


Db2 Night Show

Session Z137

Db2 & Java Performance Best Practices – Volume X

Dave Beulke, Pragmatic Solutions Inc.

 @DBeulke

 davebeulke

www.davebeulke.com

Friday, May 12, 2023

This presentation details Db2 and Java performance best practices and discusses how to optimize your processing to run 100x faster. Db2 design, partitioning strategies, coding best practices, java class frameworks and debugging/tracing practices will be presented that can immediately eliminate your bottlenecks and enhance performance.

After this discussion you will be able to dramatically improve your Db2 access, Java runtimes, minimize CPU and quickly access/process billions of rows with the best performance possible.

Dave@DaveBeulke.com - 703 798-3283

- Lifetime IBM Db2 Information Champions
- One of 45+ IBM Db2 Gold Consultant Worldwide
- Past President of International Db2 Users Group - IDUG
- Best speaker at CMG conference & former TDWI instructor
 - Designs performance for databases with billions of rows
 - Db2 Certification test
 - IBM Business Intelligence certification test
 - Former columnist for Db2 Magazine
 - Former editor of the IDUG Solutions Journal



Performance BLOG:
www.DaveBeulke.com

- **Consulting**
 - CPU Demand Reduction – Guaranteed!
 - Db2 Performance Reviews
 - Database Design Review
 - Security Audit & Assessments
 - Migration Assistance
- **Education Seminars**
 - Db2 Version 13 Transition
 - Db2 Java Performance for Developers
 - Analytic Designs for Performance
 - How to Do a Performance Review
 - Setting up Security Compliance Audits
- Extensive experience in architecture & performance of large systems, databases and DW systems
 - Working with Db2 on z/OS since V1.2
 - Working with Db2 on LUW since OS/2 Extended Edition
 - First data warehouse in 1988 for E.F. Hutton
- Programming in Java for *Syspedia* since 2001 Find, understand and integrate your data faster!

3

I continue to provide Data Engineering Design, Security Audits and overall Db2 and Java performance help to a wide variety of clients.

Please let me know if your company might be interested in my assistance to beef up security, improve your systems/applications efficiencies and save on overall processing costs.

I am honored to have been a presenter at 30+ years of Db2 conferences

- 2023 – Java & Db2 Performance Best Practices Volume X
- 2022 – Java Db2 Performance Best Practices Volume IX+ Security Best Practices Volume IV
- 2021 – Java Db2 Performance Best Practices Security Best Practices Volume III
- 2020 – SQL Performance for Big Data
- 2019 – Best Design and Performance Practices for Analytics
- 2018 – Philadelphia - Security Best Practices Volume II -Best Design and Performance Practices for Analytics
- 2017 – Anaheim -Understand IDAA Performance and Justify an IDAA Appliance
- 2016 – Austin Performance Enterprise Architectures for Analytic Design Patterns How to do your own Db2 Security Audit
- 2015 - Valley Forge Db2 Security Practices Big Data Performance Analytics Insights
- 2014 – Phoenix Big Data SQL Considerations
- 2013 – Orlando Big Data Disaster Recovery Performance
- 2012 – Denver Agile Big Data Analytics
- 2011 – Anaheim Db2 Temporal Tables Performance Designs
- 2010 - Tampa - Java DB2 Developer Performance Best Practices
- 2009 – Denver -Java Db2 Perf with pureQuery and Data Studio Improve Performance with Db2 Version 9 for z/OS
- 2008 – Dallas - Java pureQuery and Data Studio Performance
- 2007 - San Jose - Developing High Performance SOA Java Db2 Apps Why I want Db2 Version 9
- 2006 - Tampa - Class - How to do a Db2 Performance Review Db2 Data Sharing Data Warehouse Designs for Performance
- 2005 – Denver - High Performance Data Warehousing
- 2004 – Orlando – Db2 V8 Performance President of IDUG
- 2003 - Las Vegas - Db2 UDB Server for z/OS V8 Breaking all the Limits Co-author IBM Business Intelligence Certification Exam
- 2002 - San Diego - Db2 UDB for LUW 8 - What is new in Db2 Version 8 Data Warehouse Performance
- 2001 – Orlando -Data Sharing Recovery Cookbook Designing a Data Warehouse for High Performance Co-authored the first IBM Db2 z/OS Certification Exam
- 2000 – Dallas - Db2 Data Warehouse Performance Part II
- 1999 – Orlando - Store Procedures & Multi-Tier Performance Developing your Business Intelligence Strategy Evaluating OLAP Tools
- 1998 - San Francisco - Db2 Version 6 Universal Solutions Db2 Data Warehouse Performance Db2 & the Internet Part II
- 1997 – Chicago - Db2 & the Internet
- 1996 – Dallas- Sysplex & Db2 Data Sharing Best Speaker Award at CMG Conference Mullen Award
- 1995 – Orlando - Practical Performance Tips Improving Application Development Efficiency
- 1994 - San Diego - Database Design for Time Sensitive Data & Guidelines for Db2 Column Function Usage
- 1993 – Dallas - High Availability Systems: A Case Study & Db2 V3: A First-Cut Analysis
- 1992 - New York -Db2 –CICS Interface Tuning
- 1991 - San Francisco - Pragmatic Db2 Capacity Planning for DBAs
- 1990 – Chicago - Performance Implication of Db2 Design Decisions
- 1989 – Chicago - Db2 Performance Considerations

© Copyright 2023

Dave@davebeulke.com

4

Please let me know if would like a copy of any of these previous presentations I have done over the previous decades.

