IBM® DB2® Content Manager V8 Implementation on DB2® Universal Database™: A Primer

By
Kenneth Chen, Peter He, Tawei Hu, Ed Gallagher
IBM Corporation
May 2003
Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other
countries: IBM, AIX, DB2, DB2 Universal Database, Domino, Domino.Doc, EDMSuite, ImagePlus Tivoli,
VideoCharger, VisualInfo, Websphere, z/OS.

Windows and Windows-based trademarks and logos are trademarks or registered trademarks of Microsoft
Corporation.

UNIX and UNIX-based trademarks and logos are trademarks or registered trademarks of The Open Group.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in
the United States and other countries.

Other company, product or service names may be the trademarks or service marks of others.

The furnishing of this document does not imply giving license to any IBM patents. References in this document to
IBM products, Programs, or Services do not imply that IBM intends to make these available in all countries in which
IBM operates.

© Copyright International Business Machines Corporation 2003. All rights reserved.
## Table of Contents

1. Introduction ....................................................................................................................................................................... 3
2. CM solution products .......................................................................................................................................................... 3
3. CM V8.1 key enhancements ............................................................................................................................................... 4
4. CM V8.1 architecture .......................................................................................................................................................... 5
5. CM control flow terminology ............................................................................................................................................... 6
6. Content Manager data flow .................................................................................................................................................... 7
   6.1 Storing an object ........................................................................................................................................................... 7
   6.2 Replacing an object ....................................................................................................................................................... 8
   6.3 Deleting an object ......................................................................................................................................................... 9
   6.4 Retrieving an object ..................................................................................................................................................... 10
7. CM workflow ....................................................................................................................................................................... 11
8. DB2 implementation ............................................................................................................................................................ 12
   8.1 Performance ................................................................................................................................................................. 14
9. WebSphere Application Server tuning ....................................................................................................................................... 24
   9.1 WebSphere JVM and transport settings .......................................................................................................................... 25
10. Summary ............................................................................................................................................................................. 26
1 Introduction

The inevitability of e-business is a reality. Today, as e-business evolves, it is more than Websites that offer e-Commerce transactions. Instead it represents an entire business process transformation, from back office to front office, requiring not only an efficient collaboration between intra-corporate boundaries, but also better links to suppliers, vendors and partners, to provide a new class of customer services.

E-business infrastructure is the new phase of Information Technology evolution that encompasses wide-ranging solution initiatives and services in customer relationship management (CRM), supply chain (SCM), fulfillment, e-commerce, procurement, enterprise relationship planning (ERP), and business intelligence (BI). Information, both structured data and unstructured content, is at the very core of this e-business infrastructure.

The information growth in the enterprise has been explosive. This information can take the form of:

- Operational content, from paper documents, to facsimiles, mail, invoices, computer output and scanned images that constitute the backbone of a business operation
- Digital media assets - audio, video (which represents the multimedia assets in business and used in marketing), corporate communications and training activities
- Workgroup documents such as email, publications, spreadsheets, notes, and presentations
- Web content, including HTML and dynamic content for integration with Internet technologies.

The real challenge is how to leverage this information content in a cost effective, timely, accurate, and complete fashion in a day-to-day business activity. A key approach to addressing this issue is to provide an information repository that will capture, store, and manage the diverse forms of information content, and make this integrated content available to a common application programming interface (API), unified search and retrieval. The IBM DB2 Content Manager (CM) family of products is a scalable, extensible, high performance solution that provides the features and functions to meet the information integration requirements.

This paper describes IBM DB2 Content Manager (CM) for MultiPlatforms V8.1 key concepts and data flow, and also discusses the implementation on IBM DB2 Universal Database™ for Linux, UNIX® and Windows® (DB2 UDB), including performance issues and tuning information on IBM AIX®.

2 CM solution products

The need for document management is absolutely crucial to the success of an enterprise. Content Manager has evolved from three predecessor products which have been available since 1985. It is a merge of IBM EDMSuite™, ImagePlus®, VisualInfo™, and IBM DB2 Digital Library. Depending on the extent of a document management implementation, the Content Manager solutions include, but are not limited to, the following products:

- IBM DB2 Content Manager (CM) for MultiPlatforms V8.1 - Supported on AIX, Windows, and Solaris, and very soon will be supported on z/OS®. It is a repository for business information that provides integrated access to enterprise users. The CM distributed, multi-tier architecture is a triangular architecture comprising a Library Server, and one or more Resource Managers (RMs) and one or more clients. CM is integrated with DB2 Universal Database and requires the database to be installed on the Library Server and the Resource Managers. CM V8.1 is integrated with Siebel Call Center to improve call center efficiency for content integration and MQ Series for workflow and process management.

- IBM DB2 Information Integrator for Content for MultiPlatforms V8 (formerly Enterprise Information Portal (EIP)) - Supported on AIX, Windows, and Solaris. A light version of EIP which does not include extended search
and information mining, it is a single point of entry for accessing both structured and unstructured content residing across the CM backends, including homogeneous and heterogeneous sources such as Lotus® Domino.Doc® and CM OnDemand. EIP also provides a set of APIs for portal application development to facilitate search and retrieval. Many connectors, such as federated, ODBC 3.0, and JDBC 1.3 are available to extend the content search to diverse relational data sources.

- **IBM DB2 Content Manager OnDemand for MultiPlatforms V7.1** - Supported on AIX, Windows, Solaris, and HP-UX. Complementary to CM, it is optimized to manage high volume storage and retrieval of documents such as microfiche, reports, statements and checks. OnDemand has a graphical tool for administering capture and indexing of documents. It also leverages IBM Tivoli® Storage Manager for retaining documents on a secure basis.

- **IBM DB2 Content Manager VideoCharger™ for MultiPlatforms V8** – Supported on AIX and Windows. It adds value to CM’s rich media features by providing real time multimedia streaming capabilities, and delivering digital media over the Internet.

- **IBM Lotus Domino.Doc V3.5** - Supported on Windows. A document management product, it provides document versioning, checkin/checkout, and security. It is highly integrated with many desktop applications that are Open Document Management API (ODMA) compliant, as well as non-ODMA applications. Domino.Doc is also integrated with e-mail applications.

- **IBM MQSeries Workflow V3.3** - Supported on AIX, Windows, HP-UX, and z/OS. It provides the capability of defining business processes and automating process workflows.

- **IBM WebSphere Application Server V 4.0.3** – Supported on AIX, Windows, Solaris, HP-UX, Linux, and OS/390. It is business integration software that enables the integration of enterprise data and transactions with e-business infrastructure. A Java™-based Web application server, it provides an application deployment platform for e-business applications.

### 3 CM V8.1 key enhancements

Content Manager V8.1 is significantly enhanced over CM V7 to better address users’ content management requirements. It delivers leading-edge information assets management solutions spanning the entire content or document life cycle, including document creation, capture, storage, management and distribution. In addition, it leverages the capabilities of DB2 Universal Database - federation, stored procedures, cost-based optimizer, and text search via IBM DB2 Text Information Extender for scalability, extensibility and performance.

The major enhancements include:

- **Open framework** - CM V8.1 provides an open framework through support for standards including XML and a unified application programming interface including C++, and Java API, known collectively as OOAPI. This unified API can be used by applications including ERP, ERM, SCM and CRM. The API provides search and retrieval functions, workflow control, access control and the ability to administer diverse enterprise content. CM eClient can also exploit Java Beans to implement features such as links, child components, document versioning and document routing. You can select unicode support at install time for the eClient, Library Server, and Resource Manager (RM). The Java API can also be unicode-enabled.

