February 2005

DB2 Information Management Software

IBM DB2 Universal Database for Linux scales with the FinTime benchmark in a customer environment

DB2 UDB for Linux proves its scalability and flexibility

Boris Bialek
IBM Toronto Lab

1. Overview

IBM® DB2® Universal Database™ (DB2 UDB) is the acknowledged leader for Linux*[1] technology in database deployments. The highly scalable DB2 clustering technology has been renowned for its performance since its introduction in 1996. As of DB2 UDB Version 8.1, the Enterprise Extended Edition clustered offering became the Enterprise Server Edition with the Distributed Partitioning Feature, which merges the clustered and non-clustered database software into a single entity. For DB2 Universal Database servers, a single SMP database is nothing more than a special case of a cluster. The SMP database is technically a cluster of one node and can grow from a single node to a 1000-node ultimate performance cluster environment. For a Linux environment, the DB2 clustering solution has been formalized into the DB2 Integrated Cluster Environment (DB2 ICE).

Many IBM customers deploy DB2 ICE solutions today, but the specific DB2 ICE environment described in this paper is a textbook example. Although it was "only" a benchmark, the DB2 ICE environment was executed as if it were a live environment – otherwise, a cluster of 64 nodes and 4 standby nodes on IBM eServer® 326 servers would simply not be manageable.

The database challenge was much bigger than a single partition. The task was to deliver a 2-TB and a 16.5-TB benchmark using the FinTime benchmark kit from the Computer Science Department of New York University[2] (NYU). The customer set a very rigid time limit for the execution: from the first moment of planning the hardware system layout to the final benchmark result, only four weeks were available. The prerequisite for this benchmark was the implementation of the benchmark kit on a Linux operating system (including all data generation stages).

This paper describes the decision points at the various stages of the benchmark, the implementation of the system architecture, and the results achieved. It has an introduction for non-financial people to the FinTime benchmark to make the database work easier to transfer to other fields and industries.

[1] See the trademark attributions on the last page of this white paper.
[2] http://www.cs.nyu.edu/cs/faculty/shasha/fintime.html

2. FinTime benchmark

Everything in the world of corporate IT starts with the need to solve a business problem. The FinTime benchmark is an example of a solution for a problem based on a specific requirement from the financial industry. Before jumping into solution mode and addressing the computer side of the problem, we spent some time looking into the actual problem that we were about to solve.

The FinTime benchmark has a long track record in the industry. It was originally developed by Prof Dennis Shasha of the NYU as a tutorial for time series databases and then evolved into a complete vendor-independent benchmark kit that allows a fair comparison of results between vendors on completely different data sets.

The benchmark uses the most basic information of the stock market that is commonly available and splits it into two major components, the “historical data” and the “tick data”.

Historical data describes each stock in terms of its behavior over an amount of time in a consolidated snapshot. The kit uses four base tables to describe the stock. The base table gives the metadata for each instrument traded. The original benchmark specification calls for a CHAR(30) data type as the unique key, but to adhere to the customer requirement, this was changed to an INTEGER data type to accelerate the database behavior.

Field Name	Data Type	Comments
Id (KEY)	INTEGER	Unique Key and Identifier for the financial instrument, e.g. IBM.N
Exchange	CHAR(3)	Stock Exchange where the instrument is traded
Description	VARCHAR(256)	Short description of the financial instrument, .e.g “Company name, Location”
SIC	CHAR(10)	Standard Industry Code, e.g. “COMPUTERS”
SPR	CHAR(4)	S&P Rating for the company
Currency	CHAR(5)	The currency used, e.g. “CAD” or “USD”
CreateDate	DATE	Date when the security came into existence

A typical event i during the growth of the Internet bubble was a stock split. Sadly, this is no longer such a common event, but they still can play a big role in the financial markets and need to be accounted for.

Field Name	Data Type	Comments
Id (KEY)	INTEGER	Key
SplitDate (KEY)	DATE	Date when the split was executed
EntryDate	DATE	Date the split was announced
SplitFactor	DOUBLE	The split factor as a decimal value, e.g., a 0.5 (for a 2:1 split) or 0.75 (for a 4:3 split)

Another typical event is the payout of dividends which leads to the dividend table.

Field Name	Data Type	Comment
Id (KEY)	INTEGER	Key
XdivDate (KEY)	DATE	Date of the dividend disbursement
DivAmt	DOUBLE	Amount based on the currency of the instrument
AnnounceDate	DATE	Date when the dividend is announced

And finally we want to know what actually was traded on the stock market each trading day--the so called "regular time series" of a specific financial instrument--and we place those data into the market data table.

Field Name	Data Type	Comment
Id (KEY)	INTEGER	Key
Tradedate (KEY)	DATE	
HightPrice	DOUBLE	Highest price for the day
LowPrice	DOUBLE	Lowest price for the day
ClosePrice	DOUBLE	Closing price for the day
Openprice	DOUBLE	Opening price for the day
Volume	LONG	Number of shares traded

Based on those four tables we can create a real live behavior of a time series environment. The daily market data for a stock is received from a data provider and used to populate the market data table. For example, in the real world those services are Reuters, Bloomberg or others. While the irregular activities like splits and dividends are added in a more irregular interval (and that is where its name comes from) the market data grows fast and regularly on a daily basis. In a

normal business environment almost every stock gets traded, or at minimum is offered to sell for a certain price, every day.

Historical evidence gives a volume size of 50,000 equity securities for the US, 100,000 equity securities for the G7 nations and about 1,000,000 equity securities for the world. For the actual benchmark with DB2 UDB for Linux 1,000,000 equities were utilized with a time span of 4000 days as requested in the benchmark description. More details on the historic data population and the actual data generation can be found in the original benchmark documents at http://www.cs.nyu.edu/cs/faculty/shasha/fintime.d/gen.html .

Based on the data a user is able to apply a number of scenarios to the data, for example, what is the value of $100,000 now if 1 year ago it was invested equally in 10 specified stocks (so allocation for each stock is $10,000). The trading strategy is: When the 20-day moving average crosses over the 5-month moving average the complete allocation for that stock is invested and when the 20-day moving average crosses below the 5-month moving average the entire position is sold. The trades happen on the closing price of the trading day.

The second part of the benchmark works directly with trades and reflects the OLTP like behavior of modern databases for data analytics. Ticks representing each trade are added at real time to the database while the database is used to identify certain stock market behaviors.

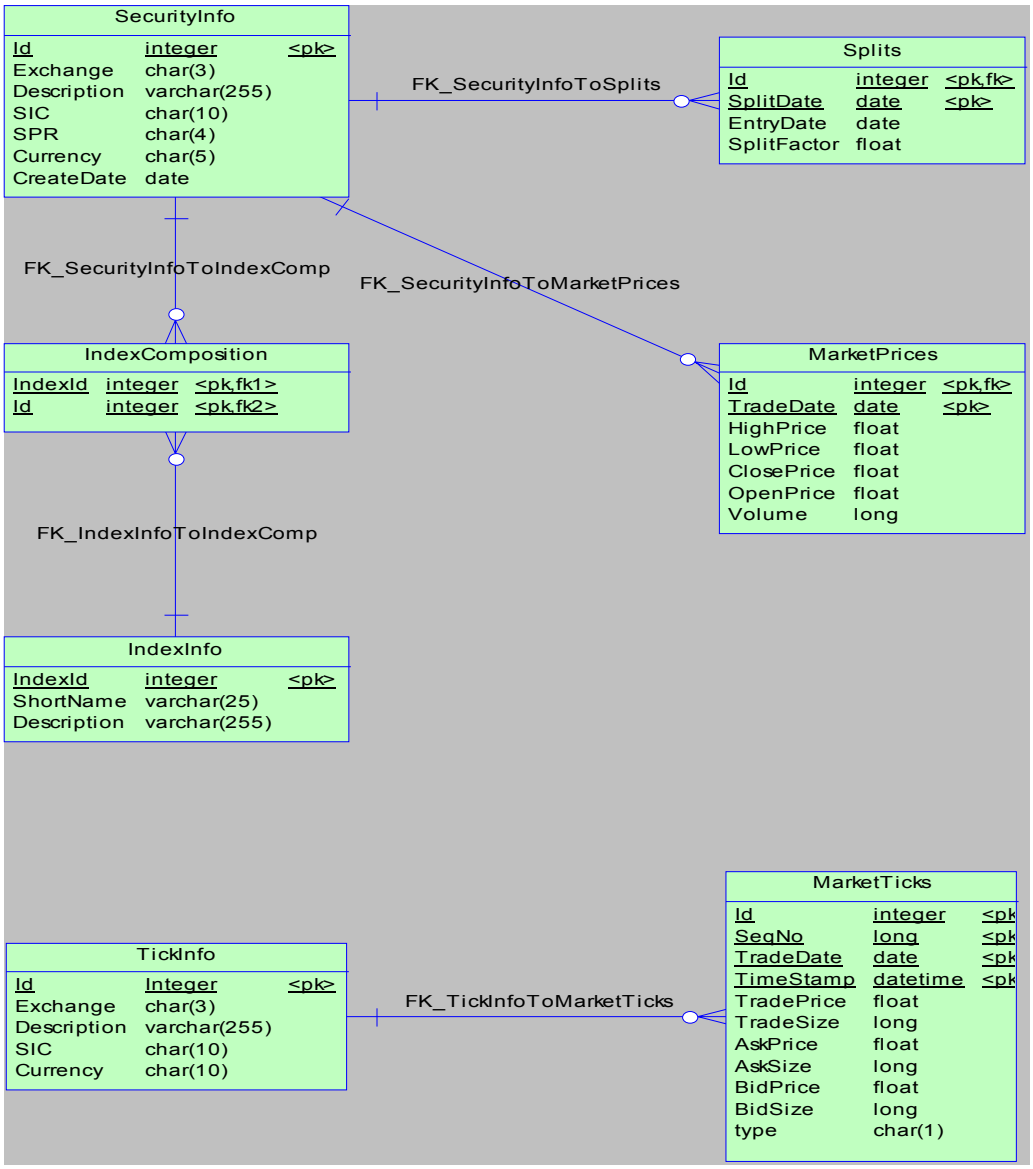
Field Name	Data Type	Comment
Id	INTEGER	Identifier for the security and key
Exchange	CHAR(3)	Exchange on which the instrument is actually traded
Description	VARCHAR(256)	A description for the security, e.g. long company name
SIC	CHAR(10)	The standard industry code
Currency	CHAR(10)	Base currency for the security trading

Now a table with the actual trading data is needed.

