The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Session #S317067
Oracle WebLogic Management for DBAs
Glen Hawkins, Director of Product Management
Madhav Sathe, Senior Product Manager
Agenda

- Oracle Enterprise Manager Overview
- Application Performance Challenges
- Database Access Scenarios
- Performance Issues and Diagnostics
- Best Practices
- Key Takeaways
- Q&A
Oracle Enterprise Manager Overview
Business-Driven IT Management

Business Users and Customers

User Experience

Business Transactions

WEB PORTAL
PRODUCT CATALOG
ORDER ENTRY
OTHER SERVICES

Business Services

Business-Driven Application Management
- Understand business needs
- Manage from business perspective

Integrated Application-to-Disk & Cloud Management
- Eliminate management silos
- Create agile IT for dynamic business

Integrated Systems Management & Support
- Proactively identify and fix problems
- Maximize business productivity

APPLICATIONS
MIDDLEWARE
DATABASES
SERVERS
STORAGE

Services Cloud

Integrated Application-to-Disk and Cloud Management

Oracle Support

Integrated Systems Management and Support
Integration with My Oracle Support

Oracle Support

Knowledge Management  Problem/SR Management  Configuration Management

Customer

Performance Management  Problem Diagnosis  Provisioning & Patching

Hardware  Databases  Middleware  Applications

© 2010 Oracle
Application Performance Challenges
Application Performance Challenges

• Slow and slower response time
• Slower under heavy load
• Sporadic hangs and aberrant errors

• HTTP errors, content errors
• System locks up
• Sudden chaos / Unexpected Errors

- Users
  - Web customer
  - Internal user
  - Batch job

- Middleware
  - Web store
  - Billing
  - Financial System

- Database

Oracle
Who will own the issue?

MW Admin/Developer: It is a DB issue

User: What happened to my order transaction?

DBA: Your application is broken!
Lack of Visibility
Typical Multi-Tiered Java EE System Leaves the Administrators Blind

Service Levels Measured Here...

Apache OHS
iPlanet

Web Server

Business Transactions Application Artifacts
Metadata Caching Tier

Real issue is here

WLS, Java & Data Grid

...Blame assigned Here

Database
Why Do Conflicts Arise?
Two Silos – Two Disconnected Islands

• Application Performance Monitoring Tools
  • Application and Middleware health and performance monitoring
  • Identify flow of business transactions

• Database Monitoring
  • Database health and performance issues
  • Visibility into SQL performance

Challenges for DBAs
• Little or zero visibility into how application SLAs are measured
• Database shared across multiple applications and DBAs lack ability to associate performance issues with business transactions
• Numerous ways the database is accessed
Database Access
Scenarios
Java Database Connectivity (JDBC) is an industry standard for connecting Java applications to databases. Java code needs JDBC driver to be able to access the database. Driver manages connection with the database which used by Java. PreparedStatement, CallableStatement:
- Used to make SQL, PLSQL calls over JDBC connection respectively
  - E.g. `stmt = connection.prepareStatement("update Orders set pname = ? where Prod_Id = ?"); stmt.setString(1, "Tom");`
- ResultSet – used to return the result of the SQL calls
Most Popular Oracle JDBC Drivers

• JDBC OCI client-side driver (a.k.a. “Thick” or “Native” driver)
  • Type 2 driver; Uses Java native methods to call entry points in an underlying C library (OCI); so platform specific.
  • Requires an Oracle client installation of the same version as the driver.

• JDBC Thin client-side driver
  • Type 4 driver; Pure java to connect directly to Oracle, so Platform independent; Does not require Oracle client software to be installed; Require the server to be configured with a TCP/IP listener
  • Most favored in Java applications
Additional Oracle JDBC Drivers

- JDBC Thin server-side driver
  - **Type 4 driver**: Pure java to connect directly to Oracle
  - Runs inside Oracle database and used to access remote Oracle database

- JDBC Server-Side Internal driver
  - **Type 2 driver**: Uses Java native methods to call entry points in an underlying C library which is part of Oracle server process and communicates directly with the internal SQL engine
  - Fastest way for Java code within Oracle to access the same database
JDBC Datasource In WebLogic

- 'Lookup' Datasource in application using JNDI name

- Datasource encapsulate connection details in connection pool

- Multi Datasources for load balancing & failover

- Connection Pool - helps to increase the performance by keep active connections to the database in the pool

  Applications 'borrow' connection from the pool, use it and return it

- Type 4 JDBC driver (Client side thin driver)
Key Application Data Source Concepts for DBAs