Summary - Java & Db2 Performance Consulting/Class

| | | |
|--|---|--|
| Chapter 1- JDBC Optimization 14 | Chapter 5- Application SQL Optimization 44 | Understand your JVM settings/constraints against different machine/CPU's 71 |
| Create/Validate JDBC parameters 14 | Validate against SQL injection 44 | Monitor your Java and especially your Db2 in getting CPU, I/O and storage resources with the LPAR/Cluster/Pod 73 |
| Use the correct type of JDBC driver 16 | Validate high performance SQL is being Cached 45 | Chapter 6- Cloud/Container considerations 71 |
| Know how to wait up a JDBC Trace 21 | Remove/reduce application joins to SQL JOINS 46 | Kubernetes add-on scale vertical and horizontal controls configuration 73 |
| Provide IBM Db2 specific configuration 24 | EXPLAIN all SQL for optimized SQL and index usage 47 | Know CPU, memory and storage requirements for a Cluster/Pod 74 |
| DB2 data sources 24 | Separate optimize Insert/Updates/Deletes to limit deadlocks 49 | Optimize your Pods within your load balanced Cloud configuration 75 |
| JDBC connection security options and parameters 25 | Minimize network round trips through SQL Array & Java batch processing 51 | Know your API server, including DNS round robin or third party load balancing solution 76 |
| Improper JDBC Connection Commit Scope 26 | Use Stored Procedures SQL as called web services 52 | Make sure to do your Node Conformance Test for your clusters. Clean up any dead containers 77 |
| JDBC Objects correctly OPENED/CLOSED 28 | Eliminate Aborts, OPEN/CLOSE Cursor issues, DESCRIBE, LOCKTABLES PREPARES 54 | Know the layers of your Cluster(s) / Node(s) / firewalls, regulatory and security configurations 78 |
| COMMIT Frequency is optimized 28 | Make sure the correct DB2 isolation level is used for the application processing 55 | Chapter 7- Cloud thinking considerations 79 |
| Configure your Connection Pooling properly 28 | Chapter 6- Leveraging the database partitioning 57 | Cluster container(s) resilience considerations configuration 80 |
| Chapter 2- Java Optimization 97 | Database partitioning 57 | Single cloud, multi cloud, and multi cloud vendor scale out considerations 81 |
| How to verify that Java ODEBUG-MODE is off 98 | Minimize index(s) overhead 58 | Cloud approval regulatory bureaucracy considerations 82 |
| JUNIT testing checklists 106 | Design processing flow to reflect the stored order of the physical data definition 58 | Security audits of the multi cloud environments 83 |
| Optimize the Unit-of-Work/Commit Scope restartability 124 | Limit the granularity of the number of locks taken within a Unit-of-Work 60 | Chapter 11- Cloud IaaS considerations 84 |
| Garbage Collection is avoided and known for UJOW 144 | Unit of work within partition 60 | Cluster container(s) resilience considerations configuration IaaS 84 |
| Verify that your Java application is THREAD-SAFE 146 | Chapter 7- Leveraging the cluster/optimize Db2 flow configurations 62 | Single cloud, multi cloud, and multi cloud vendor scale out considerations 85 |
| Chapter 3- Java data type optimization 166 | Optimize the JVM memory available within the LPAR/Cluster/Pod 62 | Cloud approval regulatory bureaucracy considerations 86 |
| LinkedLists/Vector/Lists is optimized for the application purpose 166 | Multithread for multi processor LPAR/Cluster/Pod 63 | Security audits of the multi cloud environments 87 |
| JVM Settings and consistency throughout the lifecycle 176 | Optimize Java for number of CPU engines available per LPAR/Cluster/Pod 64 | Chapter 12- EXTRA Section 89 |
| Coverage/Right size using Cache objects 188 | Adjust Java Heap and Garbage Collection 65 | Standard Practices 89 |
| Know data type conversions issues 198 | Best references are www.java-performance.com 66 | Configure a DB2 JDBC connection pool 89 |
| EBCDIC, ASCII, UNICODE, BIT-DATA, LOBs/BLOBs handled correctly 214 | Java Tuning Mantra 66 | Encrypting JDBC connections 91 |
| Chapter 4- Java Application Basics 254 | Java Sorting versus Db2 Sort usage utilize in JVM memory or SSD capacity 67 | Best performing Db2 JDBC application flow 92 |
| Java program startup or initialization 254 | Understanding your JAVA run-time constraints 68 | The different Db2 JDBC isolation levels 92 |
| Java looping is done inefficiently 265 | Chapter 8- Understanding your JAVA run-time constraints 68 | SQL CODE checking is vital for data integrity 94 |
| Finishing up—Rollback/Termination 286 | Everything within IaaS has limits—know your environment constraints 68 | Batch processing using Prepared Statement 94 |
| Avoid Frameworks 306 | Workflow weekly/daily schedule constraints 69 | |
| Hibernate—Comoros to have SQL that is not optimized 314 | Network I/O considerations and optimal flow 70 | |
| Spring Batch—SQL SELECT sub tasks were locking UPDATES and INSERTS 328 | | |
| Some dynamic SQL make sure to use the correct data types 334 | | |
| SQL usage requires SQLCODE checking 406 | | |
| Security set up within cloud cluster and tied into normal validation processes 436 | | |

The Table of Contents for my Java Performance Class

Tuning Java processing that executes **billions** of SQL statements every night!

6

Working with billions of SQL statements, executed every day fine tuning java becomes extremely important to meet transaction response time, processing windows and overall performance.

Tuning a highly used web service or SQL statement to the forth decimal (0.0000) when it is executed billions of times daily can make or break processing windows, processing capacity limits, overall efficiencies and general performance.

Java Performance Problems – Only 4 things

- Connections managed poorly - or are really bad
- Java coding is trying to do too much
- Processing - Unit of Work (UOW) is confused
- Database or SQL are being used poorly

There are four main areas that are common places where performance can be optimized.

JDBC connections are the first place to start. Handling JDBC settings, optimizing JDBC connection and JDBC commit scope can play a major part in all java performance scenarios.

Using all programming languages to do much processing within one web service can be a problem. Sometimes the web service is trying to do an application Join, using too much data or performing too much house keeping when it is doing its service.

Unit of work optimization needs to be coordinated across any application frameworks, the database tables, SQL statements, SQL isolation types, and all their related component settings.

Database partitioning and SQL optimization, especially for web services are extremely critical for performance. All these runtime attributes should be analyzed for their I/O performance, parallelism and locking attributes.

Story behind every performance problem

- Java Performance issues always have a story
 - Need to understand the background, context and issues
- Java Performance issues are thrust upon us
 - Seems that every time it is an emergency
 - Always time to fix it later
- The java stories you are about to hear are true. The names and circumstances have been changed to protect the innocence. I am a consultant and I fix these complex situations.
- From this session you can learn how to fix them too
 - The mainframe or your server is not the problem!!
 - Problems are in the config, code and how it works with the database!



© Copyright 2023

Dave@davebeulke.com

8

I have been doing Db2 z/OS and LUW, systems/applications and design reviews that use COBOL, CICS, SAS, Assembler, C++, & Java for various financial, medical, government and other applications for decades. There are always stories behind the stories of how and why performance problems happen within these companies.

The performance problems' details are always different but have several themes have emerged that can help you debug your system, database, SQL or java application problems.

Usually the problems are not related to the mainframe or server or CPU or I/O capacity. Look deeper into the application configuration and the application code to truly understand the performance details.

Started Monday morning (of course)miserly loves company

Several Vice Presidents,
Developer Team
message me first thing
in the morning

- Processing has been executing for 2+ hours

Always ask
these five
questions!

- **Who** are the developers?
- **What** was the performance history in development and QA?
- **When** was the process supposed to finish? Runtime expectations
- **Where** was it tested? Where did it come from-home grown/vendor?
- **Why** is it a performance issues? SNAFU

© Copyright 2023

Dave@davebeulke.com

9

Who are the developers?

- You need to understand the level of programmer your application issue might be related to. If the programmer is experienced and has worked in the company environment for a long time and following the company standards the basic performance questions can be verified later.

What was the performance history in development and QA?

- Having a good development and QA testing regime is very important for gathering the performance history of the application or service. Gathering detailed history can quickly help focus the performance research. Know the history of your application and overall performance.

When was the process supposed to finish? Runtime expectations

- Sometimes management and programmer expectations are really misguided. The people involved need to understand what their application is trying to do.

Where was it tested? Where did it come from-home grown/vendor?

- Some of the biggest problems are vendor software and this is why development and QA testing are so important.

Why is it a performance issues? SNAFU

- The workload and performance expectations for processing billions of rows or SQL statements needs to set **before** execution. Communicate to everyone beforehand what performance expectation is for your application.

Starting the Java performance investigation

Gather all statistics from every or any monitor available

- Best if the performance problem is still running
- Immediately grab every data point available

Plan on the worst case scenario

- Production access
- Authorization for the monitor access
- Monitor always on/started
- Correct traces turned on via cloud, Pod, zOS, zLINUX, Server, Db2 etc....
- Capture the Web Server Logs & application logs
- Traceable situation – dump of the application runtime orabend?

Gather snapshot of cloud, Pod, cluster, z/OS Sysplex, your LPAR and any other tasks running

- New or existing workloads consuming normal or abnormal resources
- Gather overall capacity consumption, Db2 system settings, WLM setting/priorities and utilization reports
- Gather LPAR CPU utilization, zIIP exploitation /availability, Db2 parallelism usage

© Copyright 2023

Dave@davebeulke.com

10

Performance problems are never convenient. Before performance problems happen prepare your environment for the gathering statistics immediately. Statistics should be gathered for all levels within the processing stack.

