November 2003

DB2 Information Management Software

High Volume OLTP Database Cluster for the Financial Industry with IBM® DB2® for Linux

Project Report

Boris Bialek
IBM Toronto Lab

1. Introduction

Starting Point
A typical project report starts with a simple statement of the target objective and then moves forward towards the implementation. The project described in this report is different as it not only describes the aspects of a solution to a business problem, but it touches on a fundamental subject for many people in the IT industry: How does the Linux operating system and changing paradigms affect the execution of a project? What are the drivers in a project where the steps are not always well defined and the road to success has many questions? This paper describes one specific project and the experience in terms of Linux implementation is easily transferable to other projects.

The Customer
In November 2002, IBM sat down with a major Wall Street customer to discuss the possibility of using Linux as an operating platform for one of their major back-office applications involving a database. The customer had broad experience with the Windows®, UNIX® and z/OS® operating systems, but no experience with Linux. This customer questioned whether Linux could perform in their stringent business environment.

The usual way to tackle a business proof-of-concept would be to take a smaller database from an open system platform and migrate it to Linux. The customer decided to focus on one of the “most challenging and business critical back-office databases” running their “balances & positions” application. This was their primary accounting application for depots and trades.

The customer’s technology direction required application and solution deployment on “every application platform at any time, at any location”. This kind of flexibility often adds significant cost to any project. However, it allows the customer to have the freedom of a global standardized infrastructure that can be managed locally, but in a unified fashion. For example, Tokyo takes over the operation when Wall Street is closing for the day and then Tokyo hands operations over to London, and so on. This requirement for complete seamless flexibility across locations starts the discussion relating to the ease and flexibility of Linux.

The technical goal of this project was to deliver performance throughput of 1.2 Million Posting “events” per hour with the same level of reliability as the existing implementation running on a non-Linux platform.

In addition, the following additional criteria were also needed:

- Integration of the database in the existing corporate Linux infrastructure and utilization of the customer network environment, including remote boot-up and central AFS
- Implementation of security through the corporate Kerberos system
- Integration with the Veritas Volume Manager, Veritas File System and Veritas Cluster Server as preset corporate standards
- Interfacing with the existing z/OS Enterprise Server infrastructure through Software AG Natural 4GL
- Measuring the true cost of Linux
- Measuring the cost of the migration effort to Linux
- Measuring commodity-based hardware infrastructure

IBM provided the recommended hardware solution, although the architecture of the solution and the final choice of components were still open and subject to discussion. The customer had a large number of highly skilled database administrators working with DB2 for z/OS. The decision was made to educate those DBAs in DB2 for Linux instead of working with open system DBAs coming from a different database on UNIX. This decision reflected the customer’s desire for utilization of available knowledge in-house rather than hiring new external resources for later deployment.

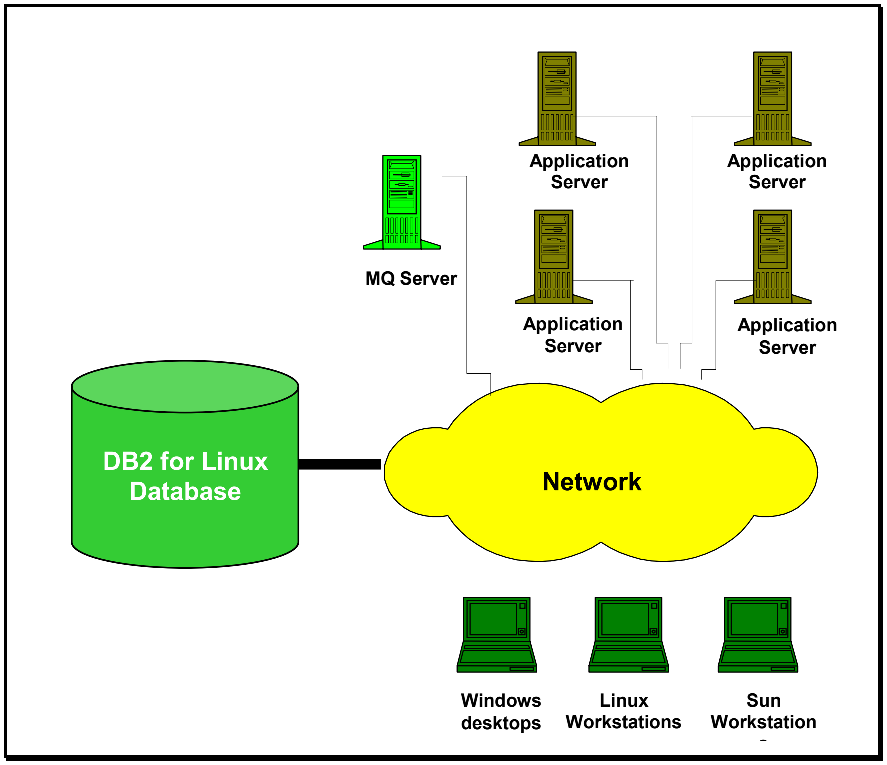
The work was clearly defined and the team began looking at the actual database design and the finer aspects of the application.