Field Name	Data Type	Comment
Id	INTEGER	
SeqNo	LONG	Unique Sequence identifier for each trade
TradeDate	DATE	Date the trade was executed
TimeStamp	TIME	Exact time of the execution
TradePrice	DOUBLE	Exact price at which the trade was executed
TradeSize	LONG	Volume in number of shares
AskPrice	DOUBLE	The price a seller asked for

AskSize	DOUBLE	Size of the transaction offered
BidPrice	DOUBLE	Price offered by a buyer
BidSize	DOUBLE	Volume of the transaction at the specific bid price
Type	CHAR	Indicator whether this is a quote or an executed trade

The following picture shows the actual implementation of the benchmark including the historical market data and the tick data.



The first part of the benchmark is of course the data generation but that is rather uninteresting for the actual execution. More important are load times, (how much data per minute could be imported into the cluster) and of course the actual queries executed in detail as described in the appendix.

3. Sizing

Through broad experience with a large number of customers the sizing for DB2 UDB for Linux database clusters follows a set of very well established rules. The sizing of business intelligence clusters has been improved and perfected since its beginnings in 1997 and all rules for existing clusters under UNIX® or Windows® operating systems apply naturally to Linux operating systems as well. Having a large number of customers running in the multiple terabyte class on a Linux distribution as well as executing a large number of benchmarks – published and internal engineering ones – offers assurance to customers that a proposed sizing and configuration will actually work.

Historically the development of business intelligence solutions like the deployment of applications like the FinTime has been a one-off project for each deployment. More recently, the introduction of Linux commodity clusters and the DB2 ICE architecture have strived to shorten the implementation and risk by developing a blueprint for the design and implementation of database clusters.

The DB2 development and technical support organization jointly developed the concept of the balanced configuration unit (BCU). The BCU defines an exact model of a configuration for a given business intelligence workload. Depending on the customer query and response time requirements this model can be adapted from the baseline for any sizing by simply multiplying the number of needed nodes. Additional BCU components such as extract, transform, and load (ETL) or administration can be added if needed and asked for by the customers – they are preconfigured and sized as well.

At the time of this whitepaper the Linux BCU defines two base configuration models. More configurations may be added in the future to adapt to changing hardware options on the market:

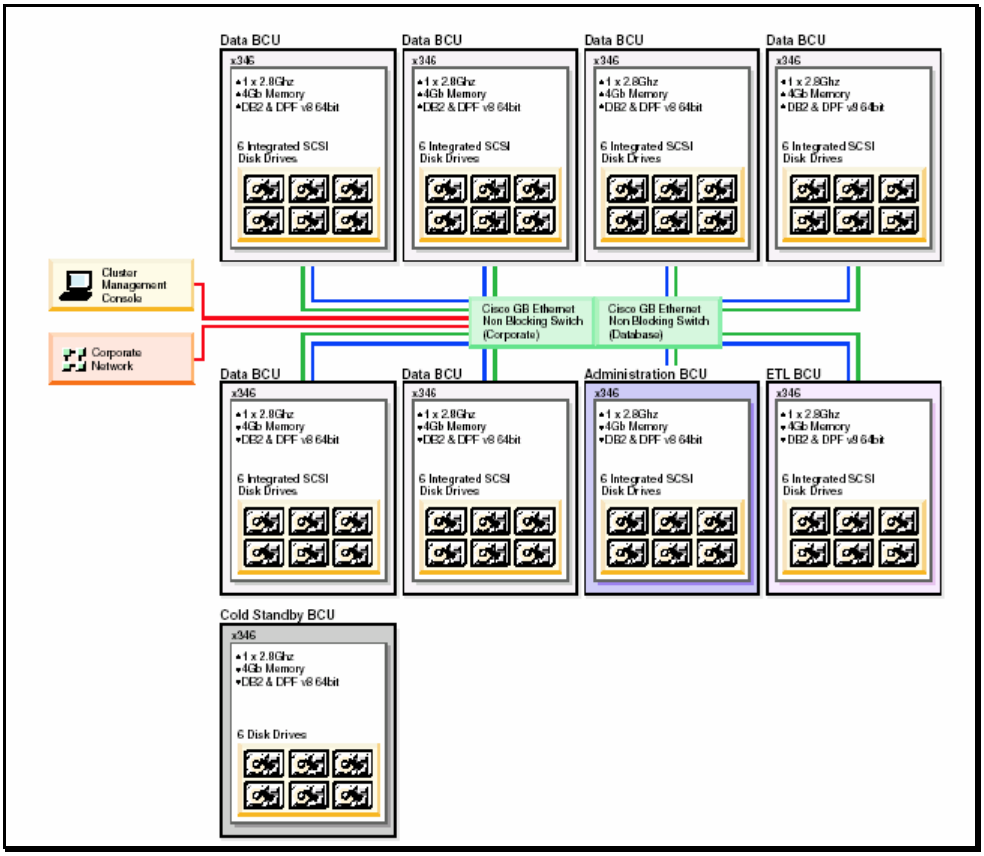
- *Small:* the “small” BCU configuration is optimized for very low entry point of costs as well as ease of implementation. Each server node operates with a single CPU and contains the local storage.
- *Large:* the “large” BCU configuration allows high availability and utilizes external fibre channel storage for high performance databases.

Small BCU

The small BCU is based around the concept of a database appliance of smallest possible denominator. The purpose can be a collection of data marts or a larger cluster that has not the needs of a highly available environment or a 24x7 availability. The selected platform is an IBM eServer xSeries® 346 server. The server is dual processor capable but is only equipped for the BCU with a single CPU and 4 GB physical memory. The server internally allows for up to 6 hard drives with 146 GB capacity each, enableing the configuration of between 80 GB and 100 GB raw data per node. This very simple concept scales excellent to a large number of nodes

The major disadvantage of the small BCU configuration is its lack of any additional availability beyond the server-associated ones like redundant power supplies or redundant network paths or RAID support for the disks. If a complete node fails there is no failover functionality designed into those systems. A

potential failover solution is a secondary cluster in a separate building that is running as hot standby: as DB2 UDB charges only for one additional CPU for a secondary standby cluster and the costs for the hardware are so low, it is more cost effective to build up a second cluster than to have the external storage required for failover added to the system.



The preceding diagram gives an impression of a small BCU configuration which has all options enabled. It is important to see that the cluster management console is actually nothing else than any workstation connection to the administrator BCU or simpler to the coordinator node of the cluster. Unlike typical high performance computing clusters such as Linux Beowulf clusters, DB2 UDB for Linux does not need an additional management or head node.

For small clusters Ethernet could be used for the cluster interconnect. However, businesses requiring larger clustered databases of 1 TB and above should use InfiniBand architecture, the ultimate clustering and grid interconnect. InfiniBand architecture provides superior price/performance solutions over Ethernet.

Large BCU

Compared to the small BCU configuration, the large BCU configuration is geared toward the highly available enterprise that has close to high availability needs and high performance data access requirements.

The selected server models are either the IBM xSeries or the IBM eServer 326. Both server models are configured with dual processors and 8 GB physical memory so having the same memory to CPU ratio as the small BCU

configuration. The IBM eServer 326 sever utilizes the AMD Opteron processor technology while the IBM eServer xSeries 346 server is equipped with Intel® Xeon® processors. The IBM eServer 326 is the preferred platform as it features a smaller foot print and its hyper-transport technology is well-proven with DB2 UDB deployments.

In contrast to the limited disk drive count in the small BCU configuration, the large BCU configuration utilizes 12 active disk drives per processor for a real optimum balanced performance between actual CPU performance and available disk I/O throughput. Additional hot spare drives are included so the actual count per processor is 14 disk drives. Further the I/O throughput needs to be enabled efficiently so Fibre Channel based storage is the right option for the solution. The IBM TotalStorage® DS4300 disk system is the backend of the large BCU configuration. It holds a maximum of 14 hard drives in its cabinet and together with an additional IBM EXP710 storage cabinet the needed 28 disks for the BCU are perfectly configured.

Voltaire InfiniBand Grid Interconnect infrastructure

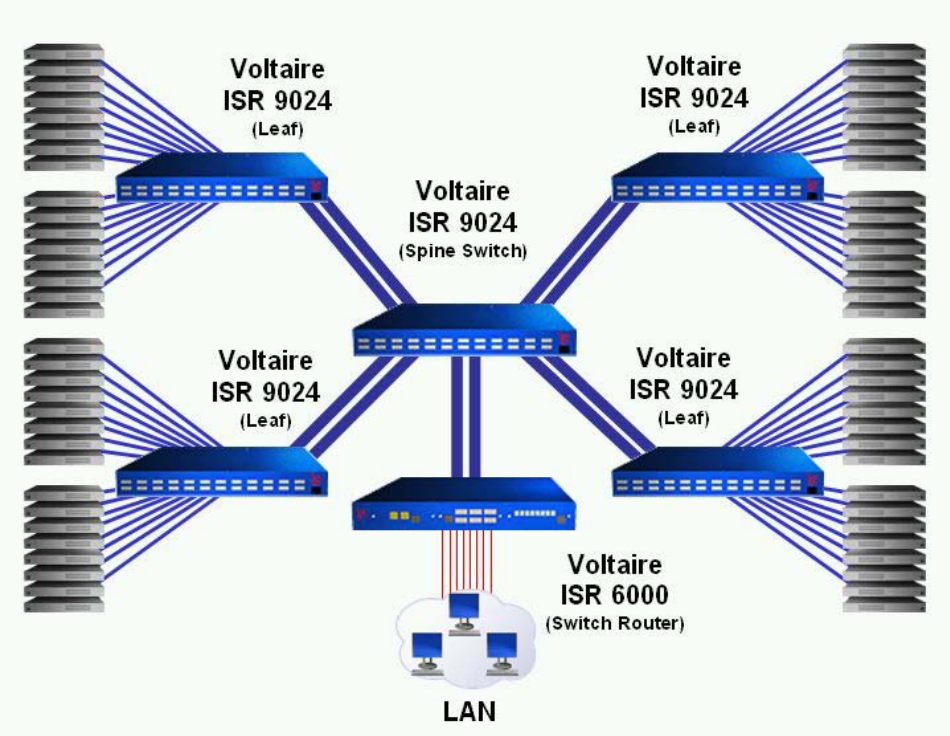
While for a very small cluster 1 GB Ethernet may be a good-enough solution as a cluster interconnect, the bandwidth requirements increase as the configuration grows larger. For larger clusters, 1 GB Ethernet becomes a bottleneck on the performance of the cluster. Aggregating both Ethernet ports on the servers doubles the port count on the GbE switches. In addition these switches must be non-blocking as they need to allow the throughput for the given effective communication levels.