- **Enhanced physical data model** - The data model is supports hierarchical item types for expressing the complex metadata hierarchy. A hierarchy consists of a root component and child components in a 1:n relationship, as well as multi-valued attributes. For example, an article may have multiple authors. The article can be represented by the root component, with the author information stored in child components each having a parent ID referencing the root. The model includes support for versioning, checkin/checkout, access control, typing and classification. Moreover, the hierarchy and the inter-item relationships can be explored using Xpath query.
- **Improved performance** - Performance improvement results from exploitation of DB2 performance features (optimized SQL, Stored Procedures, implementation of enhanced model etc.) and a mean-and-lean API implementation. In addition, performance has been enhanced by increased efficiency in network communication, along with search improvements based on TIE integration with Library Server (providing attributes search as well as full-text search capabilities), and using precomputed access control lists for document access, as well as XPath query to navigate the hierarchical model. Improvements in document routing and workflow control (including branching, collection points, decision points, and workflow) speed up the management functions. In addition, the Resource Manager providing connection pooling and optimized handling of delivery of contents to the eClient contributes to performance improvement.

4  **CM V8.1 architecture**

The Content Manager architecture is based on a triangular client/server model comprised of one Library Server, one or more Resource Managers, and one or more clients, offering a truly extensible, distributed and Web-enabled architecture. Figure 1 illustrates the architecture.

![Figure 1. Content Manager V8.1 architecture](image)

The **Library Server** (LS) sits on top of DB2 Universal Database and is the central source for indexing, describing, locating, organizing, and managing enterprise content. In addition, LS enforces controls to content access and transactions management. The core LS logic (except for system utilities and housekeeping tasks) is packaged as a set of stored procedures (SPs) containing embedded SQL statements. The interface to LS is SQL (CLI, JDBC), through which either SPs can be called via API or SQL SELECT. Remote access to LS is available via a CM Windows client.

CM supports both implicit transaction, which is just a single API call, and explicit transaction - a user-determined, recoverable unit of work, consisting of a sequence of consecutive API calls made through a single connection to a LS. CM provides synchronization logic to assure data consistency between a LS and its RMs, with the LS serving as commit coordinator, to accomplish the effect of two-phase commit.

The **Resource Manager** is a repository for managing the storage, retrieval and archival of enterprise content. The store/retrieval of RM objects is controlled by LS via an access token. During the retrieval of an RM object, the
primary RM may forward the request to other (types of) RM servers; thus the the application is transparent to the actual location of the objects. RM supports multiple access protocols including HTTP, FTP, and RTSP, a streaming protocol for data with real time properties.

The **Application Server** acts as the broker between the client and the Library Server, and provides support to the enhanced Content Manager API in C++, Java, and XML, which all applications must write to. The Application Server manages connections to the Library Server and, optionally, to the Resource Managers.

The **Client for Windows** is the desktop client that exploits the client-server architecture. It interfaces with the LS via SQL to access document applications, using OODAPI (C++ API). The Client for Windows is also known as the *production client*, or thick client, and provides support for integrated text search, document routing, and a hierarchical model.

The **eClient** is the Web application that communicates with the LS via the Application Server or the mid-tier server and to Resource Managers as well, enabling direct, secure and optimized delivery of objects with full transactional integrity from CM content servers, including OnDemand and ImagePlus. The eClient Web application uses OOAPI (Java API) and deploys JSPs and servlets on the Application Server. The eClient can connect to EIP and supports integrated text search, advanced EIP workflow and CM document routing.

### 5 CM control flow terminology

Before we describe the particulars of CM data flow, let’s define the terminology we’ll use:

- **Items, objects, and versioning** - The basic information entities managed by LS are *items*. Items in CM are of 2 types - simple items and resource items. Resource items can have content (*objects*) associated with them stored in one or more CM Resource Managers. Resource items point to their content via a CM Resource URL-RELATED DATA. An item may reference resources managed by RMs through document-part relationships maintained by the LS. *Versioning* is handled by LS on individual items, not on individual resources. Therefore versioning is unknown to RMs, i.e., a RM handles different versions of an resource as different resource, or they can be on stored different geographically dispersed RMs.

- **Object identification** - In general, the object is identified by an access token that includes item id and version id. The item id is generated based on the timestamp with additional prefixes. The access token is encrypted by LS before giving to the application. It is decrypted by the token verifier in RM. A token certifies certain access privileges (that is, authorizes certain RM operations) until a specified expiration time. The privileges can be bound to a RM, a specific object, a group of objects, a specific user, an IP address, access conditions, and so on. A token can be invalidated before it expires if an invalidation mechanism is implemented.

- **CM clients** - A client node can be a mid-tier server (for example an application server or a Web server) that supports many end-users. A mid-tier server may authenticate end-users, handle concurrent user sessions, map end-user to CM users (which is used for CM access control), and maintain a pool of reusable or shared processes/threads and LS-connections. A CM userid may or may not be the same as a RDBMS userid that is used to access the LS.

- **Checkin/checkout** - The items need to be checked out for editing. The checkout process imposes a persistent write to prevent concurrent update. This lock can span multiple CM transactions. The checkout does not prevent another CM user from retrieving the item or retrieving objects belonging to the item. The write lock is released by checkin.

- **CM transactions** - There are 2 kinds of CM transaction. A CM implicit transaction is an API call that will commit or roll back the data. An explicit transaction is a user-determined sequence of consecutive API calls that starts with startTransaction() call and ends with commit() or rollback(). This is a recoverable CM unit of work. CM (LS) is responsible for transaction integrity between LS and RM, as a result of any changes to RM, implemented via an asynchronous recovery technique. This is achieved via a transaction ID, a LS tracking table called ICMSTTXLT, an
RM tracking table called RMTRACKING used to record transactions that are not committed, and the transaction state. A CM transaction that does not make change to any RM can be handled as a database transaction on LS.

- **Asynchronous recovery** (AR) - The AR processes use the tracking tables and the transaction ID to maintain data consistency between a LS and its RMs in case of a failure. There is an AR process running on each CM server as a daemon. The tracking tables maintain a sparse and transient log of RM updates for incomplete CM transactions. If a CM transaction ends successfully (either committed or rolled back), with CM servers synchronized for this transaction, no record of this transaction remains in the TTs in a steady state.

6 **Content Manager data flow**

Now we’ll look at the control flow of the documents during the document operations store, replace, delete and retrieve. The asynchronous recovery process for maintaining consistency between LS and RM in the case of error recovery is not shown.

6.1 **Storing an object**

Figure 2 shows the data flow during the storing of an object. The communication path is described in detail here.

Figure 2. Dataflow to store and replace an object

1. The client first invokes the begin transaction to indicate the start of the transaction. No call to the Library Server is made here.

2. The begin transaction API returns to the client.

3. The client invokes the “prepare create” API, which in turn, invokes its LS counterpart stored procedure. The stored procedure performs privilege checking, determining whether or not the user has the authority to perform the create. If so, it continues to generate an item ID, inserting the item ID into ICMSTITEMS001 table with an “in progress” flag set on. It also returns the version ID (which will be 1), and a generated ‘create object token’. If the caller indicates this is an explicit transaction, no commit is done on the LS
database inserts. Otherwise, the inserts are committed. It also retrieves RM information needed by the client to store the object, and generates an 'object store' token (required by the RM).

4. The prepare create SP returns this information to the prepare create API, and the prepare create API returns to the client.

5. The client establishes a connection with the Resource Manager via the store object API. The store object API builds an HTTP request which includes the item id, version id, and token returned from the prepare SP in step 4, and sends it to the Resource Manager. The RM first validates the token that was passed in. If it is valid, it stores the object, writes a record to its tracking table, and commits the updates.