2. Architecture Discussion

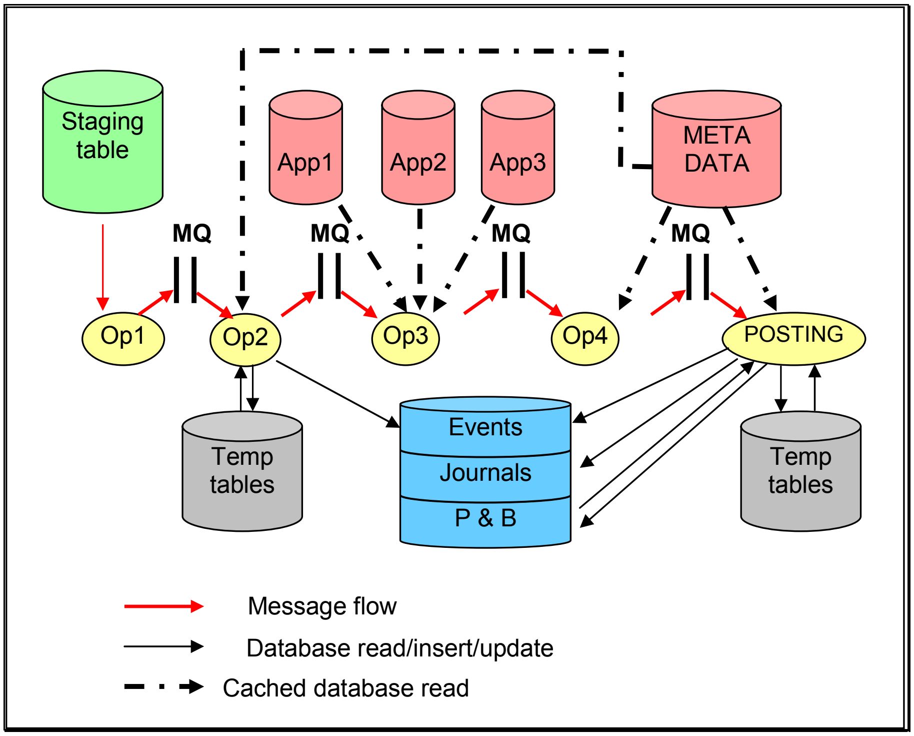
Application Environment

The application gateway was predefined through a framework of MQSeries® message queues that ensured the reliability of the various steps in the processing of a posting event.

The message queues were run on a 4-processor xSeries® server, which fed into a group of 5 application servers equipped with 4 processors each, and then into the database.



The application used the database as an intermediate staging area for the message processing as well as the final data store for the posting application. During execution of the application, high volumes of inserts and updates were processed. This led to a requirement for a very effective storage infrastructure with large I/O capability. The team sized the disk storage requirements for the application at 12 terabytes overall, with 4 terabytes of raw data at any time.



The application design required a large number of temporary tables for pre-computation prior to posting the actual event data into the database.

The Database

The database design was laid out in the usual fashion for parallel OLTP systems. The design used a rather small number of very large tables to hold the actual event information and the related entries in the transaction logs called journals (these are record logs in the application context and should not be confused with database logs used for transaction consistency). In addition to the events and journals, a larger number of less used tables containing reference data were identified in the design phase. In order to optimize performance, replication MQTs were implemented on each database server in the Linux cluster to perform the consistency checks and data enrichment for the application. Examples for this processing are stock ticker symbols and backup information of each stock traded. This led to a fairly straightforward layout of the database following the flow of the data into the different segments: event tables, journal tables, account information, stock information and the system catalog, of course.

One of the requirements was for a commodity-based hardware infrastructure. This meant implementing a scale-out solution using DB2 for Linux clusters.

We then had to determine how to implement the database design for the layout of the application servers and message queues over the cluster. Two options were available:

- Feed all traffic over a single partition as the sole coordinator of all transactions. The application did not require modifications with this approach and could benefit from the parallelism of the database servers.
- Adopt the application framework to scale out with the number of nodes and distribute the workload to as many coordinators as possible in a real distributed grid-like fashion.

The first approach fit well into the customer's experience on other platforms and the existing database design required minimal modification. The proof-of-concept team implemented both database designs to evaluate both architectures, which were easy to implement.

The results of this quick test proved several interesting points:

The existing design of the database did not need to be changed when the coordinator was distributed across the database servers. The application required a two-line application code modification to identify the best-suited node for the query (insert, update, stored procedure, etc.). The only action needed to implement this feature was to get the optimal partition information from the database cluster. Here is the raw code written during the project:

```
EXEC SQL VALUES CURRENT DBPARTITIONNUM INTO :part_num;
EMB_SQL_CHECK("VALUES CURRENT");
sprintf(declare_stmt,"DECLARE GLOBAL TEMPORARY TABLE MYTEMP
(KEY INTEGER NOT NULL, DATA INTEGER NOT NULL) ON COMMIT
PRESERVE ROWS NOT LOGGED WITH REPLACE IN
USERTEMP%0.3ld",part_num);
EXEC SQL EXECUTE IMMEDIATE :declare_stmt;
EMB_SQL_CHECK("DECLARE GLOBAL");
```

This application code identifies the partition for optimal inserts set as a variable for use by subsequent application code.

After reviewing the actual change, the customer determined that this change should be the standard for any future applications that are deployed in a clustered infrastructure where the components interact with each other, independent of DB2 for the Linux database cluster.

The second difference between a cluster and a non-clustered system layout was the use of replication for metadata tables. The amount of data was so small (less than 1% of the overall database size) and had such a low update rate that it would have been a waste of resources to run these from a single node. When the decision was made to choose the metadata replication, the team still expected a high networking load through the application. As it turned out, the DB2 performance for identifying the optimum node to connect to -- delivered a well-balanced behavior, making the precaution unnecessary. Nonetheless, the replication of metadata is an easy way to reduce network traffic in a cluster and would have allowed the team to use standard Gigabit Ethernet for the interconnect, versus choosing a high-speed interconnect.

The rest of the database layout decisions were actually uneventful since the database was designed already for high volume, and was simply migrated to the Linux platform.

Hardware

With the deployment requirement of “every application, every platform, any place,” the base decision for the servers was made for IBM eServer™ xSeries x360 servers. The xSeries x360 is a 4-way server with the latest Intel® Xeon server processors and has the required number of PCI slots to meet the rather broad customer requirements. The customer requirement for complete redundancy of components was also considered, which eliminated the use of dual processor 1U “pizza boxes”, such as the x335. Larger servers with more than 4 processors would also have been an option for the implementation. For example, the IBM eServer xSeries x440 or x445 were considered, but then the number of physical servers would have been reduced to 2, and the customer clearly required a larger number of physical nodes for the project in order to prove the scalability, extensibility and flexibility of the planned approach. We also wanted to keep the infrastructure costs as low as possible.

The final decision was to deploy with 5 servers with 4 processors each, following the sizing for the needed raw processing power. Some growth in the application space was accounted for as well. The system was intentionally not oversized, and the configuration was selected for the exact amount of data. If the number of servers turned out not to be sufficient (as some of the team members expected), then additional server units could be made available. This proved the “scale out” paradigm that was considered critical to the overall business value of the pilot.