- Operating system processing environment: Cloud, POD, z/OS CEC, Linux virtual server . Any hardware that is involved should be able to have statistics captured.
- Database systems: All data stores, files and databases involved should be able to have their performance statistics gathered. Having authorizations to get all statistics and everything related instantaneously is mandatory. Prepare for it.
- Any area that your performance team is not authorized for should have a partner with the other teams to be ready to capture statistics for these types of performance problems. Communicate and prepare for the performance problems ahead of time so you and your performance problem partners can immediately capture all the performance statistics available.

Java Process is locking itself - 1 out of 8 isn't bad

- The Framework to partition the work along the data partition boundaries
- Monitor shows
 - One job doing the work
 - Seven others doing nothing

| + Elapsed | PlanName | DB2 | Status | GetPg | Update | Commit | CORRID/JOBNM |
|--------------|----------|------|--------------|--------|--------|--------|--------------|
| + 02:03:52.1 | P J312NT | DSN3 | WAIT-SYNC-IO | 21311K | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |

© Copyright 2023

Dave@davebeulke.com

11

This is a screen of an application performance problem. The display shows the java application has spawned multiple sub-tasks using the company endorsed java framework.

The display shows that the one main processing module is doing all the work and none of the sub-tasks have done any database GetPages or processing.

Java frameworks such as Kubernetes, Hibernate and SPRING Batch can be a major headaches when debugging application performance issues. Since the frameworks can be customized for types of specialized processing your performance analysis needs to understand whether generic sub-tasks are being used or if there is only certain types of processing or SQL that is processed.

Found another Java program locking Db2

| + Elapsed | PlanName | DB2 | Status | GetPg | Update | Commit | CORRID/JOBNM |
|----------------|----------|-------|--------------|--------|--------|--------|--------------|
| + ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| + 167:11:22. P | J429JHS9 | DSN3 | WAIT-SYNC-IO | 64573M | 0 | 0 | J429JHS9 |

- Monitor shows this gem
 - One java job running for almost 7 days – **64 billion Get Pages**
 - Java process trying to work thru 330M customers – 18.9 billion rows of data
 - Using Frameworks within the processing
- Complete mess - Three Frameworks within this process
 - Using web services CALLS to some Db2 Native SPs
 - Also using:
 - Hibernate – SQL wasn't optimized for index structures
 - Spring Batch – SQL SELECT sub tasks were locking UPDATES and INSERTS
 - Some dynamic SQL – not using host variables

© Copyright 2023

Dave@davebeulke.com

12

It is also important to have a holistic overall perspective of the total environment when evaluating your performance problem,

In this example when the one java performance problem was happening another major processing effort was executing. Making sure there is enough capacity for every application can be a problem in shared clouds, z/OS or virtual Ansible, Kubernetes environments.

Be able to fully analyze your system capacity to validate performance problems are not being caused by other applications or lack of overall capacity. Sometimes one performance can cause another because of capacity, memory or I/O bottlenecks. When it rains it pours.

Architecture Frameworks

Discoveries and Recommendations

Avoid Spring Batch! Db2 can do it faster? but if you must consider it

- Architecture – Read, Process, Write concept
 - Know your database partitioning ranges
- Calculate result set size (#Rows * Column width)
 - Align JVM memory settings/Chunk with partitioning
 - Realize GCs done through one iteration of the process
 - Realize updates lock – ***isolate all locking*** appropriately
- Adjust minimize JVM settings per chunk
- Runtime Memory allocations appropriate
 - Optimize the Chunk Size
- Must have a CPU processor per JVM/subtask
- Chunk size determined by
 - One chunk “read” at time processes
 - Logical Read Process Write



© Copyright 2023

Dave@davebeulke.com

14

Transaction scope sometimes gets lost within the design team of the distributed java applications. Since the java development world of small reusable services are spread across the development staff sometimes the java web service does not concern itself with the UOW and the transaction scope.

This UOW and the transaction scope can be a major problem as the number of services called or involved in a transaction escalate and locking becomes a problem. Problems with sharing/maintaining the JDBC connection across UOW commits, the amount of in-memory data used in a Chunk, recoveries from application abends and java calls across to other JVMs can be main causes of performance problems and need to be researched and documented.

Avoid Hibernate!

- UOW & Transaction Scope
 - One hibernate definition many uses
- Persistence cache control
 - How much cache is enough
- Hibernate and persistence layer issues
 - Lazy, Evict, etc..... Data refresh rates/timing
 - Regular Dynamic SQL versus Hibernate SQL
 - Optimistic-lock – levels of locking
- Logging Considerations
 - How many logs are your transactions writing to?
 - Transaction archive, app server, web server, database(s).....



Sometimes the database access has totally been handled through a persistence layer such as Hibernate. These persistence layers are very convenient but are just another layer of debugging and complexity.

In most cases they are just another problem by rewriting the SQL, flushing their persistence poorly or forcing another layer for the transactions to go through.

Framework Configuration & Deployment Issues

Framework configuration data is difficult to deploy

- Has repeatedly delayed testing efforts
- Beware of tooling or loading configuration data to development, testing, and production database
- Configuration/reference data will not accelerate application development effort

Incremental framework config. data deployment can't easily be done

- Can't be shared - usually multiple versions on server add complexity to the issues
- Incremental improvements or changes can't be done online
- Framework doesn't provide availability, security and reliability
 - Fragile frameworks can ruin your application availability, reliability and scalability

When dealing with frameworks close attention needs to be paid to the configuration files and their detailed settings. These framework configuration settings handle all aspects of the framework JVM settings, their sub-tasking options, processing limits and other critical potential performance factors.

Also managing the framework configuration files is very important for the different development, QA and production environments. These frameworks configurations settings sometimes need to be customized for the environment size, memory and parallelism offered through sub-tasking. Pay particular close attention to any changes or adjustments that are needed for the different environments. Insights into these framework settings can help you optimize them for performance.

Collect more info on the running Java process

- Spring configuration & framework spawned the other processes
- Monitor shows
 - One job UOW is out of control, locking/syncing with other 7 processes

| + Elapsed | PlanName | DB2 | Status | GetPg | Update | Commit | CORRID/JOBNM |
|--------------|----------|------|--------------|--------|--------|--------|--------------|
| + 02:03:52.1 | P J312NT | DSN3 | WAIT-SYNC-IO | 21311K | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |
| + 02:03:51.9 | * J312NT | DSN3 | WAIT-LOCKPQS | 0 | 0 | 0 | J312NT09 |

© Copyright 2023

Dave@davebeulke.com

17

Within our java framework performance problem we can see that the sub-tasks are not performing any database SQL operations. This is evident through zero GetPages.

Also evident is that the main task and its sub-tasks could be interfering with each other given the WAIT-LOCKPQS status message from the monitor. The main processing UOW, its commit scope and analysis of how the sub-tasks are executed and managed needs further detailed performance analysis.

Researching the WAIT-LOCKPQS status message indicates that these sub-tasks are locking with other java processing within the database table structures. The SQL done through the main application and the SQL done in the sub-tasks needs to be fully researched to validate that they can coexist in a concurrent runtime environment.

Module Considerations – Thread Safe?

