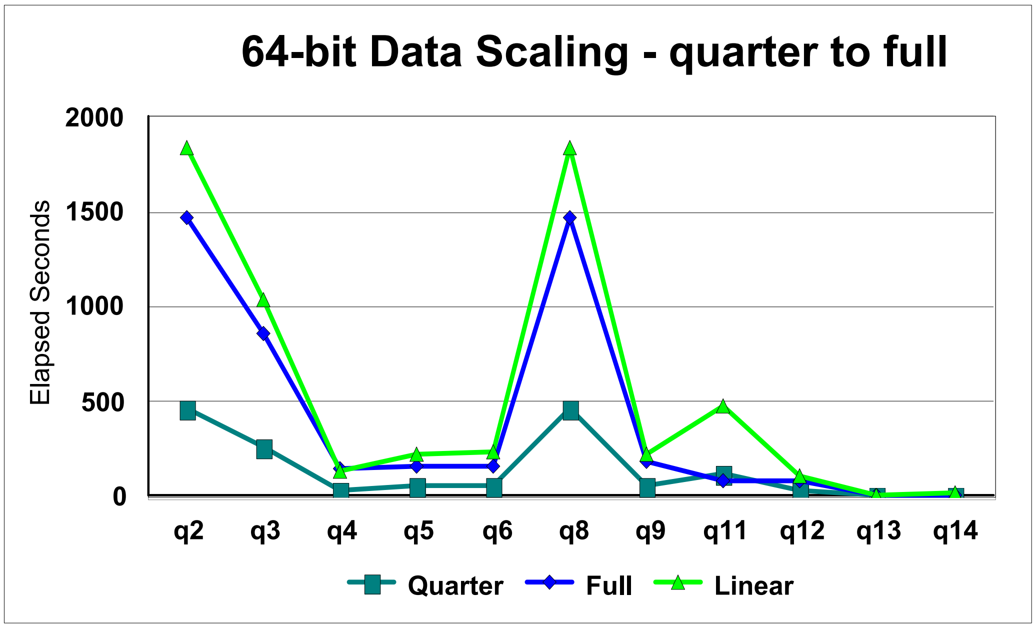


Figure 7. 64-bit scaling -- quarter to full

Figure 8 compares the elapsed time for each query at full scale for both the IA32 cluster and the

AMD cluster. Note that the AMD 64-bit cluster was not tuned to take advantage of the additional memory that could be made available to the database. With additional tuning we would expect that these results could improve further.

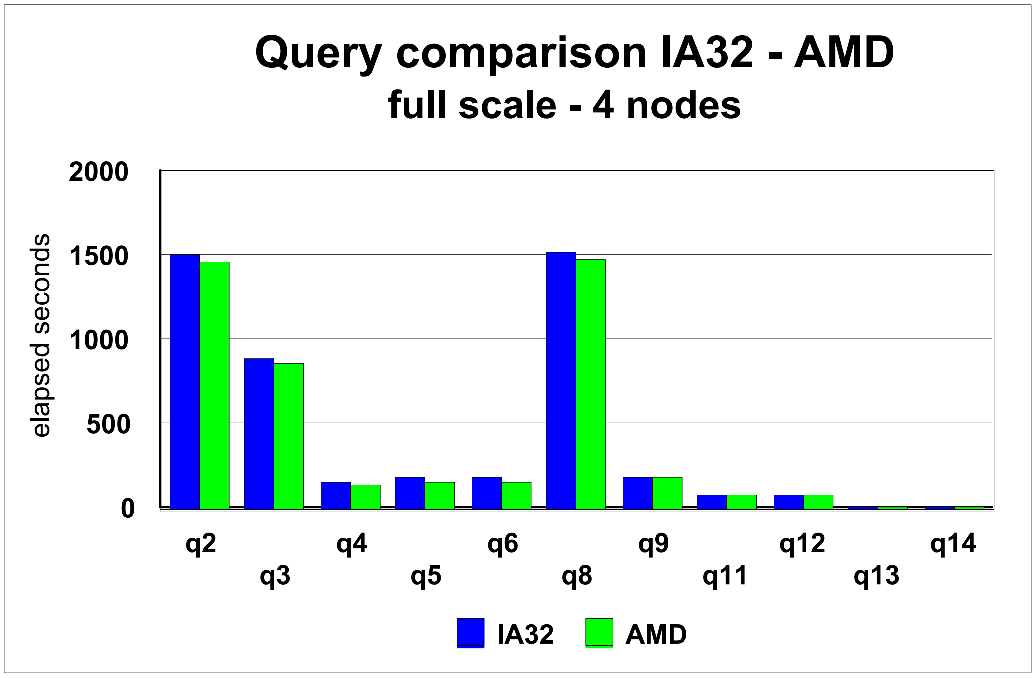


Figure 8. Query comparison IA32 to AMD

Phase III. Provisioning additional server and storage capacity

In a fast growing business a data warehouse may quickly require additional nodes to support growing data volume and queries. Processor and disk resources may need to be added quickly to meet the business growth and to be able to provision the needed resources without affecting the business operations. There are a number of solutions to this problem.

DB2 UDB with the data partitioning feature (DPF) allows processing nodes and disk to be added incrementally. The almost linear scalability available through the partitioned environment's architecture allows us to add processing power and disk in manageable quantities as needed, as opposed to having to "split the warehouse" and double the disk requirements when alternative platforms reach their limits of scalability.

Phase III had three parts:

1. Provisioning the server hardware resources into the cluster
2. Making the DB2 instance and databases aware of the new resources
3. Redistributing the data in the warehouse across the new resources so that the parallelism inherent in the DB2 with data partitioning feature can be realized

Provisioning the server and storage

For this PoC, we chose to automate the process of going from four to six and then to eight nodes for the query scalability phase. We did this by leveraging the work the HiPODS team has done with IBM's Tivoli Provisioning Manager, as well as the DB2 installation workflows available from Tivoli's Orchestration and Provisioning Automation Library (OPAL). (See the HiPODS white paper Provisioning Best Practices for On Demand Data Centers and link to information on IBM Tivoli Provisioning Manager in References)

Recently IBM announced the availability of scripts to provision the DB2 UDB database server code for AIX. For our PoC, we modified the scripts to work with Linux. It was fairly easy to make the command and other changes so that the DB2 install scripts would install DB2 on the new nodes to be added to the DB2 cluster as part of the automated provisioning workflow.

