A technical discussion of the next evolution of database technology
April 2003

DB2® Information Management Software

IBM Informix Dynamic Server 9.4: Unequalled performance, scalability and availability

by Carlton Doe
Technical Sales Manager,
IBM Information Management

Contents

Highlights

Introduction

IBM Informix® Dynamic Server™ 9.4 (IDS) continues a long-standing tradition within IBM and Informix of delivering a first-in-class database engine. It combines the robustness, high performance, availability and scalability needed by today's global e-businesses.

Today's mission-critical database management applications need a database engine that can scale in performance as well as functionality.

Complex, mission-critical database management applications typically require a combination of online transaction processing (OLTP), batch and decision-support operations, including online analytical processing (OLAP). Meeting these needs is contingent upon a database engine that can scale in performance as well as in functionality. It must dynamically adjust as requirements change—from accommodating larger amounts of data to changes in query operations to increasing numbers of concurrent users. The technology should be designed to efficiently use all the capabilities of the existing hardware and software configuration, including single and multiprocessor architectures. Finally, the database engine must satisfy users' demands for more complex application support, which often uses nontraditional or "rich" data types that cannot be stored in simple character or numeric form.

IBM IDS is built on the IBM DSA, providing a highly effective database architecture that delivers mainframe-caliber performance characteristics and the capability to extend the server to handle new types of data.

IBM IDS is built on the *IBM Informix Dynamic Scalable Architecture™ (DSA)*, which is discussed in Appendix A. It provides one of the most effective solutions available—a next-generation parallel database architecture that delivers mainframe-caliber scalability, manageability and performance; minimal operating system overhead; automatic distribution of workload; and the capability to extend the server to handle new types of data. With version 9.4, IBM IDS dramatically alters the database landscape with significantly larger storage and processing capabilities than are available in virtually any other product.

Highlights

IBM IDS helps businesses lower their TCO by leveraging its well-regarded ease of use and administration as well as support for existing standards.

IBM IDS delivers proven technology that efficiently integrates new and complex data directly into the database. It handles time-series, spatial, geodetic, XML (Extensible Markup Language), video, image and other user-defined data side by side with traditional legacy data to meet today's most rigorous data and business demands. IBM IDS helps businesses to lower their total cost of ownership (TCO) by leveraging its well-regarded general ease of use and administration as well as its support of existing standards for development tools and systems infrastructure. IBM IDS is a development-neutral environment and supports a comprehensive array of application development tools for rapid deployment of applications under Linux, Microsoft® Windows® and UNIX® operating environments.

The maturity and success of IBM IDS is built on more than ten years of widespread use in critical business operations, which attests to its stability, performance and usability. IBM IDS 9.4 moves this already highly successful enterprise relational database to a new level.

About IBM Informix Dynamic Server 9.4

IBM IDS 9.4 sets new standards for scalability, reliability and availability that cannot be matched by other products.

The IBM IDS version 9.4 engine represents the next evolution of database technology by merging the world-class performance and scalability of the DSA architecture with cutting-edge object-relational technology. IBM IDS 9.4 sets new standards for scalability, reliability and availability that cannot be matched by other products on the market today. It is also the fastest of the IBM IDS engines. A number of significant changes were made in various components to streamline processing and increase efficiency. The net result is an engine whose performance eclipses that of IBM IDS 7.x.

Highlights

Object-relational extensibility in IBM IDS supports transactional consistency and data integrity while simplifying database optimization and administration.

Given the wide range of data types used by IBM IDS 9.4, database administrators and application developers can truly define data structures and rules that accurately reflect the business environment.

A brief look at extensibility

IBM IDS provides a complete set of features to extend the database server, including support for new data types, routines, aggregates and access methods. With this technology, in addition to recognizing and storing standard character and numeric-based information, the engine can, with the appropriate access and manipulation routines, manage non-traditional data structures that are either modeled more like the business environment or contain new types of information never before available for business application processing. Though the data may be considered "nonstandard," and some types can be table-like in and of themselves, it is stored in a relational manner using tables, columns and rows. In addition, all data, data structures created through Data Definition Language (DDL) commands, and access routines recognize objected-oriented behaviors such as overloading, inheritance and polymorphism. This object-relational extensibility supports transactional consistency and data integrity while simplifying database optimization and administration. Other database management systems (DBMS) rely on middleware to link multiple servers, each managing different data types, to make it look as if there is a single processing environment. This approach compromises not only performance, but also transactional consistency and integrity because problems with the network can corrupt the data. This is not the case with IBM IDS. Its object-relational technology is built into the DSA core and can be used, or not, at will within the context of a single database environment.

Data types

IBM IDS 9.4 uses a wide range of data types to store and retrieve data. The breadth and depth of the data types available to the database administrator and application developer is significant—allowing them to truly define data structures and rules that accurately mirror the business environment rather than trying to approximate it through normalized database design and access constraints.

Highlights

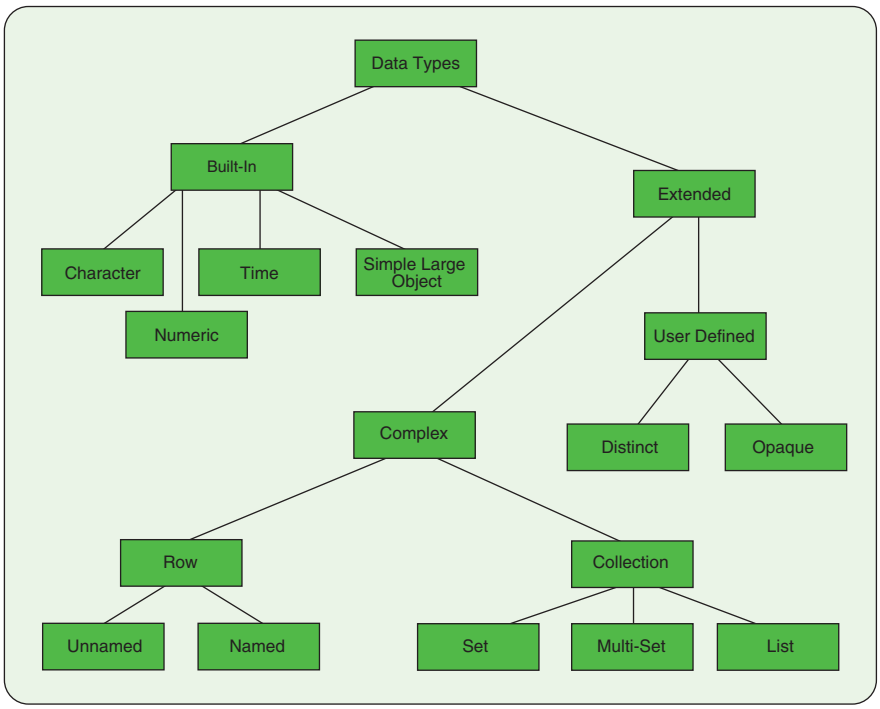

Figure 1:
The IBM Informix Dynamic Server 9.4 data type tree.

IBM has added additional built-in data types to IBM IDS, including boolean, int8, serial8 and an even longer variable-length character string, the lvarchar.

Some types, referred to as *built-in types*, include standard data representations such as *character(n)*, *decimal*, *integer*, *serial*, *varchar(n)*, *date*, and *datetime*, alias types such as *money*, and *simple large objects (LOBs)*. IBM has also added additional built-in types to recent releases of IBM IDS 9, including *boolean*, *int8*, *serial8* and an even longer variable length character string, the *lvarchar*.

Extended data types themselves are of two classes, including:

- *Super-sets of built-in data types with enhanced functionality*
- *Types that were not originally built into the IBM Informix database engine but that, once defined, can be used to intelligently model data objects to meet business needs.*

Highlights

The *collection* type is used to store repeating sets of values within one row of one column that would normally require multiple rows or redundant columns in one or more tables in a traditional database. The three collection types enforce rules on whether or not duplicate values or data order is significant. Collection data types can be nested and contain almost any type, built-in or extended.

Row data types facilitate the building of new data types that are composed of other data types.

With *row* data types, a new data type can be built that is composed of other data types. The format of a row type is similar to that used when defining columns to build a table—a parameter name and data type. Once defined, row types can be used as columns within a table or as a table in and of themselves. With certain restrictions, a row type can be dynamically defined on the fly as a table is being created or can be inherited into other tables.

Named:

```
create row type name_t
  (fname char(20),
  lname char(20));

create row type address_t
  (street_1 char(20),
  street_2 char(20),
  city char(20),
  state char(2),
  zip char(9));

create table student
  (student_id serial,
  name name_t,
  address address_t,
  company char(30));
```

Unnamed:

```
ROW (a int, b char(10))

  Note: is also equal to
  ROW (x int, y char(10))

create table part
  (part_id serial,
  cost decimal,
  part_dimensions row
    (length decimal,
    width decimal,
    height decimal,
    weight decimal));
```

Figure 2:
Examples of named and unnamed row data types and their application.

Highlights

Since the engine can natively store or process any data that can be represented in C or Java, IBM IDS can encapsulate applications that have already implemented data types as C or Java structures.

IBM IDS access routines are fully and automatically recoverable, and they benefit from the proven manageability and integrity of the IBM DSA.

A *distinct* data type is an alias for an existing data type. A newly defined distinct data type will inherit all of the properties of its parent type (for example, a type defined using a float parent will inherit the elements of precision before and after the decimal point) but because it is a unique type, its values cannot be combined with any other data type but its own without either "casting" the value or using a user-defined routine. Finally, *opaque* data types are those created by developers in C or Java™ and can be used to represent any data structure that needs to be stored in the database. When using opaque data types, as opposed to the other types already mentioned, the engine is completely dependent on the type's creator to define all access methods that might be required for the type including insert, query, modify and delete operations.

Extended data types can be used in queries or function calls, passed as arguments to database functions, indexed and optimized in the same way as the core built-in data types. Since any data that can be represented in C or Java can be natively stored and processed by the engine, IBM IDS can encapsulate applications that have already implemented data types as C or Java structures. Because the definition and use of extended data types is built into the DSA architecture, specialized access routines support high performance. The access routines are fully and automatically recoverable, and they benefit from the proven manageability and integrity of the IBM Informix database architecture.

Casts and casting

With the enormous flexibility and capability that both built-in and extended data types provide to create a database environment that accurately matches the business environment, they must often be used together. To do so requires functionality to convert values between types. This is generally done through the use of *casts* and, quite often, the casting process will use *user-defined functions*.

Highlights

IBM IDS includes casts, which enable a developer to manipulate the values of different types of data together or to substitute the value of one type in place of another.

While explicit casts take more effort on the part of the developer, they provide more program options based on the desired output type.

Casts enable a developer to manipulate values of different data types together or to substitute the value of one type in the place of another. While casts, as an identifiable function, have only been recently added to the SQL syntax, IBM IDS administrators and developers have been using casts for some time; however, they've been hidden in the engine's functionality. For example, to store the value of the integer "12" in a table's character field requires casting the integer value to its character equivalent, and this action is performed by the engine on behalf of the user. The inverse cannot be done because there is no appropriate cast available to represent a character, such as an "a," in a numeric field.

When using "user-defined types" (UDTs), casts must be created to change values between the source type and each of the expected target data. For some types, such as collections, LOBs and unnamed row types, casts cannot be created due to the unique nature of these types. Casts can be defined as either "explicit" or "implicit." For example, with an implicit cast, a routine is created that adds values of type "a" to the value of type "b" by first converting the value of one type to the other type and then adding the values together. The result can either remain in that type or be converted back into the other type before being returned. Any time an SQL operation requires this operation to occur, this cast is automatically invoked behind the scenes and a result returned. An explicit cast, while it may perform the exact same task as an implicit cast, only executes when it is specifically called to manipulate the values of the two data types. While it requires a little more developer effort to use explicit casts, there are more program options available with their use based on the desired output type.

Highlights

IBM IDS 9 provides the ability to create significantly more robust and higher-performing application or data management logic in the engine where it can benefit from the processing power of the physical server and the DSA.

Regardless of their location, C and Java routines execute as if they were a built-in component of the engine.

User-defined routines, aggregates and access methods

In earlier versions of the IBM IDS engine, developers and administrators who wanted to capture application logic that manipulated data and have it execute within the engine only had stored procedures to work with. Although stored procedures have an adequate amount of functionality, they may not optimize performance. IBM IDS 9 provides the ability to create significantly more robust and higher performing application or data manipulation logic in the engine where it can benefit from the processing power of the physical server and the DSA.

A "user-defined routine" (UDR) is a collection of program statements that—when invoked from an SQL statement, a trigger, or from another UDR—perform new domain-specific operations, such as searching geographic data or collecting data from Web site visitors. UDRs are most commonly used to execute logic in the engine, either generally useful algorithms or business-specific rules, reducing the time it takes to develop applications and increasing the applications' speed. UDRs can be either functions that return values or procedures that do not. They can be written in *IBM Informix Stored Procedure Language (SPL)*, C or Java. SPL routines contain SQL statements that are parsed, optimized and stored in the system catalog tables in executable format—making SPL ideal for SQL-intensive tasks. Since C and Java are powerful, full-function development languages, routines written in these languages can carry out much more complicated tasks than SPL routines. C routines are stored outside the engine with the pathname to the shared library file registered as the UDR. Java routines are first collected into "jar" files, which are stored inside the engine as "smart large objects" (SLOs). Regardless of their storage location, C and Java routines execute as if they were a built-in component of the engine.

Highlights

A UDA is a UDR that can either extend the functionality of an existing built-in aggregate or provide new functionality.