- Java class overall module size
- Discover the Synchronizing within the application
 - Static method forces each calling thread to block until no other thread is executing that static method
 - Is your framework and application thread safe?
- Where within the application is the module Serialization?
 - Application waits for processing of another framework UOW portion
- Duplicate java classes within projects
 - Or duplicate packages or paths within the applications

There are several signs that your Java processing may not be thread safe:

- Exceptions: If your Java process throws exceptions such as `ConcurrentModificationException` or `IllegalStateException`, this may indicate that the process is not properly handling concurrent access to shared objects or resources.
- Inconsistent or incorrect results: If your Java process produces inconsistent or incorrect results when run concurrently, this may indicate that the process is not properly handling UOW commits and/or synchronizing access to shared data or shared state reference data.
- Unexpected behaviors: If your Java process exhibits unexpected behaviors such as hanging, crashing, or deadlocking when run concurrently, this may indicate that the process is not properly handling concurrency and may not be thread safe.
- High CPU or memory usage: If your Java process uses a high amount of CPU or memory when run concurrently, this may indicate that the process is not efficiently managing concurrent access to shared resources and may be suffering from concurrency-related issues such as contention or memory leaks.

Overall, testing the thread safety of a Java process involves a combination of automated unit tests and manual testing, using tools and techniques that are specifically designed to verify the behavior of the process in a parallel environment.

Java Debugging

Discoveries and Recommendations

Gather runtime information is the best

- Thread details are critical performance & debugging information

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSAF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 05:29:59.162472 End=01/19/2023 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

Gathering runtime statistics is one of the most important performance tuning tasks. Regular monitoring of the production or QA environment and being able to gather runtime real time statistics is vital for evaluating SQL and Java performance.

The best practice for tuning is to ALWAYS have your Db2 basic monitoring on to gather run time SQL and Commit type statistics provided by most of the current Db2 tool vendors. The display above shows a typical set of statistics for our example performance problem.

All the statistics should be saved for historical performance documentation so that if any code updates occur performance can be maintain and not degrade. The statistics gathered above show several issues with the java application, commit scope, error handling, looping logic, SQL and unit-of-work commit frequency.

Java Debugging display – Looking at the code logic

- Debug display showing JDBC Db2 prepared statement
 - No parameter markers
 - Using invalid java types to receive Db2 column values
- SQLCODE ignored, copied and set
 - Variety of SQLCODE(s) - +100, -100, -811 etc.....
 - Framework runtime documentation there are no memory errors logged
 - Need to remember to CLOSE all SQL objects
 - Any opened Cursors are closed
 - stmt(s), resultSet(s) and Connection object
- Verify that the Java processing is THREAD SAFE
 - Subroutine integrity = transaction integrity = database data integrity

Also another best practice is to get a copy of the java code itself to inspect it for following coding standards.

Looking at the java code the first issue pops out that the program logic is manipulating the SQLCODE returned from the Db2 system based on the SQL executed. Also the Java module does not capture or Throw a java exception if the SQLCODE returns a negative SQLCODE,

Next examining the Java SQL statements shows that the WHERE clause values are using java variable instead of parameter markers. This is a very dangerous situation found sometimes in older java applications. Using variables is dangerous because the variables could hold malicious code passed from a web page or hacker. Using parameter markers filters and forces the correct data types for the Db2 values comparison.

Java debugging resources – Good test data is key

- Developer testing environment
 - Not enough data/performance testing resources
 - No End-to-End performance documentation for fixes
 - 1 in a Million data situations happen thousand of times - 100s of billions of rows in the application
- Performance needs to be given enough time and be a priority
 - Any changes especially SQL are documented, need to be given priority for performance analysis
- Object point analysis/Process point
 - Need to monitor memory objects within the application
 - Arrays, Hash Maps, Vectors, caching within any java object or framework component
- Management support structures
 - CM procedures drive performance analysis – checklist of components (SQL) tested/analyzed
 - Documentation gathered while going through the business requirements and functionality

© Copyright 2023

Dave@davebeulke.com

22

The development environment is critical for testing java applications that handle billions of transaction and databases with billions of rows. The development environment should be able to reflect the java runtime statistics to the forth or fifth decimal point to show their relationship and performance impact when executed a billion times.

Data analysis needs to be done on the development data to reflect all the boundary conditions of the indexes' columns, the data used in major and minor process logic and all invalid data or unvalidated data. Examples of all these various situations is the absolute minimum which can still be 10s of million of test data rows.

Java Memory analysis should be done against all the Arrays, HashMaps, Vectors and memory objects referenced within your application. Any shared state reference data or data referenced across JVMs needs detail concurrency validation.

Also any Kubernetes, Spring Batch or Hibernate setting standards or customization needs to be well documented and communicated because it can have a huge performance impact as it is customized through the development, QA and finally the production environments.

Debugging is a matter of Dev/QA – JUnit Testing

Automated tools should be a requirement

- Assign resources from development team
- Integrate automated tests during development
 - Develop bulk and unique test data for validation and certification
 - SQL EXPLAINS captured within the test – Production-Like statistics environments

Generated test data for services

- Understanding of business service usage – workload business variables
- Data missing keys, codes and proper optional types parameters
- Invalid type data flowing into services parameters
- Platform to platform conversions
 - (EBCDIC to UNICODE) or (UNICODE to UNICODE) or (ASCII to ASCII to ASCII) or XX to YY

To test whether a Java process is thread safe, you can use a combination of unit tests and manual testing.

For unit tests, you can use JUnit or a similar testing framework to create test cases that exercise the code in the Java process concurrently, using multiple threads. For example, you can use JUnit's `@Test` and `@Before` annotations to create a test case that initializes a shared object, creates multiple threads, and has each thread perform operations on the shared object.

To verify that the process is thread safe, you can use JUnit's `assertThat()` method to make assertions about the state of the shared object after the threads have completed their operations. For example, you can assert that the shared object is in the expected state, that no concurrent modifications have occurred, or that no exceptions have been thrown.

In addition to unit tests, you can also manually test the Java process to verify its thread safety. To do this, you can use a tool such as `jconsole` or `jstack` to monitor the process and see how it behaves when multiple threads are executing concurrently. You can also use a tool such as `jvisualvm` to monitor the memory usage and garbage collection behavior of the process, to see if there are any concurrency-related issues.

Benefits of Profiling Tools – Know memory/GC

- Integrated Profiling within RAD or *<insert your IDE here>*
 - Provide a trace of the normal flow of the application
 - Discover the exceptional java classes
 - Discovered 60,000+ SQL calls for process
 - Uncovered java classes used from old releases
 - Realize where cache is being used for transactions
 - Discovered data validation being done multiple times
- Web services/applications logic flow being verified
 - Uncover the performance truth about an application
 - See the time spent within each java package, *class* and *method*
 - Understand the modules referenced and their size within JVM
 - Discover unnecessary looping, arrays, memory objects and tables being built
 - Look for Java Race conditions



The introduction of the Java Flight Recorder (JFR), which is a low-overhead profiling and event collection framework built into the JVM, allows for deep and detailed analysis of the performance of a Java application. The Profiling tool is commonly built into the Java IDE and references this infrastructure for producing statistics.

In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the number of leaks caused by the execution of a test case is so minimal that it is hardly noticeable in one run.

You can use repetitive tests at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks is easier. First, the memory usage before running the module is recorded. Then, a fixed set of test cases are run repeatedly. At the end of the test run, the current memory usage is recorded and checked for significant changes.

Remember, garbage collection must be suggested when recording the actual memory usage by inserting `System.gc()` in the module where you want garbage collection to occur, or using a profiling tool, to force the garbage collection event to occur.

Java Profiling – Memory Objects- HashTable, Vectors etc..