- Multi data sources
  - Load balancing
  - Failover
- Pool size
  - Inadequate pools size adversely impacts performance
  - Too many connections in pool will waste resources on database and middleware
- Statement cache
  - PreparedStatement and CallableStatement are costly on database and even on communication between database and WebLogic
  - Cache executable statements that are used repeatedly, such as in a loop or in a method that is called repeatedly
Key Application Data Source Concepts for DBAs

• Connection leaks
  • Connection that does not return to the pool after use
  • Waste valuable resources

• Transactions
  • UserTransactions: programmer defines transaction boundaries
  • Container Managed Transactions: container defines transaction boundaries
  • For distributed transactions use XADataSource
  • Problems: Dirty Read, Non Repeatable Read, Phantom Read
  • Isolation Levels: READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE
Object-Relational (O/R) Frameworks

- Java Persistency API (JPA) layer provides OR mapping
- Application code does not use SQLs directly; instead perform operations on the Objects and JPA layer auto-generates SQLs
- E.g. ADF-BC, Toplink, Hibernate, iBate

[Diagram showing the relationship between JEE & Web Services, Persistence Layer, JDBC, SQL, and objects.]

- Persistence Layer:
  - object-level querying and creation results are objects
  - results are returned as raw data

- JDBC:
  - API uses SQL or database specific calls

- SQL:
  - object creation and updates through object-level API
Using Database Adapters in SOA and Composite Applications

- SOA allows loose coupling of disparate components based on standards
- Database adapters are used to integrate database access logic into the SOA processes
- For example, you may want to integrate a PL/SQL block in your BPEL process or a SOA Composite Application
Coherence (Distributed In-Memory Caching) & Its Impact

- In-memory data grid layer between database and application
- Works on principle – 80% of the requests use 20% of the data
- Cached data is typically partitioned across multiple nodes running on different machines
- Coherence nodes are Java processes providing storage for cache
  - Each node holds a partition of a cache with a backup partition on another node to guarantee reliability
- Ideally should reduce database access for most requests
- Cache persistence policies: Refresh-Ahead, Read-Through, Write-Behind, Write-Through
BPEL Dehydration Store

- Dehydration is a process by which BPEL engine stores process metadata and (sometimes) state of process instance to the database.

- When does BPEL dehydrate process instance?
  - At the end
  - In flight, at breakpoint activities (if any)
  - If idempotent activity is set to false

<table>
<thead>
<tr>
<th>Transient process</th>
<th>Durable process</th>
</tr>
</thead>
<tbody>
<tr>
<td>No breakpoint activities</td>
<td>Contains breakpoint activities</td>
</tr>
<tr>
<td>Save state only once</td>
<td>Save state more than once</td>
</tr>
<tr>
<td>Nothing to recover</td>
<td>Can recover from the last dehydrated state</td>
</tr>
</tbody>
</table>
Oracle Advanced Queuing

- JMS: Java Messaging Service is a messaging standard used by disparate systems to exchange messages
  - Queues are used as ‘message buckets’
- Point to Point: Senders and receivers decide on a common queue in which to exchange messages. Each message is consumed by only one receiver.
- Publish-Subscribe: messages can be consumed by multiple receivers
Performance Issues and Diagnostics
Cross-tier Issues

- Connection availability
  - Inadequate connections in JDBC pool
  - Max allowed DB sessions
- Long running SQLs
  - DB slowness
  - SELECT on large (un-indexed) tables
- No of database cursors

- Chatty sessions
- Table locks, transactions, timeouts
- SQLs generated by Java Persistence API (JPA) layer
- Low cache throughput
- Coherence departed nodes
- Cascade delete

**Typical questions that arise…..**

- How to identify issues in production?
- How to reproduce production issues in test/development environment?
- How will Administrators map these issues to high level business transactions?
Datasources, Connection Pools and Transactions

• JDBC Open connections, connection requests, in use, pool size, etc
• Cached statements used, transactions committed, rollbacks, etc
• Proactive monitoring using Alerts and Notifications
Manage Configuration Assets

- JDBC Resource configuration
  - Connection Pool – Connection URL, Driver classname, Max, Min, Inactivity timeout, Statement cache size, etc.
  - Datasources – JNDI name, pool name, prefetch size, etc.