A "user-defined aggregate" (UDA) is a UDR that can either extend the functionality of an existing built-in aggregate (for example, *SUM* or *AVG*) or provide new functionality that wasn't previously available. Generally speaking, aggregates return summarized results from one or more queries. For example, the built-in *SUM* aggregate adds values of certain built-in data types from a query result set and returns their total. An extension of the *SUM* aggregate can be created to include user-defined data types, enabling the reuse of existing client application code without requiring new SQL syntax to handle the functionality of new data types within the application. To do so, using the example of the *SUM* aggregate, would require creating (and registering) a user-defined function that would overload the "plus" function and take the user-defined data types, which needed to be added together, as input parameters.

To create a completely new user-defined aggregate requires creating and registering two to four functions to perform the following:

- *Initialize the data working space*
- *Merge a partial existing result set with the result of the current iteration*
- *Merge all the partial result sets*
- *Return the final result set with the associated closure and release of system resources to generate the aggregate.*

Functions created and registered for UDAs can be written in SPL, C or Java.

In defining the ability to work with partial result sets, UDAs can, like built-in aggregates, execute in parallel. Functions created and registered for UDAs can be written in SPL, C or Java. Like built-in aggregates, the engine wholly manages a UDA once it's registered (as either an extended or user-defined aggregate).

Highlights

IBM IDS provides primary and secondary access methods to access and manipulate data stored in tables and indexes.

IBM IDS provides primary and secondary *access methods* to access and manipulate data stored in tables and indexes. Primary access methods, used in conjunction with built-in data types, provide functionality for table use. Secondary access methods are specifically targeted to indexes and include *B-tree* and *R-tree* indexing technologies. Additional user-defined access methods can be created to access other data sources. IBM IDS has methods that provide SQL access to data in either a heterogeneous database table, an external sequential file or to other nonstandard data stored anywhere on the network. Secondary access methods can be defined to index any data as well as alternative strategies to access SLOs. These access methods can be created using the Virtual Table Interface (VTI) and the Virtual Index Interface (VII) server application programming interfaces (APIs).

DataBlades

IBM Informix DataBlade modules bring additional business functionality to the engine through specialized user-defined data types, routines and access methods.

IBM Informix DataBlade™ modules bring additional business functionality to the engine through specialized user-defined data types, routines and access methods. Developers can use these new data types and routines to more easily create and deploy richer applications that better address a company's business needs. IBM IDS provides the same level of support to DataBlade functionality that is accorded to built-in or other user-defined types/routines. With IBM Informix DataBlade modules, almost any kind of information can be easily managed as a data type within the engine.

There is a growing portfolio of third-party DataBlade modules, or developers can use the *IBM Informix DataBlade Developer's Kit (DBDK)* to create specialized blades for a particular business need.

Highlights

The TimeSeries DataBlade provides a better way to organize and manipulate any form of realtime, time-stamped data.

The NAG DataBlade provides the ability to perform quantitative analysis of tick-based financial data within the engine itself through the use of routines.

The following is a partial list of available IBM Informix DataBlade technologies (a current list is available at *ibm.com/informix*):

- *IBM Informix TimeSeries DataBlade*–This DataBlade provides a better way to organize and manipulate any form of realtime, time-stamped data. Applications that use large amounts of time-stamped data, such as network analysis, manufacturing throughput monitoring or financial tick data analysis, can provide measurably better performance and reduced data storage requirements with this DataBlade than can be achieved using traditional relational database design, storage and manipulation technologies.
- *IBM Informix NAG DataBlade*–IBM partnered with the Numerical Algorithms Group (www.nag.co.uk) to provide the ability to perform quantitative analysis of tick-based financial data within the engine itself through the use of routines from their Fortran-based library. These libraries can be applied to the analysis of currency, equity and bond instruments to identify over- and under-valued assets, implement automated trading strategies, price complex instruments such as derivatives, or to create customized products for an institution's corporate customers. Because the analysis occurs in the engine where the data is stored, response times are a fraction of those achieved by systems that must first transfer the data through middleware to a client-side application.
- *IBM Informix TimeSeries Real-Time Loader*®–A companion piece to the IBM Informix TimeSeries DataBlade, the TimeSeries Real-Time Loader is specifically designed to load time-stamped data and make it available to queries in realtime.

Highlights

The Geodetic DataBlade is designed to manage spatio-temporal data in a global context, such as satellite imagery and related metadata, or trajectory tracking in various environments.

- *IBM Informix Spatial DataBlade and the IBM Informix Geodetic DataBlade*—Provide functionality to intelligently manage complex geospatial information within the efficiency of a relational database model. The IBM Informix Geodetic DataBlade stores and manipulates objects from a "whole-earth" perspective using four dimensions—latitude, longitude, altitude and time. It is designed to manage spatio-temporal data in a global context, such as satellite imagery and related metadata, or trajectory tracking in the airline, cruise or military environment. The IBM Informix Spatial DataBlade is a set of routines that is compliant with open-GIS (geographic information system) standards, which take a "flat-earth" perspective to mapping geospatial data points. Based on ESRI technology (www.esri.com), routines and utilities, this DataBlade is better suited for answering questions such as, "how many grocery stores are within 'n' miles of point 'x'?", or "what is the most efficient route from point 'a' to point 'b'?" All IBM Informix geospatial DataBlades take advantage of the built-in IBM Informix R-tree multi-dimensional index technology, resulting in industry-leading spatial query performance. While the IBM Informix Geodetic DataBlade is a for-charge item, the IBM Informix Spatial DataBlade is available at no charge to licensed users of IBM IDS 9.4.

The Excalibur Text DataBlade performs full-text searches of documents and supports any language, word or phrase that can be expressed in an 8-bit, single-byte character set.

- *IBM Informix Excalibur Text DataBlade*—Performs full-text searches of documents stored in database tables and supports any language, word or phrase that can be expressed in an 8-bit, single-byte character set.
- *IBM Informix Video Foundation DataBlade*—Allows strategic third-party development partners to incorporate specific video technologies, such as video servers, external control devices, codecs or cataloging tools, into database management applications. It also provides the ability to manage video content and video metadata regardless of the content's location.

Highlights

- *IBM Informix Image Foundation DataBlade*—Provides functionality for the storage, retrieval, transformation and format conversion of image-based data and metadata. While this DataBlade supplies basic imaging functionality, third-party development partners can also use it as a base for new DataBlade modules to provide new functionality, such as support for new image formats, new image processing functions and content-driven searches.
- *IBM Informix C-ISAM DataBlade*—Provides two separate pieces of functionality to the storage and use of Indexed Sequential Access Method (ISAM)-based data. In those environments where the data is stored in its native flat-file format, the DataBlade provides engine-based SQL access to the data. From a user or application developer perspective, it's as if the data resided in standard database tables. The second element of functionality enables the storage and retrieval of ISAM data in the database itself while preserving the native C-ISAM application access interface. From a C-ISAM developer's perspective, it's as if the data continued to reside in its native flat-file format; however, with the data stored in the engine, transactional integrity can be added to C-ISAM applications. Another benefit to storing C-ISAM data in the engine is gaining access to the more comprehensive backup and recovery routines provided by IBM IDS.

The C-ISAM DataBlade provides two separate pieces of functionality to the storage and use of ISAM-based data.

The IBM Informix DataBlade Developer's Kit is a single development kit for Java-, C-, and SPL-based DataBlades and the DataBlade API.

The DBDK is a single development kit for Java-, C- and SPL-based DataBlades and the DataBlade application programming interface. The DataBlade API is a server-side "C" API for adding functionality to the database server, as well as for managing database connections, server events, errors, memory and processing query results. Additional support for DataBlade module developers includes the *IBM Informix Developer Zone* available at *www7b.boulder.ibm.com/dmdd/zones/informix/*. Developers can interact with peers, pass along information and expertise, and discuss new development trends, strategies and products. Examples of DataBlades and Bladelets, indexes and access methods are available for downloading and use. Online documentation for the DBDK and other IBM Informix products is available at *ibm.com/informix/pubs/library/*.

Highlights

IBM IDS and Foundation bundles

IBM IDS 9.4 is the foundation of several functional bundles targeted to specific vertical markets. These Foundation bundles usually include additional DataBlades that best serve target markets by providing turnkey database and application capabilities. With one exception, the components in any given Foundation bundle can be purchased separately and added to an IBM IDS 9.4 installation. Purchasing the Foundation bundle does, however, provide some additional cost savings over buying the components separately.

IBM offers three IBM IDS Foundation bundles, which provide turnkey database and application capabilities for target markets.

IBM offers three IBM IDS Foundation bundles. The first is the *IBM IDS with J/Foundation*, which includes a Java HotSpot™ virtual machine (JVM™) as a database engine extension delivering scalable, high-performance Java UDRs by executing Java code directly in the server. Rather than building a single, proprietary JVM, J/Foundation uses standard JVMs from IBM platform vendor partners. This standard environment provides compatibility with third-party Java products and helps when migrating existing Java applications into the database engine. J/Foundation is required for businesses planning to develop user-defined routines in Java.

The IBM IDS Foundation bundles include the IBM IDS J/Foundation, IBM IDS Financial Foundation for Capital Markets and IBM IDS Law Enforcement Foundation.

The *IBM IDS Financial Foundation for Capital Markets*, targeted at the financial services market, includes the IBM Informix TimeSeries, IBM Informix NAG, and IBM Informix TimeSeries Real-Time Loader DataBlades and IBM Informix Office Connect. The *IBM IDS Law Enforcement Foundation* includes the IBM Informix Video Foundation DataBlade as well as biometric technology from Identix and Cogent, which provides facial, retinal, fingerprint and other biometric analysis and recognition functionality. This Foundation would be of interest to any organization looking to build strong, uniquely personalized authentication routines or recognition capabilities.

Highlights

IBM IDS 9.4 provides new growth capabilities that practically eliminate the limitations to what an IBM IDS engine can do.

Unmatched scalability

Today's data processing requirements have exceeded Moore's Law, and what once appeared to be unreachable processing limits are becoming part of tomorrow's systems design. With Dynamic Scalable Architecture, the IBM IDS engine is well recognized for its ability to fully leverage the processing power of today's SMP (symmetric multiprocessing) servers. As more processors or memory are added to the server, the engine can efficiently use them to support database activity. While adequate for most companies' needs, earlier versions and capabilities of the IDS engine were hampered by a few infrastructure limitations. IBM IDS 9.4 provides new growth capabilities that practically eliminate the limitations to what an IBM IDS engine can do.

Large chunk and file sizes

In 1990, when DSA was first designed and construction started on the IBM IDS engine, hard disk drive capacity and system memory limits were measured in the low hundreds of megabytes, and database capacities were measured in the low tens of gigabytes. Trying to work with then-existing technology yet plan for future growth and capacity, the DSA designers set several infrastructure limits that have remained in place until now. Database engine memory capacity, output file size and individual chunk (used to create dbspaces to store data) size was limited to 2GB. With an additional limit of 2,048 chunks, IBM IDS instances were limited to 4TB of total capacity. Needless to say, data processing capacities and demands have grown significantly since then, and these limits needed to be changed.

IBM IDS 9.4 has a total storage capacity of just under 128 petabytes—about 1 quadrillion ($128 * 10^{15}$) bytes.

In IBM IDS 9.4, the maximum chunk size has been increased to 4TB. In addition, the maximum number of instance chunks has increased to 32,767. As a result, IBM IDS 9.4 has a total storage capacity of just under 128 petabytes—about 1 quadrillion ($128 * 10^{15}$) bytes. This assumes that the engine administrator is not using IBM IDS-based disk mirroring for fault tolerance. Moreover, using IBM IDS mirroring technology doubles the total number of chunks and storage capacity.

Highlights

To put this capacity into perspective:

- *With a sustained load rate of 1TB per hour, it would take more than 15 years to fill 128 petabytes of storage.*
- *It is estimated that the U.S. Library of Congress contains the equivalent of 10TB of printed material. IBM IDS 9.4 could store approximately 12,800 Libraries of Congress.*
- *Based on current prices, it would cost more than US$3 billion to purchase 128 petabytes of storage.*

Engine changes made to accommodate the capacity increases in IBM IDS 9.4 occurred in the page header structure.

The engine changes made to accommodate this capacity increase occurred in the page header structure. Without affecting the existing 24-byte structure and requiring an off-line conversion, the chunk number and page-offset ranges were increased while the time-stamp structure was changed to take less space. The net effect of these changes is that in upgrading to IBM IDS 9.4, little to no disk conversion work is required. Page header information is converted on the fly and as needed once the engine administrator turns on full support for this feature.

In upgrading to IBM IDS 9.4, little to no disk conversion work is required.

Conversion to large chunk sizes is a two-step process. Initially, access to large chunks is disabled and the engine acts in legacy mode. An administrator must execute an *onmode* command to activate Stage 1 access to large chunks. In Stage 1 mode, existing chunks are left in legacy mode; however, newly created chunks can be larger than 2GB. If desired, existing dbspaces can be upgraded to support the new functionality by adding a large chunk (greater than 2GB) to the dbspace. If a chunk smaller than 2GB is added to a legacy dbspace, it will not trigger a conversion to the new functionality.

Highlights

After executing another onmode command, the engine will be in Stage 2 conversion, from which there is no reversion path except reloading from a Level 0 archive taken prior to the command's execution. In Stage 2 mode, whenever a legacy page on disk is to be modified, its header will be converted to the new format. Since it could take some time, if ever, for all pages in an instance to be converted into the new format, the engine will support both legacy and "new" page formats. An engine administrator can force a mass conversion of all pages to the new format by executing a dummy SQL update operation that affects all rows in a table but does not write new data. For example, "UPDATE orders SET order_date = order_date". However, IBM does not recommend executing this type of conversion since there is no real performance benefit.

In conjunction with the increase in total storage capacity, the output limits for many engine utilities have been increased from 2GB to 8 trillion MB or 2^{63}-1 bytes.