Simpler models allow the connection speed of 1 GB/sec per port but are quickly limited with the overall switch throughput. High-end GbE switches like the CISCO Catalyst 6500 ones can get more expensive than the actual database servers. For a hypothetical cluster with 60 nodes and a given 50% network utilization at peak times, the cluster would need a switch with a backplane throughput of 30 GB/sec or in the case of a redundant load balancing environment 60 GB/sec.

In contrast to the Ethernet environment, an InfiniBand fabric allows 10 GB/sec per each adapter port (and the common adapter comes with two of those for redundancy). The additional costs of the InfiniBand adapter is quickly amortized through the performance and scalability that is achieved when using InfiniBand. Voltaire InfiniBand Grid Interconnect switches are also simple to manage and provide efficient management of the entire cluster. The Voltaire switches do not need additional configuration or special certified staff.

From a system administration perspective the InfiniBand fabric at the network level represents a standard cluster of TCP/IP devices interacting with standard network management tools. However, when it comes to database communication, the performance provided by the InfiniBand interconnect's low latency and fast response time make it clear that this is no ordinary network. In the case of the FinTime benchmark, the InfiniBand performance was most evident when returning queries with large result sets and during the load phases where the data from the four servers that generated the test data needed to be moved to each individual node.

The graphic below visualizes the 68 node cluster utilizing the Voltaire InfiniBand Grid Interconnect infrastructure. The five ISR 9024 switches connect the servers into a single 10 GB/sec fabric through an ISR 6000 Switch Router that provides the gateway to the corporate Ethernet environment and potential application servers.



In addition to the interconnect functionality of the cluster, the 10 GB/sec fabric allows convenient fast backups that can be directed to an additional backup server. It reduces the costs for the backup infrastructure and can shorten the backup windows. While a usual backup using fibre connection to a backup system is limited again to 2 GB/sec (assuming single path fibre connections) the InfiniBand based solution features a five times higher bandwidth. In this case the file based backup can be made to an additional backup storage system that can be a simple server with serial ATA disks on the database InfiniBand fabric and the final tape backup is then executed asynchronously from there saving the expensive fibre channel ports at each server and simplifying the backup process.

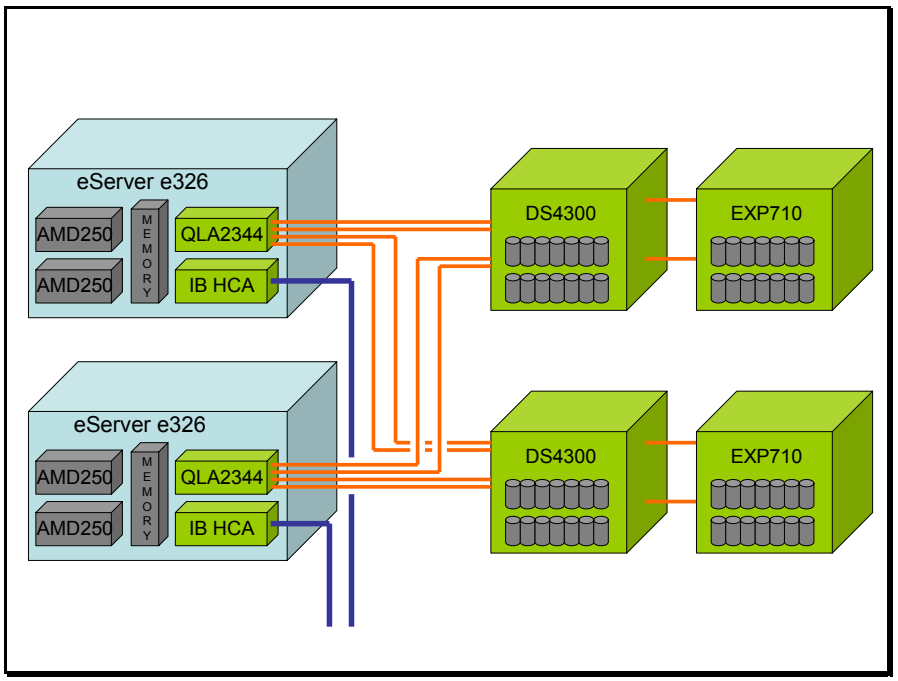
The benchmark configuration

After discussing the various generic hardware options the actual benchmark configuration needs to be implemented. For the FinTime benchmark the choice fell to the IBM eServer 326 with the AMD Opteron CPUs. The longer experience with its x86-64 architecture and its proven deployment in customer environments gave the final arguments for the decision. The eServer 326 also met the customer requirement for the smallest possible foot print with maximum performance. Following the BCU guidelines each server was equipped with 8 GB memory (4 GB per CPU) and a 4 channel QLogic QLA2344 fibre channel adapter. The server interconnect between the nodes is provided through the Voltaire InfiniBand Grid Interconnect solution that is well proven in customer environments and fully validated for DB2 UDB for Linux.

The storage configuration is also taken from the BCU guidelines, resulting in a single TotalStorage DS4300 storage controller for each server plus an IBM DS4000 EXP710 Storage Expansion Unit. The IBM TotalStorage DS4300 storage system allows a maximum of 4 fibre channel ports but has only an average throughput measured of about 300 MB/sec. This configuration allows

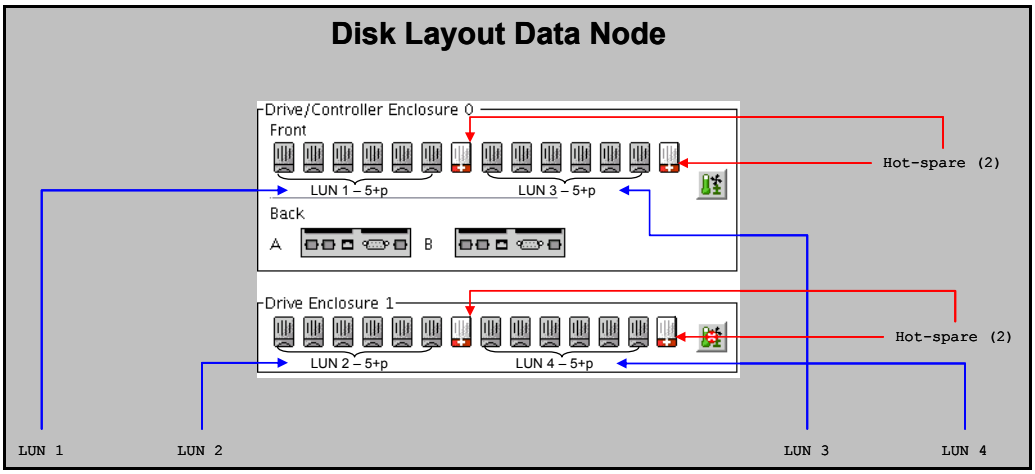
the maximum throughput channel through 2 fibre channel ports while the other ports connect to a second physical server for high availability that allows sustained throughput in the case of a failure of the primary server.

This leads to the following connectivity for each building block consisting of two servers and two storage servers (in orange). The Voltaire InfiniBand connection appears in the picture as well (in blue color).



Each server has a theoretical fibre channel throughput of 600 MB/sec. Various benchmarks have demonstrated that the server actually can operate at this throughput for a sustained period of time. Each of the DS4300 storage controllers can be active with one server at a time to deliver the full 300 MB/sec throughput per each storage system. The redundant connections between the DS4300 and the EXP710 offer additional reliability for the system.

Relying on the BCU specifications, the 28 hard drives per server are configured into four volumes of six drives plus one hot spare drive. The following diagram depicts the setup, including the distribution of the volumes between the DS4300 and the EXP710. The diagram comes directly from the DS4300 configuration utility.



This configuration provides the following available disk space for each server in the FinTime benchmark:

```
28 drives * 73.6GB       = 2060.8 GB
- 4 drives spare         = 1766.4 GB
- RAID 5 protection      = 1472 GB
- OS overhead 5%         = 1398.4 GB net space for data
```

While it can be argued that this kind of redundancy is over cautious it should be taken into account that a cluster like the one used in this benchmark has thousands of hard drives and with those numbers the probability greatly increases that one disk will fail at any given time. This probability is a simple statistical fact that has nothing to do with the quality of the disks or storage solution per se. In the actual benchmark several storage systems had bad drives. The benchmark team initially encountered more than one bad disk per day because of faulty connections or loose disks that were not seated properly. However, the system still functioned as designed and after the initial faults were corrected the system operated flawlessly. Historical evidence proves that this is normal but after several years of production the probability for failures grows and this is when the redundancy is critically necessary for a business

The Scale out

The benchmark requirement called for a 2 TB and a 16.5 TB configuration. Additional machines for data generation and ETL were needed. To satisfy these requirements the decision was made to have four additional server nodes as DB2 UDB coordinator nodes configured with slightly different storage layout (in the BCU terminology this is called an administrative BCU component). With those coordinator nodes outside of the actual data nodes, we needed to distribute 2 TB on the number of nodes defined in the BCU for “data nodes” and could simply ignore any additional nodes for administrative or ETL purposes. The BCU specification allows for a range of between 100 GB and 200 GB per CPU, which would result in a requirement of between 10 and 20 CPUs for 2 TB and 80 to 160 CPUs for the 16-TB layouts. Another aspect was the required storage to data ratio. While the 100 GB per CPU (or 200 GB per server node) would use only about 800 GB disk space, following common sizing rules of a 1:4 ratio between raw data and 200 GB would result in the requirement for a 1600 GB disk space--which would be already over the maximum that we have available. The decision was made to have *128 GB of raw data per CPU* which led to 16

CPUs in total or 8 server nodes equaling 4 pairs of the BCU configuration. While this is slightly oversized in storage now, the assumption was that the additional space may be needed for additional data sets or redundancy in multiple tests. However, at the end of the benchmark the first database design was so solid performing and stable that neither backups for system purposes (e.g. the danger of “burned up hardware”) nor backups for database specific reasons (e.g. schema design errors or any other implementation errors) were needed. So the space saved for this purposes was actually not needed at this time. In a customer environment this would be perfect for additional tests, staging areas and deployment space.