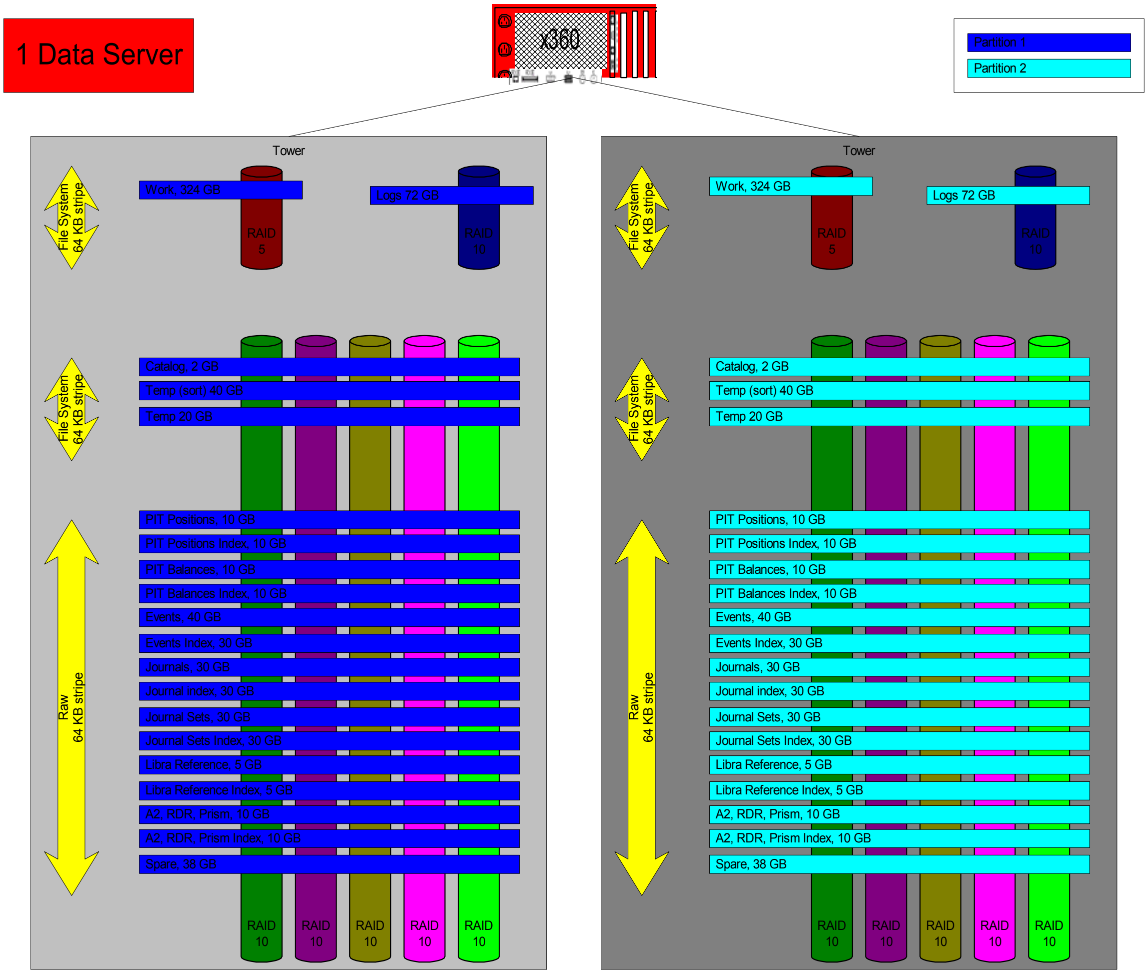
Each server had additional components added as follows:

- The onboard 1 Gbps Ethernet was duplicated, to have full redundant Ethernet connectivity,
- Remote system management for complete control, from the hardware boot cycle (power on/off), to complete console control. (In the future, the servers may reside at different locations, so this was considered critical. No redundancy was needed since the operation could still have been executed with local operation in the event of an emergency.)

We added InfiniBand interconnect for Inter-Process-Communication (with socket offload). We did not use redundancy here, since redundant Ethernet adapter was considered the fallback solution for this component.
Two redundant dual-channel Fibre Channel adapters with 2Gpbs capacity, which ran in full failover mode with channel-bundling, were added to allow a maximum of 8Gbps I/O per single server.

On the other hand, a solution of two independent InfiniBand adapters would have delivered the same functionality and even easier deployment. In this case, the two adapters would have delivered twice the redundancy. This would have also provided for 40Gbps I/O capacity. This solution might be deployed at a later stage, but the chosen infrastructure also considered the most highly available components at the time.

The server picture was as follows:



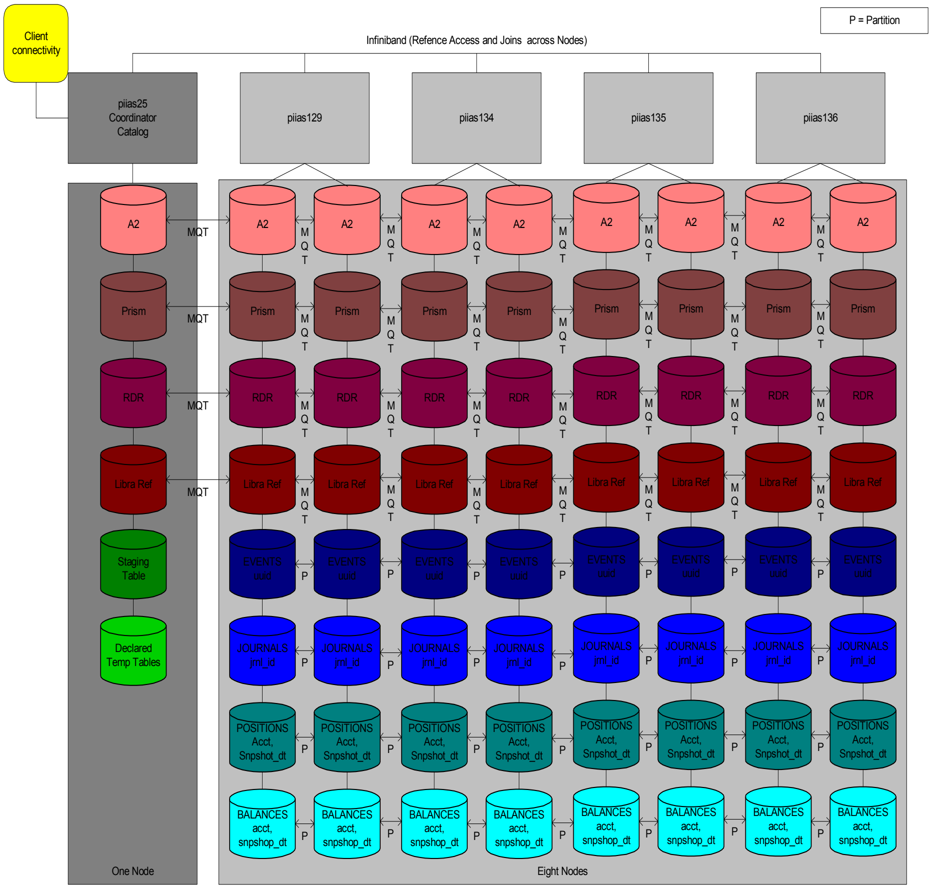
Software

The customer has a broad deployment of Red Hat Enterprise Linux 2.1 in house, trained personnel and an existing support contract with Red Hat. Therefore, the specific Linux distribution (or better, the discussion about "the right one") did not come up. The professional system layout in the customer infrastructure allowed the remote installation of the servers through the default image from the customer without any further need for customization. This was the first real proof point for the customer that DB2 is transparent for the choice of Linux distribution, contrary to other products tested earlier. DB2 for Linux has a rigorous validation program available to various distribution partners (see http://www.ibm.com/db2/linux/validate for details).

During the actual testing, a kernel threshold bug was identified and fixed in parallel through Red Hat. A kernel update solved the issue. This was only visible in very specific ultra-high load situations. Without a support contract, it could have become a very annoying problem for the customer. The existing support contract IBM has with Red Hat allowed a smooth interaction with Red Hat. For new deployments in customer environments, it should be a mandatory item to ensure complete sufficient support at the expected service level for the Linux operating system.

The next layer of software was the Veritas Volume Manager and the Veritas File System. Both were broadly deployed as a standard in the customer environment across all operating system and system architectures. While the local hard drives in the server hosting the operating system and swap space were using the Linux default file system (ext3), the actual data and log volumes used the Veritas Vxfs. In the end, the solution used the Veritas Volume Manager for managing the 12 terabytes of storage in the IBM FAStT Storage Servers. Again, the file system was the ext3 format from Linux. Technically, there was no notable difference that could be seen and all Veritas components worked as expected.

Here is a short view of the actual deployment plan:



The customer used Veritas Cluster Server for failover purposes. Although the detailed description about this option would go beyond this paper, a detailed overview is available in a white paper about Veritas Cluster Server on the DB2 Web pages.

Security

Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography.

The customer utilizes a corporate wide Kerberos system that allows global access to the IT infrastructure. Behind the Kerberos system was a vast definition of up to 10,000 UNIX groups and dedicated permissions for each user.

The DB2 open philosophy allows the binding from the user context to a Kerberos ticket via an open code segment that expands the default DB2 library with the binding for each individual security product. The integration in a standard environment as used by this customer took less than 10 minutes and delivered complete transparency of the customer security concept into the deployed solution.

Here is a short overview of the implementation:

DB2 Registry Variables

Set the DB2_NUM_CKPW_DAEMONS registry variable to disable the password checking daemons and use the db2ckpw utility for Kerberos authorization.
Command: db2set DB2_NUM_CKPW_DAEMONS=0
Note: Stop and start the database instance to activate this variable.

Permanent Kerberos tickets

To allow production users to execute remote commands and cron jobs, a permanent Kerberos ticket is needed. Thus the permanent ticket is your “secret” for authentication against the overall infrastructure.

Create executables

There are 2 routines that must be replaced with “kerberized” routines. The *db2ckpw* routine is utilized to authenticate database users. The *db2dassec* routine is used to authenticate IBM DAS tool users. The DAS tools include all GUI tools. To enable the specific Kerberos environment, utilize the db2ckpw.c and db2dassec.c from the sample code to replace db2ckkpw and db2dassec in the actual implementation, respectively.

Environment variable

Set the Kerberos credential cache environment variable: KRB5CCNAME=/var/spool/tickets/udbinst. In your environment, you would want to set this to the permanent Kerberos ticket.

Userprofile

Modify the PATH environment for DB2. For each instance, the ~instanceowner/sqllib/userfile will contain the following statement to pre-pend the “non-kerberized” remote commands to the PATH.

```
export PATH=/usr/bin:$PATH
```

Kerberos authentication is associated with a server. In a clustered environment, authentication is only required on one server in the cluster. For DB2 ESE in a clustered environment, it is not necessary to use kerberized remote commands. Kerberos tickets are granted per server. For a clustered and trusted environment, “kerberized” commands would over-complicate the environment as well as add performance overhead unnecessarily.

Debugging

- If you have problems authenticating, the “kerberized” routines should be verified. After making these changes, the database security will implement “kerberized” routines. Thus these routines must be maintained with all other “kerberized” routines.
- New software or fixpacks may overwrite these routines. It is recommended to keep copies of the “kerberized” routines outside the standard IBM installation directories. Implement these changes at your own risk.

The table below applies when creating Kerberos IDs for the database. Kerberos changes are noted in Red.

Purpose	Kerberos Principal	Primary GROUP	Secondary GROUP	LOGIN-ABLE
Instance owner	udbinst	iadmusr	aadmusr	No
Fenced user	udbfenc1	fadmusr	N/A	No
DAS owner	udbconn	aadmusr	iadmusr	No

Caveats

The Set-Uid bit is not necessary with kerberized routines. The IBM installation assumes a UNIX installation and changes the ownership of db2ckpw and db2dassec to root and sets the Set-Uid bit. When installing the "kerberized" routines, change the ownership and group of the routines to the instance owner. The permissions should allow everyone to read and execute the executable and remove the Set-Uid bit.

Do not write a Kerberos ticket in the DB2 security routines. If a ticket is written, the permanent ticket gets overwritten with the user ticket. See Appendix A for sample db2ckpw.c and db2dassec.c routines.