In conjunction with the increase in total storage capacity, the output limits for many engine utilities has been increased from 2GB to 8 trillion MB or 2^{63}-1 bytes. Those utilities affected include *oncheck*, *onload*, *onlog*, *onparams*, *onstat*, *onunload*, *DB-Access*, *dbload*, *dbschema* and *onpload*. The output capacity for other utilities, including *ontape*, *load*, *unload*, *dbexport* and *dbimport*, has increased to 4TB.

More practical changes to IBM IDS include increasing the number of DBSERVERALIAS entries as well as the number of columns in a functional index.

Other parameters have also been increased as well; however, for some, their real-world application is currently limited. For example, a single IBM IDS instance will support over 21 million unique databases, more than 477 million tables, and almost 34 billion bytes per partitioned table fragment. The maximum transaction size supported in Enterprise Replication (ER) is now 4TB. While the engine can have a virtually unlimited number of logical logs to capture transactional changes, each log cannot be greater than 1GB. More practical changes include increasing the number of *DBSERVERALIAS* entries from 10 to 32 and changing the number of columns in a functional index from 16 to 102 (C-based) or 341 (SPL or Java-based) columns.

Highlights

Support for full tape sizes

With the increase in output file sizes for DBA utilities, IBM changed how IBM IDS manages tape devices. Prior to IBM IDS 9.4, tape device configuration within the engine required not only a read and write block size but the media storage capacity as well. While these parameters were not enforced when the *ON-Bar* backup and restore interface was used in conjunction with a third-party tape management system, they otherwise affected tape output from engine utilities. If the media length varied in a cartridge during a write operation, there was either unused or wasted space at the end of the media or the job aborted with an unexpected end-of-media error.

IBM IDS 9.4 now supports a *TAPESIZE* and *LTAPESIZE* $ONCONFIG value of "0" (zero) which causes an output stream through the ontape and dbexport utilities to continue until the end-of-tape (EOT) marker code is returned. If more data needs to be written, the administrator will be prompted to insert new media.

This functionality is also supported in the High Performance Loader and its corresponding *ipload*, *onpload* and *onpladm* utilities by setting the "-s" parameter to "0" (zero) for an onpload or *onunload* job. The "-Z" option can also be set in onpload or onpladm runjob commands to write to the end of the media.

64-bit support

With large memory addressability, IBM IDS can support tens of gigabytes of physical memory and hundreds of gigabytes of address space, enabling the creation of larger buffer pools to provide increased data caching.

Memory addressability schemes built around a 32-bit architecture severely limit systems to 2GB of shared memory or less. These limitations are relaxed in 64-bit architectures, providing a significant breakthrough in database performance. With large memory addressability, IBM IDS can support tens of gigabytes of physical memory and hundreds of gigabytes of virtual address space, enabling the creation of larger buffer pools to provide increased data caching. This, in turn, increases the number of users that can be supported, improves database throughput and reduces swapping in heavy OLTP environments.

Highlights

IBM IDS 9 supports the use of long identifiers for both user identification and database object naming, allowing logical database and application models to represent the object's business use or reflect its true identity or purpose.

Long identifier support

As databases and their applications have become more complex, so has the need to create meaningful names for database objects. Previously, user names were limited to eight characters while other object names couldn't be longer than 18 characters. This length is insufficient when trying to build logical database and application models to represent the object's business use or reflect its true identity or purpose. IBM IDS 9 now supports the use of long identifiers for both user identification and database object naming. Databases, tables, views, constraints, stored procedures, indexes, columns, dbspaces, blobspaces, triggers and other database object names can now be up to 128 characters long. User names can be up to 32 characters in length.

IBM IDS 9.4 provides two longer variable-length data types, the lvarchar and the lvarchar(n).

Lvarchars

IBM IDS has supported the use of both fixed-length and variable-length character data types for some time, though early support was limited to 255 characters for the variable-length type. IBM IDS 9.4 provides two longer variable-length data types, the lvarchar and lvarchar(n). Without specifying a size, a lvarchar can be up to 2KB. A column defined with this lvarchar cannot be used in an index. If a larger capacity is needed, a size of "$0 < n < 32k - 2$" bytes can be specified. If even more text needs to be stored as a columnar element, database designers should use the *CLOB (Character Large Object)* "smart" large object. Columns defined with this larger capacity lvarchar can be used in indexes, although it is not recommended for performance and storage reasons.

Dynamic and byte-range locking

To maintain data integrity and concurrency within the database, IBM IDS uses locks to help ensure an orderly process of isolating data that is about to be changed or needs to be protected for the duration of a query operation. In earlier versions of the engine, the number of locks was a fixed resource: It had to be managed by the administrator to ensure enough were configured not only for normal operations but for the odd runaway query as well. With the object-relational extensions available in IBM IDS 9 and the need to have multi-user access to a single SLO, IBM implemented a new locking mechanism.

Highlights

IBM IDS 9.4 uses a dynamic locking mechanism, with the engine dynamically adding and deleting locks as needed for operations.

IBM IDS 9.4 uses a dynamic locking mechanism, with the engine dynamically adding and deleting locks as needed for operations. While the *LOCKS* $ONCONFIG parameter is still in place, it now represents a minimum number of locks that should be configured. As such, it should be configured for normal day-to-day operations. These locks will reside in the resident portion of the instance's shared memory. If more locks are needed, the engine will allocate additional locks that are managed in the virtual portion of the instance's memory.

Concurrent and multi-user access to SLOs is managed through byte-range locks, where, as the name suggests, a lock is placed on portions of the SLO rather than the entire SLO as happens with simple LOBs. To prevent an excessive number of locks being allocated within a SLO, a range lock will be upgraded to a whole SLO lock if transactional integrity requires locks covering more than ten percent of the SLO. Byte-range locks can be split, joined, overlapped or separated as needed to support user operations.

Performance improvements

Compared to IDS version 7, IBM IDS 9.4 showed a 10 to 15 percent increase in engine performance during the product build and test phase.

The continued enhancement of engine performance beyond that provided by IBM IDS version 7.x was of primary concern in the development of IBM IDS 9.4. This was accomplished through various improvements within the code stream and new technologies. In fact, during the build and test phase of this release, engine benchmarks regularly showed a 10 to 15 percent improvement in engine performance when compared to IDS version 7.

Highlights

IBM IDS uses a hierarchical system of cascading nodes called a B+ tree indexing method with root, branch and leaf nodes for indexing most data types.

B-tree index cleaning improvements

Where possible, the query optimizer uses indexes for an operation's access plan to locate data that does not already exist in the instance buffers. IBM IDS uses a hierarchical system of cascading nodes called a *B+ tree* indexing method with *root*, *branch* and *leaf* nodes for indexing most data types. Each branch node contains pointers to values in the next-lower level of the index that are either greater than or less than its own value. Leaf nodes contain page addresses of the data within the table, referred to by the key value, as well as a pointer to the leaf node on either side of it.

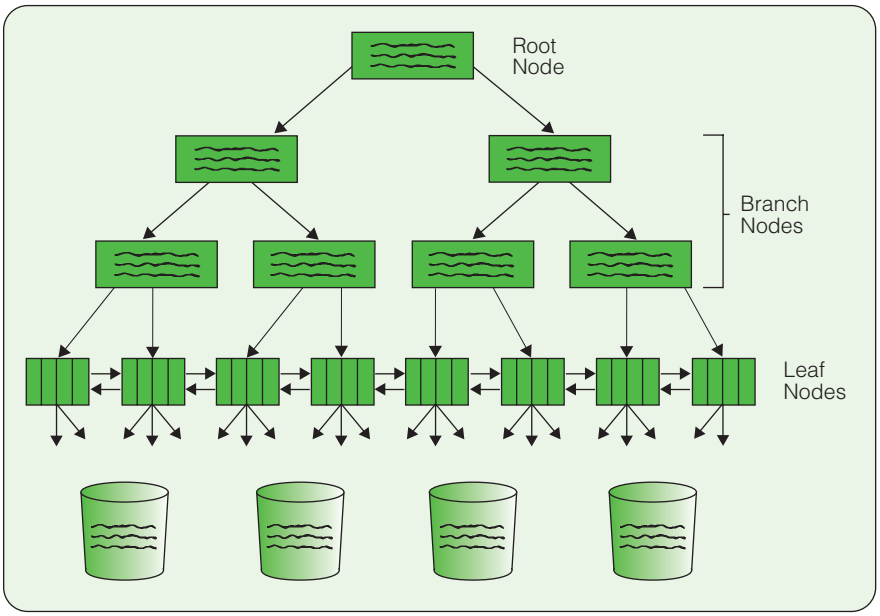

Figure 3:
A conceptual view of IBM Informix Dynamic Server B+ tree indexing.

Highlights

Previously in IBM IDS, when transactions deleted a row from a table in a logged database, a considerable amount of code and time was required to manage the index-cleaning process.

Prior to IBM IDS 9.4, whenever a transaction deleted a row from a table in a logged database, the space in the data page was marked open for reuse. The index entry for the row was not deleted immediately, however. Instead it was marked for deletion and an entry was made in the B-tree cleaner pool in the virtual portion of the instance's shared memory. These entries can be viewed by executing the *onstat -C* command. Whenever the pool entries reached a predetermined depth or a specific time interval elapsed, the B-tree cleaner thread was activated. As the cleaner parsed the entries in the pool, it executed various tasks and safety checks in addition to locating the index node entry to be deleted and clearing the key value and associated page entry. Given that one or more partitions of a partitioned table could be deleted in one transaction—or that the cleaner thread might have to return to the same index multiple times while deleting all the entries listed in the cleaner pool since it ran in unsorted mode—a considerable amount of code was required to manage the index cleaning process, in addition to the time required to execute the clean.

IBM IDS 9.4 uses a "B-tree scanner," which optimizes and prioritizes index entries to be deleted and can take advantage of additional cleaner threads added by the engine administrator for the processing of index deletes.

IBM IDS 9.4 completely replaces this process with a more efficient system called the "B-tree scanner" which, among other things, optimizes and prioritizes index entries to be deleted and can take advantage of additional cleaner threads added by the engine administrator for the processing of index deletes. The new removal process uses *dirty counts*, or the number of times a user or administrative thread encounters index entries marked for deletion during normal processing, to compile a *hot list* or prioritized list of indexes to be cleaned. The more hits an index receives, the higher it climbs in the hot list so that the index gets cleaned earlier by the B-tree scanner thread(s). Hot lists expire over time and are automatically regenerated but the engine administrator can manually force a regeneration of the hotlist with the onmode utility.

Highlights

Under specific conditions, the engine administrator can configure index cleaning to use light range scans, which helps achieve batch-processing-like speed and efficiency.

The number, execution priority and other operating parameters of the B-tree scanner threads can be monitored or tuned by the engine administrator.

When the B-tree scanner threads are active, they pick from the top of the list and, starting from the root node, traverse to the outmost left leaf node then perform a left-to-right light scan of the leaf nodes utilizing the index buffer pools to locate and delete the necessary entries. As the deletes occur, page compression occurs with node redistribution where possible. Under specific conditions, the engine administrator can configure index cleaning to occur by using light range scans rather than left-to-right leaf node scans. This cleaning mode achieves batch-processing-like speed and efficiency.

The number, execution priority and other operating parameters of the B-tree scanner threads can be monitored or tuned by the engine administrator through the onmode and onstat utilities or the Informix Server Administrator (ISA).

LRU fractional values and buffer management

The *Least-Recently Used (LRU)* queue mechanism is actually sets of two queues—the *free least-recently used (FLRU)* and *modified least-recently used (MLRU)* queues—and contains addresses to the buffers in the resident portion of the instance's memory configured through the *BUFFERS*$ONCONFIG parameter. Within these queues, there is a "most-recently used" and "least-recently used" end of each queue.

Highlights

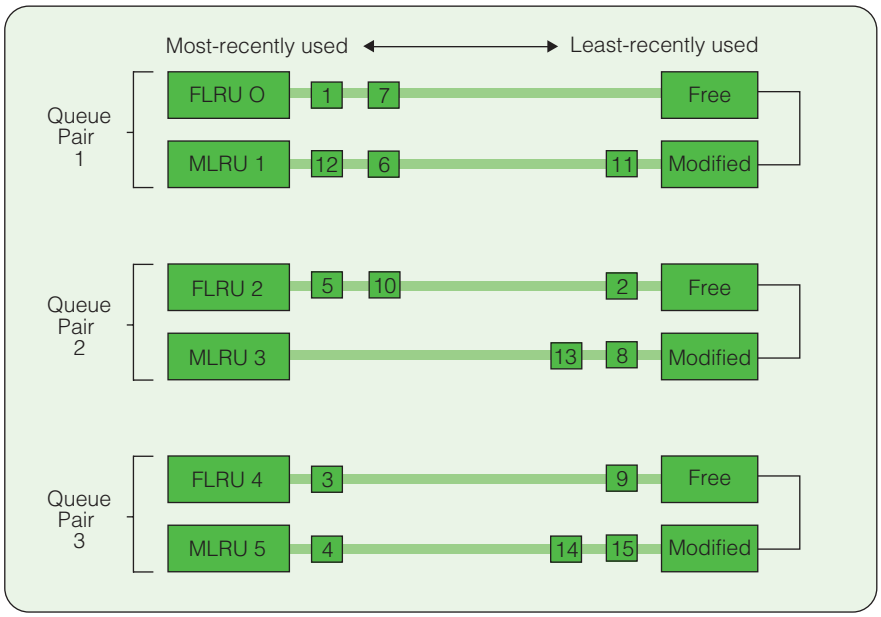

Figure 4:
Architectural diagram of LRU queue pairs.

Buffer addresses can be moved to either the most-recently or least-recently used end of the FLRU or MLRU queue during instance processing.

This helps ensure that modified buffers are not immediately reused before their contents are written to disk, available buffers are evenly cycled back into use and buffer contents are preserved for as long as possible.