6. The RM builds a reply which includes the object metadata (the object name it generated, its size and create timestamp) and returns to the store object API, which returns to the client. The reply includes a return code indicating the success or failure of the object store.

7. The client invokes the "create item" API, which invokes its LS counterpart stored procedure. The create SP sets the flag off in the ICMSTITEMS001 table, and populates the ICMUTnnnnnssss and other tables (version, auto link, etc., etc.). The object metadata is also passed in with this stored procedure call.

8. The create item SP returns to the create API, which returns to the create item API.

9. If this is an explicit transaction, the client invokes the "end transaction" API, indicating the transaction is to be either committed or rolled back. The end transaction API first performs a SQL COMMIT to commit or roll back its LS updates, as indicated by the caller.

10. The end transaction API will start a new thread to send an HTTP "endtran" request to the RM. Upon receiving it, the RM will delete its tracking table record for the object store it performed. If it is a rollback, it will delete the object it stored.

11. The asynchronous recovery (AR) process is run as a separate batch process. It will recognize when a transaction has aborted, and take corrective action depending upon the point of failure.

6.2 Replacing an object

The data flow for updating an object is similar to the data flow for storing an object, but there are some fundamental differences in the communication path, again referring to Figure 2 above above (Notice step 9 is different for store versus replace).

1. The client first invokes the begin transaction API to indicate the start of the transaction. No call to the Library Server is made here.

2. The begin transaction API returns to the caller.

3. The client establishes a connection with the LS via the "prepare update" API, which invokes its LS stored procedure counterpart. The item is marked as 'checked out'. An access token is generated for each item. In addition, if this is the first call to "prepare update" since the start of the transaction, the stored procedure will generate a transaction ID, insert a record for this transaction ID into the LS tracking table. Note: there is no need for a DB2 commit in this step (as is done in the store scenario). It will be performed optionally based on the caller’s understanding of an implicit versus an explicit transaction.

4. The LS returns the tokens, RM host names, port numbers, collection names, and transaction ID to the API. The API returns to the client.
5. The client establishes a connection with the Resource Manager (RM) via the “replace object” API. The HTTP protocol is used to transmit the object token, collection name, information about the object, transaction ID and the object itself (one at a time). The RM HTTP Server receives the request, and invokes the RM Replace Agent program which validates the token, replaces the object (actually storing the replaced object under a different object name, leaving the old version alone for roll-back purposes), inserts a "post replace" record in the RM Tracking Table, and then commits all updates.

6. The RM returns to the client, indicating the success or failure of its updates. It also returns the object metadata to the client. Steps 5 and 6 are repeated until all objects are replaced.

7. The client invokes the "update item" API, which invokes its LS counterpart stored procedure to update the item and object metadata. It will update the LS Tracking Table record as complete.

8. The "update item" stored procedure returns to the API, which returns to the client. The client then issues the DB2 commit.

9. The client will invoke the "end transaction" API. The API will start a new thread to delete all RM tracking table records for the transaction ID (i.e., it happens asynchronously). The RM commits the deletes. If this asynchronous cleanup failed, the AR process will redo the cleanup. (This asynchronous cleanup initiated by EndTran aims to keep the size of RM Tracking Table small and to reduce AR workload.) Finally, the EndTran API deletes all transaction-specific info from the API cache before returning to the application.

10. The RM returns after deleting tracking table records for the transaction ID. (Client does not need to wait for return)

11. The Asynchronous Recovery (AR) Process is run as a separate batch process. It will recognize when a transaction has aborted, and take corrective action depending upon the point of failure.

### 6.3 Deleting an object

Figure 3 shows the data flow during the delete of an object. There is no communication with the RM in the process, and also a token is not needed.
The communication path is as follows:

1. The client initiates a connection with the Library Server (LS) Delete Stored Procedure via the delete item API to insert a record for each object in the item in the LS tracking table with a status of “delete pending” and with the transaction ID. The item is then deleted. The object(s) can no longer be retrieved or worked with by any application through the LS, although they are still available from the respective RMs if an application still holds a valid access token.

2. The LS returns to the client indicating the success or failure of this update.

3. The asynchronous recovery (AR) process is run as a batch job. It checks the LS “to be deleted” table for items pending. It deletes the corresponding object.

### 6.4 Retrieving an object

Figure 4 shows the communication flows during the retrieval of an object. There is no implications for transaction management. The communication path is provided here for reference only, as retrieves are often done prior to other transaction-related actions.
Figure 4. Dataflow to retrieve an object

1. The client establishes a connection with the LS via a call to the “get item” API. The LS then generates an access token(s) for one or more items using the Gen_Token( ) function.

2. The LS then returns the token(s), the RM URL-RELATED DATA(s) and the collection name(s) to the client.

3. The client makes an HTTP request to the RM via the retrieve object API. It passes the token for the object to be retrieved. The RM then validates the token using the Validate_token( ) function. (The RM will attempt to validate the token up to two times; the first time using the “read forever” key, and the second time using the "specified duration" key.) The client also specifies whether the whole object is to be retrieved, or the offset and length of the part of the object it wants. Also, the object’s “changed timestamp” (from the LS metadata) is passed to the RM. If the RM determines this timestamp is earlier than that of the RM’s own timestamp for this object, a replace may have occurred which has not yet been committed in the LS. The RM may reject the retrieve, or interrogate the RM TT to determine the name of the “old object”, and return that one instead.

4. The RM then returns the object (or part of the object) to the client.

Steps 3 and 4 are repeated if the client wants to retrieve the object in chunks, and continues either until the client decides it has what it needs, or until the Resource Manager determines there is nothing more to return to the client (i.e. it has reached the end of the object).

7 CM workflow

CM V8.1 offers two workflow implementations, an integrated document routing model, and EIP Advanced Workflow model. While they have common features, there are unique differences. The CM document routing has an embedded workflow engine dedicated to the Content Manager, while EIP Advanced Workflow has a workflow engine based on MQSeries Workflow.

The document routing process supports features including sequential routing, collection points and dynamic routing. The EIP workflow model provides features that include sequential routing, parallel routing, collection points, decision points, sub-workflow and user exits.
The document routing process and EIP workflow can be consolidated in CM V8.1 to provide a more complete document-based workflow solution.

Figure 5. Content Manager workflow

In Figure 5, CM workflow is illustrated using Advanced EIP Workflow and MQSeries Workflow. Here are some of the terms used:

- **The workflow** (or process) represents a series of worknodes through which an item routes. A worknode is either a workbasket or collection point.
- **The collection point** is a special worknode which allows suspension of folders until all documents are collected.
- **A worklist** is a group of worknodes from which the user obtains a list of work or next work item.
- **Parallel routing** represents multiple routes which the item flow through, for example multiple review of an insurance claim.
- **Decision point** represents possible alternate branching in the routing.

8 DB2 implementation

This section describes the characteristics of DB2 Universal Database from the perspective of the Content Manager implementation. After the installation of the LS and RM components, CM creates one LS database and one RM database. The LS database can be installed as either a unicode or a non-unicode database, but the RM database is unicode enabled by default. These 2 databases contain the CM metadata for the management of the document life cycle. The install script uses a set of configuration parameters that are intended to address the needs of a typical installation. Database configuration changes can be applied to the configuration after a sizing and performance exercise.

The LS and RM seed databases are small (< 100M each). The LS database contains 103 tables, and the RM has 26 tables. Based on implementation considerations, rather than functional, the LS and RM metadata can be divided into categories based on typical CM usage of data.