- Multi datasource – JNDI, datasource list, algorithm

- Automated, daily collection of configuration data, installations and patches
- Search across enterprise for specific information
- Make configuration changes
- Compare configuration

- Save configuration to repository as ‘Gold Image’
- Compare with other WebLogic servers in environment
JVM Diagnostics With Database Visibility

- Trace Java thread to database session
  - Identify in-flight Java threads waiting for database resource
  - Drill to SQL

- Trace database session to Java thread
  - View database sessions waiting for or holding locks
  - Identify Java thread holding DB session
Quickly Identify Cross-Tier Issues
Bridge the gap between Database and Middleware

MW Admin: Look, my order request is stuck because of this SQL
DBA: Looks like your SQL is waiting for table lock taken by another SQL
MW Admin: Ok, lets kill the other session so that this key user transaction can go ahead

DB State Causing Application Issues

Are majority of the threads waiting for DB?
In-Context Architecture & SQL Performance

- Detailed method and SQL metrics for robust monitoring and troubleshooting
- Architecture display gives context to metrics for easy navigation and comprehension

Find SQL performance based on caller
Monitoring & Managing Coherence Caches

- Identify caches using backend too frequently
- Check cache persistence policy
- Find cache health
  - Find cache balance on multiple nodes
  - Compare performance of multiple nodes side by side
  - Correlate metrics for diagnostics
- Check cache eviction policy, expiry policy, etc
- Ensure cluster stability by auto-starting perished nodes
Best Practices
JDBC Connection Pool Size

Ensuring application gets connection quickly

• Connection is not shared by multiple threads/requests
  • Thread *acquires a lock* on JDBC connection once `getConnection()` returns the connection, *releases the lock* when connection is closed (returned to pool)

• Consider following WebLogic monitoring metrics for pool size
  • JDBC load & efficiency
    • `WaitingForConnectionsTotalCount`, `WaitingForConnectionsSuccessCount`, `ActiveConnectionsCurrentCount`
  • Application load
    • Completed Requests (per minute/second, etc)
    • Pending Requests (requests in queue)
JDBC Connection Pool Size

Ensuring application gets connection quickly

• Set pool Initial Size to Max Size
  • Datasource maintains these many connections in the pool. IF this is not set response time may be impacted as for some requests getConnection() will lead to actually creating a new connection on database

• Do not set the pool size too high
  • DBA controls ‘Max Allowed DB Sessions’ on database
  • Maintaining an ‘open connection’ is resource intensive

• Configure datasource to make connection requests wait if pool is exhausted
JDBC Statement Caching
Improve statement execution by caching

• Use statement cache to reduce CPU load on database and network roundtrips
  • Can prevent the overhead of repeated cursor creation
  • Can prevent repeated statement parsing and creation
  • Cache is maintained at connection level but size is configured at datasource level
• Default ‘LRU’ caching algorithm is best for most situations
• Increasing cache size will dramatically increase application performance
JDBC Statement Caching
Improve statement execution by caching

• Notes for DBAs (Database specific behavior)
  • Each cached statement may consume a ‘cursor’ on database so if too many statements are cached in WebLogic cluster it may exceed maximum open cursors limit on database & will lead to error
  • Any structural change to table which cached statement references may lead to statement execution failure
Transaction Performance Tips

• Define transaction boundaries in code, setAutoCommit(false)
• Do not use 'Serializable' isolation level unless there is a clear need in the application for Zero Concurrency
• Never have a database transaction span user input
  • SQLs may lock rows/tables for updates
  • Use ‘optimistic locking’ if user input is required in between a transaction
Transaction Performance Tips

• Make Transactions Single-batch

  BEGIN TRANSACTION
  UPDATE TABLE1…
  INSERT INTO TABLE2
  DELETE TABLE3
  COMMIT

• Preferable because the database obtains all the locks necessary on the various rows/tables, and uses & releases them in one step

• Using separate statements and commits results in many more client-to-database transmissions and holds the locks in the database for much longer
Important Design Time Considerations
Ensuring developers and DBAs are synchronized

• Process as much data as possible inside database
  • Put data processing logic where data is
  • Avoid needless data transfer from database to application

• JDBC Batch updates
  • Group queries in a ‘batch’ to reduce no of database calls and improve performance

• Optimize queries
  • Reduce the no of rows returned as far as possible

• Control connection leaks
  • Close connection in ‘finally’ block
  • InactiveConnectionTimeout on datasource to forcibly return a connection to the datasource if there is no activity
Important Design Time Considerations

Ensuring developers and DBAs are synchronized

- Keep operational dataset small
  - Move non-current data to other tables and do joins to these tables for the rarer historical queries
  - Index and cluster your table so that the most frequently used data is logically and physically localized

- Row Prefetch
  - JDBC fetches that number of rows at a time from the database during the query to get better performance
  - The extra data is stored in client-side buffers for later access by the client
  - Value ’10’ is optimum for most situations
Tuning Tips for JPA O/R Frameworks
ADF-BC, CMP, Toplink, Hibernate, etc

- Make sure that SQL generated by your finder/select method uses indexes
- Avoid findAll() methods on large tables
- Use read-only beans when appropriate
  - Reduce number of activations and “passivations”
- Enforce primary key constraint at database level
  - Avoid extra SQL for constraint checking by container/JPA provider
- Exploit DELETE CASCADE at database level
  - Container generates multiple SQL statements