At instance activation, all buffer addresses are evenly divided into the FLRU component of the queue pairs. As threads need buffers for processing, buffer addresses are transferred to the MLRU component of the pair. Depending on what happens to the data in the buffer during the SQL operation, the buffer address could be moved to the most-recently used or least-recently used end of either queue. For example, if the data is modified, the buffer address is moved to the most-recently used end of the MLRU queue. If the data was simply read and then discarded, the buffer address is also moved to the most-recently used end of a queue, but it is the FLRU queue. This helps ensure that modified buffers are not immediately reused before their contents are written to disk, available buffers are evenly cycled back into use, and buffer contents are preserved for as long as possible in case another operation needs the same data. If so, the request can be satisfied from the buffer pool as opposed to having to go back to disk. The distribution and activity of the LRU queue pairs can be monitored using the onstat utility.

Highlights

Engine administrators have always worked to tune checkpoint durations to as short an amount of time as possible.

As the MLRU portion of the queues grows, the modified data needs to be written to disk. Assuming for the moment that the instance is not using "fuzzy" checkpoints (see page 31), these write operations usually occur in conjunction with a checkpoint. When a checkpoint occurs, the buffer entries are sorted into chunk order, then written to disk. While these writes are highly efficient, depending on the number of modified buffers to be written, the operation could take anywhere from less than a second to several minutes to complete. This time interval is called the *checkpoint duration*. Since these types of checkpoints interrupt some end-user access to the engine, administrators have always worked to tune checkpoint durations to as short an amount of time as is reasonably possible.

One tuning method is to set an upper and lower bound for the percentage of buffers in an LRU pair that could be "dirty" or contain modified data needing to be written to disk. Called *LRU_MAX_DIRTY* and *LRU_MIN_DIRTY* respectively in $ONCONFIG, when the max limit is reached a trickle feed of data from the dirty buffers to disk begins. It stops when the *LRU_MIN_DIRTY* limit is reached or a checkpoint happens, whichever occurs first. These write operations are not as efficient as sorted chunk writes but the ability to trickle data to disk does help reduce checkpoint duration.

IBM IDS 9.4 enables administrators to use decimals with up to four places of precision instead of whole numbers when setting the LRU_MAX/MIN_DIRTY values, which helps reduce checkpoint duration.

IBM IDS 9.4 enables administrators to use decimals with up to four places of precision instead of whole numbers when setting the *LRU_MAX/MIN_DIRTY* values. Prior to this change, the lowest possible combination (other than 0/1 that continually fed data out of the system) was "2" and "1". If the instance supported a large number of users or transactions and had a lot of buffers, even a 2/1 combination was too large to prevent end-user interruption during a checkpoint.

Highlights

With the ability to use decimal values, the administrator can set the max and min boundaries to optimize checkpoint efficiency.

For example, if the instance had 750,000 buffers, a *LRU_MIN_DIRTY* value of 1 percent would leave 7,500 buffers to be flushed to disk at a checkpoint. Conservatively assuming that the instance uses at least 1 percent of its buffer pool per minute, the engine would have to flush at least 7,500 buffers per minute, to stay within the max and min boundaries. With the ability to use decimal values, the administrator could set the boundaries at 1.125 percent and 1 percent, triggering writes at 8,438 buffers but stopping at 7,500, or about 900 buffers every eight seconds.

The buffers whose addresses are in the LRU queues store more than just data for end-user operations. Tables or fragments thereof can be declared "memory-resident" and stored in the buffers. Index pages that are regularly used can also be stored in buffers for quicker index response. This can cause a problem in OLTP environments where many indexes are "wide"—containing a lot of fixed-length character columns, which generates an index that is larger than the table itself. In addition, based on the pre-IBM IDS 9.4 buffer design, cached index entries always included the root and most of the branch entries since they were used to get to leaf nodes. They were given a "high" priority value, which was used to determine what buffers remained in cache as opposed to being flushed out to accommodate new requests for space. The leaf nodes that actually pointed to data were given a lower priority for remaining in cache. Depending on the number of *BUFFERS* configured, this could result in "buffer thrashing," causing premature foreground data writes to disk and the flushing of leaf index pages out of the buffers to make room for other data only to have to reload the leaf index information again after the other operations completed.

Highlights

To help prevent buffers from being overrun with index or data use, IBM IDS 9.4 includes a new buffer management algorithm that balances the needs of high-priority and low-priority buffers.

IBM IDS 9.4 introduces a new buffer management algorithm that balances the needs of high-priority buffers (containing commonly used data) with low-priority buffers whose values are used less often. This prevents the buffers from being overrun with index or data use. Within these limits, the engine has the ability to dynamically shift buffers closer to or farther away from being flushed and reused based on actual use. The process is simple—the more times the values of a specific buffer are used, the longer it stays in place. As a buffer's values are used less and less often, the buffer is migrated to the end of the list where it is freed. Its address then returns to the "most-recently used" side of either the FLRU or MLRU, depending on what happened to the data it contained. In this way, infrequently used branch index node entries are gradually dropped from the buffers. With these entries out of the buffer pool, there is more space to cache data for end-user operations or other index entries.

Shared-statement cache

IBM IDS 9 uses a SQL statement cache to cache query access plans so that similar statements do not need to be processed by the query optimizer every time they're executed.

It is not uncommon in OLTP environments to have more than one user session executing the same query, albeit with different data values in the WHERE clause. For example, an order-processing system application repeatedly queries the "customer" table for billing and shipping information, the "items" table for product- and stock-level information. Rather than treating each query as a "never-before-seen" entity, IBM IDS 9 uses a SQL statement cache to cache query access plans so that similar statements do not need to be processed by the query optimizer every time they're executed. Instead, they can use preexisting query plans.

Highlights

The SQL statement cache can be configured to dynamically increase and decrease its size as needed, within pre-set boundaries, to support valid entries.

The engine administrator can configure and tune the SQL statement cache to function optimally.

Primarily of benefit to queries that have complex WHERE clauses, the SQL statement cache only accepts SQL DML statements that:

- *Connect to the local instance using built-in data types*
- *Does not contain an embedded sub-select*
- *Is not executed from within a stored procedure or user-defined function*
- *Does not create an explicit temporary table.*

The cache can be configured to dynamically increase and decrease its size as needed, within pre-set boundaries, to support valid entries.

After enabling the cache, an SQL operation is first hashed to see if it already exists in the cache. Several factors can affect the statement's hash value, including the case (upper versus lower) of the statement's text, any optimization directives or parallel data query (PDQ) prioritization. Some elements of the statement are ignored so that the hashing process can focus on finding commonality in the statement. This includes variable names or placeholders representing values in the WHERE clause. If a hash match is found, the existing plan is used and immediately executed. If a hash match is not found, the statement is evaluated for inclusion in the cache. If all tests prove true, it is parsed and prepared by the optimizer, and the plan is entered into the cache while the statement is executing.

The engine administrator can configure and tune the SQL statement cache to function optimally. Parameters include the initial and total size of the cache, the number of times a statement must be executed before it's entered into the cache, and the number of cache pools created. Statement caching can be turned on or off at an instance, session or statement level through $ONCONFIG parameters, environment variables, onmode commands or SQL statements. The cache can also be flushed to force reoptimization of all statements to pick up changes in table statistics.

Highlights

The shared statement cache reduces the cost of executing SQL operations and frees up memory that would otherwise be used to store query plans.

IBM IDS 9 supports new technology, called "fuzzy" checkpoints, that significantly reduces checkpoint duration.

The use of the shared statement cache reduces the cost of executing SQL operations and frees up memory that would otherwise be used to store query plans. The net benefit is the faster execution of SQL operations.

Fuzzy checkpoints

During a standard, or "sync" checkpoint, like that discussed in the LRU fractional values and buffer management section of this paper, certain user activities are interrupted while data is flushed from the shared memory buffers to disk and the physical log and its buffers are cleared. The greater the number of buffers to flush, the longer the interruption, which can be significant for OLTP applications and users accustomed to almost instantaneous response times. IBM IDS 9 supports new technology, called "fuzzy" checkpoints, that significantly reduces checkpoint duration.

With fuzzy checkpoints enabled, certain SQL operations such as *INSERT*, *DELETE* and *UPDATE* are reclassified as fuzzy operations. During a fuzzy checkpoint, buffers containing results from fuzzy operations have their addresses recorded in a new cross-reference table created in memory. This table's purpose is to record not only the buffer address associated with the transaction but also the position in the logical log where the completed transaction was recorded. As the fuzzy checkpoint ends, the cross-reference table is written into the logical logs and any outstanding non-fuzzy operations, such as table changes, are written to disk. These two operations occur quickly, enabling the engine to continue processing user requests. Dirty buffers containing transactional data whose addresses were written into the cross-reference table are, over time, gradually flushed to disk. The engine ensures that the buffer flush does not fall too far behind current processing through a series of markers placed in the logical logs. This flush to disk is independent of any buffer clearing that would occur due to *LRU_MAX/MIN_DIRTY* parameters.

Highlights

Instance recovery with fuzzy checkpoints enabled is heavily dependent on the logical logs and their archiving. With data synchronization to disk lagging behind transaction closure, the logical portion of a restore (which checks for the need to reapply transaction information recorded in the logical logs to disk) is critical. In beginning to reprocess the logical log entries, some might not be available within the instance because the log was reused. In this case, it would be imperative that the logs were properly archived to tape or other media so they could be retrieved.

With fuzzy checkpoints, BUFFERS can be increased, without penalty, so that more data can remain cached in instance memory for quicker response.

With fuzzy checkpoints, *BUFFERS* can be increased so that more data can remain cached in instance memory for quicker response without the corresponding penalty of having to flush all the buffers during a checkpoint.

Availability/replication/backup and restore

Because it can dynamically adjust a number of operational resources to meet changing workloads, the IBM IDS engine helps minimize administration overhead and virtually eliminates the need for planned outages.

The IBM IDS engine has a number of features and functions whose goal is to minimize unnecessary interruption to the data-processing environment. Since the engine can dynamically adjust a number of operational resources to meet changing workloads, it significantly reduces the overhead required to administer the engine and eliminates most of the planned outages required by other database products to maintain or tune the environment. An administrator, without interrupting user activity, can manually adjust most of the remaining parameters. The engine also provides its own disk mirroring functionality to duplicate critical data providing automatic disk fail-over capabilities. Instances can be backed up without interrupting users and without having to disassociate a set of disks. Partial restores can also be effected under normal workloads. Data consistency is provided through transaction logging and internal consistency checking, as well as by enforcing locking procedures, isolation levels and business rules. Data can be replicated from server to server in either a peer-to-peer group or enterprisewide with any level of granularity or direction. In IBM IDS 9.4, many of these features and functions received important upgrades or changes intended to enhance the engine's reliability and serviceability.

Highlights

Dynamic logical logs

A series of logs, called the *logical logs*, are used to capture transactional information. In earlier releases, the number, size and storage location of the logs was fixed at instance start-up and could only be modified during a maintenance window. Administrators had to tune these logs for expected usage as well as for the odd runaway transaction that filled a significant portion of the logs. On rare occasions, if the instance suffered an uncontrolled shutdown, sufficient log space might not be available during instance start up for previously opened transactions to roll back without intervention from support.

The IBM IDS 9 engine can now dynamically add and activate logical logs as needed without interrupting user activity or requiring a Level 0 backup.

The IBM IDS 9 engine can now dynamically add and activate logical logs as needed without interrupting user activity or requiring a Level 0 backup. Administrators can also add and drop logs manually without user interruption or Level 0 backup. Log activation is automatic, similar to logs added dynamically by the engine.

Logs added dynamically by the engine are inserted into the middle of the log stack, just after the current log, so that they are used next, preventing suspension of instance activity. Logs added by an administrator, through the onparams or ISA utility, can be inserted either after the current log or at the bottom of the stack. The engine uses a simple hierarchy of rules to determine which dbspace to use when dynamically adding a new log. The engine administrator can tune this feature so that it waits rather than automatically adding a new log in case the administrator wants to use a specific dbspace for new logical log use. Dynamic logging can be turned on or off as needed. For example, if there isn't enough log space to roll back transactions during an instance restart, the administrator can turn dynamic logging on and then restart the instance. Logs are automatically added to complete the rollback enabling the instance to come online—assuming the instance is in good condition.

While this functionality neither eliminates long transactions nor their filling of the logical logs, it does mitigate their impact to the instance, particularly at instance restart.

Highlights

The IBM IDS engine uses a fast-recovery process to verify transaction integrity and, if necessary, reapplies the transactions.

Restartable fast recovery

In instances that use sync checkpoints, the physical log is used to record "before" images of data pages that are about to be changed. The pages are flushed from the physical log during a checkpoint and, at the end of a sync checkpoint, the instance is physically and logically consistent. During an instance restart, the physical log is checked to see if it contains any images. If so, it indicates that the instance was not shut down cleanly and that transaction integrity must be checked. The engine uses a fast-recovery process, which includes the reapplication of physical log pages to disk followed by a check of the logical log records generated after the last completed sync checkpoint to see if the committed changes they contain exist on disk. If not, the transaction is reapplied.

Prior to IBM IDS 9.4, physical logging was not active during the fast-recovery process for all operations. If the fast-recovery process was interrupted before it completed, it might not have been able to restart and complete successfully. For example, during the reapplication of log records in a fast recovery a logical log record is written to disk. Prior to IDS 9.4, a before image was not written into the physical log for this change. Suppose the physical server crashes before the fast-recovery process completes. When the server is restarted and the instance fast-recovery process begins again, an attempt is made to reapply all the logical records from the last recorded checkpoint. The recovery process would fail when attempting to apply a change to disk on which the change had already been written. Another failure situation occurs if multiple rows on a page are affected by numerous transactions and a fast recovery rollback operation for one of the last transactions on that page

Highlights

is interrupted. Prior to IDS 9.4, when rolling back a transaction during fast recovery a before image was written into the physical log. In this example though, the image would contain changes made by committed transactions from earlier in the logical log. If the physical server fails before the transaction rollback completes, when the server is restarted and the instance fast recovery begins again, the before image that is reapplied to disk contains committed changes that were made prior to the rollback. When fast recovery attempts to apply the first logical log records for that page, it will fail; the change is already on disk.