Alterations to DB2 to use new resources

After the server resources are provisioned, the next step is to ensure that DB2 is aware of them and can take advantage of them for data storage. This can also be automated after some initial set-up work is complete.

For adding the fifth and sixth nodes to the cluster, we completed the alterations to DB2 manually. The first step required was to add the new data partitions. In our case, because the naming conventions of the disk were identical for each system, we were able to add data partitions using the db2start command:

```
db2start DBPARTITIONNUM 17 ADD DBPARTITIONNUM HOSTNAME testsys05 PORT 1
LIKE DBPARTITIONNUM 2
```

When the database partition has been added, these additional steps were taken for each new partition:

- Update the log directory
- Update the database configuration settings
- Alter the multinode partition groups to include the new partition(s)
- Alter the table spaces to include containers on the new partitions

When adding the seventh and eighth nodes to the cluster, we defined workflows (TPM logical operations) to automate these steps.

Redistributing the data

After the resources are defined to DB2, one final step must be taken to spread the data evenly across the new partitions. This is called a redistribute. For the redistribute from 16 to 32 processors, the work was done by entering the DB2 commands from a shell script. These steps could be automated by defining additional workflows in Tivoli.

The DB2 redistribution is done as one unit-of-work for every table that is in a database partition group. The work is logged and the expected elapsed time is generally gated by the insert rate on the new partitions. Thus it is recommended when using redistribute to schedule down time and complete the following general tasks:

- Drop indexes

- Redistribute
- Reorganize the tables (to reclaim unused space in the old partitions)
- Recreate the indexes
- Run statistics

The provisioning phase included the redistribution of data from the smaller number of data partitions to the larger number of data partitions; that is, from 16 to 24 and then from 24 to 32 data partitions.

Figure 9 shows the method that we used for estimating how long redistribution would take.

	From	**To**	**Factor**		
Servers	4	6	0.67	188.6	**Data to move**
DPs	16	24	8.00	1.6	**Estimated hours**
	From	**To**	**Factor**		
Servers	6	8	0.75	141.5	**Data to move**
DPs	24	32	8.00	1.2	**Estimated hours**

Figure 9. Redistribution estimates (spreadsheet snippet)

In Figure 9, the first factor indicates what percentage of data will remain in the old data partitions. The inverse (1-factor) is then used to calculate the amount of 'Data to move'. This number is represented in GB and based on the total GB of storage taken by the table and documented in the database catalog. The second factor indicates the number of new data partitions. The estimated number of hours is calculated in the following manner:

(Data to move) / (number of new data partitions * expected rate)

Based on historic evidence on other systems we used an expected rate of 15 GB per new partition per hour.

The redistribution time from 16 to 24 processors was 40 minutes 26 seconds for the DW_ORDERS table. We did not obtain complete timing results at the first scale factor, as we did not have enough log space on the system to redistribute the order_items table, and did not have time to allocate additional file system space to logs.

However, at the second scale factor, we completed the redistribution in 1 hour 14 minutes and 24 seconds. Redistribution of the data from 24 to 32 processors was done in 25 minutes 48 seconds for the DW_ORDERS table. The DW_ORDER_ITEMS table took 48 minutes and 39 seconds. This equates to slightly less than the 15GB rate with 14/GB per hour per new partition.

Processor scalability results

Processor scaling was accomplished by adding both CPU and disk resources to the cluster, keeping the ratio of disk to CPU the same throughout. In this case, we can consider that one system consisted of one 4-CPU server with 60 RAID1E LUNs for DB2 data.

Scale factor	Servers	Processors	RAID1E LUNs	DB2 data partitions
16	4	16	240	16
24	6	24	360	24
32	8	32	480	32

Figure 10. Processor scale factors

Due to hardware availability restrictions, we were unable to have all eight servers of the same hardware type. The first four servers consisted of IA32 processors. The second four servers were AMD64s. When configuring these to work together in a single cluster, the AMD64s were booted with the 32-bit kernel.

The queries demonstrated better than linear scaling both when comparing 16 to 24 as well as 16 to 32 processor configurations. In Figure 11, the linear line (green) is projected based on the actual elapsed time of queries at scale=16. The blue line shows the achieved performance of the queries relative to expectations at the 24 processor level.

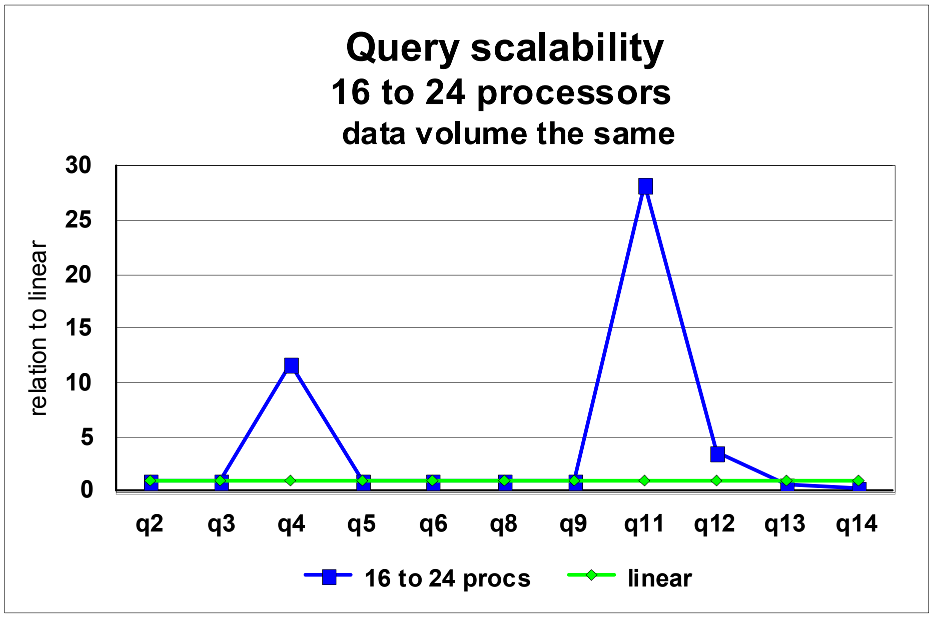


Figure 11. Processor scaling 16 to 24

In Figure 12, the linear line (green) is projected based on the actual elapsed time of queries at scale=16. The blue line shows the achieved performance of the queries relative to expectations at the 32 processor level.