As mentioned before the connection between the nodes was implemented using a Voltaire InfiniBand Grid Interconnect using a socket based InfiniBand drivers. For implementing the switching fabric a number of 24-port Voltaire ISR 9024 InfiniBand switch routers were used and the Ethernet gateway functionality was provided by a Voltaire ISR 6000 InfiniBand switch router.

To keep the scaling factor identical to the 2-TB benchmark, the configuration for the 16.5-TB benchmark is sized with 64 data nodes with 128 partitions overall .

The requirement for the coordinator node and the additional three data generation nodes was driven by the need for the short time frame of the benchmark and the directive to keep the environment as close to an operational production system as possible.

The DB2 database partitioning strategy was to split each node into 2 database partitions. Two database partitions per node increased the CPU parallelism and memory addressability across the DB2 UDB environment.

Database Partition Layout at each Benchmark database point (2 TB and 16.5 TB)

In a production environment the 4 data generation nodes can then be used as ETL nodes, additional test nodes, or spare space for special purpose queries.

Software

The operating system was pre-selected by the customer and was RedHat Enterprise Linux 3 (RHEL3) implemented with Update level 3 as it was the latest available update level at the time of the benchmark. DB2 UDB is able to exploit the various features of RHEL3 but besides the function of RAW partitioning none of the advanced features were even needed to achieve the performance in the given time for the benchmark. Further, the choice was made to execute the benchmark with standard 32-bit operating system–even though the server hardware and DB2 software are fully supported in 64-bit mode. The RHEL3 hugemem kernel still allows full exploitation of the 8 GB of memory through the DB2 software. Another option would have been the using the regular x86-64 version of the same operating system with the 64-bit release of DB2 UDB for the x86-64 platform, but the customer preferred 32-bit at the time of the benchmark.

DB2 UDB does not need any customization for the Linux kernel, but some preset parameters simplify administrators' jobs in a benchmark and production environment. The /etc/sysctl.conf file controls the standard settings for the Linux kernel. While DB2 UDB can configure those parameters at start time, other software may subsequently change those parameters, so setting the parameters in /etc/sysctl.conf is a security measure that may subsequently save administrator time debugging performance regressions. The following is the actual content of the /etc/sysctl.conf file as used in the benchmark:

```
#The following lines should be added in for a 32-bit system:
# Added for 32-bit DB2
kernel.sem=”250    256000     32    1024”
kernel.msgmni=1024
# maximum allowed size for shared memory segments in 32bit
is obviously 2^32-1
kernel.shmmax=4294967295
# overall memory for shared memory is given in bytes on the
system
kernel.shmall= 8589934591
#end additions for DB2
```

The kernel.sem parameter sets the number of the available kernel semaphores and its behavior. The kernel.msgmniparameter sets the number of available message queues inside the kernel. The kernel.shmmax and kernel.shmall parameters configure the size of the available shared memory overall and the maximum allocable size per segment.

No other parameters needed to be tuned for the operation of the DB2 database environment – nor were any drivers changed from the defaults supplied with RHEL3 Update 3.

For operational purposes an installation of the fibre channel management utilities is an option that was not needed for the benchmark as all systems were configured identically through the /etc/modules.conf file that controls all driver behavior. For the purpose of the environment the failover behavior and the redundant path options were configured (for more details of the broad options of the QLogic QLA2344 please read the related specific literature).

As various tools and the DB2 Query Patroller V8 require a Java® environment, the IBM Java Development Kit 1.4.1 service level 2 was installed on all systems. This is the default version of the IBM SDK delivered with DB2 UDB V8.2.

The DB2 UDB release implemented for the benchmark was DB2 Universal Database Version 8.2 as it comes out of the box. No additional fixes or changes were applied to it for the execution and delivery of the benchmark. In addition, DB2 Query Patroller Version 8 was installed as part of the benchmark to test the impact of the DB2 Query Patroller to the various tested specific query

environments. While these tests were not necessary requested by the customer, the impact of using Query Patroller or not was an excellent exercise on the given infrastructure and customer requirements.

For convenience the open source sysstat package was added to the setup to have more detailed views on the various components in the system. The sysstat package contains extended functions for vmstat, iostat and other commands. While it does not impact the operation of the system it provides administrators with a simple health check during the operation of the benchmark or production environment.

The installation of additional system management tools like Tivoli Enterprise Console® would simplify health monitoring at a system level, but the limited time for the benchmark did not allow for installing and configuring these tools on the cluster.

4. Physical Setup

The physical setup of a cluster in the given size is a massive endeavor. Starting from the actual assembly of the servers and configuration with the requested options, to a full cabling and system burn in, the amount of work can reach into hours per node.

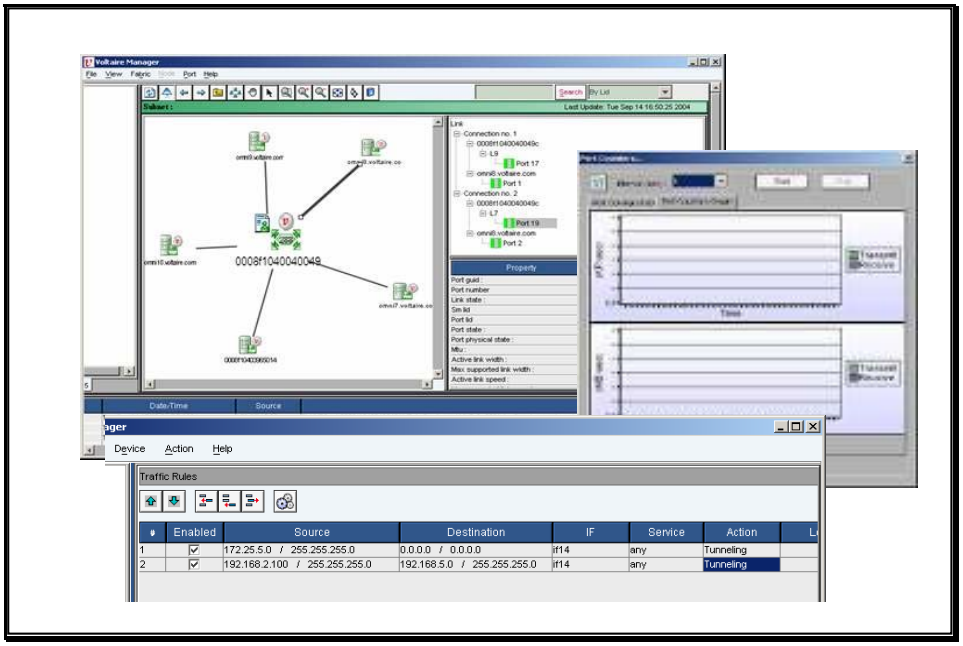
The delivery as an IBM eServer Cluster 1350 Linux cluster solution that comes fully configured, cabled and physically tested allowed a very rapid deployment for the benchmark at the IBM benchmark center in Poughkeepsie. Deviating from the normal IBM process, this specific cluster was not pre-installed with the operating system and database that comes with the IBM DB2 Integrated Cluster Environment (ICE), so additional setup was necessary.

The core setup for the eServer Cluster 1350 required minimal time for the connection of power to the ten storage racks and the two server racks was all prepared through power distribution units inside the rack to an easy single or dual plug setup and the connections between the storage servers and the database servers.

The simplicity of the BCU specification again proved its value as the cluster does not require any setup effort for the switching gear in a SAN fabric. While the usual opinion is that a SAN fabric simplifies the storage management, that is not necessarily true for large-scale environments like the business intelligence cluster used in this benchmark. The clearly defined I/O requirements are fixed for each node and do not change. If the storage capacity is exceeded in production the need for additional computing power will also be required--and it is easier to add additional BCU building blocks than to reconfigure 100+ TB storage and move disks around to fill the needs. Keep in mind that the costs for the people to manage this are higher than the costs for the additional BCU building blocks. Of course, if a customer desires it, it is possible to build a system utilizing a SAN fabric with the same characteristics of the BCU building block.

The clearly structured BCU blocks simply need direct cabling of eight cables between the two server nodes and their respective two storage servers. From the operator and user view this concept is more like an appliance than they would normally encounter in large-scale clusters and this reduces the possibility of setup failure dramatically. In this setup, all nodes came up except one on the first attempt–and that node had problems that were quickly identified and corrected by changing a single cable.

Connecting the cluster to the network was a simple matter of connecting four Ethernet connections into the ISR6000 InfiniBand Switch Router. The ISR6000 behaves as a standard L2 Ethernet bridge, and no cabling or extra configuration was required. For administrative purposes (and normal IBM procedure) additional Ethernet was configured in the benchmark center but was not utilized during the benchmark. Each IBM DS4300 storage server had an Ethernet connection as well which allowed for a quick configuration of all storage systems with a standardized layout configuration file. Once again, the simplicity and robustness of the DB2 ICE solution saved time and prevented error prone configuration sessions.



VoltaireVision InfiniBand Fabric Management Software, Voltaire's grid management suite, was instrumental in simplifying the cluster installation process. VoltaireVision is a Java based management platform embedded in the Voltaire switches that can be accessed through a Web based GUI. The management suite automatically discovered all server and networking resources, and once the servers were connected to the switching infrastructure it verified the correctness of the network topology. While the cluster was being built VoltaireVision generated alerts as soon as a cable was connected incorrectly or damaged and pinpointed the physical location of the failure. These features prevented delays during setup and production ready operation.

Individual cluster setup versus integrated cluster management tooling

With the power and network cabling set up, the installation of the operating system could begin. Before the actual setup execution several discussions were held about the implementation of specific cluster management software on top of the database cluster.

Cluster management kits like IBM Cluster Systems Management (CSM) for Linux and xCat (Extreme Cluster Administration Toolkit) allow a fast re-installation ("wiping") of nodes as is appropriate in a stateless high performance computing environment where a master node holds all control over the cluster and contains all state information. For a database cluster, however, it would be horrendous to lose all node specific state information simply to upgrade a single package through a central installation utility of any kind; Unlike database clusters, high performance computing (HPC) clusters do not have bound raw devices that use specific local IDs to identify the right volume as just one example of state information that may be required at each node.