IBM IDS 9.4 now captures all before images of pages during the fast-recovery process to support fast-recovery process completion with total transactional integrity—even if interrupted.

IBM IDS 9.4 now captures all before images of pages during the fast-recovery process to support fast-recovery process completion with total transactional integrity—even if interrupted. Now, during the roll-forward phase of recovery, checkpoints are disabled until all records are applied to prevent premature buffer and log flushing to disk. In addition, in the event more physical log records are generated than can be contained in the log itself, overflow records are written and read from the file specified by the *PLOG_OVERFLOW_PATH* $ONCONFIG parameter. If this parameter is not set, the engine default is to use $INFORMIXDIR/tmp.

Redirected restores

IBM IDS 9.4 now supports the substitution of new physical path names for storage devices during a cold restore to either shift data between physical devices or to restore to another system.

IBM IDS 9.4 now supports the substitution of new physical path names for storage devices during a cold restore. With this functionality, administrators can relocate instances to faster devices within a storage subsystem, easily migrate an instance to another machine without unloading and reloading data, change path names to reflect newer naming standards in the enterprise, and complete a restore on the same server even though some of the original devices are not available.

Highlights

Redirected restores are possible through both the ON-Bar and ontape utilities.

Redirected restores are possible through both the ON-Bar and ontape utilities when the "-rename" flag is set. During the initialization of a cold restore, the device names are read from the chunk-reserved page of the rootdbs captured to tape. Under normal conditions, the restore operation validates that the devices are accessible, writes the appropriate device information to disk to support the restore, and then begins writing to the devices. If the "-rename" flag is set, the restore operation compares the original device information with the new targets for compatibility then substitutes the new device information when writing the restore support information to disk.

Creating a redirected restore requires entering the original device path and offset as well as the new device path and offset. For a single device, this can easily be done at the command line. For example:

ontape –r –rename –p <old_chunk_path> -o <old_offset>
-n <new_chunk_path> -o <new_offset>. . .

onbar –r –rename –p <old_chunk_path> -o <old_offset>
-n <new_chunk_path> -o <new_offset>. . .

If multiple devices need to be relocated, a filename can be substituted by using "-f <file_name>" in conjunction with the "-rename" flag. For example:

onbar -r -rename -f <file_name>. . .

In the file, the old device path, old offset, new device path and new offset must be listed using one device pair to a line with whitespace or tab separation. Other file requirements include using only one device pair per line, and ignoring trailing and leading whitespaces or tabs as well as empty lines. Comments can be included in the file if they are preceded with a "#".

Highlights

When redirecting the restore, all chunk devices in a dbspace do not need to be redirected. The redirection occurs at a chunk level so only those chunks that must be moved need to be listed. If IBM IDS mirroring is used, those chunks have significance and must be explicitly redirected on a chunk-by-chunk basis like primary chunks.

If the chunks supporting the rootdbs (primary or mirror) are redirected, the appropriate changes will automatically be made in $ONCONFIG. A copy of the original $ONCONFIG will be made in the $INFORMIXDIR/etc directory. Other messages related to the redirection and the restore are written into $MSGPATH.

Once a redirected restore has completed, a new Level 0 backup must be taken to capture the new device information.

Additional chunk information to full rootdbs structures

Concurrent with the changes enabling more and larger individual chunks, IBM IDS 9.4 also changes how chunk-related information is stored in the rootdbs. Previously, all chunk descriptor information had to be stored in the primary or mirror chunk pages in the reserved section of the rootdbs. If full with information about allocated chunks, these two sections of pages could be extended provided there were sufficient unused pages within the reserved section. If not, new chunks could not be added to the instance.

With IBM IDS 9.4, chunk reserve pages can be extended with any open set of pages from the rootdbs, which allows additional chunks to be added to the instance.

With IBM IDS 9.4, chunk reserve pages can be extended with any open set of pages from the rootdbs. This allows additional chunks to be added to the instance, provided there are unused pages in the rootdbs and the instance has not reached the maximum number of supported chunks.

Highlights

In IBM IDS 9.4, not only have the HDR and ER technologies been improved, but their use is no longer mutually exclusive.

HDR and ER

IBM IDS uses *High Availability Data Replication (HDR)* and *Enterprise Replication (ER)* data replication technologies. HDR is primarily intended for immediate fail-over support to provide high-availability services to applications. Using synchronous or asynchronous transfers of log information, HDR creates a complete copy of a logged instance to another peer server. ER, as its name suggests, is used to replicate subsets of data throughout the enterprise for data consolidation, distribution or performance reasons. HDR supports a primary/target directly connected topology. Because ER provides enterprise-wide data replication, it supports additional network topologies such as fully connected, star and tree topologies.

In IBM IDS 9.4, not only have the functionalities of each technology been improved, but their use is no longer mutually exclusive. ER and HDR can be run simultaneously within the same instance to provide both disaster recovery protection and comprehensive data distribution. ER-related configuration can only be executed on the primary node of an HDR pair but will be replicated to the secondary server in the pair via HDR. Since any node within the ER domain can be defined to be part of an HDR pair, this merging of technologies allows the flexibility of ER while providing the availability of HDR.

HDR in IBM IDS 9.4 now supports the replication of all UDTs, including built-in UDTs, and ER in IBM IDS 9.4 now supports all extensible data types.

HDR in IBM IDS 9.4 now supports the replication of all UDTs including built-in UDTs, such as collection, boolean, row and SLOs, with a couple of caveats. If the access methods are not stored in the database, but reside in outside files (for example, C routines), information about the file will be replicated but the file itself will not. If the UDT has data components that are not logged (typically SLOs), those components will not be replicated either. HDR will replicate indexes created using UDTs as well as R-tree indexes. There is conditional support for replicating UDRs. The IBM Informix TimeSeries DataBlade is now supported, in addition to several of the core DataBlades.

Highlights

The individual transaction size that can be replicated using ER has been increased to 4TB through improvements in the shared memory management and spooling functions.

ER in IBM IDS 9.4 now supports all extensible data types, such as row and collections as well as serial and serial8 types. The individual transaction size that can be replicated has been increased to 4TB through improvements in the shared memory management and spooling functions. The converted queue manager and transactional statistics have also been converted to 64-bit implementations.

Considerable work has been done to ER to provide support for autonomic dynamic administration. The net result has been a significant improvement in DataSync performance as well as an improvement in overall stability. For example, the DataSync adjusts the amount of resources it uses based on its own statistical history. In addition, as the "log snooping" process monitors logical log usage and senses that an increase in log use has increased the risk of a log wrap, it will issue warnings that additional log space is needed. If the logs do fill, it invokes the dynamic insertion of a logical log provided the *DYNAMIC_LOGS* $ONCONFIG parameter has been enabled.

With the appropriate options set in $SQLHOSTS and $ONCONFIG, ER traffic can be encrypted, enabling transmission across public networks.

With the appropriate options set in $SQLHOSTS and $ONCONFIG, ER traffic can be encrypted, enabling transmission across public networks.

Metadata stealing

The addition of smart blobspaces (sbspaces) to store SLOs added new requirements to the sizing process that occurs when allocating space to store data. Administrators need to consider the size of the sbspace metadata area and whether or not it is big enough to hold all of the address entries for the expected number of SLOs. This preallocation of pages must be factored into the total sizing estimate of the sbspace so its data area is large enough for the expected number of SLOs. In most cases, administrators can rely on the engine-sized metadata area created based on input parameters in the sbspace creation command. Unfortunately, if the metadata area size turns out to be too small and fills up with entries, sbspace activity will be suspended. Until now, the only work-around was adding a new chunk to the sbspace and, optimally, reserving its pages for metadata use only.

Highlights

IBM IDS now provides metadata stealing to eliminate the end-user impact of a poorly sized metadata area.

IBM IDS now provides *metadata stealing* to eliminate the end-user impact of a poorly sized metadata area. Within the data pages area of a sbspace, 40 percent of the pages will be flagged as dual-purpose pages. They can either be metadata or user-data pages. If the original allocation of metadata space fills up, it will dynamically expand by using portions of these dual-purpose pages. If the metadata area continues to grow and all dual-purpose pages are used, activity in the sbspace will be suspended until more metadata space is added to the sbspace. Administrators can monitor the dynamic expansion of metadata areas in sbspaces through messages in $MSGPATH.

Application-oriented technologies

The IBM IDS engine is used extensively for developing and supporting high-quality and critical line-of-business applications. IBM IDS 9.4 includes several new features and technologies that may be of interest to application developers.

Client communication encryption

IBM IDS 9.4 adds the ability to encrypt all client/server communication with software developed by the OpenSSL Project for use in the OpenSSL Toolkit.

As enterprises have grown, locations have become more dispersed and more employees have begun working from home or on the road, which means that applications and data are being distributed farther from each other. Consequently, data must sometimes be accessed from across public networks. Previous releases of IBM IDS have supported the encryption of passwords sent from clients to the engine for access validation through the Communication Support Module (CSM). IBM IDS 9.4 adds the ability to encrypt all client/server communication with software developed by the OpenSSL Project for use in the OpenSSL Toolkit (www.openssl.org). It is based on OpenSSL 0.9.6g.

Managed through options set in the $SQLHOSTS file, IBM IDS 9.4 provides strong network encryption using many of the popular cyphers such as DES (Data Encryption Standard), triple DES and multiple versions of the Blowfish cypher. Additional message security is provided by use of SHA1/HMAC message authentication and periodic cypher/key renegotiation. All cyphers use block chaining to further increase the strength of encryption.

In addition to session encryption, data replication occurring with ER can be encrypted as it is transferred between nodes.

Highlights

IBM IDS conforms to the GLS Level-4 specification, which allows it to collate character strings, print dates and accept monetary input in the rules and formats required by the country where it's being used.

New Unicode and GLS support

IBM IDS Global Language Support (GLS) conforms to the GLS Level-4 specification, a coding standard allowing multi-byte characters. By providing GLS support, IBM IDS can collate character strings, print dates and accept monetary input in the rules and formats required by the country where products are being used. Furthermore, GLS provides worldwide support of database applications so they can be migrated to multiple languages while maintaining the same functionality.

IBM IDS 9.4 supports and uses the International Components for Unicode (ICU) extensions to GLS Level 4, which supports localized collation as well as the full implementation of GB18030 for the complete Chinese character set. In addition, the database limitation of a single GLS collation order has been removed. Applications can now declare the desired collation order on the fly with the *SET COLLATION* SQL command.

IBM IDS 9.4 adds several new enhancements and compatibility features to SQL processing that help simplify the work required to create full-featured applications as well as simplify migrating applications from other database engines to IBM IDS.

SQL enhancements and compatibility

IBM IDS 9.4 brings several new enhancements and compatibility features to SQL processing. Some of these features simplify the work developers must do to create robust and full-featured applications; others further simplify the process and decrease the pain involved in migrating applications from other database engines to IBM IDS.

Application developers often build nested sets of cursor-driven queries using the results of outer queries to drive inner queries. If the results returned by the innermost query are updated, unless all the cursors are opened with the WITH HOLD SQL keywords, the transaction commit closes all the cursors and releases their resources. Prior to IDS 9.4, the statement or engine's PDQ setting was ignored when processing transactions generated through these types of cursors. The optimizer and Memory Grant Manager in IBM IDS 9.4 will now consider the PDQ settings for transactions executed by "with hold" cursors and ensure they are executed with the requested parallelized resources.

Highlights

The query rewrite function in IBM IDS 9.4 has been enhanced to enable sorting by column(s) not in the SELECT clause of a query.

The query rewrite function in IBM IDS 9.4 has been enhanced to enable sorting by column(s) not in the *SELECT* clause of a query. Prior to this release, even if the application didn't need the results of a particular column, the column had to be in the query (and its values returned) for the result set to be sorted by the column's values. With this new functionality, a query such as the following is possible:

```
SELECT emp_last_name,
emp_first_name,
emp_department_number
FROM employee
WHERE emp_active = TRUE
SORT BY emp_hire_date;
```

This query would return active employee names and department numbers sorted by their hire date. When parsing this type of query, the optimizer will rewrite the query to include the *SORT BY* column in the *SELECT* clause so its values are captured. After sorting, the *SORT BY* column values will be stripped from the result set before the result is returned to the application.

With respect to Web-server-driven applications, IBM IDS 9.4 now supports the right outer and cross joins.

With respect to Web-server-driven application servers, such as *IBM WebSphere® Application Server*, IBM IDS 9.4 now supports the *right outer* and *cross* joins. The latter join creates a Cartesian product of the rows in the joined tables while the former changes the table dominance within the join. Normally, the table name following the *JOIN* keyword in a SQL operation is the subservient table. With the right outer feature, the table following the join would be the dominant table with each of its rows being returned even if there were not a corresponding *JOIN* match within the other table.

Highlights

IBM IDS 9.4 introduces a sequence generator to create unique numbers that are instance rather than table driven.

IBM IDS 9.4 introduces a *sequence generator* to create unique numbers that are instance rather than table driven. IBM IDS has long supported the creation and automatic incrementing (or decrementing) of a serialized column within a table. Often used to create unique values, such as an order or customer number, these identifiers were table-specific with no inherent significance across the database. When a universally unique value was required, a table with just a serial or serial8 column was created. To use it, however, required obtaining an exclusive lock on the table in order to increment and capture the next value. This serialized approach presented performance issues and limited its usefulness in large applications. With the new sequence generator, parallelized access is available. The generator can be configured to increment or decrement by any desired interval and its values can be either non-cyclical (and unique) or cyclical, once the domain range has been met. If necessary, more than one sequence generator can be created in the instance.