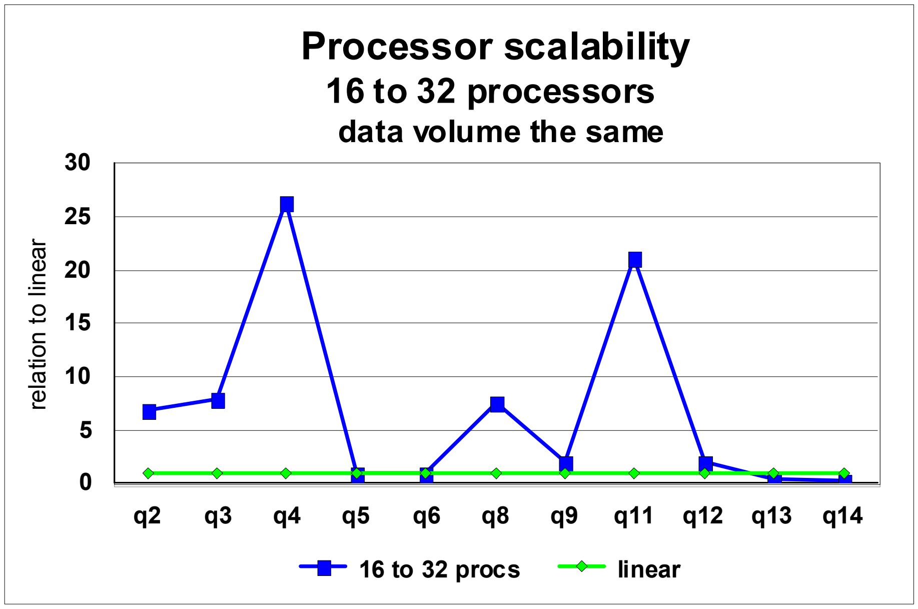


Figure 12. Processor scaling 16 to 32

Figure 13 plots the elapsed time of the queries at all three scale factors—16 (turquoise), 24 (pink) and 32 (blue). Because elapsed time is being charted, the lower the line the better the performance. As expected, when resources are added, and data redistributed, the performance improves.

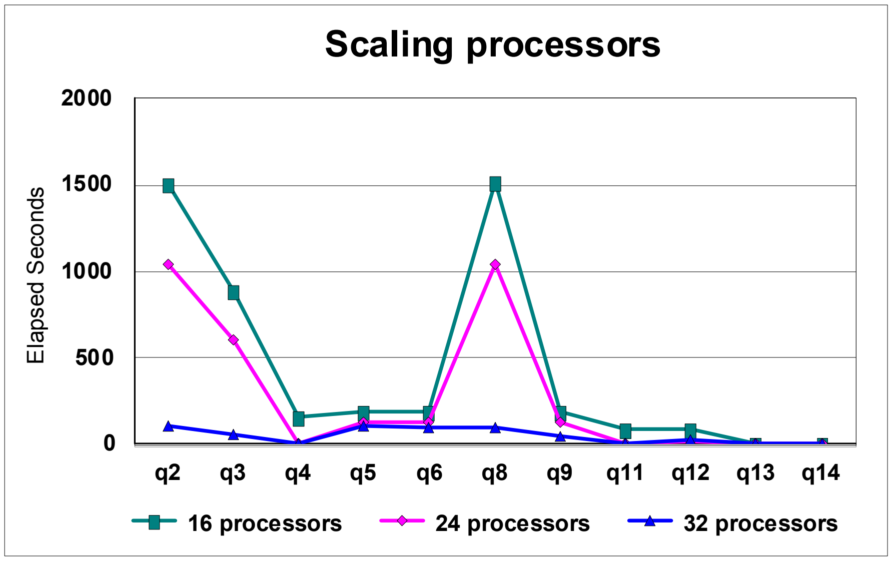


Figure 13. Processor scaling 16/24/32

Multi-stream query test results

The multistream test was set up with 5, 10, and 15 concurrent streams. A pool of 12 queries was created based on four of our representative queries: five versions of Q11, five versions of Q14, one version of Q12, and one version of Q13. The alternate versions of each query were created by modifying the predicates (where clauses). One stream then, consisted of running the entire pool at random. Query predicates were kept unique for all streams.

Number of concurrent streams	Queries per minute
1	9.6
5	31.6
10	38.7
15	40.3

Figure 14. Concurrent stream queries per minute

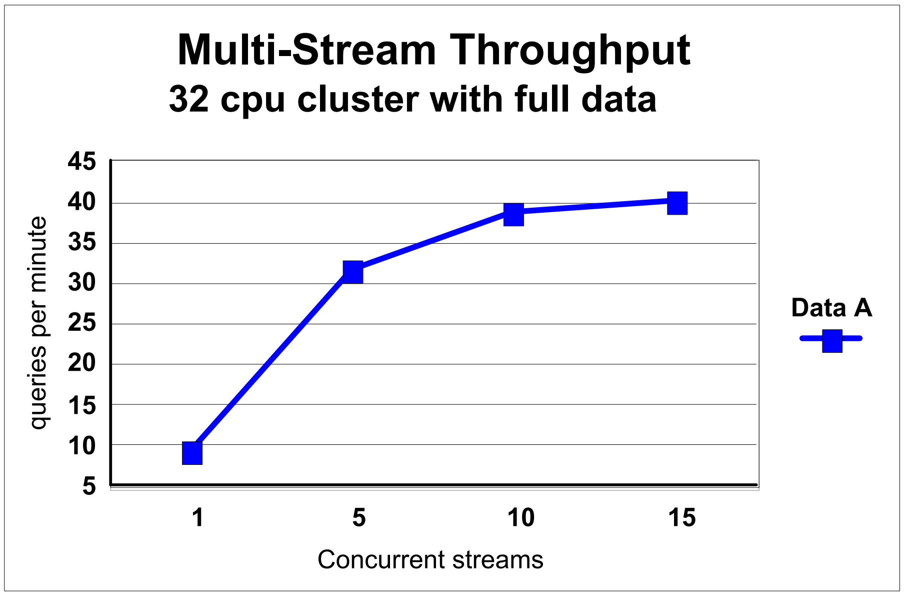


Figure 15. Multistream throughput

Figure 15 shows the results of the multistream query tests and how the throughput levels off when ‘extra’ system resources are consumed.

Hints and tips

This section is intended to provide additional information that could be useful to others looking into setting up a similar environment.

