Introduction to Oracle Coherence Distributed Data Grid
Paolo Ramasso
Principal Sales Consultant Oracle Italy

Agenda
• Why Oracle Coherence?
• What is Oracle Coherence?
• Coherence in the Application-Tier
• How Coherence Works
• Grids and Data Grids
• Coherence and DataSources
• How much effort
• Coherence and other Oracle Products
• Future Directions
• Demo (Swing-Demo and Labs)

Scalaiblity Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>How</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical “scaling-up”</td>
<td>Increase resources in existing server(s)</td>
<td>❖ Relatively simple process (can be achieved overnight) ❖ Transparent to system architecture and development</td>
<td>❖ Comparatively expensive hardware (niche) ❖ Limited Scalability (physical limits typically encountered) ❖ Increases cost of failure</td>
</tr>
<tr>
<td>Horizontal “scaling-out”</td>
<td>Add more servers</td>
<td>❖ Comparatively inexpensive hardware (commodity) ❖ Virtually unlimited scalability possible (typically greater than scale-up approach)</td>
<td>❖ Applicable only when a system is designed to “scale-out” ❖ May require months of rework to achieve ❖ Scalability may be limited by “network” ❖ Requires additional administration</td>
</tr>
</tbody>
</table>

Why Oracle Coherence?
Coherence Let Your Application Scale Predictably

- Coherence: Designed to scale-out the Application-tier
  - Standard Java Applications (JSE, non-JEE, container-less)
  - Web Applications (session state)
  - Middle-tier Applications (JEE, container-based)

- Artifacts that can be scaled
  - Application and User State (objects)
  - Object Access (crud)
  - State Mutation Notifications (events)
  - Processing (updates, transactions)

Why Scaling-out the Application-Tier is Hard!

- Anyone can write network software these days…
  - Java, .NET, Ruby etc… all provide network abstractions to transfer data between applications on separate servers, even around the world
  - You can learn it from the Internet
  - Anyone can write code to make software communicate with other bits of software

- However…
  "It's extremely difficult to write software that ensures an unpredictably (dynamically) growing collection of servers connected by an unreliable network can continuously work together without losing information (or work) in a manner that itself is linearly scalable"

- Significance…
  - Achieving all of these things in the same product
  - Working together means “consensus” has to be maintained!

The Coherence Approach…

- Traditional scale-out approaches limit
  - Scalability, Availability, Reliability and Performance

- In Coherence…
  - Servers share responsibilities (health, services, data…)
  - No SPoB
  - No SPoF
  - Massively scalable by design
  - No Masters / Slaves etc.
  - Members work together as a team
What is Oracle Coherence?

- Coherence (deployment perspective)
  - Single Library
  - Standard Java Archive “JAR” for Java
  - Standard Dynamically Linked Library “DLL” for .NET connectivity (.Net 1.1 and 2.0)
  - Other libraries for integration (Databases, Spring…)
  - No 3rd party dependencies!
  - Minimal “invasion” on standard code
  - Configurable implementations of standard Map / Dictionary interfaces (NamedCache)
  - Provides Predictable Scalable Caching
  - “RemoteException” free distributed computing

- Coherence (architectural perspective)
  - Scale-out Applications State
  - Reliable Data Management / Data Abstraction Layer
  - Effortlessly Cluster Applications (clustering infrastructure)
  - Web (session management)
  - Front, Middle, Back Tiers
  - Thick Clients (AWT, Swing, Console, RCP…)
  - JSE or JEE
  - Remote Connectivity
  - Business Continuity and Disaster Recovery
  - Provide a Data Grid