Professional Linux deployments in corporations use commercially supported Linux distributions like RedHat or SUSE LINUX – as we also used for the benchmark. But instead of using out-of-the-box installations, corporate customers deploy standardized images that have pre-customized environment configurations like directory services, network settings or even specific security fixes. These images simplify maintenance across a corporate network – and the

Linux operating system is perfectly suited as the same kernel scales from a workstation to a large SMP server. These installations are performed with a kickstart server – in the case of a RedHat environment – or with a YaST server – in the case of a Novell SUSE LINUX environment.

The decision was made to work along the same lines for the setup of the benchmark cluster. First a single node was created including:

- the kernel settings in the sysctl.conf file
- the network configuration settings
- the storage settings
- the InfiniBand drivers and configuration settings
- the users and groups at an operating system level (in a customer environment this would be normally coming from an LDAP, NIS or other central managed system of course)
- RSH and SSH functionality and necessary keys for all needs
- an NFS share for the home directory of the instance as a mount point to all nodes
- all necessary database code including the Java Runtime Environment, DB2 UDB Enterprise Server Edition, and DB2 Query Patroller
- a DB2 instance created with all needed node settings for the overall database

After having the node installed and tested a backup was taken of the system. A more elegant method would use Tivoli® Intelligent Orchestrator and Tivoli Provisioner but as we needed only a single image for a one time deployment we chose to deploy the backup through a simple restore and adoption of the server specific parameters through a script. See the appendix for the complete script used in the setup.

For the specific setup with scripts the server setup was completed within minutes per node as each server needed to be booted once and the imaging script needed to be instantiated[3]. The base image in the used version was less than 3 in size. Software on all nodes was completely set up within a few hours after the physical correctness of the nodes had been established.

Linux tuning

When database experts start working on a Linux operating system for the first time, the question is always asked about which options need to be tuned in the operating system, how to compile the Linux kernel for DB2 UDB, and how to configure DB2 UDB for Linux with Linux-specific settings. The fact is that DB2 UDB for Linux is already optimized out of the box for any kind of Linux deployment ranging for a small DB2 UDB Express to a large scaling cluster like the one used in the benchmark. Besides the previously mentioned five kernel parameters – and even those DB2 UDB tries to set itself to ensure proper operation – no operating system tuning needs to occur. The DB2 support organization only supports the binary kernel provided by the Linux distributors as those are fully validated against DB2 UDB under high stress workload conditions.

[3] A full network boot install would also have been possible, but the fact that the system was not pre-installed made it a prudent task to check each server boot process once for system errors. The complete DB2 ICE setup would not have that need obviously as it is delivered turn key through IBM.

That said, it does not mean that there are no options to further optimize the function of DB2 UDB for Linux. DB2 UDB offers a broad variety of additional advanced exploitations of the Linux operating system that are known from other operating systems like AIX®. Those options can be switched on using the db2set commands and are described in the DB2 documentation. Because of the time constraints in the benchmark there were not many options to test the available features but the benchmark itself has proven that a standard DB2 UDB performs excellent out of the box even without these advanced features. Here a short list of additional options:

- enabling I/O performance features like direct I/O, vectored I/O or asynchronous I/O
- processor affinity
- large page size support

More details can be found at ftp://ftp.software.ibm.com/software/data/db2/linux/db2stingerlinux.pdf .

DB2 UDB setup

After the installation of the software image (including the DB2 instance) only the specific mounting of the raw storage device needed to be done with another simple script and the database was ready for either 2-TB or 16.5-TB operation. The simple control of a DB2 UDB cluster through its db2nodes.cfg file first allowed the operation of the cluster with 8 data nodes (and the previously mentioned single coordinator). Then, by simply adding the additional nodes to the db2nodes.cfg file, the cluster was extended to all required nodes for the 16.5-TB tests. The standardized prepared volumes could just be added as table spaces to the cluster and then the cluster needs just a single redistribution to bring the cluster up to the full node number. To comply with the requirements of the benchmark, however, the process of loading the cluster was executed independently for the 2-TB and 16.5-TB loads, so the DB2 solution could not even utilize this convenient feature.

Overall the installation of the cluster, including the configuration and connection of storage and physical burn-in of the system, was executed by a team of three engineers and two technicians within three days from dock ready delivery onward.

That time includes complete validation of the existing hardware environment, setup of storage. and delivery up to the level of the running 16.5-TB instance. In the case of a regular delivered DB2 ICE solution the setup for the operating system and the DB2 software as well as the configuration of the cluster would have been done through IBM before system delivery, which would have saved three days.

5. Benchmark Operation and Result

Every benchmark follows a typical modus operandi. First, raw data needs to be generated following very well defined statistical rules so the data reflects realistic real life behaviour. Second, the database (including all table spaces) is created and the previously generated data is loaded. Finally, the actual database queries are executed. During each of the phases data is collected that provides further information about the overall performance of the system.

Data generation

The benchmark kit contains two data generation programs – one for tick data and one for market data generation. The program directly generates raw data for the import into the database. Following is a short example of the tick data base information consisting of the ID, trading exchange, description, industry sector and the operating currency.

```
0|LN|'Financial security number: 0'|FINANCIAL|USD
1|NY|'Financial security number: 1'|ENTERTAINMENT|JPY
2|NY|'Financial security number: 2'|ENTERTAINMENT|USD
3|TK|'Financial security number: 3'|BANKING|USD
4|O|'Financial security number: 4'|FINANCIAL|DEM
5|TK|'Financial security number: 5'|SOFTWARE|FFR
6|NY|'Financial security number: 6'|CHEMICALS|GBP
7|NY|'Financial security number: 7'|SOFTWARE|JPY
8|NY|'Financial security number: 8'|MEDICAL|DEM
9|NY|'Financial security number: 9'|CONSTRUCTION|DEM
```

All of the other raw data is generated in the same way and is stored in what are called staging files.

Database creation

The actual database creation, including the table space generation, follows. Here is an example for the market data table spaces for the 2-TB benchmark:

```
create tablespace ts_mkt in db_group pagesize 8k managed by database
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (4)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (5)
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (6)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (7)
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (8)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (9)
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (10)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (11)
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (12)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (13)
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (14)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (15)
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (16)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (17)
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088) on dbpartitionnum (18)
```

```
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088) on dbpartitionnum (19)
 extentsize 8
 prefetchsize 64
 bufferpool bp8k
;


create tablespace ts_mkt in db_group pagesize 8k managed by database
 using ( device '/dev/raw/raw1' 2001088,
         device '/dev/raw/raw2' 2001088)
               on dbpartitionnum (4, 6, 8, 10, 12, 14, 16, 18)
 using ( device '/dev/raw/raw3' 2001088,
         device '/dev/raw/raw4' 2001088)
               on dbpartitionnum (5, 7, 9, 11, 13, 15, 17, 19)
extentsize 8
prefetchsize 64
bufferpool bp8k
;
```

The difference between the 2-TB and the 16.5-TB benchmark is just the number of data partitions created. For the 16.5-TB benchmark the space is simply extended to have enough capacity for the larger amount of raw data. In the case of an operational situation where the database is outgrowing the existing table spaces, additional containers can be added or – to ensure the availability of enough computing power – more nodes can be added to extend the existing cluster with disks, CPU and memory. At this point we can create the actual tables, for example, the historic market data table:

```
create table hist_base
( id integer,
  ex char(3),
  descr varchar (256),
  sic char(15),
  spr char(4),
  cu char(5),
  createdate date
)
partitioning key (id)
in ts_mkt index in ts_mkt_ix
;
```

DB2 UDB implicitly distributes the table across all of the nodes. The only reference to the cluster is the identification of the partitioning key with the ID column. At this point the table has been created across the complete cluster.

The same approach is applied to all of the other table spaces and tables. In contrast to other databases, the DB2 ICE cluster is able to create every single table space and table in a fully parallel operation, so the size of the cluster has no impact on the execution of the creation process. The 20-TB table spaces – spread across the proportionally larger number of nodes – take the same amount as the 2-TB table spaces and table creation process.

For both benchmarks the following data volumes were loaded:

Program	Securities	Tick per day	Days	Raw size	Number of rows
histgen	1,000,00		4,00	250 GB	4,000,000,000
tickgen	25,00	5,000	365	2 TB	45,063,678,535
tickgen	50,000	3,750	1,825	16.5 TB	339,803,785,565

The amount of tick data is obviously larger than the consolidated historic market data and has more impact on the overall size of the cluster.

The first actual test is the execution of the market data queries (see queries detail in the appendix of the document). The first run is a simple execution of the

queries with a single user. The second run represents an execution of 10 users in parallel. Next is a benchmark run utilizing the DB2 Query Patroller to ensure best possible execution order of the queries and last follows the execution of the queries while loading additional data into the ticker tables. As expected the performance of the queries significantly differed between the 2-TB and 16.5-TB runs as the actual market data tables were static and no more data was added. But the performance of DB2 UDB was evident with linear scalability and reduced response time proportional to the amount of physical machines available. As the data was spread out between more nodes a typical spread rumor was put to rest: competitors of the DB2 database tout that adding physical resources to a DB2 database cluster and not having more data loaded have no impact on the performance and try to make a case for the “inferiority of the logical shared nothing architecture”. This benchmark proved them completely wrong (again – like in many other occasions) but this time the data were exactly the same and the number of nodes was added. In other customer environments this can be done exactly the same way – simply adding nodes to the DB2 ICE cluster and redistributing the existing data and the response time can be reduced to any changed requirements from the actual end users. DB2 ICE is scaling linear and the numbers proof it.

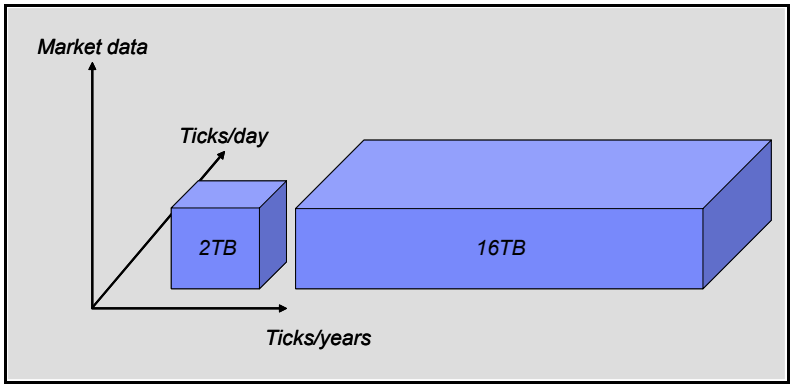
Number of nodes	Single thread	10 users	10 users with QP	10 users with QP while loading
8 nodes	48 min	47 min	41 min	43 min
64 nodes	15 min	6 min	6 min	6 min

The DB2 Query Patroller could show a little bit of its performance and runs with more users would increase the difference but were not required as part of the benchmark requirements.