For the LS database, developers have determined the following categories of data based on usage patterns:

- **Tablespace 1: Large tables that are written and read often**
  - Items Table (ICMSTITEMSnnnssss) - Tables that contain CM items
  - Item Links Table (ICMSTLINKSnnnssss) - Tables for maintaining items relationship
  - ICMUT predefined resource tables (component type 300-304) - Tables containing special item types
ICMUT user defined item type (ICMUTnnnnnsss)

- Tablespace 2: Large tables that are not frequently read, and may or may not be written often
  Item Versions Table (ICMSTITEMVERnnnsss) - Table containing old item versions
  Replicas Table (ICMSTREPLICAS) - Table containing items with replica rules
  Item Events Table (ICMSTITEMEVENTS) - Table logging create/read/update/delete items activities
  System Administration Events Table (ICMSTSYSADMEVENTS)

- Tablespace 3: Frequently used but volatile tables
  CheckOut Table (ICMSTCHECKEDOUT) - Tables for checkin/checkout of items
  Document Routing tables (component type 200-207) - Tables for routing document items in workflow

- Tablespace 4: Small, frequently used tables that are read often but not frequently written
  System Control Table (ICMSTSysControl)
  User Table (ICMSTUsers)
  Compiled ACL Table (ICMSTCompiledACL)
  Compiled Permission Table (ICMSTCompiledPerm)
  All other LS tables will also be in this tablespace.
  The EIP admin tables will all be in this tablespace.

This classification translates into physical data placement in 4 LS tablespaces with separate bufferpools, which can be separately tuned for performance.

The RM database is much smaller than the LS database, and the categorization is enforced based on the need to keep all the data in memory under all conditions. This leads to 5 RM tablespaces with separate bufferpools.

The CM indexing scheme is very elaborate to ensure optimized query performance in both LS and RM. There are nearly 250 indexes in the LS, and 36 indexes in the RM, including the primary keys. The set of indexes has been determined based on thorough database design and performance testing using both dynamic and static workloads simulating the activities performed in the items life cycle in a highly concurrent environment.

Referential Integrity (RI) is used extensively to enforce relationship between the application tables. The application relies on the RI constraints for integrity of data, and this simplifies the application design, but RI incurs some performance penalty, especially evident in the access modules generation, where the parent-child relationship grows. The complex RI relationships impact the insert/update/delete operations.

The use of DB2 user-defined functions (UDFs) in CM is limited. There are two UDFs that are used to encrypt and decrypt user information. Two other UDFs are related to Text Information Extender search, for retrieving and filtering of object content, in order for Text Information Extender indexing to work.

The CM V8 has been rearchitected to exploit the DB2 Stored Procedures (SP) executing on the LS, with the objective of reducing network data traffic between server, where a SP executes, and the client. There are 116 SPs which are tied to the CM APIs, most of the SPs are CM system administration specific, with about 25 SPs that are items related. A SP is invoked when an API like ICMGetItem is called. The SPs are C Stored Procedures with embedded static SQLs using host variables. As such, the CM workload comes from both dynamic ad hoc query, as exemplified by EIP queries, and static queries in the DB2 bind packages. This signifies that changes to database configuration parameters will require a rebind of all the packages, to take advantages of the possible access plans improvements. The following command rebinds all packages.

```
db2rbind -d dbname -l logfile all
```
8.1 Performance

In general, the performance tuning of a CM application on DB2 is similar to tuning any other application. There are two general areas:

- **Application tuning** - CM development has progressed through stringent enforcement in the areas of logical model design, good SQL practice (including the use of prepare-once-execute-many with host variables), static SQL where possible, optimize for n rows, for update/for fetch only, exploitation of DB2 special registers, implementation of volatile statistics, indexing, good CM transaction scope, stored procedures, document operations simplifications, and redundancy reduction. This creates the basis for a scalable application.

- **Database tuning** - The application installs a fairly-well optimized physical database, with a carefully chosen set of database manager parameters, database parameters, and registry variables which will be discussed in detail later.

However, special considerations need to be directed to the following CM implementation characteristics:

- Use of both static and dynamic SQL
- API logics materialized in stored procedures
- Deployment under WebSphere®
- Thick windows client and e-client configuration
- Use of Text Information Extender
- Implementation on a 2-database configuration (LS and RM)

To tune the database and applications, we need to understand their characteristics and statistics. The main tools to use are:

- Snapshot monitor
- Event monitor
- Explains
- System Tools like Performance Monitor, VMSTAT, IOSTAT, SVMON, Top, PMAP, LSPS

8.1.1 Snapshot monitor

The DB2 snapshot monitor captures the state of the DB2 system at a point in time. The state information is controlled by 6 switches (bufferpool, lock, sort, statement, table, and UOW) which provide a wealth of performance information. One method of tuning is to take a snapshot for a single user (or controlled number of users) and then iterate through the data analysis, change implementation, and measurements until the performance objective is met. We’ll discuss some of the data that can be gleaned from the snapshot monitor to analyze performance problems, and then suggest how to take a reasonable approach to solving those problems.

- **Locks** - Lock waits are common, but excessive lock waits may possibly be corrected by proper indexing, which reduces the number of rows scanned, hence reducing the locks being taken. Reducing locks minimizes memory resources needed, and avoids lock escalations. The snapshot data helps to right-size the LOCKLIST and MAXLOCKS parameters in the database configuration. An update lock is more expensive than a read lock. DB2 optimizer model does not always consider lock implications (i.e. single user based) It may generate an access path that has the lowest cost based on its best cardinality estimate, but having some lock implications; hence the choice of index is very important. You can use the DB2 RR_TO_RS environment variable to reduce next key locking, but the side effect is that it skips over rows that have been deleted (but not committed), which can cause inconsistent results if for example, a delete is rolled back. Locks and isolation levels greatly impact the application concurrency. Some of
the heavily locked tables are ICMSTCHECKEDOUT, ICMUTnnnnnnss, ICMSTTXLT, RMACCESS, RMTRACTING, and RMVOLUMES tables.

Here’s an example of output from a DB2 snapshot for locks:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locks held currently</td>
<td>0</td>
</tr>
<tr>
<td>Lock waits</td>
<td>0</td>
</tr>
<tr>
<td>Time database waited on locks (ms)</td>
<td>0</td>
</tr>
<tr>
<td>Lock list memory in use (Bytes)</td>
<td>324</td>
</tr>
<tr>
<td>Deadlocks detected</td>
<td>0</td>
</tr>
<tr>
<td>Lock escalations</td>
<td>0</td>
</tr>
<tr>
<td>Exclusive lock escalations</td>
<td>0</td>
</tr>
<tr>
<td>Agents currently waiting on locks</td>
<td>0</td>
</tr>
<tr>
<td>Lock Timeouts</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Rows processed** - Rows read should be ideally comparable to rows selected, but in the real world, rows read are a few times larger than rows selected because you cannot guarantee full index access. Excessive rows read cause excessive user CPU usage and also I/O, by flushing out possible needed data in the bufferpool. Short of changing the application access of data, indexing can reduce the rows read. Rows inserted means normal insert, but can also imply insertion of data into temporary tables as required for massing of data before it is accessed. The choice of temporary tablespace should be properly planned.