• Is there a test history of Profiling the Java application

- Thread Statistics
- Monitor Statistics
- Threads Visualizer

• Shows java module where the processing is taking the most time

| Package | <Base Time.. | Average Ba.. | Cumulative .. | Calls |
|---|--------------|--------------|---------------|-------|
| com.gsl.yespedia.cobol.V9 | 0.019530 | 0.000868 | 0.019530 | 22 |
| MoneyCalculation | 0.019530 | 0.000868 | 0.019530 | 22 |
| MoneyCalculation() | 0.018755 | 0.018755 | 0.018755 | 1 |
| log(java.lang.String) void | 0.000298 | 0.000043 | 0.000298 | 7 |
| doCalculations() void | 0.000244 | 0.000244 | 0.000695 | 1 |
| rounded() java.math.BigDecimal() java.math.BigDecimal | 0.000041 | 0.000010 | 0.000041 | 4 |
| main() java.lang.String[] void | 0.000039 | 0.000039 | 0.000775 | 1 |
| getAverage() java.math.BigDecimal | 0.000031 | 0.000031 | 0.000039 | 1 |
| getDifference() java.math.BigDecimal | 0.000028 | 0.000014 | 0.000028 | 2 |
| getPercentage() java.math.BigDecimal | 0.000027 | 0.000027 | 0.000043 | 1 |
| MoneyCalculation() java.math.BigDecimal, java.math.BigDecimal | 0.000024 | 0.000024 | 0.000041 | 1 |
| getPercentageChange() java.math.BigDecimal | 0.000023 | 0.000023 | 0.000038 | 1 |

© Copyright 2023

Dave@davebeulke.com

25

Look at areas where collections of objects are used. Typically, memory leaks are composed of objects within the same class. Also, collection classes such as Vector and HashTable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a HashTable object does not remove its reference to the retrieved object.

Heap consumption that indicates a possible leak during periods when the application server is consistently near 100 percent CPU utilization, but disappears when the workload becomes lighter or near-idle, is an indication of heap fragmentation. Heap fragmentation can occur when the JVM can free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap to larger contiguous spaces.

Another form of heap fragmentation occurs when objects that are less than 512 bytes are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation until a heap compaction occurs.

Heap fragmentation can be reduced by forcing compactions to occur. However, there is a performance penalty for forcing compactions. Use the Java -X command to see the list of memory options.

SUSPENDS, DEADLOCKS & ROLLBACKS - Concurrency

- Usually undocumented and/or tracked when Java is involved!
- Poor Java exception handling
 - No documentation on any database or general Java exceptions
 - Email all java exceptions to the developers!
- Java memory structures trying to mimic a database
 - Cache involved in problem situations/transactions/deadlocks
 - How often is a memory/cache error reported/involved in these errors?
- How many insert/update/deletes within the Java application
 - When is the processing done within the UOW – Commit points

To guarantee that a Java process is thread-safe, you need to ensure that access to shared resources is properly synchronized to avoid race conditions and other concurrency issues. Here are some steps you can take:

1. Use proper synchronization: Use synchronized blocks or methods to ensure that access to shared resources is properly synchronized. Make sure that all threads that access the shared resource use the same synchronization mechanism.
2. Use atomic operations: Use atomic classes such as AtomicInteger or AtomicReference to perform atomic operations on shared resources.
3. Avoid shared mutable state: Avoid sharing mutable objects across threads or ensure that access to the shared objects is properly synchronized. Immutable objects can be safely shared across threads without synchronization.
4. Use thread-safe classes: Use thread-safe classes such as ConcurrentHashMap or CopyOnWriteArrayList instead of non-thread-safe classes like HashMap or ArrayList when working with shared collections.
5. Test for thread safety: Use testing frameworks like JUnit to test your code for thread safety. Write test cases that simulate concurrent access to shared resources and verify that the results are correct.
6. Avoid using global variables, as they can be accessed by multiple threads and lead to unexpected behavior.
7. Test the thread safety of your code by running stress tests or simulating concurrent access to shared resources.
8. Consider using a thread-safe programming paradigm such as Actor Model, Message Passing or CSP.
9. Use synchronized blocks or methods to control access to shared resources. This will ensure that only one thread can access the resource at a time, preventing conflicts.
10. Use volatile keyword for variables that are shared among threads. This will ensure that the variable's value is always up-to-date across all threads.

By following these steps, you can ensure that your Java process is thread-safe and avoid common concurrency issues such as race conditions, deadlocks, and data corruption.

Java Class, Method Dependencies

Discoveries and Recommendations

Looking at you threads in detail – red is really bad

- Thread details are critical performance & debugging information

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSAF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 05:29:59.162472 End=01/19/2023 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

All of the highlighted Red displayed statistics need to be further investigated. The most important highlighted consideration is the ABORT = 2 situation. Given that the statistics show the process has Inserted, Deleted and Updated so many rows it could be a rollback situation or data integrity problem.

The Open/Close Cursor figures should always match and this usually indicates serious looping, UOW and commit scope issues.

We will talk about these red highlighted processing details in the coming slides.

Dependency on frameworks issues – SPs to the rescue!

- Convert frameworks to Db2 Native Stored Procedures
 - Helps remove dynamic Hibernate SQL statements
 - Convert into stored procedures will dramatically reduce CPU & overall TCO
 - Exploits Db2 zIIP CPU resources that are free
- Beware of stale cache data problems
 - Dependent on the cache for data
 - Reference data
 - Slowly changing data within the reference data
 - Transaction UOW data
- Local cache and failover dependencies
 - Beware of multiple servers/clouds for performance

© Copyright 2023

Dave@davebeulke.com

29

Db2 native stored procedures are precompiled SQL statements that are stored in the database and can be executed by applications. Using stored procedures can provide a number of benefits for application performance, including:

- Improved performance: Stored procedures are precompiled, which means that they can be executed more quickly than regular SQL statements that are executed on the fly. This can help to improve the overall performance of your application by reducing the amount of time it takes to execute database queries.
- Reduced network traffic: When you use stored procedures, the application sends a request to the database to execute the stored procedure, rather than sending the actual SQL statement. This can help to reduce the amount of network traffic between the application and the database, which can improve performance and reduce the load on the network.
- Enhanced security: Stored procedures can help to improve the security of your application by allowing you to control which operations are allowed on the database and to enforce security policies at the database level. This can help to prevent unauthorized access to the database and protect sensitive data.
- Common code: Stored procedures can be CALLED from many different types of applications are a good methodology to maintaining and centralizing code for common application logic and standardizing company processing.

Overall, using Db2 native stored procedures can provide a number of benefits for application performance, including improved performance, reduced network traffic, and enhanced security. If you're looking to optimize the performance of your application that interacts with a Db2 database, using Db2 native stored procedures can be an effective way to do so.

Leverage Java `executeBatch` against Db2 – simulate SPs

- Executing many database SQL statements as part of one network round-trip
 - Knowing your UOW
 - Turn off your JDBC - `AutoCommit(false)`
 - `PreparedStatement` of your SQL with parameter markers
 - Set the SQL parameters and use `addBatch`
 - Keep your java batch size matched to your debugging skills → 10, 25, 100?
 - Exploits Java and Db2 SQL
- Beware of stale cache data problems
 - Dependent on the cache for data
 - Reference data
 - Slowly changing data within the reference data
 - Transaction UOW data
- Local cache and failover dependencies
 - Beware of multiple servers/clouds for performance

© Copyright 2023

Dave@davebeulke.com

30

The `executeBatch` method is used to execute a batch of SQL statements. This method has several performance advantages over executing single SQL statements one at a time:

1. **Reduced network traffic:** When using `executeBatch`, multiple SQL statements can be sent to the database server in a single network call, reducing network traffic and improving performance.
2. **Reduced round-trip time:** Since multiple statements are executed in a single batch, the round-trip time between the application and database server is reduced, resulting in improved performance.
3. **Improved database performance:** By executing multiple statements in a single batch, the database server can optimize its internal execution plans, resulting in improved database performance.
4. **Reduced parsing overhead:** When executing multiple SQL statements one at a time, each statement needs to be parsed individually, which can result in significant overhead. However, with `executeBatch`, the parsing overhead is reduced since multiple statements are parsed together.
5. **Improved transaction performance:** When using `executeBatch` within a transaction, the entire batch of statements can be treated as a single transaction, resulting in improved UOW transaction performance.