Conversion notes

IBM provides a migration toolkit that can be used to assist in migrating DDL and SQL to DB2. Information about the migration toolkits IBM has available can be found at: www.ibm.com/software/data/db2/migration/mtk/

We used the toolkit to do minor SQL conversions. However, the toolkit does not migrate the (+) outer join syntax correctly when used with complex SQL, as was our case. As a result, all MERGE statements and later queries had to be manually converted. Our approach was to use the toolkit to do the initial translation, and then to manually match that to the original and modify as needed. While this does take some time, the majority of the minor transforms were done for us when we used this approach.

Other issues included:

NUMBER data type without a qualifying length attribute. These would be translated to a NUMERIC data type in DB2, which may have used more space than necessary. Careful examination of the data characteristics was needed to avoid wasted space. This included understanding minimum and maximum values of the source data as well as what transforms may have been

TO_CHAR function used extensively. The migration toolkit provides an equivalent User Defined Function (ORA8.TO_CHAR) that matches more directly the existing syntax. However, this did not cover all of the cases identified. Additional work would be required to match all cases. DB2 provides all of the functionality, but simply does not use the same syntax to accomplish it.

Space usage and data placement considerations

Table space page and extent size

The *page size* to be used was determined primarily with regard to space usage. DB2 currently limits the number of rows of data that can be placed on a single page to 255 rows. When the row widths are small, it is beneficial to use a smaller page size so that space is not wasted. However, in our case, most of the tables had wide average row widths so we used the largest page size, 32K.

The larger page size also allows for more data to be read at one time to satisfy sequential I/O requests. This can be beneficial to data warehouse type queries, which often need to perform large sequential I/Os.

The *extent size* is the number of pages that DB2 will use when allocating new space for a table during write activities. This number is set when defining the table space. We used an extent size of 8. This equates to 256K for the full extent size. This was based on previous experience with other data warehousing applications using regular DB2 tables.

With multidimensional clustering (MDC), the size of the extent can have a significant impact on data maintenance. It is strongly recommended that you consider the expected number of extents in your MDC cells when deciding the appropriate value for this. We could have benefited from a larger extent size.

Data partitioning

DB2 data partitioning feature was used to partition the larger tables into multiple pieces. A single partition per CPU was used. In a data warehousing environment, this allows for the typical large long-running queries to utilize more of the system resources. If a greater number of concurrent users were anticipated, it could be beneficial to use a smaller number of partitions—perhaps one data partition per every two CPUs in the cluster.

The columns to use for partitioning were determined with two criteria in mind. First, and most important, is that the column has a fairly high cardinality so that the amount of data in each partition is evenly distributed. Second, and also important, is that the value is commonly used to join tables together. This allows for the possibility of collocation of data during queries that results in less data to pass between data partitions.

In this case, because we were working with data that was already partitioned, we used the same column that was defined for the hash partitioning.

Multidimensional clustering

The original MDC column we used resulted in a large data skew between MDC cells. While this has some advantages for cell elimination when querying the smaller cells, it presents a problem for data maintenance for the larger cells. Although dramatic performance improvements to our data maintenance were made during the course of this PoC, additional performance improvements could possibly have been realized by selecting a larger extent size for these tables.

In this case, because we were working with data that had been partitioned (both hash and range), we used the same columns that were defined for the range partitioning as our starting point for the MDC column values. For the staging tables, this was the region_id.

We determined that alternate columns would be more appropriate for the MDC value in the data warehouse tables. This was done based upon analysis of the queries that we received. For these tables we used the market_id and the order_date as the two MDC columns.

Linux file system issues

File system vs. raw devices for data

During the course of this PoC, both DB2 and Linux came out with new versions/releases that we were able to take advantage of. We initially started out with database tables stored on Red Hat file systems. However, because we needed additional I/O performance, we reloaded our database using raw devices at the same time as we upgraded to Red Hat Enterprise Linux v 3.0. This allowed us to realize additional performance gains.

A major issue for good performance with decision support workloads is the ability to issue large block I/Os. While the Linux 2.4.x enterprise kernels support this, many of the I/O subsystem device drivers do not have these features enabled by default.

We strongly recommend that customers check that both raw I/O variable-size optimization and

Highmem I/O are enabled for their particular configuration.

While other operating systems over time are showing better file system performance — approaching that of raw devices — our experience at this writing is that there is still a significant delta on the Linux platform, with raw devices significantly outperforming file systems.

SQL considerations

Use of YEAR() function

Using a constant in a date format for comparison to a DATE field produces better results than using the YEAR() function against the date and comparing to an integer. For example, year(order_day) >= 1998 will perform worse than order_day >= date('1998-01-01').

Bulk-up using INSERT

The SQL provided by the customer related to their daily processing and required the use of the MERGE statement to avoid duplicates in the target table. However, when doing the initial bulk-up of the tables, the target tables are empty. Converting the MERGE statement to an INSERT statement for this situation produces a simpler plan and performance is improved.

Appendix A. Project plan

Figure 16 is a template of the project tasks and durations that might be useful for other projects.