In CM, the data model is highly normalized, and therefore it is necessary to have multiple joins to access complete information. For example, to get full details on itemtypes, CM has to join ICMITEMTYPEDEF, with ICMSTCOMPDEFS, with ICMSTCOMPATTRS, and with ICMSTATTRDEFS, which can cause many rows to be read. This is a classic normalization, and denormalization will not be discussed here. Here’s an example of row information returned from a snapshot:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows deleted</td>
<td>27</td>
</tr>
<tr>
<td>Rows inserted</td>
<td>73</td>
</tr>
<tr>
<td>Rows updated</td>
<td>37</td>
</tr>
<tr>
<td>Rows selected</td>
<td>4302</td>
</tr>
<tr>
<td>Rows read</td>
<td>18225</td>
</tr>
</tbody>
</table>

- **Sort** - Sorts are common place also, but the number of sorts should be reduced in order for the application to be able to scale. The sorts use memory from agent private memory, and can cause memory contention in a highly concurrent environment. CM sorts are typically small and less than 2 per statements. However, the EIP queries with parametric searches can cause large sorts, and often using a lower optimization level, or optimizing for a rows appear to reduce sorting by generating a non-sort-centric access plan. In addition, a good index is helpful to reduce the sort. Here’s an example of sort information returned by the snapshot monitor:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total sort heap allocated</td>
<td>0</td>
</tr>
<tr>
<td>Total sorts</td>
<td>354</td>
</tr>
<tr>
<td>Total sort time (ms)</td>
<td>34</td>
</tr>
<tr>
<td>Sort overflows</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bufferpool** - The bufferpool is one of the most important considerations. The separation of data in 4 tablespaces allow 4 bufferpools to be used. The most important is ICMLSMAINBP32 for the 32K tablespace (referred to as tablespace 1 earlier) and ICMLSFREQBP4 for the 4K tablespace for volatile table (referred to as tablespace 3). Ensure the choice of bufferpool parameter results in a high bufferpool ratio (> 90%), defined as:

\[
(1 - \frac{(pool\_data\_p\_reads+pool\_index\_p\_reads)}{(pool\_data\_l\_reads+pool\_index\_l\_reads)})\times 100
\]
If a higher bufferpool hit ratio cannot be achieved because of memory constraints on the system, then the alternative
will be to ensure a high index bufferpool ratio, defined as:

\[(1 - (\text{pool\_index\_p\_reads}/(\text{pool\_index\_l\_reads}))) \times 100\]

Here is an example of snapshot buffer pool output:

- Buffer pool data logical reads = 7669
- Buffer pool data physical reads = 1044
- Buffer pool data writes = 71
- Buffer pool index logical reads = 10376
- Buffer pool index physical reads = 859

**Table** - The activities on individual tables allow you to determine which tables are critically accessed. The Rows Written is typically a function of the CM operation. For example, when an itemtype is created, an entry may be in the ICMUT01006001 table assuming the correct component type ID, and the each itemtype may have child components in the ICMUT01007001 table. The Rows Read is a function of the CM operation and DB2 data access pattern. Thus the table snapshot provides information that allows you to optimize table layout, and data organization. Here is an example of table snapshot information:

- Table Name = ICMUT01007001
- Table Type = User
- Rows Read = 0
- Rows Written = 6
- Overflows = 0
- Page Reorgs = 0

- Table Name = ICMUT01006001
- Table Type = User
- Rows Read = 0
- Rows Written = 3
- Overflows = 0
- Page Reorgs = 0

Some of the above information leads us to some database parameters tuning, and that is the easy part. To further
tune the system, we need to look at the SQL execution using the dynamic SQL snapshot. It is useful to load the
dynamic SQL snapshot details into the DB2 sqlcache. From there it is possible to categorize queries for query
frequency, average execution time (when cursor opens to cursor closes), rows read/inserted/updated/deleted, and
sorting. To load the snapshot into sqlcache, execute these DB2 commands:

- get snapshot for dynamic SQL on icmnlsdb write to file

( Because the sqlcache option only persists for a connection, you can export the sqlcache data to another table for analysis)

- export to data.ixf of ixf select * from TABLE(sysfun.sqlcache_snapshot()) as t

- import from data.ixf of ixf create into data

It is now possible to query the data table to identify the most important problem queries to tackle.
8.1.2 Event monitor

The event monitor is used to complement the point-in-time snapshot information. It is a continuous picture of the execution sequence in a time interval, and in addition, it captures the static SQL statistics as well. The static statement section needs to be looked at in conjunction with using the output from the following detailed report, which maps the section number in the packages with the SQL statement texts.

```
select plname, stmtno, sectno, substr(clob(text),1,2048) from sysibm.sysstmt
```

The approach is to focus on the worst SQL statements. This can be determined by criteria like Exec Time, Sorts, or Rows Read. For example, it is very easy to generate the set of SQL statements whose execution time exceed, say, one second, as a starting point. The following is a representative section of an event monitor, showing the relevant part of a statement event.

```
102) Statement Event ...
   Type : Dynamic
   Operation: Close
   Section : 4
   Creator : ICMADMIN
   Package : ICMPLSGT

   Text : SELECT A.KEYWORDNAME, A.KEYWORDDESCRIPTION, B.* FROM ICMADMIN.ICMSTNLSKEYWORDS A, ICMADMIN.ICMSTITEMTYPEDEFS B WHERE B.ITEMTYPEID IN (200, 201, 202, 300, 301, 302, 303, 304, 400, 500, 1000, 1001, 1002, 1003, 1004) AND B.ITEMTYPEID = A.KEYWORDCODE AND A.KEYWORDCLASS = 2 AND A.LANGUAGECODE = 'ENU' ORDER BY ITEMTYPEID ASC

-------------------------------------------
Start Time: 08-11-2002 09:18:52.321887
Stop Time:  08-11-2002 09:18:52.798819
Exec Time:  0.476932 seconds
Number of Agents created: 1
User CPU:   0.000000 seconds
System CPU: 0.000000 seconds
Fetch Count: 15
Sorts: 0
Total sort time: 0
Sort overflows: 0
Rows read: 30
Rows written: 0
```

Listing 1. Event monitor statement output

8.1.3 Two real scenarios

Scenario 1 - Document store/retrieve with no resource item (i.e. no actual part is stored in the RM)

To investigate the performance issues with a performance workload called "create documents with no parts", we made one snapshot report and one monitor report from 1 run with:

1) 1 user with 200 loops

Observations for Library Sever:

1) For every document, 3 rows are inserted, and 2 rows are updated, and on average 60 rows are selected from a total of 300 rows read. The tables changed are:

* ICMUT01004001 (1 row)
There are 6 selects per document operation. The tables that are heavily read are:

- **ICMSTATTRDEFS** (212256 rows)
- **ICMSTCOMPILDAACL** (36446 rows)

**Scenario 2** - Document store/retrieve with resource item (that is, actual part is stored in the RM)

To investigate the performance issues with a performance workload called "create documents with parts", we ran one snapshot report and one monitor report from 1 run with:

1) 1 user with 50 loops

Observations for RM server:
1) 300 item parts result in 300 additional new rows in RMOBJECTS at the end of the test.
2) RMTRACKING and RMVOLUMES are heavily written to as well, but the row counts did not increase.
3) The tables that are read heavily in descending order are: RMSERVER, RMVOLUMES, RMTRACKING, RMCOLLECTIONS, RMOBJECTS, RMDEVMG0, RMSTORAGECLASS, RMMGTTRANSITION, RMSTRINGS, RMACCESS, RMMGTCLASS.
4) For 300 item parts, there are
   - 10029 commits
   - 750 rollbacks

Observations for LS server
1) We see +150 items row increase in ICMUT01004001, and +300 parts row increase in ICMUT01005001. Other reads include RI tables and user tables like ICMSTLINKS001001 (+150), ICMUT00300001 (+300), ICMSTRI001001(+300), ICMSTTXLT(+500), ICMITEMS001001(+500), ICMSTCHECKEDOUT(+50),
2) ICMUT01005001 and ICMSTITEMS001001 are heavily read registering 788550 and 1443000 reads.
3) Compared to RM, there are only 2408 commits for same number of parts