Servers	Query	Single thread	10 users	10 users with QP	10 users + Load	10 users + Load with QP
8 nodes	Q1	2	3	3	3	3
	Q2	15	38	21	40	23
	Q3	4	38	12	41	19
	Q4	1	4	1	122	8
	Q5	9	26	18	30	20
	Q6	2	15	4	17	4
	Q7	6	32	8	11	12
	Q8	2	11	5	5	5
	Q9	1	2	2	2	2
	Q10	245	2468	1345	2743	1439
64 nodes	Q1	4	11	6	11	12
	Q2	14	36	27	46	48
	Q3	3	7	9	5	6
	Q4	2	9	9	7	6
	Q5	8	20	18	21	21
	Q6	2	5	4	4	4
	Q7	11	86	80	76	73
	Q8	6	32	60	38	44
	Q9	1	4	2	3	6
	Q10	39	170	116	171	166

The behavior of query 10 was an interesting aspect of the overall benchmark run. This query was not part of the original benchmark but was requested by the customer as an additional requirement. During the benchmark, the team was confused by the length of time it took to process the query and the huge result set (in the scale of multiple gigabytes). During the results presentation the customer confirmed that they made a mistake in the parameters provided to the team and had not intended to request that extreme data set. From the database perspective, it is impressive that DB2 UDB was still able to compute the massive result without any issues. The high bandwidth fabric helped to collect the results and to consolidate those results to a single answer as it is visible in the direct comparison. The conclusion from this interesting case is that DB2 UDB clearly can scale to even the most complex of queries and the addition of resources to handle complex queries results in excellent utilization of the infrastructure. The scalability of DB2 UDB for the historical market data was linear with factor 8.

The next execution run was the actual tick data queries. In this case 50 users at high load were required by the customer. The results were disappointing at the beginning as only a limited scalability from factor 6 was measured. Detailed analysis of the loaded data amount, however, showed that the amount of data added was not linear itself, but was instead slightly shifted towards an imbalance that generated higher operational costs in I/O and CPU. The amount of tick data per day actually increased by factor 1.5 from 120 million to 180 million trade operations recorded instead of being linear so that all other dimensions of the database were scaled in proportion to the data.



Again the first run was executed with a single user, then a run with 50 users and last a run with 50 users and a parallel load operation happening.

Number of nodes	Single thread	50 users with QP	50 users with QP and parallel load
8 nodes	103 min	84 min	95 min
64 nodes	35 min	13 min	15 min

The single thread run lost some momentum in the 16.5-TB configuration as it needed to load every row of data from the disc independently and no synergies – as normally would be expected in a database – were achieved; buffer pools were empty when the queries were run, materialized query tables could not be applied… The 50 users run with DB2 Query Patroller showed the excellent

scalability again of the DB2 ICE cluster and proved the soundness of the BCU architectural specification.

The tick data queries showed a very stable performance result scaling across the bandwidth and led to the mentioned scalability factor 6. The benchmark team was sure that a more detailed look at the queries would have allowed further improvements but the time required for the investigation was not available during this benchmark.

6. Conclusion

The FinTime benchmark is a highly demanding benchmark environment that does not allow a broad number of options to tune and optimize in its original state. Naturally, like all benchmarks, it would be a candidate for improvements targeted specifically for its operation--but with its clear focus on a specific business problem, it is a perfect candidate to show that the DB2 Integrated Cluster Environment (DB2 ICE) allows rapid deployment in any scale factor for database requirements in the financial markets.

While this benchmark did not provide the testing environment to demonstrate the more OLTP-oriented features of DB2 UDB, such high availability disaster recovery, the benchmark clearly showed the key features of DB2 UDB for Linux:

- Multidimensional clustering
- Materialized query tables
- UNION ALL views

These features make DB2 UDB the superior database for business intelligence solutions.

DB2 ICE was deployed within days and would be available to any other client in the same amount of time. The balanced confugration unit (BCU) concept simplifies the planning phase of the database cluster and provides a solid base for additional functions. Its ease of use and tested configurations take the risk factor out of business intelligence when the project is defined in its early stages.

Acknowledgements

I would like to thank Daniel Hancock and James Sun from the IBM DB2 team and Lerone Latouche and Magnus Larsson from the IBM Poughkeepsie benchmark center for their focus on excellence during the planning and execution of this benchmark. Further, I would like to thank Lior Ofer and Asaf Somekh from Voltaire for help during the implementation of the Voltaire InfiniBand fabric that made our life so much easier, and Martin Schlegel for his insights and help in cluster automated setup. Lastly, I would like to thank Alexandra Bialek, who let me spend many weeks on this project.

7. Appendix

Used Scripts

Clone script for server installation

The clone script allows the automatic installation of a server based on a given image with additional software installed. It simply takes the server node number as input as well as the source where the image is copied from and then allows as option to setup the setup completely automatically (-n option) and to copy all needed scripts if booted of a remote device (-c option).

```
clone <server to be cloned> <ip address for the source> -n -c
```

```
#!/bin/bash

CMDbasename=" basename "
CMDdirname=" dirname "
CMDpwd=" pwd "
SCscriptName=`$CMDbasename $0`
SCscriptDir=`$CMDdirname $0`
SCscriptPath="`cd \"${SCscriptDir}\" 2>/dev/null && $CMDpwd -P || echo
\"${SCscriptDir}\"`"

if [ "$4" == "-c" ]; then
        echo "Copying files to /root/ ..."
        cp ${SCscriptPath}/* /root/ || exit 1

        echo "Changing permissions ..."
        chmod 600 /root/id_rsa || exit 1
        chmod 775 /root/clone || exit 1

        echo "Executing clone script on /root/ ..."
        /root/clone $1 $2 $3
        exit $?
fi

nodeNr=$1
installSrv=$2

subExt="255.255.255.0"
subInt=${subExt}

nodeNr=$[ nodeNr + 70 - 1 ]


hostExt="lcan${nodeNr}"
ipExt="129.40.101.${1}"
ipExtSub="255.255.255.0"
hostInt="lcai${nodeNr}"
ipInt="129.40.102.${1}"
ipIntSub=${ipExtSub}

instIP="129.40.101.${1}"
instSub="255.255.255.0"
instBroad="129.40.101.255"

bckDirSrv="/bck"
bckMbrFile="citadel.mbr.bin"
bckRootFile="citadel.root.tar.gz"
mntPoint="/mnt/root"
sshKey="id_rsa"
```

```
if [ "$3" == "-n" ]; then
        echo "Configuring private netwok on eth0 ${instIP} netmask ${instSub} ..."
        ifconfig eth0 ${instIP} netmask ${instSub} broadcast ${instBroad} || exit 1
fi
ping -c 30 $2

echo "Creating partition table ..."
ssh -i ${SCscriptPath}/${sshKey} root@${installSrv} "cat ${bckDirSrv}/${bckMbrFile}" |
(dd of=/dev/sda) || exit 1
echo -n "scsi remove-single-device 0 0 0 0" > /proc/scsi/scsi
echo -n "scsi add-single-device 0 0 0 0" > /proc/scsi/scsi

echo "Creating filesystem on /dev/sda1 ..."
mkfs.ext3 "/dev/sda1" || exit 1
tune2fs -L "/" /dev/sda1 || exit 1

#echo "Creating swap partition on /dev/sda2 ..."
#mkswap "/dev/sda2" || exit 1

echo "Creating mount point ${mntPoint} ..."
mkdir -v ${mntPoint} || exit 1

echo "Mounting root partition /dev/sda1 ..."
mount "/dev/sda1" ${mntPoint} || exit 1

echo "Copying root filesystem over ..."
ssh -i ${SCscriptPath}/${sshKey} root@${installSrv} "cat ${bckDirSrv}/${bckRootFile}" |
(cd "${mntPoint}" && tar -xpzf -)

echo "Changing network configuration to use hostname ${hostExt} ext. ${ipExt} /
${ipExtSub} and int. ${ipInt} / ${ipIntSub} ..."
cp "${mntPoint}/etc/sysconfig/network-scripts/ifcfg-ipoib0"
"${mntPoint}/etc/sysconfig/network-scripts/ifcfg-eth0"
"${mntPoint}/etc/sysconfig/network" "/tmp" || exit 1
sedcmd=`echo "s/^HOSTNAME.*/HOSTNAME=${hostExt}/g"`
cat "/tmp/network" | sed -e "${sedcmd}" > "${mntPoint}/etc/sysconfig/network" || exit 1
hwaddrcmd=`ifconfig eth0 | grep HWaddr | sed -e 's/.*HWaddr[ \t]*\([^ ]*\)/\1/g'`
sedcmd=`echo "s/^IPADDR.*/IPADDR=${ipExt}/g"`
sedcmd2=`echo "s/^HWADDR.*/HWADDR=${hwaddrcmd}/g"`
cat "/tmp/ifcfg-eth0" | sed -e "${sedcmd}" -e "${sedcmd2}" >
"${mntPoint}/etc/sysconfig/network-scripts/ifcfg-eth0" || exit 1
sedcmd=`echo "s/^IPADDR.*/IPADDR=${ipInt}/g"`
cat "/tmp/ifcfg-ipoib0" | sed -e ${sedcmd} > "${mntPoint}/etc/sysconfig/network-
scripts/ifcfg-ipoib0" || exit 1

echo "Reinstalling grub boot manager on /dev/sda and rebooting ..."
chroot ${mntPoint} "/sbin/grub-install" "/dev/sda" && reboot
#END . Martin Schlegel, Toronto Lab
```