IBM IDS 9.4 gives database designers and application developers the ability to create INSTEAD OF triggers on views, so that SQL operations previously denied on views can occur.

Generally speaking, triggers are most commonly executed in conjunction with SQL data operations on tables. Whenever the triggering action occurs, be it an insert, delete, modify or select, the trigger's functionality is executed—typically through a call to a UDR with its more robust functionality. IBM IDS 9.4 gives database designers and application developers the ability to create *INSTEAD OF* triggers on views so that SQL operations previously denied on views can occur. For example, in most cases views are query-only entities; new data cannot be inserted nor existing data modified in the underlying table(s) through the view. With an *INSTEAD OF* trigger on the view, an operation executed against the view will invoke the trigger which will most likely call a UDR with the data passed to the trigger. The UDR will then modify the individual table(s) that the view covers. The functionality of these triggers is somewhat restricted. For example, they are limited in scope to one action, meaning they cannot have *BEFORE* and *AFTER* functionality. The triggering actions are limited to insert, delete and modify operations, and must execute *FOR EACH ROW*.

Highlights

With IBM IDS 9.4, *union* operations are not limited to the *SELECT* clause of a query. Union functionality can be invoked in the *FROM* clause or within a sub-select in the *WHERE* clause.

Finally, prior to IBM IDS 9.4 data returned from stored procedures or DB-Access had column headings of "expression." For some application interfaces, such as OBDC and JBDC, this caused issues that had to be handled programmatically. IBM IDS 9.4 provides the ability to provide names for the result columns with the AS keyword. For example:

```
CREATE PROCEDURE new_names (parm_1 INTEGER, parm_2
CHARACTER(20)) RETURNING out_1 AS first_name, out_2 AS
last_name;

:

END PROCEDURE;
```

The identifiers used as return names have no significance within the procedure other than as a column name pushed with the result set. A duplicate of the identifier can be used as a variable or parameter name within the procedure. All return columns must have a name attached or no names will be returned.

New "explain" mode

IBM IDS better facilitates rigorous testing of the design and application logic of new database applications as they are being developed by providing the ability to interrupt statement processing.

During the design and development of new database applications, rigorous testing of the design and application logic needs to occur. As components of the application are ready, ideally they are tested against ever-increasing subsets of data to help ensure that they, and the database, react as expected. A significant portion of the testing process is the staging and re-creation of data for each test iteration. As the data loads increase, the shift turns from application functionality to database performance. With earlier versions of the engine, there was no way to review query plans created by the optimizer without actually executing the operation. While this worked in test environments where test data could be reconstructed, it prevented testing applications against production data to see how the engine would respond under full loads.

Highlights

IBM IDS now provides the ability to interrupt statement processing after the optimizer "prepare" phase completes building the query plan and outputs it for review. This feature can be enabled on a statement-by-statement basis with an optimizer directive or turned on (and off) for a block of statements with a *SET EXPLAIN* SQL command.

Management technologies

IBM IDS includes a System Master Interface, which provides administrators with monitoring access to every facet of engine activity, helping them to identify problems, monitor resource usage or track session activity.

The administration features of IBM IDS allow administrators to manage instances easily and efficiently while keeping data online and available. A key component to engine administration is the system master database, which holds information about the instance. Some of this information is transitive, meaning it resides in shared memory and is lost when the instance is shut down, and some is kept on disk mainly for recovery purposes. The System Master Interface (SMI) is used as a query interface to access the real tables and pseudo-tables in shared memory. Through the SMI, administrators have monitoring access to every facet of engine activity and can use it to identify problems, monitor resource usage or performance, or track user session activity. The onstat utility and Informix Server Administrator are used as front ends to this database.

Informix Server Administrator

IBM ISA provides an easy-to-use, browser-based interface for almost all database administrative tasks.

The *IBM Informix Server Administrator (ISA)* is a browser-based, cross-platform database server administration tool. IBM ISA provides an easy-to-use interface for almost all database engine administrative tasks. As a graphic front end to the onstat and other utilities, IBM ISA relieves new administrators of the need to learn all the various flags and permutations of the commands. The output displayed by the ISA is fully cross-referenced to other relevant pieces of information, enabling click-and-dive analysis. Written in Perl, ISA allows administrators to add their own modules to it to customize it to their needs. The ISA can be used to manage all instances, including *IBM Informix MaxConnect*™ instances from a single Web page. It is included at no additional charge on the engine distribution media.

Highlights

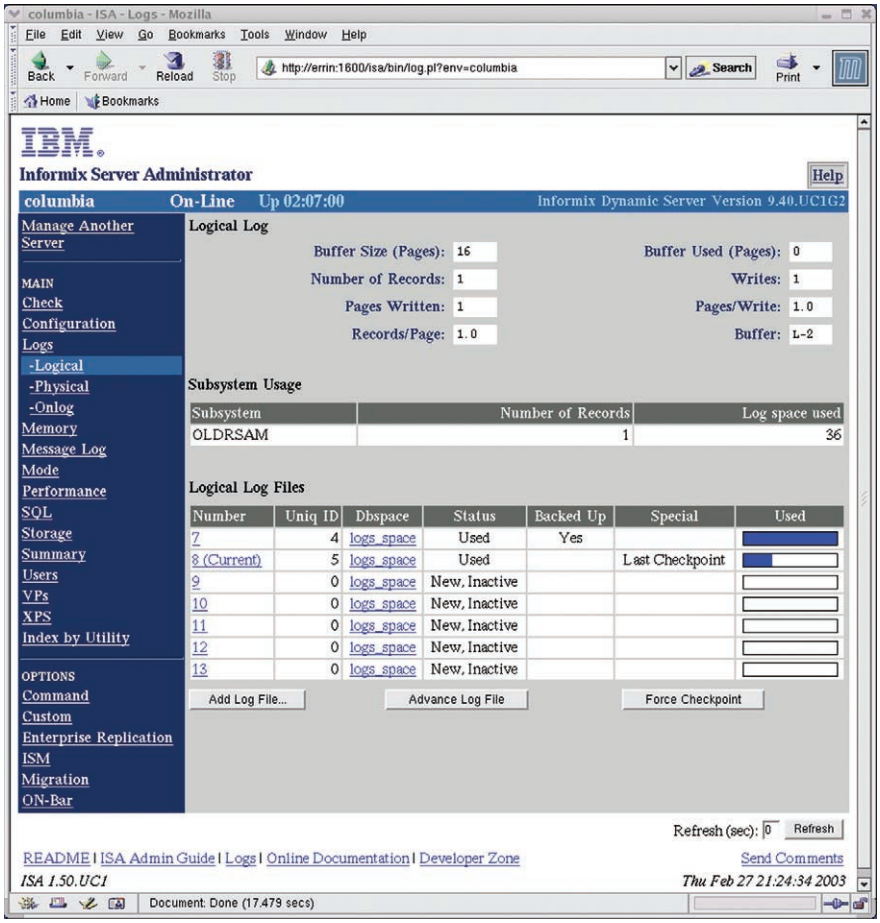

Figure 5:
IBM Informix Server Administrator enables point-and-click engine administration and monitoring functionality.

Server Studio JE provides an open-architecture tool for the management of databases and database objects.

Server Studio JE

Server Studio JE provides an open-architecture tool with an easy-to-use interface for the management of databases and database objects. The utility is composed of several base components, including the Database Object Explorer for object creation and modification plus the management of server connections; an SQL editor for writing, executing and reviewing the results of SQL operations and the creation of stored procedures; and a table editor for the creation and management of tables and indexes.

Highlights

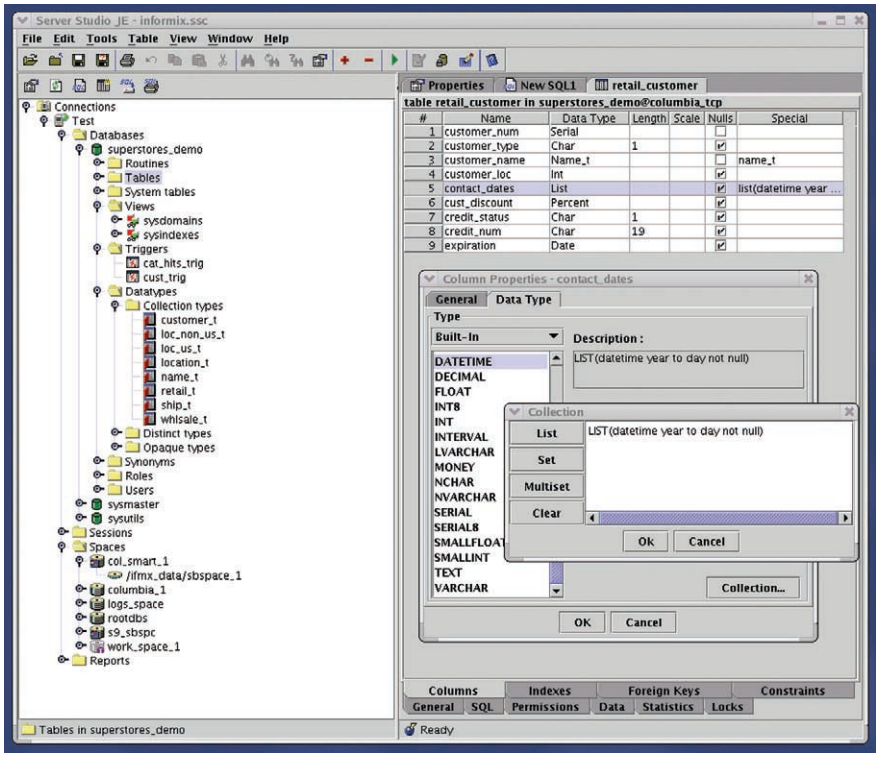

Figure 6:
Server Studio JE provides an easy-to-use interface to manage database objects.

Server Studio JE utility and base components are distributed at no additional charge, and support additional modules, which can be purchased, including the DB Difference Analyzer and the Dependency Analyzer.

The utility and base components are distributed at no additional charge, and they support additional modules, which can be purchased and added. The modules include the DB Difference Analyzer, which compares two database schemas and generates a DDL file to synchronize them, and the Dependency Analyzer, for analyzing referential constraints and other data dependencies. Additional functional enhancements to the bundled SQL editor and table manager can be purchased as well.

Highlights

The UNIX Bundle Installer, which is included in the IBM IDS engine, simplifies installation in UNIX environments by providing a way to easily install components in the correct order.

Utilities

There are a number of administrative utilities bundled with the engine. These include the *UNIX Bundle Installer*, which simplifies installation in UNIX environments by providing a way to easily install components in the correct order. Prior to IBM IDS 9.4, to install the engine an administrator had to extract the engine, connectivity libraries, the ISA, DataBlades and so on from the installation media to disk. While standard UNIX commands were used (i.e., *cpio* and *tar*), there was no consistency; some packages used cpio and others tar. Once extracted, the components needed to be installed in the correct order by executing their install scripts. Over the years, that order changed and it was important to read the release notes to verify the installation order. With the UNIX Bundle Installer, all the packages are in one distribution file that gets "tar-ed" to disk. Once in place, the administrator executes one command to launch the installation of all products. The installer will verify which packages exist and then ask what should be installed. If desired, the installer will even create and initialize a simple instance. The administrator simply selects the options needed and the installer does the rest of the work. When the installer is finished, the administrator can initialize a new instance or work with the one created by the installer.

IBM IDS includes an archecker utility, which can check the consistency of database backups without executing a restore.

The *archecker* utility checks the consistency of database backups without executing a restore. The *onsmsync* tool synchronizes backup objects maintained by the ON-Bar database, the emergency bootfile and the storage manager database to optimize backup handling and improve restore times. The oncheck utility performs consistency checks on data and indexes and can be used while the instance is online with active user sessions.

Highlights

IBM IDS 9.4 provides an array of industry-leading features and functionality that allow database professionals to create databases in a way that is meaningful to business users, and application developers to quickly and easily build robust applications.

Conclusion

The powerful and extensible IBM Informix Dynamic Server delivers breakthrough scalability, manageability and performance. IBM IDS users and administrators benefit from the performance and scalability offered by the proven Dynamic Server Architecture, as well as reap the advantages provided by object-oriented technology and unlimited extensibility. With these features and functionality, database professionals can create databases in a way that is meaningful to business users, and application developers can quickly and easily build robust applications. IBM IDS 9.4 provides an immense capacity to grow and adapt to ever-changing business needs.

For more information

To learn more about IBM IDS 9.4, additional information management technologies, and world-class IBM customer support and services, please contact your local IBM sales representative, or visit:

ibm.com/informix

Highlights

IBM IDS maintains industry-leading performance levels through multiprocessor features, shared memory management, efficient data access and cost-based query optimization.

IBM IDS is available on many hardware platforms and can be easily migrated to more powerful platforms as needs change.

Appendix A: Exploiting system power—a review of the DSA

High system performance is essential for maintaining maximum throughput. IBM Informix Dynamic Server (IDS) maintains industry-leading performance levels through multiprocessor features, shared memory management, efficient data access and cost-based query optimization. IBM IDS is available on many hardware platforms and because the underlying platform is transparent to applications, the engine can migrate easily to more powerful computing environments as needs change. This transparency enables developers to take advantage of high-end SMP (symmetric multiprocessing) systems with little or no need to modify application code.