In summary, the `executeBatch` method in Java provides several performance advantages by reducing network traffic, round-trip time, parsing overhead, and improving database and transaction performance. If Db2 native stored procedures are not available you can get the similar performance advantages by using `executeBatch` to optimize the performance of their applications that interact with a database.

Client Situations

Discoveries and Recommendations

Every picture tells a story don't it!

- First impressions

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSAF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 05:29:59.162472 End=01/19/2023 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

Having a real-time production display of your Java application is critical to understanding the performance of your code. It is especially important to understand the items displayed components.

The commits, aborts, number and different types of SQL statements, the SQL creation and runtime attributes, and their locking frequency and usage are all critical to performance. These figures are the beginning point for more detailed performance statistics.

Data Integrity – huge recovery restart issue

- ABORTS=2 Did all/any the Updates, Insert or Deletes happen?
 - When did the Abort happen?
 - Why didn't the application abend/stop?

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSAF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 05:29:59.162472 End=01/19/2023 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

There are several performance factors and considerations when application abort occurs:

Data Recovery: If the application was processing data at the time of the failure, data recovery becomes necessary to ensure that the data is not lost or corrupted. If the application UOW iteration or the commit scope/mechanisms are not designed properly the database data may need to be recovered and potentially cause application downtime.

Resource Allocation: When the application is restarted, the resources need to be allocated again. This includes CPU, memory, and I/O resources. If the application requires a lot of resources, restarting it can have a significant impact on the performance of other applications running on the same system.

Application Design: The design of the application can also have an impact on the performance impact of an abort and restart. For example, an application that has a lot of stateful connections to the database may take longer to restart than an application that uses stateless connections.

In general, the performance impact of a JVM, Java, or database application abort and restart can be significant. It is important to design the application with failure and recovery in mind and to test the application thoroughly to ensure that it can rollback UOWs properly, restart and recover quickly and with minimal impact on performance.

SQL Loops – Looping & thread safe errors

- Open Cursor \neq Closed Cursor
 - Difference spells trouble – 1.3m OPEN versus 183k CLOSE = 1.1m difference

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSASF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 05:29:59.162472 End=01/19/2023 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

In a DB2 Java application, the DB2 SQL OPEN and CLOSE cursor counts should be the same to ensure optimal performance and efficient use of system resources. When a cursor is opened, resources such as locks and buffers are allocated to the cursor. These resources are only released when the cursor is closed.

If the number of OPEN and CLOSE cursor counts is not the same, it can indicate a programming error or a design flaw in the application. In addition, a large difference between the number of OPEN and CLOSE cursor counts may indicate unused connections to the database, unused locks, and other resources being held unnecessarily potentially even impact other applications running on the same system.

For example, if an application is repeatedly opening and closing cursors in a loop, this could lead to excessive overhead and decreased throughput. It is important to carefully monitor cursor usage and ensure that cursors are being used efficiently in order to optimize application performance.

Lock Usage – Commit problems

- Why or how are Lock tables being used?
 - Which table(s)? When in the processing? Why were the locks taken?

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSAF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 08:29:59.162472 End=01/19/2023 10:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

© Copyright 2023

Dave@davebeulke.com

35

The Lock Table =7 indicates lock escalation has occurred with the application execution. Since the application was designed against the coding standards to have a regular commit frequency lock escalation should have never occurred.

When the number of row-level or table-level locks held by a transaction exceeds a threshold, lock escalation may be triggered. This lock escalation situation takes a overall table lock and further restricts access of other concurrent applications to the table's data and potentially causing an application deadlock situation.

In addition to potentially causing a deadlock situation the number of locks held within an application execution can be a severe memory requirement and dramatically impact concurrency.

SQL Statement reuse – Not required with SPs

- Number of Describes = 3.9m
 - Why?

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSAF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 05:29:59.162472 End=01/19/2023 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

The Db2 DESCRIBE operation is done to understand the metadata of the result set returned by the SQL statement. The DESCRIBE operation is required for any dynamic SQL statement within your application. The DESCRIBE is done when the PreparedStatement is executed and not when the SQL values of the parameters are set.

So it is very important to validate the application logic to verify that the PreparedStatement is done once and that the logic looping is when the parameters are set. When the parameter marker values are updated and the statement is executed, the values are substituted into the already prepared statement and it is executed without the need for a new DESCRIBE. This results in better performance since the statement does not have to be described and optimized again for each execution with different parameter values.

REUSE Pleased – Large # of PREPARES – SPs

- How about some thread or SQL reuse? 3.4m

```
THREAD HISTORY SQL COUNTS
HPLN
+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSADF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2023 05:29:59.162472 End=01/19/2023 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit = 44 Abort = 2 Select = 0
+ Open Cursor = 1344387 Close Cursor = 183722 Fetch = 1163383
+ Insert = 168902 Delete = 38088 Update = 1940714
+ Describe = 3909142 Lock Table = 7 Prepare = 3492098
+ Grant = 0 Revoke = 0 Set Rules = 0
```

The large number of PREPAREs further indicates that the java application programming iteration logic is looping through `PreparedStatement` SQL creation.

Given that the number of DESCRIBEs and PREPAREs are different shows there is no consistencies and the UOW and commit logic are flawed.

The number of DESCRIBEs and PREPAREs show further evidence that these millions of overhead java SQL operations could be avoided by encapsulating the database SQL Calls within a Native Db2 Stored Procedure. If a Native Db2 Stored Procedure was used these millions of extra DESCRIBEs and PREPAREs would be zero improving dramatically overall performance, reducing system and application CPU.

Java logic issues - Cursor loop control

1. The number of ABORTS=2. How are these impacting UOW and integrity of the data INSERTED/DELETED? Aborts should ALWAYS be = 0.
2. OPEN CURSOR<>CLOSE CURSOR which is causing a bad -479 SQLCODE error.
3. LOCK TABLE – Was there a need for the LOCK TABLE statements?
 - Which tables and where in the processing logic?
4. The number of DESCRIBES is extraordinarily high and needs to be investigated.
 - Usually there are only a few.
5. The number of SQL PREPAREs is high. Is the logic PREPAREing for every SQL statement for execution, usually these PREPAREs are based on logic unit-of-work (UOW) processing.
 - The number of PREPAREs doesn't seem to reflect UOW logic, CURSOR usage or number of DESCRIBEs.

Given that there were so many problems especially the ABORTS, the java execution was corrupted and its processing experienced a JVM memory overlay.

This memory overlay caused the SQL PreparedStatement error of a - 479 error which indicates the number of DESCRIBED columns is greater than the allocated space. The number of columns in the table is larger than the allocated descriptor.

Remember the DESCRIBE operation is only needed once for a SQL PREPARED statement. The purpose of the DESCRIBE operation is to determine the access path and validate the SQL statement syntax. This operation is performed when the PREPARE statement is executed, and the access plan is saved for subsequent executions of the prepared statement for each time the looping logic SETs the SQL PREPARED statement parameter marker values .

Number of Clouds, Servers, UOW & Connections

- How many connections does the application really need?
 - Database(s)
 - Db2 LUW, Db2 z/OS & Oracle in one transaction
 - MQ-Queues
 - Inbound and outbound
 - Web and App Server connection threads
- Parallelism and connection state
 - Mind the state of all of these connections
 - How many READ/UPDATE locks are held
 - How long each is active
 - How long the transaction UOW is!
 - What is the UOW within each OS, database, within which table(s)
 - Get Current_Time experiment demonstration!