Task	Duration	Start	Finish
Power	2d	Tue 1/27/04	Wed 1/28/04
VLAN Defined	2d	Tue 1/27/04	Wed 1/28/04
Switch Installed	11d	Fri 1/30/04	Fri 2/13/04
Switch arrives	0d	Fri 1/30/04	
Install switch	7d	Thu 2/5/04	Fri 2/13/04
Install wiring	2d	Thu 2/5/04	Fri 2/6/04
Build out environment	74d	Tue 1/27/04	Fri 5/7/04
8 Racks in place	1d	Tue 1/27/04	Tue 1/27/04
Install PDUs	2d	Wed 1/28/04	Thu 1/29/04
Disk Bays arrive	0d	Thu 1/29/04	
Install Disk Bays	4d	Fri 1/30/04	Wed 2/4/04
Disk Drives Arrive -	0d	Wed 2/4/04	
Install Disk Drives	4d	Thu 2/5/04	Tue 2/10/04
X365s arrive	0d	Thu 2/12/04	
Install memory in 2 systems	2d	Thu 2/12/04	Fri 2/13/04
Processors and Memory card arrives	0d	Fri 2/27/04	
Install additional processors and memory	1d	Fri 2/27/04	Fri 2/27/04
Install x365's (4)	1d	Mon 3/1/04	Mon 3/1/04
Install software	2d	Tue 3/2/04	Wed 3/3/04
Configure RAID	1d	Wed 3/3/04	Wed 3/3/04
Newisis arrive	0d	Fri 5/7/04	
Install Newisys & OS	6d	Mon 4/12/04	Mon 4/19/04
Get contractor site/VLAN access	2d	Tue 1/27/04	Wed 1/28/04
Produce representative retail DDL	4d	Tue 1/27/04	Fri 1/30/04
Contractor start	2d	Tue 2/3/04	Wed 2/4/04
Data Gen tool prep	2d	Thu 2/5/04	Fri 2/6/04
Generate Data on test	3d	Mon 2/9/04	Wed 2/11/04
Generate Data on x365's	10d	Thu 3/4/04	Wed 3/17/04
Perform Data Load - Merge/Insert	8d	Thu 3/18/04	Mon 3/29/04
Present results: Data Load	0d	Mon 3/29/04	
Generate new tables for queries	44d	Mon 3/29/04	Thu 5/27/04
Run 32 bit query scalability tests	3d	Fri 5/28/04	Tue 6/1/04
Run query/load 64 bit comparison test	2d	Wed 6/2/04	Thu 6/3/04
Provisioning PoC	60d	Mon 4/12/04	Fri 7/2/04
Set up TPM environment	8d	Mon 4/12/04	Wed 4/21/04
Test Linux provisioning on x360	5d	Thu 4/22/04	Wed 4/28/04
DB2 provisioning flows ready	0d	Mon 5/3/04	
Test DB2 provisioning on x360	5d	Mon 5/3/04	Fri 5/7/04
DB2 add node and redistribute data flows available	0d	Fri 5/7/04	
Provision four Newisys nodes from bare metal	3d	Mon 5/10/04	Wed 5/12/04
DB2 Add nodes and redistribute data	10d	Mon 6/21/04	Fri 7/2/04
Run 8 node Query scalability test	1d	Mon 7/5/04	Mon 7/5/04
Document set-up and results	2d	Tue 7/6/04	Wed 7/7/04
Write White paper	5d	Mon 9/6/04	Fri 10/08/04
Present Results: Executive meeting	0d	Fri 7/9/04	
Transfer machines to new owners	2d	Fri 7/9/04	Mon 7/12/04

Figure 16. Project plan

Resources were not full time and there were a number of starts and stops due to hardware acquisition and team member conflicts with other projects.

Appendix B. PoC configuration

Server hardware

Figure 17 itemizes the parts required for all of the servers used, including racks, disk controllers, and disks. We chose SCSI controllers to demonstrate the viability of low-cost commodity disk storage in the DB2 partitioned environment. DB2 can use both SCSI and fiber-attached storage.

Qty.	Part number	Description
4	4300	Newisys 4300 Chasis
16	C0	AMD Opteron Processors
4	73P9710	Memory Upgrade Card, 16-DIMM
64	73P2267	1GB PC2700 DDR ECC SDRAM RDIMM
4	8862-3RX	Loaner x365 2.8GHz/400MHz/2MB 2GB Rack (3U)
8	73P7075	2.8GHz/400MHz/2MB Xeon MP Processor
4	73P9710	Memory Upgrade Card, 16-DIMM
64	33L5039	1GB PC2100 DDR ECC SDRAM RDIMM
40	02R0988	ServeRAID 6M Controller (256MB Cache)
32	32P0734	36.4GB 15K rpm Ultra 320 HS HDD
64	1733-1RU	EXP400 External Storage Unit
896	32P0734	36.4GB 15K rpm Ultra 320 HS HDD
8	9308-42S	NeyBAY42 Enterprise Rack, (42U)
16	32P1751	DPI 30a/125v Front-End PDU
32	32P1736	DPI Universal Rack PDU
10	94G6670	Blank Filler Panel Kit
1	32P1031	1U Flat Panel Monitor Console Kit w/KB
1	1735-R16	Remote Console Manager
4	32P1652	Long KVM Conversion Option, 4-Pack

Figure 17. List of hardware parts

Disk subsystem

Every server had the same amount of disk capacity assigned. These disks were configured into the logical unit numbers in the same manner and defined to the operating system using the same naming convention as well. Assigning disk to DB2 table spaces (and thus data partitions) was also done the same for each server. This approach greatly simplifies the initial setup as well as the ability to provision servers and disk in the future.

I/O layout

All disks were 36.4GB 15K rpm Ultra 320 drives. Of the external storage, 42 disks were set aside for flat file space for data generation on each server. In addition, each server contained five disk adapters. Each of the five ServerRAID-6M cards controlled:

- Four 3-disk RAID1E LUNs for DB2 table spaces
- One 2-disk RAID1 LUN for DBPATH and logs

This allowed for a total of 20 three-disk RAID1E LUNs for DB2 table spaces. It is beneficial to evenly distribute the disk resources among data partitions. This includes adapters as well as disk. Therefore, each server also provided storage for four data partitions. Each data partition owned:

- Five 3-disk RAID1E LUNs across the 5 controllers
- One 2-disk RAID1 LUN for DBPATH and logs

Software

Operating system

The final configuration used Red Hat Enterprise Linux AS version 3 update 2. The kernel level was 2.4.21-15.ELsmp

Database

The Daily ETL tests were run using DB2 UDB Version 8.1.5 at full volume. However, the final query and scalability tests were run using DB2 UDB Version 8.2. It is assumed that 8.2 will give similar performance results for the ETL as 8.1.5 did.

Appendix C. Data characteristics

The characteristics of the data were very important to the successful completion of the tests in this PoC. For the first phase (ETL processing), the space usage characteristics were the most important. For the query phase, the relationship and referential integrity of the data within the tables was more important.

For space usage considerations, the variable character columns needed to actually vary in width while maintaining an overall average column length that matched that of the original system. In addition, the proportion of NULL values in NULLABLE fields needed to be maintained.

Using the data generation program described in Appendix D, we made sure that the initial fact tables were populated with data that would mimic the space usage of the data used by our retail customer.

The following sections attempt to describe both types of data characteristics that were implemented. The fact tables discussed are the source tables; these are later modified slightly using the ETL process.

These tables do not completely mimic the customer database design. Names, number of columns, some relationships, and some characteristics were changed to protect the customer's intellectual capital.