Database engine architecture is a significant differentiator and contributor to the engine's performance, scalability and ability to support new data types and processing requirements. Almost all database engines available today use an older technological design that requires each database operation for an individual user (for example, read, sort, write, communication, etc.) to invoke a separate operating system process. This architecture worked well when database sizes and user counts were relatively small. Today, these types of engines spawn many hundreds and into the thousands of individual processes that the operating system must create, queue, schedule, manage/control and then terminate when no longer needed. Given that, generally speaking, any individual system CPU can only work on one thing at a time—and the operating system works through each of the processes before returning to the top of the queue—this engine architecture creates an environment where individual database operations must wait for one or more passes through the queue to complete their task. Scalability with this type of architecture has nothing to do with the software; it's entirely dependent on the speed of the processor—how fast it can work through the queue before it starts over again.

Highlights

The DSA design architecture supports the most efficient use of available system resources through advanced capabilities such as built-in multi-threading and parallel processing.

IBM IDS provides the unique ability to scale the database system by employing a dynamically configurable pool of database server processes called virtual processors.

The IBM IDS engine architecture is based on advanced technology that efficiently uses virtually all of today's hardware and software resources. Called the Dynamic Scalable Architecture (DSA), it fully exploits the processing power available in SMP environments by performing similar types of database activities (such as I/O, complex queries, index builds, log recovery, inserts and backups/restores) in parallelized groups rather than as discrete operations. The DSA design architecture includes built-in multi-threading and parallel processing capabilities, dynamic and self-tuning shared memory components, and intelligent logical data storage capabilities, supporting the most efficient use of all available system resources.

Processing

IBM IDS provides the unique ability to scale the database system by employing a dynamically configurable pool of database server processes called *virtual processors*. Database operations such as a sorted data query are broken into task-oriented subtasks (for example, data read, join, group, sort) for rapid processing by virtual processors that specialize in that type of subtask. Virtual processors mimic the functionality of the hardware CPUs in that virtual processors schedule and manage user requests using multiple, concurrent threads.

Highlights

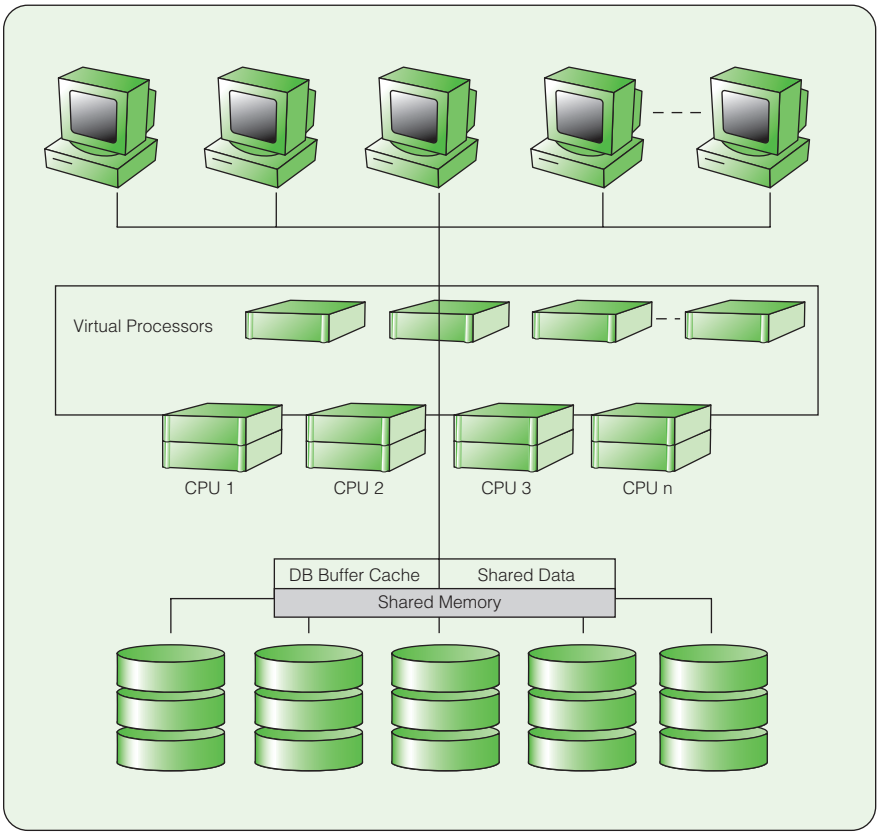

Figure a1:
IBM IDS has a configurable pool of database server processes called virtual processors, which can respond to any client's request.

Virtual processors are multi-threaded, which helps save precious CPU time by dynamically switching threads when tasks are completed.

A thread represents a discrete task within a database server process and many threads may execute simultaneously, and in parallel, across the pool of virtual processors. Unlike a CPU process-based (or single-threaded) engine, which leaves tasks on the system CPU for its given unit of time (even if no work can be done thus wasting processing time), virtual processors are multi-threaded. Consequently, when a thread is either waiting for a resource or has completed its task, a thread switch will occur and the virtual processor will immediately work on another thread. As a result, precious CPU time is not only saved, but it is used to satisfy as many user requests as possible in the given amount of time. This is referred to as *fan-in parallelism*.

Highlights

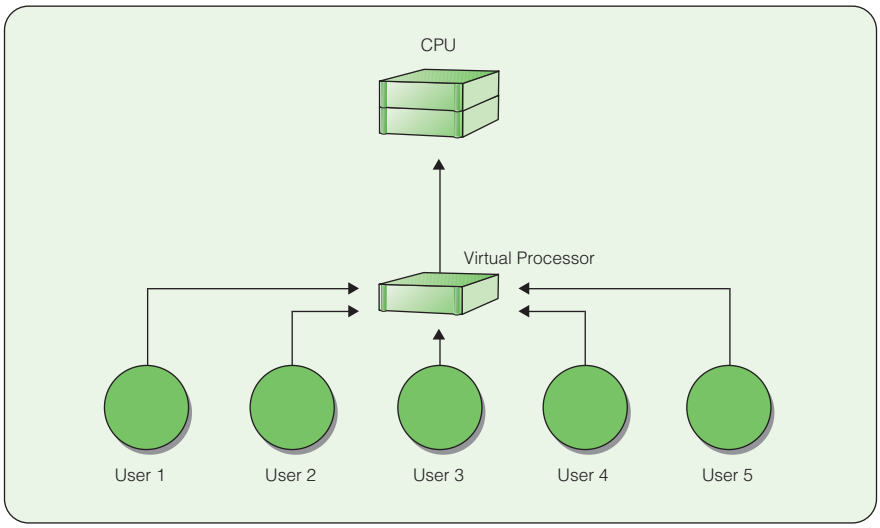

Figure a2:
A virtual processor can respond to many user requests.

Not only can one virtual processor respond to multiple user requests in any given unit of time, but one user request can also be distributed across multiple virtual processors. For example, with a processing-intensive request such as a multi-table join, the database server divides the task into multiple subtasks and then spreads these subtasks across all available virtual processors. With the ability to distribute tasks, the request is completed quicker. This is referred to as *fan-out parallelism*. Together with fan-in parallelism, the net effect is more work being accomplished quicker than with single-threaded architectures; in other words, the engine is faster.

Dynamic load balancing occurs within IBM IDS because threads are not statically assigned to virtual processors.

Dynamic load balancing occurs within IBM IDS because threads are not statically assigned to virtual processors. Outstanding requests are serviced by the first available virtual processor, balancing the workload across all available resources. For efficient execution and versatile tuning, virtual processors can be grouped into classes—each optimized for a particular function, such as CPU operations, disk I/O, communications and administrative tasks. An administrator can configure the system with the appropriate number of virtual processors in each class to handle the workload. Adjustments can be made while the engine is online without interrupting database operations in order to handle occasional periods of heavy activity or different load mixes.

For efficient execution and versatile tuning, virtual processors can be grouped into classes—each optimized for a particular function.

Highlights

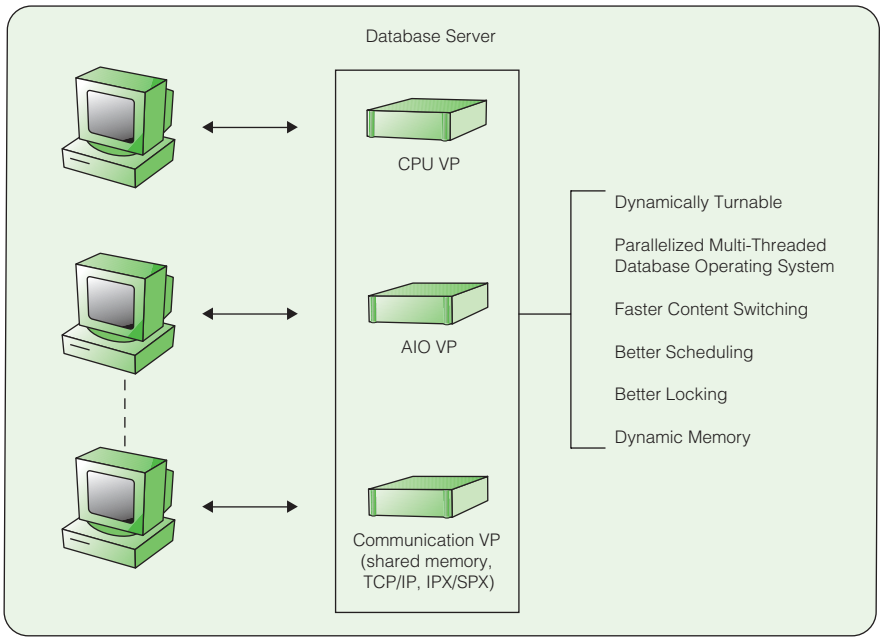

Figure a3:
Virtual procesors (VPs) are grouped into classes, which are optimized for a particular function. IBM IDS may be configured with the appropriate number of virtual processors in each class to handle the engine's workload.

Because IBM IDS includes its own threading capability for serving client requests, the actual number of Windows threads is decreased—reducing the system thread scheduling overhead and providing better throughput.

In UNIX and Linux systems, the use of multi-threaded virtual processors significantly reduces the number of UNIX/Linux processes and, consequently, less context switching is required. In Microsoft Windows systems, virtual processors are implemented as threads to take advantage of the operating system's inherent multi-threading capability. Because IBM IDS includes its own threading capability for servicing client requests, the actual number of Windows threads is decreased—reducing the system thread scheduling overhead and providing better throughput.

In fully utilizing the hardware processing cycles, IBM IDS engines do not need as much hardware power to achieve comparable to better performance than other database engines.

Highlights

All memory used by IBM IDS is shared among the pool of virtual processors.

Memory

All memory used by IBM IDS is shared among the pool of virtual processors. Beyond a small initial allocation of memory for engine-level management, usually a single shared memory portion is created and used by the virtual processors for all data operations. This portion contains the buffers of queried and modified data, sort, join and group tables, lock pointers, etc. Should database operations require more (or less) shared memory, additional segments will be dynamically added and dropped from this portion without interrupting user activities. An administrator can also make similar modifications manually while the server is running. When a user session terminates, the thread-specific memory for that session is freed within the portion and reused by another session.

The buffer pool is used to hold data from the database disk supply during processing. When users request data, the engine first attempts to locate the data in the buffer pool to avoid unnecessary disk I/Os. Depending on the characteristics of the engine workload, increasing the size of the buffer pool can result in a significant reduction in the number of disk accesses, which can help significantly improve performance, particularly for online transaction processing (OLTP) applications.

Access to data in frequently used tables or indexes can be improved by keeping such tables or indexes "in memory."

Access to data in frequently used tables or indexes can be improved by keeping such tables or indexes "in memory." Depending on the amount of memory allocated to the engine, portions to entire tables or indexes can be kept "resident" in the database buffer pool, eliminating the need for physical disk I/O to satisfy requests for data. Any data that changes in these "resident" tables will be written out to disk to preserve logical consistency of the database.

Highlights

To speed up what is typically the slowest component of database processing, IBM IDS uses its own asynchronous I/O feature when available.

Disks

The parallelism and scalability of the DSA processor and memory components are supported by the ability to perform asynchronous I/O across database tables and indexes that have been logically partitioned. To speed up what is typically the slowest component of database processing, IBM IDS uses its own asynchronous I/O (AIO) feature, or the operating system's kernel AIO, when available. Because I/O requests are serviced asynchronously, virtual processors do not have to wait for one I/O operation to complete before starting work on another request. To ensure that requests are prioritized appropriately, four specific classes of virtual processors are available to service I/O requests: logical log I/O, physical log I/O, asynchronous I/O and kernel asynchronous I/O. With this separation, an administrator can create additional virtual processors to service specific types of I/O in order to alleviate any bottlenecks that might occur.

The IBM IDS read-ahead feature significantly improves the throughput of sequential table or index scans, and end-user applications spend less time waiting for disk access to complete.

The read-ahead feature enables IBM IDS to asynchronously read several data pages ahead from disk while the current set of pages retrieved into memory is being processed. This feature significantly improves the throughput of sequential table or index scans, and end-user applications spend less time waiting for disk accesses to complete.

Data partitioning

Table and index data can be logically divided into partitions, or fragments, using one or more "partitioning schemes" to improve the ability to access several data elements within the table or index in parallel as well as increase and manage data availability and currency. For example, if a sequential read of a partitioned table were required, it would complete quicker because the partitions would be scanned simultaneously rather than each disk section being read serially from the top to the bottom. With a partitioned table, database administrators can move, associate or disassociate partitions to easily migrate old or new data into the table without tying up table access with mass inserts or deletes.

Highlights

Table partitions in IBM IDS can be set and altered without bringing down the database server and, in some cases, without interrupting user activity within the table.