It is clear from the observations above that a lot of useful information can be derived to begin work on performance tuning. In conjunction with recommendations from SQL access plan analysis, we can improve the performance of the given workloads.
8.1.4  **Explains**

Access plan analysis is one of the most important techniques for performance tuning of any database application. CM 8.1 queries are well designed to start with. In addition, they have gone through careful analysis and have been tuned for good performance through appropriate choice of bufferpools, sortheap, optimization level, volatility of table, use of include clause, distribution statistics and a comprehensive indexing scheme on both the Library Server, and the Resource Manager. The "optimize for n rows “ feature is also selectively exploited to favor an access plan that returns the first n rows quickly, where n is carefully chosen so the data returned from the database server to the client is still well blocked to save on communication cost. The “optimize for n rows” is found to be very useful, and tends to generate plans with fewer sorts. Distribution statistics, though not used by applications with parameter markers, still benefit CM queries that deal with highly normalized system meta attributes tables. CM queries will be generated by customer installs, depending on implementation. As a result, additional tuning of queries may be needed at the deployment phase. Any new tuning changes should be followed by a rebind of all packages using the db2rbind utility, so that the static packages may benefit from the changes as well.

**Figure 6. Workflow Explain**

An SQL statement being investigated may be dynamic with parameter markers, and the access plan must be generated with the explain tool keeping the parameters in place, as it would be at runtime. Replacing the parameter markers with real values will likely generate invalid access plan. The access plan shown in Figure 6 is generated with “optimize for n rows” for a query in the Worklist selection. In the original version without the “optimize for n rows” clause, a lot of table scans and four sorts cause a performance issue. The new access plan is very beneficial in a highly concurrent environment, because of the absence of sorts, fewer rows read, and smaller memory footprint.
As a note of caution, while a lower cost access plan is usually faster, there are exceptions. At times, the cost-based optimizer may have insufficient information to do an accurate cardinality estimation, for example when parameter markers are used, or when the table is very small (that is, few pages). Under these circumstances, the optimizer can derive an inaccurate cost of performing a specific operation, which causes it to generate a non-optimal access plan. This costing issue is addressed as part of the continuous optimizer improvement as it evolves.

DB2 provides an index advisor (db2advis) that is useful to improve explain plans, and it can be used together with additional analysis to achieve the best access plan.

Here is the syntax to invoke the index advisor:

```
db2advis -d  <dbname> -t <time> -I <sqlfile>
```

### 8.1.5 Top

**Top** is a UNIX system monitoring tool, and is available in different flavors across most platforms. It can be used to provide information regarding the CPU consumption, memory usage, and processes. A well behaved workload will consume high User CPU, and some I/O. The memory free should be high, and swap in use will ideally be low. It is possible to estimate the memory footprint per user connection based on the memory information. The PMAP tool can also be used to look at the memory usage by process.

```
load averages: 0.03, 0.01, 0.02                                     10:48:46
352 processes: 351 sleeping, 1 on cpu
CPU states: 99.3% idle, 0.1% user, 0.6% kernel, 0.0% iowait, 0.0% swap
Memory: 4096M real, 3134M free, 1678M swap in use, 17G swap free

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>RI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>TIME</th>
<th>CPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>18590</td>
<td>icmadmin</td>
<td>1</td>
<td>58</td>
<td>0</td>
<td>2864K</td>
<td>1760K</td>
<td>cpu/0</td>
<td>0:00</td>
<td>0.26%</td>
<td>top</td>
</tr>
<tr>
<td>18587</td>
<td>icmadmin</td>
<td>4</td>
<td>58</td>
<td>0</td>
<td>17M</td>
<td>5040K</td>
<td>sleep</td>
<td>0:00</td>
<td>0.01%</td>
<td>db2bp</td>
</tr>
<tr>
<td>21863</td>
<td>db2inst1</td>
<td>4</td>
<td>58</td>
<td>0</td>
<td>56M</td>
<td>23M</td>
<td>sleep</td>
<td>0:13</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21864</td>
<td>db2inst1</td>
<td>4</td>
<td>58</td>
<td>0</td>
<td>57M</td>
<td>22M</td>
<td>sleep</td>
<td>0:04</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>270</td>
<td>root</td>
<td>115</td>
<td>55</td>
<td>0</td>
<td>5440K</td>
<td>4752K</td>
<td>sleep</td>
<td>0:03</td>
<td>0.00%</td>
<td>nscd</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>1</td>
<td>58</td>
<td>0</td>
<td>800K</td>
<td>384K</td>
<td>sleep</td>
<td>0:01</td>
<td>0.00%</td>
<td>init</td>
</tr>
<tr>
<td>21558</td>
<td>root</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>44M</td>
<td>14M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21560</td>
<td>root</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>44M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21651</td>
<td>db2inst1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>54M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21688</td>
<td>db2inst1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>54M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21858</td>
<td>db2inst1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>54M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21814</td>
<td>db2inst1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>54M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21840</td>
<td>db2inst1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>54M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21851</td>
<td>db2inst1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>54M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
<tr>
<td>21751</td>
<td>db2inst1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>54M</td>
<td>13M</td>
<td>sleep</td>
<td>0:00</td>
<td>0.00%</td>
<td>db2sysc</td>
</tr>
</tbody>
</table>
```

Listing 2. Sample Top output

### 8.1.6 VMSTAT

VMSTAT is a monitoring tool that provides a summary of total active virtual memory used by all processes. It also lists the real-memory free list. At runtime, it provides valuable information regarding the application execution. Listing 3 shows the monitor information when a problem is occurring. Notice the User CPU (useful work) is low compared to
SYSTEM CPU, and the run queue (r column) is high. SVMON also reports global and process level memory usage, and can be mapped to VMSTAT report.

<table>
<thead>
<tr>
<th>kthr</th>
<th>memory</th>
<th>page</th>
<th>faults</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r  b  avm</td>
<td>fre re  pi</td>
<td>in  sy  cs</td>
<td>us  sy  id</td>
</tr>
<tr>
<td>222 4</td>
<td>2012991 1065612</td>
<td>0 0 0 0 0 0</td>
<td>1070 60892 20286</td>
<td>45 41 13 1</td>
</tr>
<tr>
<td>51  3</td>
<td>2014542 1064060</td>
<td>0 0 0 0 0 0</td>
<td>1051 38765 11820</td>
<td>22 71 4 4</td>
</tr>
<tr>
<td>476 1</td>
<td>2015007 1063597</td>
<td>0 0 0 0 0 0</td>
<td>1197 43805 11808</td>
<td>38 57 1 5</td>
</tr>
<tr>
<td>239 1</td>
<td>2017685 1060927</td>
<td>0 0 0 0 0 0</td>
<td>911 16314 4312</td>
<td>14 82 3 0</td>
</tr>
<tr>
<td>28  2</td>
<td>2020707 1057905</td>
<td>0 0 0 0 0 0</td>
<td>850 9612 2188</td>
<td>4 79 8 8</td>
</tr>
<tr>
<td>20  2</td>
<td>2021894 1056724</td>
<td>0 0 0 0 0 0</td>
<td>942 23899 7434</td>
<td>18 55 12 16</td>
</tr>
<tr>
<td>321 2</td>
<td>202341 1055269</td>
<td>0 0 0 0 0 0</td>
<td>1256 39582 12042</td>
<td>29 63 5 4</td>
</tr>
<tr>
<td>89  1</td>
<td>2025414 1053198</td>
<td>0 0 0 0 0 0</td>
<td>949 20744 5970</td>
<td>13 63 14 10</td>
</tr>
</tbody>
</table>