Fact tables

ORDERS and ORDER_ITEMS are the two main fact tables. Every order has one orders table row and one or more order_items table row. Orphan orders are not allowed. Orphan order_items are also not allowed. There is an average of two order items per order.

SHIPMENTS and SHIPMENT_ITEMS tables were also defined. However, these were not used after the on-site ETL testing.

The average row width maintained for the ORDERS table was 366 bytes per row.
The average row width maintained for the ORDER_ITEMS table was 785 bytes per row.
The average row width maintained for the SHIPMENTS table was 387 bytes per row.
The average row width maintained for the SHIPMENT_ITEMS table was 769 bytes per row.

Data skew

Data skew by region_id was important to maintain in order to ensure that the ETL performance issues reported is representative of a real customer environment. Figure 18 describes the skew that we were able to generate into our data.

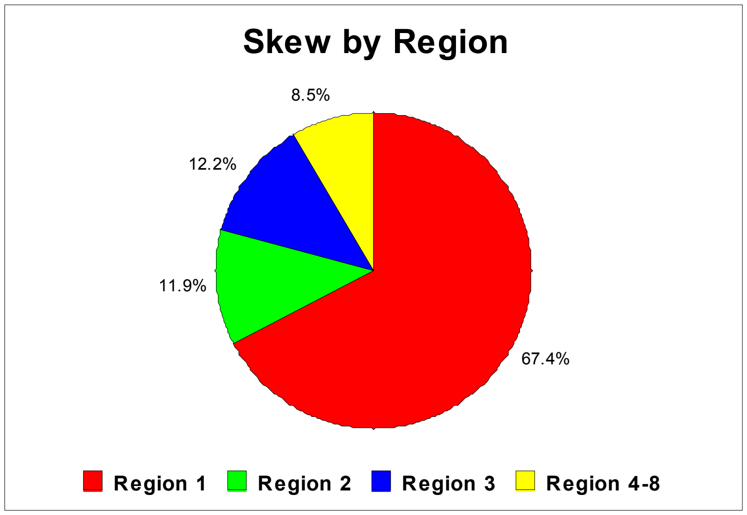


Figure 18. Skew by region

Figure 19 shows the skew by year. In addition to yearly data skew, we incorporated some skew into the months—November and December being the largest volume months for each year.

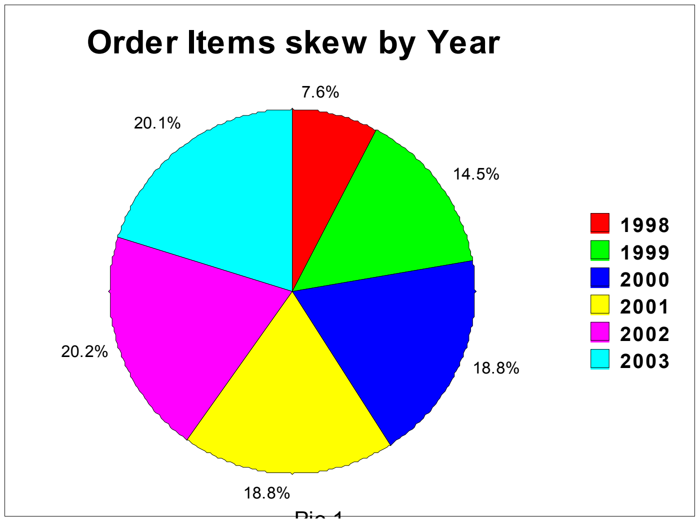


Figure 19. Order items skew by year

Dimension tables

Most of the dimension tables were small enough to be placed into a single partitioned table space. The CUSTOMERS table started out this way. It was determined that partitioning the customer table would allow for better performing queries. We would recommend this in the future.

The PRODUCTS table is the only other dimension table which was partitioned from the beginning.

There are several tables which make up the PRODUCTS information. The products table itself and several tables which define a categorization hierarchy. As noted before, the complete customer data model was not implemented for this PoC.

In addition to the customer and product dimensions, we also defined dimensions for the sellers. Sellers are organized as well into store groups.

When the dimensions were defined, the fact tables were altered so that orders generated included only valid customer numbers, products, and sellers. An attempt to ensure that related denormalized data (e.g. pricing pulled into the order from the products table) was also kept in-synch with the dimension tables. This was not done 100%, as the idea was that over time, some of these values may have changed. And the possible impact on queries was minimal.

Data skew

Customers were generated with similar data skew to the fact tables for region_id and dates. In addition 95% of the customers generated were considered active, while 5% were considered inactive. Also, 80% of the customers generated were considered 'individuals', while the remaining 20% were considered 'institutional'.

Sellers are also divided into three categories with 90% being individual sellers and 9% representing large merchants. 1% represented small merchants.

We used the UNSPSC code to generate half of the products in the table. The other half were randomly generated in a 'Books' category. Books and non-book items are both stored using the same column definitions, however, for books some columns related to non-books will be NULL and vice versa.

UNSPSC for product groupings

In order to generate a set of products, we relied on the UNSPSC coding system. This enabled us to create individual products descriptions within a hierarchy that we could use to populate the category tables. Once the following table was populated, a set of SQL was run to populate the product groups, categories and sub-categories tables. The PRODUCTS table is still generated using the DataGen program, however, it also relies on the UNSPSC coding system. The UNSPSC code is available online.

Appendix D. Data generation program

Characterization and proof-of-concept efforts need a reliable, reproducible, and easily extensible data set on which to base their work. Often, the precise data population that will be required to highlight a new functionality or diagnose an on-going bottleneck is not fully understood until the engineering effort is underway. The successful completion of a project can hinge on the engineering team's ability to respond quickly to a new insight or a discovered problem by adjusting the underlying data set. As this project progressed, it became clear that additional fidelity within the data set and added complexity within the operation data model would help demonstrate the power of the solution that IBM could provide. The data warehouse PoC relied on dbgg, a new tool from Gradient Systems, to provide the sophisticated data set and quick modification time that the project required.

dbgg is an automated tool that produces a portable, parallel data generator from standard ANSI SQL DDL. In addition to the underlying data model information in the DDL (for example, primary and foreign keys, column definitions), dbgg relies on annotations to the DDL that are transparent to DB2. The result is a single DDL file that simultaneously defines both the data model and the data set. The annotations provide control over all aspects of a data set, including:

- Arbitrarily complex, user-defined data distributions
- Business rule logic
- Complex join relationships
- Linear, logarithmic and static table scaling
- Correlated and skewed data within and between tables
- Incorporation of public data sources (for example, census data)

For the data warehouse project, dbgg allowed the DB2 team to focus on engineering, while the underlying schema and data set changed repeatedly to model more of the customer's operational schema, and to include external data sets to model geographic information and product classifications. For more information on dbgg, contact:

Gradient Systems, Inc.
643 Bair Island Road
Suite 103
Redwood City, CA 94063-2755

Appendix E. Queries

Nine representative queries were run against the data warehouse. Seven were created by IBM as examples of SQL that would answer general retail business questions. Four were obtained from the customer as examples of production queries. All queries ran optimally and used MDC block filtering as appropriate.