© Copyright 2023

Dave@davebeulke.com

40

When analyzing java application performance it is critical to pay attention to a number of performance critical factors.

The number platforms referenced, their JDBC commit frequency parameters, the JDBC variables, the transaction isolation level, SQL statement types used and the Result Set types thread used within the application.

In Java JDBC Db2 database connection, there are two commit modes that can be set using the "setAutoCommit" method of the "Connection" class:

- **Auto-commit mode:** `connection.setAutoCommit(true);` In this mode, each SQL statement is automatically committed to the database as soon as it is executed. This can be a big performance over head and needs to be noted and monitored.
- **Manual commit mode:** `connection.setAutoCommit(false);` In this mode, multiple SQL statements can be grouped together as a single transaction, which can then be committed or rolled back together. This mode is good for controlling commit scope and potential rollback situations.

It is recommended to use manual commit mode for transactions that involve multiple SQL statements to ensure data consistency and rollback of all the SQL statements involved in the UOW.

Developers needs help – edge/boundary test data

- Developers want their own Cloud/fork/dev environment
 - Consolidate for common testing environments
 - Helps team communicate
 - Test and debug without worrying about locking rows used by others
- Provide standardized test data to reflect complexity of the business
 - Isolated testing for only a small variety of data
 - Don't have full range or set of data or complexity of data needed for all tests
 - How many different tests are runs? How many are enough?
- How many network round trips to get the transaction completed?
 - Within the application, UOW and java class(es)

Generating test data for edge cases can be challenging, but here are some techniques that can be used:

1. **Boundary values:** Determine the minimum and maximum values for each field and use those values to test the limits of the application. For example, if a field is defined as an integer between 1 and 100, test it with values 1, 100, and values just below and above those.
2. **Random values:** Generate random values within the valid range of each field. This can help identify unexpected behavior that may not have been considered in the application logic.
3. **Invalid values:** Test the application with invalid values that fall outside of the valid range for each field. This can help identify how the application handles errors and exceptions.
4. **Combination of inputs:** Test the application with various combinations of Boundary input values. This can help identify issues that may only occur when multiple fields are used together.
5. **Real-world scenarios:** Generate test data based on real-world scenarios that the application is intended to handle. This can help ensure that the application is behaving correctly in expected use cases.

Overall, a combination of these techniques can be used to generate test data that covers a wide range of scenarios and edge cases. It is important to consider the specific requirements and constraints of the application and data model when generating test data.

System z/OS Java Considerations – Bigger problems

- Know the number of zIIPs/GCP within your Sysplex/CEC-LPAR
 - Know the number of z/OS Db2 zParms for Db2 SQL DBM1 parallelism
- Know your WLM scope, its workloads divisions and different priorities
 - 36+ WLM categories can be used to define CPU priorities
 - Between all Production and test subsystems
 - Where are your GCP, zIIPs being used
 - What is the wait time for CPU and I/O resources
 - Know the Service Classes of your workloads, Db2 Address Spaces
- Monitor the Storage I/O Subsystem(s)
 - Schedule of different heavy I/O Activities
 - DR offloads, CLONEing systems & Image Copy databases
 - Priorities of the different Java workloads –

© Copyright 2023

Dave@davebeulke.com

42

The key control parameters in WLM that can impact priorities, velocities, CPU usage, and I/O usage are:

Service Classes: WLM uses service classes to define the priority of workloads on the system. Service classes can be assigned a relative importance value, which is used to determine the priority of the workloads in that service class.

Goals: Goals are used to define performance objectives for service classes. Goals can be defined for response time, throughput, and other performance metrics. WLM uses these goals to allocate CPU resources to service classes in a way that meets the defined objectives.

Velocity: Velocity is a measure of the relative importance of a workload compared to other workloads in the same service class. Velocity is calculated based on the workload's importance value and its resource usage. WLM uses velocity to allocate CPU resources to workloads within a service class.

Dispatching Priority: Dispatching priority is a value that is assigned to each workload by WLM. Workloads with higher dispatching priorities are given access to CPU resources before lower-priority workloads.

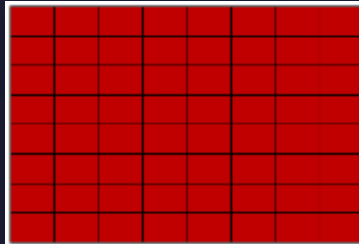
CPU Usage: WLM can limit the amount of CPU resources that are allocated to each service class or workload. This can be done using various techniques, such as capping, goal mode, and shared processor pools.

I/O Usage: WLM can also limit the amount of I/O resources that are allocated to each service class or workload. This can be done using techniques such as priority queuing and I/O pacing.

Overall, these parameters can be configured and tuned to ensure that the system resources are allocated in a way that meets the performance objectives of the workloads running on the system.

Developer trust and verification – recovery

1. Resolve all Java JVM, GC, Race, Serialization, memory, framework, Sysplex/Cluster/Pod/VM capacity, rollback issues
 - Repeatably run, rollback, recovery & restarted successfully
 - Against the entire database
 - Backups and recovery procedures well documented
 - Understand all table(s) locks acquired and released



It very difficult to pinpoint a single Java application parameter that impacts performance the most as it largely depends on the specific application and its workload. However, in general, memory allocation and garbage collection tend to be critical factors in Java performance.

Inefficient memory usage can lead to frequent garbage collection, which can cause long pauses and negatively impact response time. Additionally, poor garbage collection settings can result in excessive heap usage and long garbage collection times. Therefore, optimizing memory usage and garbage collection settings can often have a significant impact on Java application performance.

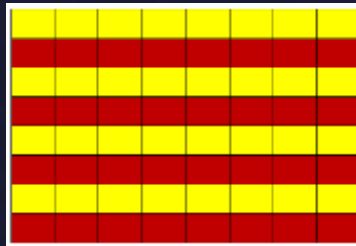
The best way to test a Java Db2 program iteration pattern, commit processing, and checkpoint restart capabilities is through a combination of unit testing and integration testing.

To effectively test iteration patterns and commit processing, it's important to use test data that covers a range of scenarios, including edge cases and boundary conditions. Additionally, test cases should be designed to validate the program's behavior in the event of errors or unexpected conditions, such as network or rollback/restart failures.

Overall, a comprehensive testing strategy that includes both unit testing and integration testing, as well as a variety of test scenarios and data, is the best way to ensure that a Java Db2 program is functioning as intended and is capable of handling various complex rollback/restart processing scenarios.

Developer trust and verify – partition independence

2. Run against a single partition of the driver of the processing
 - Validate that there absolutely no issues
 - Certify application partition level restart-ability
 - Repeatably run, rollback, recover & restarted successfully
 - Against the single database partition



© Copyright 2023

Dave@davebeulke.com

44

The optimum number of Java programs that use a Db2 database that can be run in a standard 1GB JVM depends on several factors such as the amount of memory required by each program, the complexity of the SQL queries executed by each program, and the amount of data processed by each program.

As a general rule, it is recommended to allocate at least 256MB of memory for the JVM itself, leaving 750MB for application code and data. Based on this assumption, and assuming that each program requires around 50MB of memory, it would be possible to run around 15 to 20 Java programs concurrently in a single 1GB JVM.

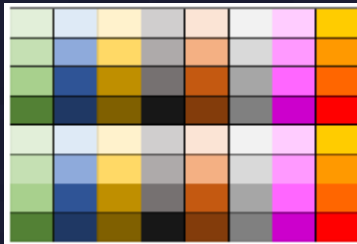
In addition to z/OS WLM settings if you are running in a Linux environment there are additional considerations. Linux Containers (LXC) is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel.