Listing 3. VMSTAT output

8.1.7 Database manager parameters

We chose the following database manager parameters after repeated testing based on analysis using the performance techniques we’ve discussed, including the PS command and PMAP on Solaris.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Recommended Value</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDF_MEM_SZ</td>
<td>256</td>
<td>This parameter is common to both fenced and unfenced user defined functions (UDFs). For a fenced UDF, it specifies the default allocation for memory to be shared between the database process and the UDF. CM uses Fenced UDF for encryption and decryption on a user basis. This parameter is found to affect the memory footprint of DARI process</td>
</tr>
<tr>
<td>JAVA_HEAP_SZ</td>
<td>256</td>
<td>One for each process, reduces memory footprint</td>
</tr>
<tr>
<td>SHEAPTHRES</td>
<td>10,000</td>
<td>CM sorts are small, but the numbers can be large as the number of CM users increase. Some of the queries go against predefined system metadata, and sort is characteristics in those queries, and also many work assignment queries use sorts</td>
</tr>
<tr>
<td>ASLHEAPSZ</td>
<td>15</td>
<td>The application support layer heap represents a communication buffer between the local application and its associated agent. This is found to reduce memory footprint</td>
</tr>
<tr>
<td>RQRIOBLK</td>
<td>65,535</td>
<td>communication buffer between remote applications and their database agents on the database server. CM data being passed requires higher than default value</td>
</tr>
<tr>
<td>QUERY_HEAP_SZ</td>
<td>16,384</td>
<td>Needed to accommodate CM queries, especially those involving LOBs. Any value lower will generate error</td>
</tr>
<tr>
<td>MAXAGENTS</td>
<td>2,500</td>
<td>To handle up to 2,500 users</td>
</tr>
<tr>
<td>Parameter</td>
<td>Recommended Value</td>
<td>Rationale</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUM_POOLAGENTS</td>
<td>600</td>
<td>CM application has many users concurrently connected. High value of num_poolagents needed to avoid the costs associated with the frequent creation/termination of agents.</td>
</tr>
<tr>
<td>NUM_INITAGENTS</td>
<td>500</td>
<td>The initial number of idle agents created in the agent pool at DB2START time to save allocation time later.</td>
</tr>
<tr>
<td>KEEPDARI</td>
<td>YES</td>
<td>To keep the DARI process from being destroyed after the DARI or stored procedure, particular critical in an environment where LS/RM are in the same box</td>
</tr>
<tr>
<td>INTRA_PARALLEL</td>
<td>NO</td>
<td>Individual queries making up CM transactions are almost always in the 10 ms ranges. Parallelism is not very beneficial</td>
</tr>
</tbody>
</table>

### 8.1.8 Database parameters

We chose the following database parameters after repeated testing based on analysis using the performance techniques as discussed, including PS command and PMAP on Solaris.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Recommended Value</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFT_QUERY_OPT</td>
<td>2</td>
<td>The use of optimization level 2 suits all kinds of queries in LS, RM, and eClient. Opt level provides distribution statistics, and makes sure of optimize for n rows clause</td>
</tr>
<tr>
<td>MULTIPAGE</td>
<td>YES</td>
<td>CM installs an SMS databases for both LS and RM, Multipage allocation saves page allocation cost by allocation in extent.</td>
</tr>
<tr>
<td>BUFFPAGE</td>
<td>Overridden by multiple bufferpools in sysibm.sysbufferpools with size chosen to maximize bufferpool hit ratio</td>
<td></td>
</tr>
<tr>
<td>ICMLSFREQBP4</td>
<td>1000(4K)</td>
<td>ICMLSVOLATILEBP4 8000(4K) ICMLSMAINBP32 8000(32K) CMBMAIN4 1000(4K)</td>
</tr>
<tr>
<td>ICMLSFREQBP4</td>
<td>1000(4K)</td>
<td></td>
</tr>
<tr>
<td>DBHEAP</td>
<td>1,200</td>
<td>One database heap per database. This value is reasonable based on snapshot and there is enough storage available for processing the statement.</td>
</tr>
<tr>
<td>LOGBUFSZ</td>
<td>32</td>
<td>Amount of DB heap that is used as a buffer for log records.</td>
</tr>
<tr>
<td>LOCKLIST</td>
<td>1,000</td>
<td>CM has very well indexed schema, the rows read are typically small, reducing the need for large locklists</td>
</tr>
<tr>
<td>APP_CTL_HEAP_SZ</td>
<td>4,096</td>
<td>Set for CM to handle complex CM queries</td>
</tr>
<tr>
<td>SORTHEAP</td>
<td>128</td>
<td>Because of high number of potential users, and small sorts, this value is chosen low to ensure efficient memory usage, also to influence optimizers to choose less plans with sort.</td>
</tr>
<tr>
<td>STMTHEAP</td>
<td>16,384</td>
<td>CM needed to avoid error</td>
</tr>
<tr>
<td>APPLHEAPSIZE</td>
<td>4,096</td>
<td>Memory initialized for an agent by the application. Needed to avoid memory error</td>
</tr>
<tr>
<td>PCKCACHESZ</td>
<td>16,000</td>
<td>For caching dynamic and static sql statements.</td>
</tr>
<tr>
<td>Variable</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MAXLOCKS</td>
<td>25</td>
<td>Avoid need to recompile for dynamic SQL and need to access system catalog for packages. Snapshot of CM workload shows this value is a safe value.</td>
</tr>
<tr>
<td>LOCKTIMEOUT</td>
<td>300</td>
<td>On a highly concurrent connected environment, locking is inevitable. Too low a value may cause unnecessary timeout. 300 is a good value.</td>
</tr>
<tr>
<td>NUM_IOCLEANERS</td>
<td>DISKS</td>
<td>Dependent on the environment. To be determined via snapshot using pool_data_writes and pool_async_data writes, as well as pool_index_writes and pool_async_index writes. This value for the asynchronous can start with the value of the number of disks.</td>
</tr>
<tr>
<td>NUM_IOSERVERS</td>
<td>CONTAINERS</td>
<td>No of Containers in tablespaces.. A higher value is harmless.</td>
</tr>
<tr>
<td>MAXAPPLS</td>
<td>2,000</td>
<td>To handle 2000 users potentially.</td>
</tr>
<tr>
<td>AVG_APPLS</td>
<td>5</td>
<td>To take into a concurrent user environments, not all the bufferpools will be used to optimize 1 user query.</td>
</tr>
<tr>
<td>LOGFILSZ</td>
<td>1,000</td>
<td>During the measurement of 1000 users, log activity is not excessive.</td>
</tr>
<tr>
<td>LOGPRIMARY</td>
<td>10</td>
<td>During the measurement of 1000 users, log activity is not excessive. The primary logs seem to be suffice.</td>
</tr>
<tr>
<td>LOGSECOND</td>
<td>20</td>
<td>During the measurement of 1000 users, log activity is not excessive. Just in case secondary is needed.</td>
</tr>
<tr>
<td>MINCOMMIT</td>
<td>1</td>
<td>As there are not excessive log records write, this value is the default.</td>
</tr>
</tbody>
</table>

### 8.1.9 DB2 registry variables

We chose the following registry variables for CM applications to enhance performance and handle operational improvements.

- **DB2_SPNOMEMSET=1**
  The DB2 engine will not memset the LOB memory. This reduces the cost of passing a BLOB/CLOB pair via stored procedure, that is, the cost for a 5 MB BLOB is the same for a 10 KB BLOB.