A slight disadvantage was the complete unification of the environment. DB2 had to scan all 10,000 user groups to identify the permission set of permissions and denials. During the default logon, this led to a slight delay in connection time, which was negligible as the message queues and application servers connect and disconnect only during start and stop. From the usability perspective, a more compartmentalized approach may deliver better results.

3. Migration

The migration process from the host database was expected to be painless and, in fact, was trivial. A short overview about the logical order of the actual process is given here for completeness. As this was done during the early prototyping, it actually happened in parallel with the final implementation of the hardware platform, and was done independently at that time.

The list below contains the steps necessary to migrate the database objects and data.

1. Create tables
2. Create indexes
3. Create views
4. Create triggers
5. Grant privileges on table and views to public
6. Unload database from MVS™
7. FTP data to Linux
8. Load data to DB2 ESE
9. Set integrity for any tables in check-pending state
10. Runstats
11. Create UDFs
12. Create stored procedures

To simplify the process, scripts were used to modify syntax and functional differences.

Reference Tables

The smaller application and non-application reference tables were exported from the mainframe in IXF format using DB2 Connect™ and then loaded into the DB2 for Linux tables. IXF format automates the movement of data across the DB2 family by creating the tables/indexes at the target database as well as moving the associated data. The table spaces were created manually to the appropriate size.

The viability of this approach depends on the size of the table and whether it can be moved via DB2 Connect in an acceptable timeframe. The amount of data loaded for representative data amounts in the tables “A2. RDR” and “PRISM” was approximately 10 GB. The total amount of time spent in this process was approximately 2 to 3 hours.

Partitioned Tables
DB2 communication uses the DRDA protocol, which requires the receiver of the data to do the data conversion. DB2 Connect adheres to this protocol. When DB2 Connect sends out an SQL statement and its input host variables, DB2 for OS/390® converts them into EBCDIC before processing (DB2 for z/OS can be configured to store information in an ASCII format). DB2 for z/OS sends back data in EBCDIC, and DB2 Connect converts the result to ASCII/ISO representation before returning it to the user. DB2 Connect uses the application's code page when returning the result to the application. The application code page is derived from the active environment when the database connection is made. If the DB2CODEPAGE registry variable is set, its value is taken as the application code page. It is not necessary to set the DB2CODEPAGE registry variable because DB2 will determine the appropriate code page value from the operating system.

The conversion of larger, partitioned tables was less automated:

The actual data was unloaded from DB2 for z/OS in EBCDIC delimited format, and converted to ASCII by the FTP program that moved the data to the Linux platform. Additional BMC utilities were utilized to get column-delimited output from ShadowE. This functionality will not be provided by the IBM utilities until DB2 for z/OS Version 8. The BMC utilities were also needed for column support; both the NULLSTRING and ENCLOSED BY features were utilized. In several tables within the columns, a quote was used as valid data and not as a terminator that was passed by using the column delimiter of x’09’.

Some observations on the BMC unload jobs that were set up to dump the application data from Shadow SYSE in delimited format:

- No sorting is done by the BMC utility, so the sortwkxx datasets are not required
- If jobs are being set up to run concurrently, different utility IDs must be used (in PARM='DPAE, BMCUIS08, NEW, MSGLEVEL(1)' BMCUIS08 is the utility ID - this must be unique for concurrent execution).
- REGION=64M instead of REGION=4M should be specified on each unload
- BUFNO=30 should be added to each SYSREC DD statement

A problem was encountered with embedded data in the application tables. In loading data from the mainframe to DB2 ESE with the Data Partitioning Feature (DPF) platform, there were two tables that had embedded CRLF delimiters within the column data. The DB2 ESE with DPF data load has the following priority order for delimiters: record, character then column. As soon as DB2 for Linux detects the CRLF, it will mark that as the termination of a row. In the case of these two tables, the CRLF occurred in the middle of a character field. The workaround was to have a Perl program that went through the input unloaded file and changed the CRLF to another value when the CRLF was detected in the middle of a column.

Special Considerations
The following considerations are incompatibilities or differences in implementation that had a minor impact on the conversion effort.

- A performance bottleneck for INSERT processing on tables with Identity columns occurred due to traffic to the catalog node. This was resolved by increasing Identity Cache Size from 20 to 50000 for the table containing identity columns (Journal_Set0, Journal_dsc0, and LB_JRNL_STAGING tables).
- Indexes were created on the “Partnum” column that was used for special handling of DB2 for zOS partitioning. The Partnum column was either removed from the indexes or placed at the end. The

primary purpose of this column on the mainframe was to assist with utility work. In DB2 for Linux, hashing accomplishes the need to spread out the data.

- In DB2 for z/OS, indexes do not automatically get built for primary key definitions. DB2 for z/OS will allow the creation of the table but render it unusable until a primary index is created. DB2 for Linux will automatically build indexes on top of primary keys if one is not available. DB2 for Linux was set up to generate the primary keys. When the supporting create unique index DDL was executed, it received a message saying the index already exists. These warnings were ignored.
- The application data model had several different partitioning keys that did not facilitate collocation in the DPF environment. **During the tests, the application achieved satisfactory performance in spite of the lack of collocation.**
- The syntax within DB2 for Linux does not support DBNAME and TSNAME. There is a direct correlation between TSNAME and TBNAME. For the DBNAME the column definition was hard-coded with the literal of "MYAPPTST".
- In allocating table spaces in DB2 for Linux, the entire file is being formatted during its allocation. This is different from DB2 for z/OS in that formatting occurs upon usage and the size is 2 or 3 cylinders. For a 50 GB table space in the database, this process took about 90 minutes.
- Two application tables use identity definitions for the “journal_set_id” and “journal_id” columns. When the data was migrated from the DB2 for z/OS, these values were loaded into DB2 for Linux. DB2 for Linux had no knowledge of the sequence numbers that were assigned for both tables. SQLCODE 803 (duplicate keys) started to occur once the pipeline started to assign values for these columns, which were previously given by DB2.