Root command for all server nodes

In theory an additional execution of a command across all nodes is not often necessary but conveniently was implemented as a small script as well. The script expected simply the given “ssh” acknowledgement between the various nodes (as it was automatically setup as part of the installation itself).

```
cmdall 129.40.102. 1 68 “ssh {} reboot”
```

```
#!/bin/bash

if [ "${CLUSTER}" == ""  ]; then
        pref=$1
        from=$2
        to=$3
        to=$[$to + 1]
        cmd=$4
        for ((i=$from ; i < $to ; i=i+1 )); do
                j=`printf "%02d" $i`
                sedcmd=`echo "s/{}/${pref}$j/g"`
                ncmd=`echo $cmd | sed -e $sedcmd`
                msg="Executing command \"${ncmd}\""
```

```
                echo $msg
                echo $msg | sed -e 's/./-/g'
                $ncmd
                echo -e "=========================\n"
        done

else
        for h in ${CLUSTER}; do
        sedcmd=`echo "s/{}/${h}/g"`
        ncmd=`echo $1 | sed -e $sedcmd`
        msg="Executing command \"${ncmd}\""
        echo $msg
        echo $msg | sed -e 's/./-/g'
        #echo "h=\"$h\", sedcmd=\"${sedcmd}\", ncmd=\"${ncmd}\""
        $ncmd
        echo -e "=========================\n"
        done
fi

#END. Martin Schlegel, IBM Toronto Lab
```

Benchmark queries

Market data Query #1

```
-- Get the closing price of a set of 10 stocks for a 10-year period and group
-- into weekly, monthly and yearly aggregates.
-- For each aggregate period determine the low, high and average closing price
-- value. The output should be sorted by id and trade date

SELECT id, year(tradedate) as year,
month(tradedate) as month,
week(tradedate) as week,
max(closeprice) as max,
min(closeprice) as min,
avg(closeprice) as avg
FROM
hist_price2
WHERE tradedate between '2006-09-26' and '2016-09-26'
--WHERE tradedate between '2010-02-12' and '2020-02-12'
AND id in (select id from mkt_q1_id1)
--AND id in (select id from mkt_q1_id5)
group by rollup ((id,year(tradedate)),
(id,month(tradedate)),
(id,week(tradedate)))
order by id, year(tradedate),
month(tradedate),
week(tradedate)
with ur
;
```

Market data query #2

```
-- Adjust all prices and volumes (prices are multiplied by the split factor
-- and volumes are divided by the split factor) for a set of 1000 stocks
-- to reflect the split events during a specified 300 day period,
-- assuming that events occur before the first trade of the split date.
-- These are called split-adjusted prices and volumes.

SELECT a.id,
tradedate,
value(exp(sum(ln(value(b.splitfactor,1)))),1) adj_split_factor,
closeprice*value(exp(sum(ln(value(b.splitfactor,1)))),1) adj_price,
volume / value(exp(sum(ln(value(b.splitfactor,1)))),1) adj_volume,
closeprice,
volume
FROM
 hist_price2 a left outer join hist_split2 b
on a.id=b.id
AND a.tradedate<b.splitdate
```

```
WHERE tradedate between '2008-05-14' and '2009-03-10'
and a.id in (select id from mkt_q2_id1)
GROUP BY a.id, tradedate, closeprice, volume
with ur
;
```

Market data query #3

```
--For each stock in a specified list of 1000 stocks, find the differences
--between the daily high and daily low on the day of each split event
--during a specified period.

SELECT a.id, tradedate,
lowprice, highprice
FROM
hist_price2 a,
hist_split2 b
WHERE  a.id=b.id
AND a.tradedate=b.splitdate
and a.tradedate >= '2011-09-12'
and a.id in (select id from mkt_q3_id1)
order by a.id
with ur
;
```

Market data query #4

```
--Calculate the value of the S&P500 and Russell 2000 index for a specified day
--using unadjusted prices and the index composition of the 2 indexes
--(see appendix for spec) on the specified day

SELECT avg(closeprice) SP5_close_price
FROM
hist_price2 a
WHERE tradedate='2010-12-06'
--WHERE tradedate='2005-07-01'
AND a.id in (select id from INDEXCOMPOSITION where indexid=1)
;
SELECT avg(closeprice) R2000_close_price
FROM
hist_price2 a
WHERE tradedate='2010-12-06'
--WHERE tradedate='2005-07-01'
AND a.id in (select id from INDEXCOMPOSITION where indexid=2)
with ur
;
```

Market data query #5

```
--Find the 21-day and 5-day moving average price for a specified list
--of 1000 stocks during a 6-month period. (Use split adjusted prices)


WITH
splitadj (id, tradedate, adjprice, adjvolume) as (SELECT a.id,
tradedate, closeprice*value(exp(sum(ln(value(b.splitfactor,1)))),1) adj_price,
volume / value(exp(sum(ln(value(b.splitfactor,1)))),1) adj_volume
FROM
 hist_price2 a left outer join hist_split2 b on a.id=b.id
 AND a.tradedate<b.splitdate
WHERE a.id in (select id from mkt_q5_id1)
AND tradedate between '2010-02-02' and '2010-08-02'
GROUP BY a.id, tradedate, closeprice, volume)
SELECT
id,
tradedate,
avg(adjprice) OVER (PARTITION BY id ORDER BY tradedate asc ROWS between 21 prece
ding AND current row) day21,
avg(adjprice) OVER (PARTITION BY id ORDER BY tradedate asc ROWS between 5 preced
ing AND current row) day5
FROM
splitadj
with ur
;
```

Market data query #6

```
--(Based on the previous query) Find the points (specific days) when the
-- 5-month moving average intersects the 21-day moving average for
-- these stocks. The output is to be sorted by id and date.


WITH
splitadj (id, tradedate, adjprice, adjvolume) as (SELECT a.id,
tradedate, closeprice*value(exp(sum(ln(value(b.splitfactor,1)))),1) adj_price,
volume / value(exp(sum(ln(value(b.splitfactor,1)))),1) adj_volume
FROM
 hist_price2 a left outer join hist_split2 b on a.id=b.id
 and a.tradedate<b.splitdate
WHERE
tradedate between '2009-11-23' and '2010-04-23'
and a.id in (select id from mkt_q6_id1)
GROUP BY a.id, tradedate, closeprice, volume),

mov21_5 (id, tradedate, day21, day5)
as (SELECT id, tradedate, avg(adjprice) over (PARTITION BY id order by tradedate
 asc ROWS between 21 preceding and current row) day21,
avg(adjprice) over (PARTITION BY id order by tradedate asc ROWS between 5 preced
ing and current row) day5
FROM
splitadj),

mov21_5cross (id, tradedate, day21prev, day5prev,day21, day5)
as
(SELECT id, tradedate, avg(day21) over (PARTITION BY id order by tradedate rows
between 2 preceding and 1 preceding),
avg(day5) over (PARTITION BY id order by tradedate rows between 2 preceding and
1 preceding),
day21,
day5
FROM mov21_5)

SELECT  *
FROM  mov21_5cross
WHERE sign(day21-day5)*sign(day21prev-day5prev) < 0
with ur
;
```

Market data query #7

```
-- Market Query : Q7
-- Determine the value of $100,000 now if 1 year ago it was invested equally
-- in 10 specified stocks (i.e. allocation for each stock is $10,000).
-- The trading strategy is: When the 20-day moving average crosses over
-- the 5-month moving average the complete allocation for that stock is
-- invested and when the 20-day moving average crosses below the 5-month
-- moving average the entire position is sold. The trades happen on the closing
-- price of the trading day.

-------------------------------------------------------------------------------
--
--  Temp Table - Stock Moving Averages (20-day, 5-month)
--
-------------------------------------------------------------------------------


DECLARE GLOBAL TEMPORARY TABLE session.moving_average
(
  id            INT             NOT NULL,
  tradedate     DATE            NOT NULL,
  day_seq       INT             NOT NULL,
  closeprice    REAL            NOT NULL,
  avg_20_day    REAL,
  avg_5_month   REAL
)
ON COMMIT PRESERVE ROWS
NOT LOGGED
WITH REPLACE
;
```

```
-------------------------------------------------------------------------------
--
--  Calc moving averages (20-day, 5 month)
--
-------------------------------------------------------------------------------
INSERT INTO session.moving_average
SELECT  id,
        tradedate,
        ROW_NUMBER() OVER(PARTITION BY id ORDER BY tradedate),
        closeprice,
        AVG(closeprice) OVER(PARTITION BY id ORDER BY tradedate
                                    ROWS BETWEEN  20 PRECEDING AND 1 PRECEDING),
        AVG(closeprice) OVER(PARTITION BY id ORDER BY tradedate
                                    ROWS BETWEEN 160 PRECEDING AND 1 PRECEDING)
FROM
        hist_price2
WHERE
        tradedate between '2006-01-17' and '2007-01-17'
and     id in (select id from mkt_q7_id1)
with ur
;
CREATE INDEX session.ix1movingavg ON session.moving_average
(
        id,
        day_seq
)
ALLOW REVERSE SCANS
;

RUNSTATS ON TABLE session.moving_average
  AND INDEXES ALL ;

-------------------------------------------------------------------------------

--
--  Calculate buys/sells/positions
--
-------------------------------------------------------------------------------

WITH
recur (id,tradedate,day_seq,cash,stock_value,shares_held,action,closeprice,avg_2
0,avg_5)
AS
(
  SELECT id,
         tradedate,
         day_seq,
         CAST(10000 AS REAL) AS cash,
         CAST(    0 AS REAL) AS stock_value,
         CAST(    0 AS SMALLINT) AS shares_held,
         CHAR(' ',4) AS action,
         CAST(    0 AS REAL) AS closeprice,
         CAST(    0 AS REAL) AS avg_20,
         CAST(    0 AS REAL) AS avg_5
  FROM
         session.moving_average
    where
        tradedate = '2006-01-17'

  UNION ALL

  SELECT ma.id,
         ma.tradedate,
         ma.day_seq,
         case
           WHEN avg_20_day > avg_5_month AND r.shares_held = 0 THEN
                  r.cash - (FLOOR(r.cash / ma.closeprice) * ma.closeprice)
           WHEN avg_20_day < avg_5_month AND r.shares_held > 0 THEN
                  r.cash + (r.shares_held * ma.closeprice)
           ELSE r.cash
         END AS cash,
         case
           WHEN avg_20_day > avg_5_month AND r.shares_held = 0 THEN
                  FLOOR(r.cash / ma.closeprice) * ma.closeprice
           WHEN avg_20_day < avg_5_month AND r.shares_held > 0 THEN 0
           ELSE r.shares_held * ma.closeprice
         END AS stock_value,
         case
           WHEN avg_20_day > avg_5_month AND r.shares_held = 0 THEN
                  FLOOR(r.cash / ma.closeprice)
           WHEN avg_20_day < avg_5_month AND r.shares_held > 0 THEN 0
           ELSE r.shares_held
         END AS shares_held,
```

```
         case
           WHEN avg_20_day > avg_5_month AND r.shares_held = 0 THEN 'BUY'
           WHEN avg_20_day < avg_5_month AND r.shares_held > 0 THEN 'SELL'
           ELSE ' '
         END AS action,
         ma.closeprice,
         ma.avg_20_day,
         ma.avg_5_month
  FROM
         recur                   r,
         session.moving_average  ma
  WHERE
         r.id = ma.id
  AND    (r.day_seq + 1) = ma.day_seq
  AND    ma.tradedate BETWEEN '2006-01-17' AND '2007-01-17'
)
 SELECT id,
        tradedate,
        DEC(closeprice,5,2),
        DEC(cash,8,2)         AS cash,
        DEC(stock_value,8,2)  AS stock
FROM
        recur
WHERE
        tradedate = '2007-01-17'
        -- to verify initial allocation uncomment these
        -- or tradedate = '2005-03-16'
        -- or tradedate = '2005-03-15'
ORDER BY 1
;
terminate;
```