The Linux kernel provides the cgroups functionality that allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any virtual machines, and also the namespace isolation functionality that allows complete isolation of an application's view of the operating environment, including process trees, networking, user IDs and mounted file systems.

Developer trust and verify – lock isolation verification

3. Run against a fraction of a single partition of the processing

- Validate absolutely no issues, especially rollback, recovery or restart issues
 - Verify no Sysplex/Cluster/Pod/VM *capacity*, complexity or configuration issues
 - When/how/what automation to recover and then restart processing
 - Certify no intra-partition locking or lock escalation issues
 - Validate storage efficiency and network capacity can handle throughput



Kubernetes and frameworks provides several settings that can be used to share JVM resources among multiple PODs:

- CPU Limits and Requests: These settings can be used to limit and request CPU resources for a POD. By setting appropriate values, CPU resources can be allocated to the PODs in a way that ensures efficient utilization of resources.
- Memory Limits and Requests: These settings can be used to limit and request memory resources for a POD. By setting appropriate values, memory resources can be allocated to the PODs in a way that ensures efficient utilization of resources.
- Resource Quotas: These settings can be used to limit the amount of resources that a namespace can consume. By setting appropriate values, the overall usage of resources across all the PODs in a namespace can be controlled.
- Horizontal Pod Autoscaler: This setting can be used to automatically scale the number of PODs in a deployment based on the resource utilization of the existing PODs. By setting appropriate values, the number of PODs can be increased or decreased dynamically based on the resource demands of the application.
- Affinity and Anti-Affinity: These settings can be used to ensure that PODs are scheduled on nodes that have sufficient resources and are not overloaded. By setting appropriate values, PODs can be scheduled in a way that ensures efficient utilization of resources across the cluster.
- Node Selector: This setting can be used to ensure that PODs are scheduled on nodes that have the required resources. By setting appropriate values, PODs can be scheduled in a way that ensures efficient utilization of resources on the nodes.

Overall, these settings can be used to ensure that JVM resources are shared efficiently among the PODs running in a Kubernetes cluster.

Infrastructure -no java, JVM, config, performance issues

- Sysplex/Cluster/Pod/VM capacity constrained
- Limited zIIP Processor(s) availability
- Workload Capacity Mismanagement
- Poor Processing Capacity Prioritization
- Storage Capacity misallocation
- Network efficiencies are adequate
- Suboptimal database orchestration flow
- Identify poor processing design and iteration sequence
 - Note normal, exception and balancing workloads

Kubernetes can be run on z/OS using IBM's z/OS Container Extensions (zCX) feature. When running multiple Kubernetes PODs within a single z/OS container, the following performance parameters should be considered:

- CPU: The z/OS container should be allocated enough CPU resources to support the workload of multiple PODs. The CPU resources can be allocated using Workload Manager (WLM) on z/OS.
- Memory: The z/OS container should be allocated enough memory to support the workload of multiple PODs. The memory resources can be allocated using z/OS Resource Management Facility (RMF).
- I/O: The z/OS container should be allocated enough I/O resources to support the workload of multiple PODs. The I/O resources can be allocated using z/OS Workload Manager (WLM) and z/OS Workload Manager Extended (WLM-Extended) on z/OS.
- Network: The z/OS container should have enough network bandwidth to support the workload of multiple PODs. The network bandwidth can be allocated using z/OS Network Control Program (NCP) and z/OS Communications Server.
- Storage: The z/OS container should be allocated enough storage to support the workload of multiple PODs. The storage resources can be allocated using z/OS DFSMS (Data Facility Storage Management Subsystem) and z/OS Workload Manager (WLM).

Overall, the performance of multiple Kubernetes PODs running within a single z/OS container will depend on the available resources, workload characteristics, and configuration parameters of z/OS and Kubernetes. It is recommended to perform thorough testing and tuning to ensure optimal performance.

Summary - Java

- Avoid any and all Java Frameworks
- Know your JDBC Connection setting – commit scope, security authorization
- Know your config – JDBC settings, JVM, GC frequency, Heap, max memory requirements
- Understand usage of Java Packages, Classes and versions – hopefully only 1
- Minimize Java open source packages since they are updates & security liabilities!!
- Understand usage of JVM, memory cache, java types ie; Hash Maps etc.
 - Use IDE Profiling to understand memory usage
 - In terms of database information cached
 - Use IDE Debug mode to fully document logic looping
 - Automate JUnit testing
 - Understanding how much Garbage your program produces during UOW
 - Fully document GC overhead and frequency at full capacity performance load
- Test your UOW java processing, transaction and for being Thread Safe/serialization
- Understand failover and scalability liabilities limitations – Cloud/LPAR/server/app

While Spring Batch, Hibernate and Kubernetes can provide benefits for managing containerized applications in a z/OS environment, there are also some potential disadvantages to consider:

- Complexity: Frameworks can be complex to set up and manage, especially for organizations that do not have prior experience with container orchestration tools.
- Resource overhead: Running frameworks requires additional resources, including memory, CPU, and storage, which can impact the performance of other workloads running on the system.
- Compatibility: Some z/OS applications may not be compatible with containerization or may require significant modification to run in a framework or containerized environment.
- Cost: Implementing frameworks in a z/OS environment may require additional hardware, software, and licensing costs.
- Security: While frameworks can provide security benefits, it also introduces new attack surfaces and potential vulnerabilities that need to be carefully managed.
- Availability: If Kubernetes itself fails, it can impact the availability of all applications running on the platform.
- Skills: These frameworks requires specific skills to operate and maintain, which may not be available in-house, leading to additional training costs or reliance on external consultants.

Summary - Java Db2

- Fully understand connection usage and UOW scope
 - Logic loop reflection of UOW
 - Understand Locking scope of UOW
 - Locks usage, locks acquired, retained and released during UOW and full runtime
 - Understand COMMIT scope and failover
- SQL etiquette
 - SQL column to Java data type usage
 - SQL parameter markers
 - SQL EXPLAIN captured with production level statistics
 - SQLCODE checking/handling immediately after SQL executed
 - CLOSE - clean up all database resources – SQL Cursors, StmtS, ResultSets, Connection
 - Leverage Db2 Native Stored Procedures

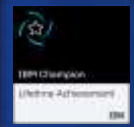
There are several important considerations for guaranteeing SQL processing performance:

- **Efficient SQL statements:** Write efficient SQL statements that optimize data access and minimize data processing. This includes using proper indexing, avoiding unnecessary joins, and minimizing the amount of data returned.
- **Proper configuration of the database and operating system:** Configure the database and operating system to optimize performance for SQL processing. This includes configuring buffer pools, sort memory, and other database and system parameters.
- **Proper resource allocation:** Ensure that sufficient resources, such as CPU, memory, and I/O, are allocated to support the SQL processing workload. This may involve tuning the number of threads, connection pool size, and other settings to optimize resource usage.
- **Effective use of caching:** Use caching techniques, such as database caching or application caching, to reduce the amount of time required to access frequently used data.
- **Monitoring and analysis:** Monitor SQL performance metrics, such as response times and throughput, and analyze performance data to identify potential bottlenecks or issues that may impact SQL processing performance.

Overall, a combination of efficient SQL statements, proper configuration, resource allocation, effective caching, and monitoring and analysis can help to guarantee SQL processing performance.


Thank you!!

Please fill out your session evaluation before leaving!



Share these tips
with everyone!

Performance BLOG:
www.DaveBeulke.com

 @DBeulke

 davebeulke

✓ Send any questions or comments
to Dave@DaveBeulke.com

© Copyright 2023

Dave@davebeulke.com

49