4. Implementation

The implementation phase was scheduled to be an 8-week exercise of basic implementation system setup, followed up by a performance phase of equal time. The implementation phase was actually much shorter than planned. The hardware setup was finished within three days, including the basic installation of the operating system. The database installation took less than 15 minutes across all servers through the use of DB2 response files. The first DB2 server was installed manually and generated a single response file that could be applied for a silent installation of the following servers. A different way to perform the installation would have been to use the Red Hat “kickstart” that permits the copying of full systems and automatically updates the hostname and TCPIP addresses.

Following the core installation of DB2, the seamless compatibility between clusters and non-clusters for DB2 was evident. The DDL from the early prototype experiments could be directly applied to the DB2 cluster, with just the addition of the node groups for the cluster and defining the partitions of the cluster. (The DB2 experienced reader may know that this is accomplished simply by adding an NFS share and one line in the db2nodes.cfg file).

The largest amount of setup time (next to the physical assembly of the hardware) was the creation of the needed volumes and actual table space initialization. After that was completed, the database was ready for its first transactions.

During the prototype phase the following subjects were taken into account: These issues were addressed next.

General Changes

- Hard-coded DB2 OS/390 SQL codes in application were transformed into sqlstates instead of sql codes to ensure compatibility between the DB2 platforms.
- DB2 for z/OS applications expect results to be returned in EBCDIC sort order. This code is related to mapping of account types which are 9 character alphanumeric strings. Assuming that UDB returns ASCII sort order, the code has to be modified.
- Received errors after "rollback to savepoint" calls, called "release to savepoint" after "rollback to savepoint"; as suggested in the DB2 documentation.

- DB2 for Linux does not support certain nuances of creation of temporary tables so the application code was altered to use declared temporary tables. JAVA code was updated to declare the global temporary tables and reference them as SESSION.table name. The stored procedures that access the temporary tables were also updated to refer to them as SESSION.table name. In the SQL proc(s) we had to include the DECLARE for the temp table in the SQL proc (inside an IF block that would never be executed) before the proc would build successfully. The C stored procedure did not have the same issue, so no DECLARE was included.
- DB2 for Linux does not permit a column definition to default its value to the CURRENT SQLID. The comparable syntax in DB2 UDB is “Current User”.

SQL User Query Changes

- User Query stored procedures on the mainframe have a specific "with ur for fetch only" syntax. This needs to be coded as "for fetch only with ur" in DB2 for Linux. Additionally, the declare cursor needs to be within a begin-end block in DB2 for Linux.
- Stored procedures executed from “Software AG Natural” programs contain logic to check if given range numbers are within a range specified on ACCT_TYPE_MAP columns, LGCY_FROM_ACCT_NUM and LGCY_TO_ACCT_NUM. The columns contain character data representing account number ranges. That is, from: 07AAAAAA0 to: 079999999. These ranges need to be modified to account for differences between DB2 for Linux and DB2 for z/OS, because of ASCII vs. EDCDIC sequencing. Otherwise, the stored procedures will not find any rows that satisfy the selection criteria.

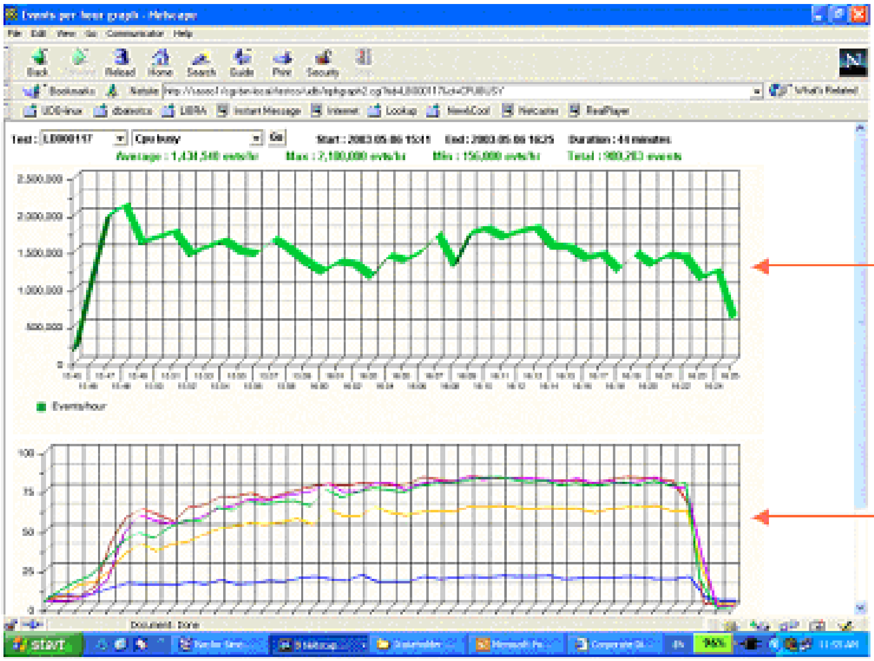
C Procedure Changes

- The application UDF utc.timestamp was coded in PL/I on the mainframe and had to be converted to an SQL UDF.
- A specific decimal SQL data type doesn't exist in DB2 for Linux. The decimal variables in the C stored procedures were converted to type double per the DB2 for Linux Application Development Guide. Since no precision can be specified on the double type, DB2 maintained the precision based on the table column definition.
- UDFs in a DB2 for Linux cluster environment that are written in C (or languages other than SQL procedure language) cannot contain SQL code. The application UDF was rewritten as an SQL procedure language UDF.
- In a C stored procedure, variable scope isn't recognized by the DB2 preprocessor, causing errors in code that used the same local variable name in multiple functions. These variables were renamed to unique names in the UDB version of the procedure to circumvent this.
- Our C stored procedures run on UDB in fenced mode. For performance reasons, KEEPFENCED=YES was specified in the DBM Configuration to prevent stored procedure binaries from being reloaded on every invocation.
- The default path for non-SQL stored procedure binaries applies at the instance level. This means that stored procedures created with the same name for different DB2 would overlay each other's binaries. The team got around this issue by creating a separate directory for each DB and specifying the full path to the binary in the EXTERNAL NAME parameter of CREATE PROCEDURE.
- All references to SQLSTATE were changed to sqlca.sqlstate (DB2 for Linux SQLCA is set up differently than DB2 OS/390 to reflect the different physical environments).
- IN some cases SQL SET statements were replaced with VALUES statements e.g. the statement EXEC SQL SET :V_JRNL_SET_ID = IDENTITY_VAL_LOCAL(); became EXEC SQL VALUES IDENTITY_VAL_LOCAL() into :V_JRNL_SET_ID,

All of these items are rather minor and were addressed quickly. The reader should note that the application changes were required for standard DB2 compatibility, versus being specific to a clustered implementation.