Use Coherence as In-Memory Data Grid to drastically reduce load
Improve application performance, scalability & reliability
Monitor and Configure Coherence For High Performance

- Plan coherence cache capacity to *minimize database access*
- Ensure coherence cluster stability
  - Nodes may crash or leave cluster because of various reasons directly impacting available cache size, throughput
- Monitor performance trends for cache hit ratio, database access, work load distribution on nodes, etc
Monitor and Configure Coherence
For High Performance

• Key configuration parameters that affect database access
  
  • **Persistence Policy** – Refresh-Ahead, Read-Through, Write-Behind, Write-Through
  
  • **Eviction Policy** – LRU, LFU, **Hybrid** (evicting those that are accessed least frequently and were not accessed for the longest period first)
  
  • **High Units** – Maximum number of units that can be placed in the cache before pruning occurs
  
  • **Low Units** – Number of units that the cache will be pruned down to when pruning takes place
  
  • **Expiry Delay** – Amount of time from last update that entries will be kept by the cache before being marked as expired
  
  • **Flush Delay** – Time interval between periodic cache flushes, which will discard expired entries from the cache
Key Takeaways

• Never leave any middleware or database tier unmanaged
  • Application issues can be found anywhere and often leads to fire fighting everywhere if unprepared
• Use a single console to monitor the entire stack
• Use top-down application monitoring tool that provides end-to-end visibility across the stack vertically and horizontally
Key Takeaways

- **Diagnose** the issues in the *production* environment
  - Extremely difficult, impossible in most cases, to reproduce the issue in staging, QA, or development
  - Protect your top and bottom line by *catching and diagnosing* issues in production
- Fine tune configuration at all tiers for optimized performance
Product Roadmap
# Roadmap for WebLogic Server Management

## Patches

**Platform Support**
- WebLogic Server Virtual Edition
- Additional 64-bit IBM JREs
- Exalogic Management

**JVM Diagnostics**
- JRockit Virtual Edition
- Continued new JRE support
- SQL Hash look-up

**Request Monitoring**
- Single request instance transaction tracing

**EM / RUEI Integration**
- Enhance contextual drill-down

**Provisioning/Cloning**
- Expand platform support (Solaris, AIX, Windows)

## Next Release

**Agent/Manager update**
- Consolidate agent deployment
- Bi-directional impact analysis with Database diagnostics
- Full Configuration Management (ACC/CCC) Integration
- JVM as a target

**Lifecycle Management**
- WLS Patching
- WLS Gold Image Provisioning
- Java EE application provisioning

**Application Performance Management**
- Middleware Diagnostics Advisor (MDA)

## Future Releases

**EM / E2E Integration**
- BTM Integration
- Consolidated end-to-end agent (ADP/JVM/BTM)
- Consolidated automated installation
- RUEI data integration for dimensional analysis of transactions

**Platform Support**
- Latest versions of Fusion Apps and FMW

**Support Integration**
- Offline Diagnostics expansion framework
Oracle Enterprise Manager 11g Resource Center
Access Videos, Webcasts, White Papers, and More
Oracle.com/enterprisemanager11g
### Additional Sessions / Demos / Hands-On Labs

<table>
<thead>
<tr>
<th>SESSION ID</th>
<th>TITLE OF TECHNICAL SESSION</th>
<th>DATE &amp; TIME</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>S317067</td>
<td>WebLogic Server Management for Oracle DBAs</td>
<td>Thursday 9:00 am</td>
<td>Marriott Hotel, Salon 9</td>
</tr>
<tr>
<td>S316996</td>
<td>Oracle Enterprise Manager Grid Control Deployment Best Practices</td>
<td>Thursday 10:30 am</td>
<td>Moscone South, Room 102</td>
</tr>
<tr>
<td>S317066</td>
<td>Deep Java Diagnostics and Performance Tuning: Expert Tips and Techniques</td>
<td>Thursday 1:30 pm</td>
<td>Marriott Hotel, Salon 9</td>
</tr>
<tr>
<td>S317060</td>
<td>Managing User Experience in the Cloud: Lessons from eBay</td>
<td>Thursday 1:30 pm</td>
<td>Marriott Hotel, Golden Gate A</td>
</tr>
</tbody>
</table>

Enterprise Manager DEMOgrounds in Moscone West

Fusion Middleware Management Hands-On Lab on Tuesday 2:00-3:00 pm and Thursday 10:30 – 11:30 am in Marriott Hotel, Salon 12/13, YB Level
SOFTWARE. HARDWARE. COMPLETE.