IBM IDS has two major partitioning schemes that define how data is spread across the fragments. Regardless of the partitioning scheme chosen, or even if none is used at all, the effects are transparent to end users and their applications. Table partitions can be set and altered without bringing down the database server and, in some cases, without interrupting user activity within the table. When partitioning a table, an administrator can specify either:

- *Round robin—Data is evenly distributed across each partition with each new row going to the next partition sequentially.*
- *Expression-based—Data is distributed into the partitions based on one or more sets of logical rules applied to values within the data. Rules can be "range" based, using operators such as "=", ">", "<", "<=", MATCHES, IN, and their inverses, or "hash" based where the SQL MOD operator is used in an algorithm to distribute data.*

Depending on the data types used in the table, individual data columns can be stored in different data storage spaces, or "dbspaces" than the rest of the table's data. These columns, which are primarily smart large objects, can have their own unique partitioning strategy that effectively distributes those specific columnar values in addition to the partitioning scheme applied to the rest of the table. Simple LOBs can and should be fragmented into simple blobspaces, however, because they are black-box objects, as far as the engine is concerned, no further fragmentation options are possible.

Partitioned indexes can be placed on a different physical disk than the data, resulting in optimum parallel processing performance.

Indexes can also be partitioned using an expression-based partitioning scheme. A table's index partitioning scheme need not be the same as that used for the associated table. Partitioned indexes can be placed on a different physical disk than the data, resulting in optimum parallel processing performance. Partitioning tables and indexes improves the performance of data-loading and index-building operations.

Highlights

With the help of expression-based partitioning, multiple operations can be executing simultaneously on the same table, each in its unique partition, resulting in greater system performance than typical database systems.

With expression-based partitioning, the IBM IDS cost-based SQL optimizer can create more efficient and quicker plans using *partition elimination* to only access those table/index partitions where the data is known to reside or should be placed. The benefit is that multiple operations can be executing simultaneously on the same table, each in its unique partition, resulting in greater system performance than typical database systems.

Depending on the operating system used, IBM IDS can use "raw" disks when creating dbspaces to store table or index data. When raw disk space is used, IBM IDS uses its own data storage system to allocate contiguous disk pages. Contiguous disk pages reduce latency from spindle arm movement to find the next data element. It also allows IBM IDS to use direct memory access when writing data. With the exception of Windows-based platforms, where standard file systems should be used, using raw disk-based dbspaces provides a measurable performance benefit.

Leveraging the strengths of DSA

With an architecture as robust and efficient as IBM IDS, the engine provides a number of performance features that other engines cannot match.

The *High-Performance Loader (HPL)* utility can load data very quickly because it can read from multiple data sources (for example, tapes, disk files, pipes or other tables) and load the data in parallel. As the HPL reads from the data sources, it can execute data manipulation operations such as converting from EBCDIC to ASCII (American Standard Code for Information Interchange), masking or changing data values, or converting data to the local environment based on Global Language Support requirements. An HPL job can be configured so that normal load tasks, such as referential integrity checking, logging and index builds, are performed either during the load or afterwards, which speeds up the load time. The HPL can also be used to extract data from one or more tables for output to one or more target locations. Data manipulation similar to that performed in a load job can be performed during an unload job.

Highlights

A properly designed database can leverage IBM IDS features such as parallel data query, parallel scan, sort, join, group and data aggregation for larger, more complex operations.

The IBM IDS parallel data query feature benefits complex SQL operations that are more analytical, or OLAP oriented, than operational, or OLTP oriented.

The speed with which IBM IDS responds to a data operation can vary depending on the amount of data being manipulated and the database's design. While many simple OLTP operations such as single row inserts/updates/deletes can be executed without straining the system, a properly designed database can leverage IBM IDS features such as parallel data query, parallel scan, sort, join, group and data aggregation for larger, more complex operations.

The parallel data query (PDQ) feature takes advantage of the CPU power provided by SMP systems and the IBM IDS virtual processors to execute fan-out parallelism. PDQ is of greatest benefit to more complex SQL operations that are more analytical, or OLAP oriented, than operational, or OLTP oriented. With PDQ enabled, not only is a complex SQL operation divided into a number of sub-tasks but the sub-tasks are given higher or lower priority for execution within the engine's resources based on the overall "PDQ-priority" level requested by the operation.

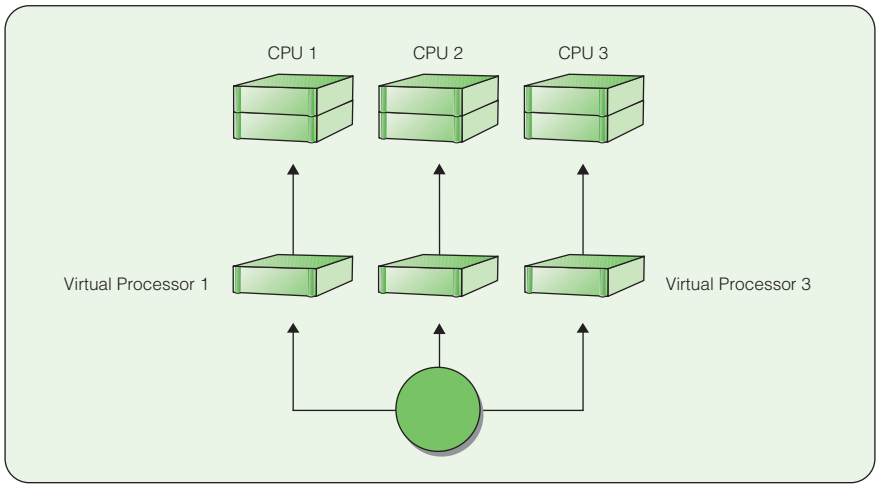

Figure a4:
Many virtual processors can be used to respond to a single user's request.

Highlights

The Memory Grant Manager (MGM) works in conjunction with PDQ to control the degree of parallelism by balancing the priority of OLAP-oriented user requests with available system resources, such as memory, virtual processor capacity and disk scan threads. Each OLAP query can be constructed to request a percentage of engine resources (i.e., PDQ priority level) for execution. The IBM IDS administrator can set query type priorities, adjust the number of queries allowed to run concurrently, and adjust the maximum amount of memory used for PDQ-type queries. The MGM enforces the rules by releasing queries for execution when the proper amounts of system resources are available.

The parallel scan feature of IBM IDS, which facilitates the simultaneous execution of multiple operations across the table/index, can provide a significant performance boost.

The parallel scan feature takes advantage of table partitioning in two ways. First, if the SQL optimizer determines that each partition must be accessed, a scan thread for each partition will execute in parallel with the other threads to bring the requested data out as quickly as possible. Second, if the access plan only calls for "1" to "N-1" of the partitions to be accessed, another access operation can execute on the remaining partitions so that two (or more) operations can be active on the table or index at the same time. Since disk I/O is the slowest element of database operations, to scan in parallel or have multiple operations executing simultaneously across the table/index can provide a significant performance boost.

As data is being retrieved from disk or from memory buffers, the IBM IDS parallel sort and join technology takes the incoming data stream and immediately begins the join and sorting process rather than waiting for the scan to complete. If several join levels are required, higher-level joins are immediately fed results from lower-level joins as they occur.

Similarly, if aggregate functions such as *SUM*, *AVG*, *MIN* or *MAX* need to be executed on the data or a *GROUP BY* SQL operator is present, these functions execute in realtime and in parallel with the disk scan, join and sort operations. Consequently, a final result can often be returned to the requester as soon as the disk scan is completed.

Highlights

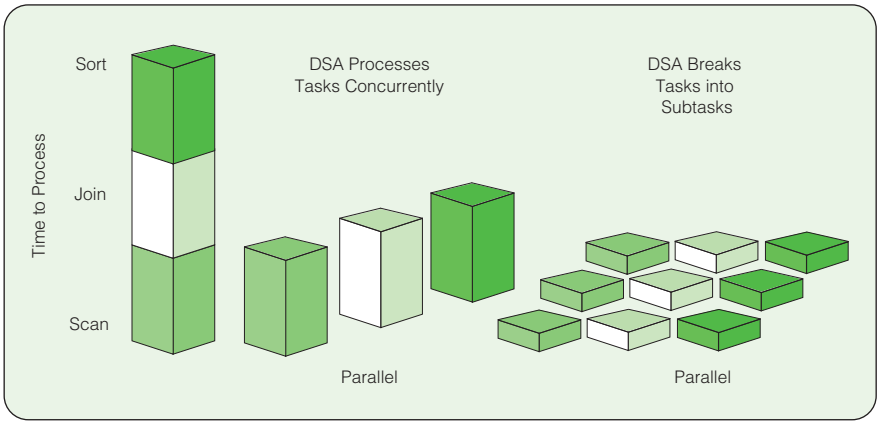

Figure a5:
The IBM DSA is designed to process tasks (such as scan, join and sort) concurrently and breaks them into subtasks to greatly reduce processing time.

Like the parallel scan, a parallel insert takes advantage of table partitioning allowing multiple virtual processors and update threads to insert records into the target table(s) in parallel. This can yield performance gains proportional to the number of disks on which the table was fragmented.

IBM IDS uses parallelized index-building technology to significantly reduce the time needed to build indexes.

With single-threaded database engines, index building can be a time-consuming process. IBM IDS uses parallelized index-building technology to significantly reduce the time needed to build indexes. During the build process, data is sampled to determine the number of scan threads to allocate. The data is then scanned in parallel (using read-ahead I/O where possible), sorted in parallel

Highlights

Parallelization produces a dramatic increase in index-build performance when compared to serial index builds.

and then merged into the final index. As with other I/O operations already mentioned, everything is done in parallel; the sort threads do not need to wait for the scan threads to complete and the index builds do not wait for the sorts. This parallelization produces a dramatic increase in index-build performance when compared to serial index builds.

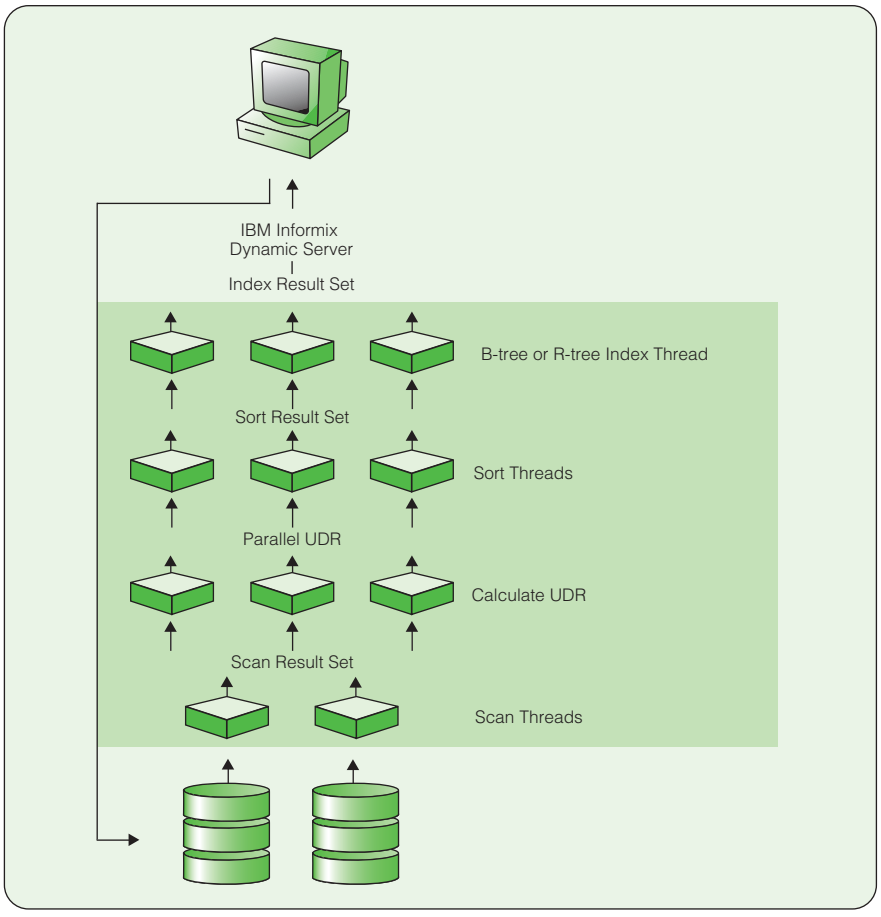

Figure a6:
An example of parallel index build using read-ahead, parallel scans and parallel sorts.

Highlights

IBM IDS uses a cost-based optimizer to determine the fastest way to retrieve data from database tables and/or indexes.

IBM IDS uses a cost-based optimizer to determine the fastest way to retrieve data from database tables and/or indexes based on detailed statistical information about the data within the database generated by the *UPDATE STATISTICS* SQL command. This statistical information includes more than just the number of rows in the table; the maximum and minimum values for selected columns, value granularity and skew, index depth and more are captured and recorded in overhead structures for the optimizer. The optimizer uses this information to pick the access plan that will provide the quickest access to the data while trying to minimize the impact on system resources. The optimizer's plan is built using estimates of I/O and CPU costs in its calculations.

Access plan information is available for review through several management interfaces so developers and engine administrators can evaluate the effectiveness of their application and/or database design. The SQL operation(s) under review do not need to actually execute in order to get the plan information. By either setting an environment variable, executing a separate SQL command or embedding an instruction in the target SQL operation, the operation will stop after the operation is "prepared" and the access plan information is output for review. With this functionality, application logic and database design can be tested for efficiency without having to constantly rebuild data back to a known "good" state.

In some rare cases, the optimizer may not choose the best plan for accessing data. This can happen when, for example, the query is extremely complex or there is insufficient statistical information available about the table's data. In these situations, after careful review and consideration, an administrator or developer can influence the plan by including *optimizer directives* (also known as optimizer hints) in the SQL statement. Optimizer directives can be set to use or exclude specific indexes, specify the join order of tables, or specify the join type to be used when the operation is executed. An optimizer directive can also be set to optimize a query to retrieve only the "N" rows of the possible result set.

IBM Corporation
Silicon Valley Laboratory
555 Bailey Avenue
San Jose, CA 95141
U.S.A.

Printed in the United States of America
04-03

GC18-7624-02