5. Performance

The base environment for testing was prepared through the “workbook” application that controlled the actual trading application (start/stop of the various complex components in a controlled fashion). Further, it collected all necessary data for later evaluation and performance analysis, including the number of events processed through the DB2 cluster.


Events per hour graph - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Netsite What's Related
UDB-linux dbstatics LIBRA Instant Message Internet Lookup New&Cool Netcaster RealPlayer
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
Events/hour
100
75
50
25
0
Document: Done
start

Test : LD000117 Cpu busy Go Start : 2003.05.06 15:41 End : 2003.05.06 16:25 Duration : 44 minutes

Average : 1,431,540 evts/hr Max : 2,100,000 evts/hr Min : 156,000 evts/hr Total : 980,203 events

NOTE: This chart shows some of the best throughput seen with an "empty" database, populating libra tables from scratch

Application throughput, shown as events posted per hour

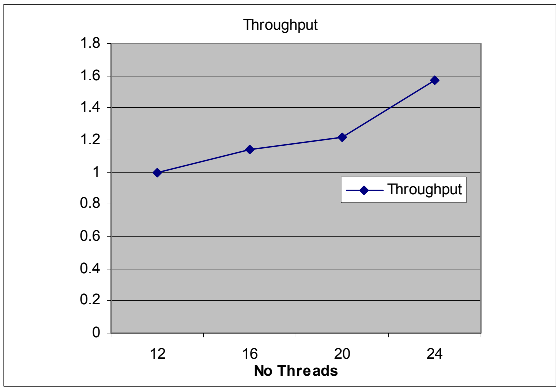
Database or OS metric: this chart shows aggregate CPU utilization on database servers as % busy

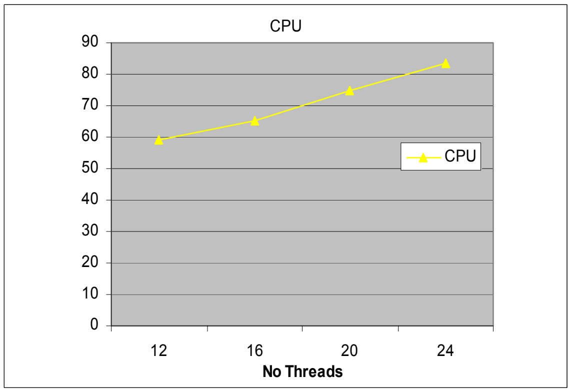
The picture below shows one executed run with the integrated comments. At the point shown, it was clear that the input stream of the data was maxed out.

The first test runs were executed with a single coordinator node for the database. This was a simple test to validate the actual application data flow. As expected, the bottleneck was the connectivity between the servers and the solution maxed out with about 250,000 “events” posted through the application pipeline. The limiting factor was clearly the single insert message queue through the system catalog node and then redistribution of the data flow across all other nodes for the actual processing.

The application changes discussed in the earlier chapter were not included at this point. Identifying the various steps and their implementation led the team to drive the event rate up to 300,000 events per hour. At this point, it was clear that a single coordinator node was a limiting factor so the team switched to a distributed environment with multiple coordinators. The expectation was set at that point that a limit of about 1.5 million events for the overall cluster should be in the range of a realistic goal.

A single message queue was set up per database partition, and the system throughput approached one million events per hour. Now the number of queues were extended to a maximum number of 24 message queues overall and the system hit 1.6 million events per hour, about 25% above the specified target goal. The increase from 16 to 20 message queues was later identified as a problem.

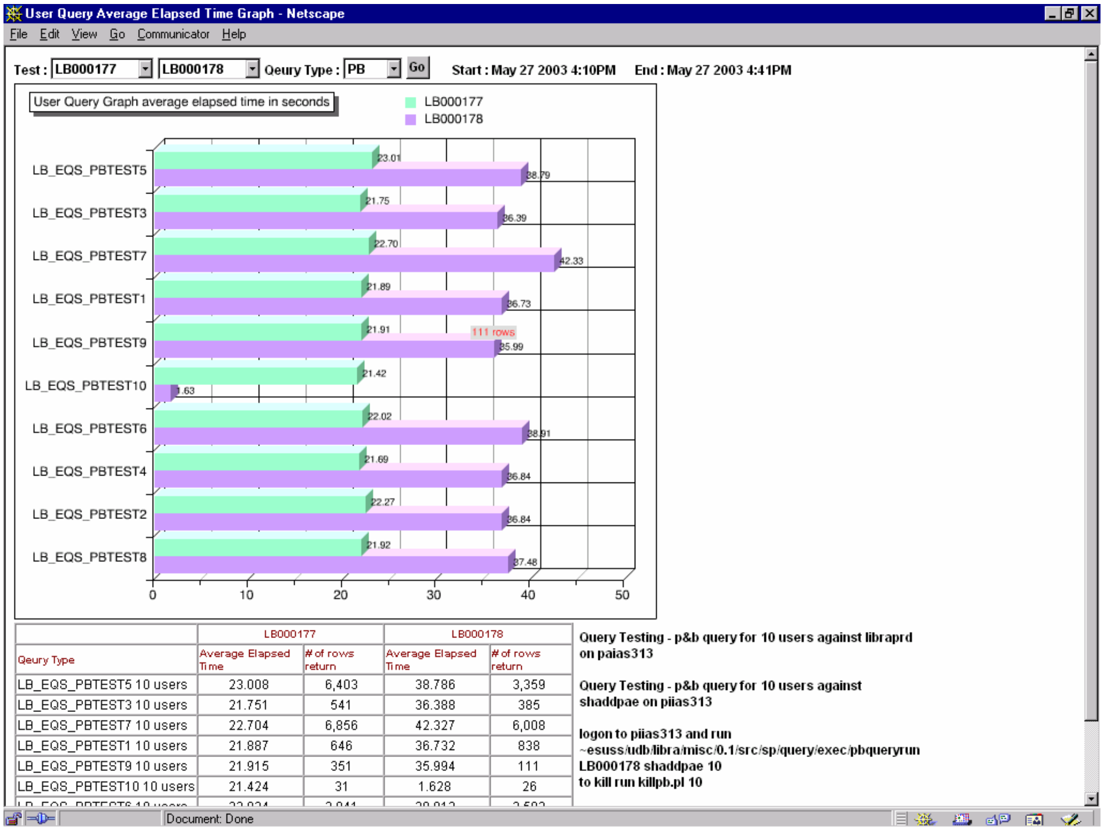




Interestingly enough, the CPU utilization increased directly proportional to the performance gains of the system. The number of clients had no impact on the system at all and no thrashing occurred.