Query description and SQL

Q2: How many items (or how much revenue) were sold for a particular seller for a given time period? count(*) is number of order_items, whereas sum(quantity*price) is revenue

```
select count(*) as count, sum(quantity*price) as revenue
from dw_order_items
where seller_vendor_id = 87466
and year(order_day) between 1998 and 2003;
```

Q3: How many items (or how much revenue) of a particular product were sold for a given time period?

```
select count(*) as count, sum(oi.quantity*oi.price) as revenue
from dw_order_items oi
where product_id = '20691' and
year(order_day) between 2001 and 2003;
```

Q4: Joins of orders to order_items (two large fact tables) What is the average number of items per order for customers in a given zipcode for purchases in the last year?

```
select avg(count) as avgcount
from
(select o.order_number, count(*) as count
  from dw_orders o, dw_order_items oi, customers c1
 where o.order_number = oi.order_number
  and o.customer_number = c1.customer_number
  and c1.postal_code = '00110-5660'
  and year(o.order_day) > 1997
 group by o.order_number) as x;
```

Q5: Seller (Merchant) ranking metrics. Queries in this category would answer the business questions: Which sellers are most successful in a particular category? this version is for one particular category based on number of items (count(*))

```
select seller_vendor_id
from (select  oi.seller_vendor_id,
     rank() over (order by count(*) desc) as r
     from dw_order_items oi, products p
     where oi.region_id = p.region_id
     and oi.product_id = p.product_id
     and category = 31210000
     group by oi.seller_vendor_id) as x(seller_vendor_id, r)
where r = 1;
```

Q6: Which sellers are most successful in a particular category? this version is for one particular category based on revenue (sum(quantity*price))

```
select seller_vendor_id
from (select  oi.seller_vendor_id,
     rank() over (order by sum(oi.price*oi.quantity) desc) as r
     from dw_order_items oi, products p
     where oi.region_id = p.region_id
      and oi.product_id = p.product_id
     and category = 31210000
     group by oi.seller_vendor_id) as x(seller_vendor_id, r)
where r = 1;
```

Q8: How did a seller's performance in the last quarter compare to the same quarter one year ago? added a predicate to limit years

```
select year(oi.order_day) as year,
    quarter(oi.order_day) as qtr,
      sum(oi.price*oi.quantity) as curr_rev,
      min(sum(oi.price*oi.quantity)) over
      (partition by quarter(oi.order_day)
              order by year(oi.order_day)
                 rows between 1 preceding and 1 preceding)
    as prev_rev
from dw_order_items oi, products p
where oi.region_id = p.region_id
      and oi.product_id = p.product_id
      and oi.seller_vendor_id = 246549
      and year(oi.order_day) > 1997
group by year(oi.order_day), quarter(oi.order_day);
```

Q9: Of the sellers that sold best in category X, what other products did they sell? -- added year constraints to limit results

```
select distinct oi.seller_vendor_id, p1.product_id,
substr(p1.description,1,60)
from products p1, dw_order_items oi,
  (select oi.seller_vendor_id,
      rank() over (order by sum(oi.price*oi.quantity) desc) as r
      from dw_order_items oi, products p2
      where oi.region_id = p2.region_id
        and oi.product_id = p2.product_id
        and category = 41110000
        and year(oi.order_day) = 2001
      group by  oi.seller_vendor_id) as x(seller_vendor_id, r)
where p1.region_id = oi.region_id
  and p1.product_id = oi.product_id
  and x.seller_vendor_id = oi.seller_vendor_id
  and year(oi.order_day) = 2001
and x.r = 1;
```

Q11. How many products and what is the total revenue for a particular market group(s) and date?

```
SELECT cop.MARKET_GROUP, cust.EMAIL AS cust_email_address,
  case when pg_blend.product_group > 0 then
            concat(rtrim(pg_blend.description), ' Only')
            else pg_blend.description end AS order_blend
  ,COUNT(DISTINCT cop.product_id) AS cop_product_count_d
  ,sum( cop.QUANTITY ) AS quantity
  ,sum( cop.OUR_PRICE * cop.QUANTITY ) AS sales_dollars
FROM
  dw_order_items cop, customers cust,
  dw_orders co, product_groups pg_blend
WHERE
co.order_blend*1000000 = pg_blend.product_group
AND cust.customer_number = cop.customer_number
AND co.order_number = cop.order_number
AND cop.ORDER_DAY = date('2003-11-27')
AND cop.MARKET_GROUP IN (4,5)
AND (case when cop.dw_data_source in ('MCO','MCOI')
         then case when cop.restriction = 2
                  then 3
                  when cop.restriction = 3
                  then 4
                  when cop.restriction = 5
                  then 4
                  else 6
               end
         else cop.restriction
      end ) <> 6
AND co.order_day = cop.order_day
AND co.market_group = cop.market_group
```

```
GROUP BY cop.MARKET_GROUP, cust.EMAIL,
        case when pg_blend.product_group > 0 then
            concat(rtrim(pg_blend.description), ' Only')
            else pg_blend.description end;
```

Q12. What products and how many of each (including sales $) has a particular seller sold for a given date?

```
SELECT cop.product_id AS product_id
      ,prod.DESCRIPTION AS prod_desc
      ,sum( cop.QUANTITY ) AS quantity
      ,sum( cop.OUR_PRICE * cop.QUANTITY ) AS sales_dollars
FROM
  dw_order_items cop
 ,dw_orders co
 ,products prod
WHERE
co.order_number = cop.order_number
AND cop.product_id = prod.product_id
and cop.region_id = prod.region_id
AND cop.MARKET_GROUP IN (1,2)
AND cop.SUPPLIER_ID = 17
AND cop.PRODUCT_GROUP IN (38,92,140)
AND cop.ORDER_DAY >= DATE('2003-01-01')
AND (case when cop.dw_data_source in ('MCO','MCOI')
         then case when cop.restriction = 2
                  then 3
                  when cop.restriction = 3
                  then 4
                  when cop.restriction = 5
                  then 4
                  else 6
              end
         else cop.restriction
     end ) <> 6
and co.order_day = cop.order_day
and co.market_group = cop.market_group
GROUP BY cop.product_id, prod.DESCRIPTION
ORDER BY sales_dollars DESC;
```