- **DB2_STPROC_LOOKUP_FIRST=1**
  CM Stored Procedure are not cataloged as PARAMETER TYPE DB2DARI, so the `DB2_STPROC_LOOKUP_FIRST` registry variable is set to ON. DB2 will look up the name of the shared library for the stored procedure in the system catalog before searching the SP directories.

- **DB2MEMMAXFREE=3000000**
  For UNIX environments, this specifies the maximum amount of unused memory that is held by DB2 processes when the process finishes executing.

- **DB2MEMDISCLAIM=YES**
  On AIX, DB2 processes have associated paging in reserved memory. This value helps reduce paging space requirements.
• DB2ENVLIST=LIBPATH ICMROOT ICMDLL ICMCOMP EXTSHM CMCOMMON LUMDISABLE

• DB2_RR_TO_RS=YES
  This registry variable forces Repeatable Reads to Read Stability, and the next key locking is turned off during index key insertion and update. The side effect is that scans will skip over rows that have been deleted but not committed, even though the rows may be qualified for scans.

8.1.10  DB2 optimization level 2 features

The optimization level 2 used in CM V8.1 has the following features:

<table>
<thead>
<tr>
<th>DB2 Optimization level 2</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join Enumeration</td>
<td>Greedy</td>
</tr>
<tr>
<td>Frequency statistics</td>
<td>Y</td>
</tr>
<tr>
<td>Quantile Statistics</td>
<td>Y</td>
</tr>
<tr>
<td>List Prefetch</td>
<td>Y</td>
</tr>
<tr>
<td>Index ANDing</td>
<td>Y</td>
</tr>
<tr>
<td>Index ORing</td>
<td>Y</td>
</tr>
<tr>
<td>List Prefetch</td>
<td>Y</td>
</tr>
<tr>
<td>Nested Loop</td>
<td>Y</td>
</tr>
<tr>
<td>Sorted Nested Loop</td>
<td>Y</td>
</tr>
<tr>
<td>Merge Scan</td>
<td>Y</td>
</tr>
<tr>
<td>Hash Join</td>
<td>Y</td>
</tr>
<tr>
<td>Star Join AND</td>
<td>Y</td>
</tr>
</tbody>
</table>

9  WebSphere Application Server tuning

The level of WebSphere that is required is the WebSphere Application Server Advanced Edition for multiplatform 4.0.3 (with PTF3). You must use WebSphere JDK in /usr/WebSphere/AppServer/java directory to avoid any timeout problems.

As CM is a WebSphere-based application, the overall CM performance depends on tuning the WebSphere Application Server. Generally, the following components affect the performance of RM deployment in WebSphere Application Server:

• HTTP Server
• WebSphere Application Server process
• Java Virtual Machine
• DB2 UDB

In the CM V8.1 design, the RM assumes a number of roles from WebSphere such as doing its own connection pooling. However, the eClient does use WebSphere connection pooling, but not the EJBs or persistent session in WebSphere Application Server. In this regard, only the Web Container and Java virtual machine tuning will impact CM performance.

A RM client request flows through a network of queues that must be harmoniously tuned to achieve optimal performance. These queues include the network, HTTP server, Web Container and DB2. Each of these resources represents a queue of requests waiting to use the resource.
Each queue provides settings to limit the maximum number of requests flowing through it. The settings are as follows:

- **HTTP Server** - The Server maintains a thread (process) pool to process the incoming HTTP requests. Its size can be controlled by the parameters defined in the HTTP configuration file in `conf/httpd.conf`. Two parameters are particularly important:
  - ThreadsPerChild for Windows
  - MaxClients for UNIX

- **Web Container** - The Web Container handles the request for server-side entities including servlet and JSPs. The following 3 parameters are critical:
  - Maximum thread size - the thread pool size to process servlet and JSP requests.
  - Transport maximum Keep-Alive.
  - Transport maximum requests per Keep-Alive.

An important rule of tuning is to minimize the number of requests in the queues. In general, it is better for requests to wait in the network (in front of the HTTP server), than it is for them to wait in WebSphere Application Server. WebSphere Application Sever provides a Resource Analyzer tool to monitor the performance of the Web Container. A high value of threads being created/destroyed, that is, Percent Maxed is > 10, indicates that the Web Container could be potential bottleneck. In this case, the number of threads should be increased.

Java Virtual Machine (JVM) tuning is also critical. There are two important Java heap parameter settings:

- Initial Java Heap size
- Maximum Java Heap size

Increasing these parameters allocates more memory space for objects creation. The application can run longer before a garbage collection (GC) occurs. However, a larger heap also takes longer to sweep and compact the freed objects, so garbage collection will take longer.

### 9.1 WebSphere JVM and transport settings

#### JVM key settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout</td>
<td>120</td>
</tr>
<tr>
<td>Inactivity</td>
<td>60</td>
</tr>
<tr>
<td>Initial heap size</td>
<td>512</td>
</tr>
<tr>
<td>Max heap size</td>
<td>512</td>
</tr>
</tbody>
</table>

#### Web container service key settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min thread size</td>
<td>150</td>
</tr>
<tr>
<td>Max thread size</td>
<td>150</td>
</tr>
<tr>
<td>Thread timeout</td>
<td>60</td>
</tr>
<tr>
<td>Allow thread allocation beyond maximum</td>
<td>yes</td>
</tr>
<tr>
<td>Maximum keep alive</td>
<td>135</td>
</tr>
<tr>
<td>Maximum requests per keep alive</td>
<td>150</td>
</tr>
<tr>
<td>Keep alive timeout</td>
<td>600</td>
</tr>
</tbody>
</table>
HTTP Server key settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxClients for UNIX</td>
<td>200</td>
</tr>
</tbody>
</table>

10 Summary

This article provides an introduction to implementing IBM DB2 Content Manager on DB2 as well as performance considerations. We have described the Content Manager architecture, the dataflow, and the methodology for tuning the Content Manager system from both the DB2 UDB and WebSphere Application Server perspectives. The dataflow details the steps in the data access, and provides clear relationship between the meta data and the actual document data that reside in the Library Server and Resource Manager. The default installation of Content Manager works for the general case, but depending on the specific environment, some customization may be needed. The tools you can use for monitoring and tuning were also described.
Related Information

IBM WebSphere Application Server Web site http://www.ibm.com/software/webservers/
IBM DB2 Content Manager Web site http://www-3.ibm.com/software/data/cm/

About the Authors

Kenneth Chen is a Consulting Software Product Design Professional from the DB2 Technical Alliance organization at the IBM Toronto Lab, specializing in business partner application enablement - application design, development, and performance. His areas of focus included distributed database platforms, as well as IBM DB2 Content Manager. He may be reached at kchen@ca.ibm.com.

Peter He is a Software Developer with the DB2 Integration Team at IBM Toronto Lab. Peter has been working extensively with DB2 and WebSphere technologies and is the architect of the IBM Video Central software based on Web Services architecture. He holds a Ph.D. from York University and is a DB2 Certified Solutions Expert. Peter may be reached at peterzh@ca.ibm.com.

Tawei Hu is the key technical leader for Library Server, and well recognized across the Content Management community. Tawei is also the key architect for the document model, and his technical expertise in both the workstation and OS/390 platforms are well recognized. He may be reached at tawei@us.ibm.com.

Ed Gallagher is a member of the IBM DB2 Content Manager Version 8 software development team, working at the Silicon Valley Lab in San Jose, CA. He has 20 years of software development experience. Ed may be reached at egallagh@us.ibm.com.