Besides the processing of the events coming through the message queues, there was another application aspect that was critical for the deployment of a DB2 for Linux cluster. Could the cluster deliver real-time access to the data where DB2 was used simultaneously as a data warehouse, as well as servicing the OLTP workload? Would the performance be comparable to the DB2 z/OS expectations? The result was exceptional performance of the DB2 for Linux cluster. The project proved that the high I/O ratio, in conjunction with the processor performance and amount of memory had a significant impact on the overall performance, when compared to the existing mainframe. Clearly, DB2 for Linux is ready for enterprise deployment.

The following picture is a comparison of a number of customer queries run on DB2 for Linux versus the equivalent queries run on DB2 for z/OS.



Test : LB000177 LB000178 Qeury Type : PB Start : May 27 2003 4:10PM End : May 27 2003 4:41PM

	LB000177		LB000178	
Qeury Type	Average Elapsed Time	# of rows return	Average Elapsed Time	# of rows return
LB_EQS_PBTEST5 10 users	23.008	6,403	38.786	3,359
LB_EQS_PBTEST3 10 users	21.751	541	36.388	385
LB_EQS_PBTEST7 10 users	22.704	6,856	42.327	6,008
LB_EQS_PBTEST1 10 users	21.887	646	36.732	838
LB_EQS_PBTEST9 10 users	21.915	351	35.994	111
LB_EQS_PBTEST10 10 users	21.424	31	1.628	26
LB_EQS_PBTEST6 10 users	22.024	2,041	38.912	2,592

Query Testing - p&b query for 10 users against libraprd on paias313

Query Testing - p&b query for 10 users against shaddpae on piias313

logon to piias313 and run ~esuss/udb/libra/misc/0.1/src/sp/query/exec/pbqueryrun LB000178 shaddpae 10
to kill run killpb.pl 10

6. Conclusion

The customer's primary objective for the pilot was to determine whether Linux could deliver the reliability and scalability required. The complexity of the application framework and the sheer amount of data was a challenge to any database and application infrastructure, but the IBM DB2 for Linux solution delivered on all aspects of the project.

The Linux OS part of the project can only be described as uneventful. The operating system worked as expected under the high load tests, and no major flaws or architectural roadblocks were uncovered.

The complete planning and sizing at the beginning of the project definitely had an impact on the project overall. The customer considered Linux as a business platform from the start, and not just as an engineering environment. There was no "tinkering" or experimenting with the kernel of any sort. This was considered one of the major achievements of this project, since it demonstrated that you do not need to "custom tune" the operating system as a prerequisite for a successful project.

Another great lesson learned is the fact that amazing scalability can be achieved using commodity Intel servers in a clustered OLTP DB2 environment. While clustering is standard practice for warehouse and data mart applications, distributed clusters have not been widely adopted for OLTP applications. In addition, clustered configurations have always been considered difficult to optimize or manage.

This project demonstrated that, with DB2 ESE Version 8, the complexity of clustering has been reduced dramatically. There are applications that might be ideally suited for deployment in a clustered environment (independent of Linux as the operating system), but the specific scenarios where clustering should not be considered has been reduced. More significantly, the combination of using large-scale data warehouses with OLTP-like behavior, or continuous inserts and deletes in real time will clearly drive the size of DB2 for Linux database clusters into larger dimensions.

The scalability exhibited in this project was almost linear and would have allowed even larger amounts of data if it had been available. Besides the performance DB2 demonstrated in this project, the combination of the Java™ environment and MQSeries cannot be stressed enough. MQSeries delivered amazing capabilities for scalability. Using MQSeries, the scalability of the input data streams could be adapted to the changing database configurations, considering available memory, the number of database partitions and the available computing power of the processors, without changes to the application.

The final question that the team considered was if Linux was any easier to deploy and manage that any other UNIX variant. The answer was clearly "yes". Especially when it comes to clusters of the size used in this project. Even if standard configuration procedures such as the configuration of the Veritas VLM volumes is the same on other UNIX systems, the easy and clear structure of the Linux distributions makes it very easy to replicate large numbers of servers with individual, discreet requirements without any changes to the operating system kernel settings or parameters. The best proof of this is that DB2 for Linux was deployed on the customer-specific version of Linux, which had been optimized for Web serving and other front-tier applications, demonstrating the interoperability of DB2 with the Linux kernel. .

In conclusion, is Linux ready for enterprise computing workloads? Based on the results of this project, absolutely!

Acknowledgements

I would like to acknowledge the performance from our team with Kerry Bruington as our Project Manager, Bo Taramina, Gus Branich, Christopher Tsounis and Robert Bergman as the DB2 consultants, Florbela Vieira and Yvonne Chan from the Linux team in the IBM Toronto Lab, Anil Kappor and Mahmood Hussein for their great work on system and storage infrastructure (surviving my comments for that amount of time) and the whole team from our customer. Without all of them, this project would not have been possible.

© Copyright IBM Corporation 2003
IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

Printed in United States of America
10-03
All Rights Reserved.

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

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

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

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

The information in this white paper is provided AS IS without warranty. Such information was obtained from publicly available sources, is current as of 01/30/2003, and is subject to change. Any performance data included in the paper was obtained in the specific operating environment and is provided as an illustration. Performance in other operating environments may vary. More specific information about the capabilities of products described should be obtained from the suppliers of those products.