Market data query #8

```
-- Find the pair-wise coefficients of correlation in a set of 10 securities
-- for a 2 year period. Sort the securities by the coefficient of
-- correlation, indicating the pair of securities corresponding
-- to that row. [Note: coefficient of correlation defined in appendix]


declare global temporary table session.temp2
like hist_price2
partitioning key (tradedate)
on commit preserve rows
not logged
;
insert into session.temp2
SELECT * FROM hist_price2 a
WHERE a.id in (select id from mkt_q8_id1)
AND a.tradedate between '2005-10-19' and '2007-10-19'
with ur
;
SELECT
a.id,
b.id,
correlation(a.closeprice, b.closeprice)
FROM
session.temp2 a, session.temp2 b
WHERE
a.tradedate=b.tradedate
group by a.id, b.id
order by correlation(a.closeprice, b.closeprice)
;
terminate;

Market data query #9
-- 1.   Find Price Gaps over a 5 Year Period for a single security
-- A price gap is defined as the opening price for a security is outside the
-- trading range of the prior day.  The data may not generate price gaps.
-- If this is the case, please modify the data to provide 10 different price gap
s
-- for each of the price gap queries executed.

WITH getallprice as (
        SELECT
                id,
                tradedate,
                ROW_NUMBER() OVER() as rownum,
                openprice as openprice,
                highprice as highprice,
```

```
                lowprice as lowprice
        FROM
                hist_price2 a
        WHERE
                a.id = 513571
                AND a.tradedate between '2005-07-24' and '2010-07-24'
        )

           SELECT
                a.id,
                a.tradedate,
                a.openprice,
                b.highprice,
                b.lowprice
        FROM
                getallprice a,
                getallprice b
        WHERE
                a.id = b.id
                AND a.rownum =  b.rownum - 1
                AND ( a.openprice > b.highprice or a.openprice < b.lowprice )
with ur
 ;
```

Market data query #10

```
-- 1.   Find any Price Gaps over a 6 month  Period
-- A price gap is defined as the opening price for a security is outside the
-- trading range of the prior day.  The data may not generate price gaps.
-- If this is the case, please modify the data to provide 10 different price gap
s
-- for each of the price gap queries executed.

declare global temporary table session.temp
(
                id integer,
                tradedate date,
                openprice real,
                highprice real,
                lowprice real
)
on commit preserve rows
not logged
;
insert into session.temp
WITH getallprice as (
        SELECT
                id,
                tradedate,
                ROW_NUMBER() OVER(partition by id  order by tradedate ) as rownu
m,
                dec(openprice,6,2) as openprice,
                dec(highprice ,6,2) as highprice,
                dec(lowprice, 6,2)  as lowprice
        FROM
                hist_price2 a
        WHERE
                a.tradedate between '2009-01-22' and '2009-07-22'

        ),

        count_gap as (
        SELECT
                a.id,
                a.tradedate,
                a.openprice,
                b.highprice,
                b.lowprice
        FROM
                getallprice a,
                getallprice b
        WHERE
                a.id = b.id
                AND a.rownum =  b.rownum - 1
                AND ( a.openprice > b.highprice or a.openprice < b.lowprice )

        )
        SELECT * from count_gap
with ur
;
```

```
select count(*),'TOTAL RECORDS' from session.temp;
terminate;
```

Tick data query #1

```
-- Get all ticks for a specified set of 100 securities for a specified
-- three hour time period on a specified trade date


SELECT *
FROM tick_price2
WHERE tradedate = '2009-04-13'
AND timestamp BETWEEN '09:00:37'  AND '12:00:37'
AND id in (select id from t_lge_q1_id1)
with ur
;
```

Tick data query #2

```
-- Determine the volume weighted price of a security considering only the ticks
-- in a specified three hour interval

SELECT id,
sum(tradesize*tradeprice)/sum(tradesize)
FROM tick_price2
WHERE tradedate = '2009-06-22'
AND timestamp BETWEEN '15:43:27' AND '18:43:27'
AND id = 45978
GROUP BY id
with ur
;
```

Tick data query #3

```
-- Determine the top 10 percentage losers for the specified date on the
-- specified exchanges sorted by percentage loss. The loss is calculated
-- as a percentage of the last trade price of the previous day.

WITH
LAStTs (id, tradedate, tradeprice, lASttime) AS
(SELECT id,
tradedate,
tradeprice,
ROW_NUMBER() OVER (PARTITION BY  id, tradedate ORDER BY timestamp DESC) AS rown
FROM Tick_price2
WHERE tradedate between '2005-04-14' and '2005-04-15'),

CurrTs (id, tradedate, tradeprice, prevprice) AS
(SELECT id,
tradedate,
tradeprice,
avg(tradeprice) OVER (PARTITION BY  id ORDER BY tradedate ASc ROWS BETWEEN 2 pre
ceding AND 1 preceding)
FROM LAStTs
WHERE lASttime=1),

result (id, percLoss,percLossRank) AS
(SELECT  id,
(prevprice-tradeprice)*100/prevprice perc_loss, rank() OVER (ORDER BY (prevprice
-tradeprice)*100/prevprice)
FROM CurrTs
WHERE tradedate='2005-04-15')

SELECT *
FROM result
WHERE percLossRank<=10
with ur
;
```

Tick data query #4

```
-- Determine the top 10 most active stocks for a specified date
-- sorted by cumulative trade volume by considering all trades

WITH
allids (id, rank) AS
(SELECT id, rank() OVER (ORDER BY sum(tradesize))
FROM tick_price2
WHERE tradedate='2005-04-04'
```

```
GROUP BY id)

SELECT id
FROM allids
WHERE rank<11
with ur
;
```

Tick data query #5

```
-- Find the most active stocks in the "COMPUTER" industry (use SIC code)

WITH
allids (id, rank) AS (SELECT a.id, rank() OVER (ORDER BY count(1) DESC)
FROM tick_price2 a,
tick_bASe b
WHERE
a.id=b.id
AND b.SIC='COMPUTERS'
and tradedate = '2008-04-25'
GROUP BY a.id)

SELECT id
FROM allids
WHERE rank <=1
with ur
;
```

Tick data query #6

```
-- Find the 10 stocks with the highest percentage spreads. Spread is the
-- difference between the last ask-price and the last bid-price.
-- Percentage spread is calculated as a percentage of the mid-point
-- price (average of ask and bid price)

WITH
LAStB (id, bidprice, lASttime) AS
(SELECT id, bidprice, ROW_NUMBER() OVER (PARTITION BY  id ORDER BY
tradedate,timestamp DESC) AS rown FROM Tick_price2
WHERE bidprice is not null and tradedate= '2009-09-07'),

LAStA (id,ASkprice, lASttime) AS
(SELECT id, ASkprice, ROW_NUMBER() OVER (PARTITION BY  id ORDER BY
tradedate,timestamp DESC) rown FROM Tick_price2
WHERE ASkprice is not null and tradedate = '2009-09-07'),

allids (id, rank) AS
(SELECT  a.id, rank() OVER (ORDER BY (2*(b.ASkprice-a.bidprice) / (b.ASkprice+a.
bidprice)) DESC)
FROM
LAStB a, LAStA b
WHERE
a.id=b.id
AND a.lASttime=1
AND b.lASttime=1)

SELECT id
FROM allids
WHERE rank < 11
with ur
;
```

Tick data query #7

```
-- Get a 5 minute sample (last tick of interval) of data for 1 security over a 6
 month period.
--


WITH t_5min_set AS (
  SELECT
        a.*,
        ROW_NUMBER() OVER ( PARTITION BY
                              tradedate ,
                              HOUR(timestamp),
                              FLOOR(MINUTE(timestamp) / 5)
                            ORDER BY timestamp DESC) AS seq
   FROM tick_price2 a
```

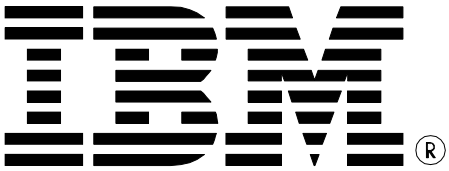
```
   WHERE  a.id  = 44550
          AND a.tradedate BETWEEN '2009-02-07' AND '2009-08-07'
  )
  SELECT *
  FROM t_5min_set
  WHERE seq = 1
  ORDER BY tradedate,timestamp
with ur
;
```

Tick data query #8

```
-- Get a 10 minute sample (last tick of interval) of data for 1 security over a
12 month period.
--


WITH t_10min_set AS (
  SELECT
        a.*,
        ROW_NUMBER() OVER ( PARTITION BY
                              tradedate ,
                              HOUR(timestamp),
                              FLOOR(MINUTE(timestamp) / 10)
                            ORDER BY timestamp ASC ) AS seq
   FROM tick_price2 a
   WHERE  a.id  = 117
          AND a.tradedate BETWEEN '2004-01-11' AND '2005-01-11'
  )
  SELECT *
  FROM t_10min_set
  WHERE seq = 1
  ORDER BY tradedate,timestamp
with ur
 ;
```

.



IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

Printed in United States of America
02/05