Q13. Report sales $ for a market group and supplier for a 2 year period.

```
SELECT co.ORDER_NUMBER AS oltp_order_id
      ,sum( co.SUBTOTAL ) AS sales_dollars
FROM dw_order_items cop, dw_orders co, dss_segment_1
WHERE
cop.ORDER_NUMBER = dss_segment_1.ORDER_NUMBER
AND co.ORDER_NUMBER = dss_segment_1.ORDER_NUMBER
AND co.MARKET_GROUP = 7
AND co.SHIP_TYPE = 'std-us'
AND co.SUBTOTAL < 100.00
AND co.ORDER_DAY BETWEEN DATE('01/01/2002') AND DATE('12/31/2003')
AND cop.SUPPLIER_ID = 167
AND cop.MARKET_GROUP = 7
AND cop.ORDER_DAY BETWEEN DATE('01/01/2002') AND DATE('12/31/2003')
AND (case when cop.dw_data_source in ('MCO','MCOI')
        then case when cop.restriction = 2
                then 3
                when cop.restriction = 3
                then 4
                when cop.restriction = 5
                then 4
                else 6
             end
        else cop.restriction
     end ) <> 6
GROUP BY co.ORDER_NUMBER;
```

Q14. Report which sellers were active on a certain day in a selected market group

```
SELECT
  co.single_seller_ID
  ,co.ORDER_CHANNEL AS ORDER_CHANNEL
  ,COUNT( DISTINCT co.customer_number ) AS new_customer_count
FROM
  dw_orders co,customer_org_units cfo
WHERE
cfo.order_number = co.order_number
AND co.MARKET_GROUP = 1
AND co.ORDER_DAY = DATE('2003-11-27')
AND (case when co.dw_data_source in ('MCO','MCOI')
        then case when co.restriction = 2
                then 3
                when co.restriction = 3
                then 4
                when co.restriction = 5
                then 4
                else 6
             end
        else co.restriction
     end ) <> 6
GROUP BY co.single_seller_ID, co.ORDER_CHANNEL;
```

Query characteristics

Figure 20 outlines the query name, the major tables involved, a brief description of the query, the access method used for the larger tables, an approximation of how many rows were accessed and the number of rows which were returned for the query.

Query	Tables joined	Description	Access	Number rows read	Number rows returned
Q2	dw_order_items	Aggregate over time for given seller	MDC	98%	1
Q3	dw_order_items	Aggregate over time for given product	MDC	59%	1
Q4	dw_order_items dw_orders customers	Aggregate over time for set of customers	RID index		1
Q5	dw_order_items products	Rank sellers on items sold in category	scan	100%	N
Q6	dw_order_items products	Rank sellers on revenue in category	scan	100%	N
Q8	dw_order_items products	Seller revenue in quarter compared same quarter in previous year	MDC scan	98%	24
Q9	dw_order_items products dw_order_items products	Sellers who sold best in category in given year plus what else they sold	MDC	19%	400
Q11	dw_order_items dw_orders, customer, product_group	Aggregate set of market-groups for given date	MDC	.027%	8000
Q12	dw_order_items dw_orders products	Aggregate set of market-groups for given seller in current year, order by revenue	MDC	5.7%	2000
Q13	dw_order_items dw_orders dss_segment_1	Aggregate set of market-groups for two year period	RID index		10
Q14	dw_orders customer_org_units	Aggregate data on given date for given market-group	MDC	.014%	1000

Figure 20. Query characteristics

Query CPU and I/O usage

Query	**I/O rate**	**CPU usr**	**CPU sys**	**CPU idle**	**CPU WIO**
Q2	62M/sec	3	2	10	85
Q3	62M/sec	3	2	10	85
Q4	62M/sec	3	2	10	85
Q5	520MB/sec	43	13	1	43
Q6	520MB/sec	43	13	1	43
Q8	62M/sec	3	2	10	85
Q9	190M/sec	14	14	0	72
Q11	5M/sec	0	0	75	25
Q12	28M/sec	2	1	53	44
Q13	62M/sec	3	1	9	86

Figure 21. Query CPU and I/O usage

Figure 21 shows the I/O and CPU resources consumed during sequential query runs.

References

InfoCenters

DB2 Information Management Software Information Center at
ibm.com/infocenter/dzichelp/index.jsp

White papers

See all the IBM High Performance On Demand Solution white papers at ibm.com/websphere/developer/zones/hvws Of particular interest, see *Provisioning Best Practices for On Demand Data Centers.*

The **DB2 Universal Database** automation package is available for Tivoli Intelligent Orchestrator v1.1 and Tivoli Provisioning Manager v1.1 for AIX. Visit the IBM Orchestration and Provisioning Automation Library (OPAL) Web site at www.ibm.com/software/ondemandcatalog/automation

Acknowledgements

We thank the many who contributed to this project and this paper. We especially appreciate the contributions of:

Pat Selinger, Vice President, Data Management Architecture and Technology Executive Sponsor
Berni Schiefer, Manager of DB2 UDB Performance, Platform Exploitation and Autonomic Computing
Haider Rizvi, Manager of DB2 UDB Data Warehousing Performance
Bill O'Connell, Senior Technical Staff Member, Chief BI Architect, DB2 UDB Development
Larry Lange, Project Manager, HiPODS Center
Dawn Seymour, Distributed DB2 Performance and Technical Sales Support
James Cho, DB2 UDB Performance
Shu Lin, DB2 UDB Performance
My-ha To, HiPODS Software Engineer
Stew Edelman, HiPODS Systems Engineer
Latha Colby, DB2 Analytics, DBTI
Susie Holic, Editor
Allen Kliethermes, Business Intelligence Solutions Architect

And the SEEL team: Jerry Heglar, Manager SEEL Integration Center

Special thanks to:
Jack Stevens, Gradient Systems
Doug Norton, Newisys
Conor Malone, AMD

Notices

Trademarks

Special Notice