Scaling the Application-tier with Coherence

<table>
<thead>
<tr>
<th>Approach</th>
<th>How</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Coherence to share and manage objects</td>
<td>- Introduce Coherence into Application(s)</td>
<td>Simple</td>
<td>New paradigm</td>
</tr>
<tr>
<td>(application state)</td>
<td></td>
<td>Transparent and</td>
<td>People tend to use old</td>
</tr>
<tr>
<td>&quot;Coherence is responsible for my objects&quot;</td>
<td></td>
<td>Automatic Partitioning of</td>
<td>patterns with it – that</td>
</tr>
<tr>
<td></td>
<td>- Use Coherence</td>
<td>Data</td>
<td>don’t work or are overly</td>
</tr>
<tr>
<td></td>
<td>NamedCache API (derived from</td>
<td>RemoteException-free</td>
<td>complicated</td>
</tr>
<tr>
<td></td>
<td>java.util.Map)</td>
<td>distributed computing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Start multiple Coherence-enabled</td>
<td>Itself is massively</td>
<td>Configuration isn’t easy</td>
</tr>
<tr>
<td></td>
<td>processes to scale-out (load balance)</td>
<td>scalable</td>
<td>(at first) mainly</td>
</tr>
<tr>
<td></td>
<td>objects (data)</td>
<td>Displaces other</td>
<td>because of the new paradigm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>technology (messaging)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extremely</td>
<td>Takes time for people</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configurable</td>
<td>to “trust” the technology</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Use Coherence to share and manage objects (application state)
- "Coherence is responsible for my objects"
Coherence in the Application-Tier

How Coherence Works
Introduction to NamedCaches

- Developers use NamedCaches to manage data

- NamedCache
  - Logically equivalent to a Database table
  - Store related types of information (trades, orders, sessions)
  - May be hundreds / thousands of per Application
  - May be dynamically created
  - May contain any data (no need to setup a schema)
  - No restriction on types (homogeneous and heterogeneous)
  - Not relational

Introduction to NamedCaches

- NamedCache implementations are configurable
  - Permit different mechanisms for organizing data
  - Permit different runtime characteristics (capacity, performance etc…)

- A mechanism for organizing data is often called a Topology or more generically, a Scheme

- Coherence ships with some standard schemes
  - You may configure / override / create your own!

Coherence Schemes

The Local Scheme

- Non-Clustered Local Cache
  - Contains a local references of POJOs in Application Heap
  - Why:
    - Replace in-house Cache implementations
    - Compatible & aligned with other Coherence Schemes
  - How:
    - Based on SafeHashMap (high-performance, thread-safe)
    - Size Limited (if specified)
  - Configurable Expiration Policies:
    - LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable
The Local Scheme

The Distributed Scheme

- Sophisticated approach for Clustered Caching
- Why:
  - Designed for extreme scalability
- How:
  - Transparently partition, distribute and backup cache entries across Members
  - Often referred to as ‘Partitioned Topology’
- Configurable Expiration Policies:
  - LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable
The Distributed Scheme

- Each Member has logical access to all Entries
  - At most 1 network-hop for Access
  - Regardless of Cluster Size

- Linear Scalability
  - Cache Capacity Increases with Cluster Size
  - Coherence Load-Balances Partitions across Cluster
  - Point-to-Point Communication
  - No multicast required (sometimes not allowed)

- Seamless Failover and Failback
  - Backups 'promoted' to be Primary
  - Primary 'makes' new Backup(s)

- Invisible to Application
  - Apart from some latency on entry recovery
  - Recovery occurs in Parallel
  - Not 1 to 1 like Active + Passive architectures

- Any Member can fail without data loss
- Configurable # backups
- No Developer or Infrastructure intervention
The Distributed Scheme

• Benefits:
  • Deterministic Access and Update Latency (regardless of Cluster Size)
  • Cache Capacity Scales with Cluster Size Linearly
  • Dynamically scalable without runtime reconfiguration

• Constraints:
  • Cost of backup (but less than Replicated Topology)
  • Cost of non-local Entry Access (across the network)
    • (use Near Scheme)

The Distributed Scheme

• Lookup-free Access to Entries!
  • Uses sophisticated ‘hashing’ to partition and load-balance Entries onto Cluster Resources
  • No registry is required to locate cache entries in Cluster!
  • No proxies required to access POJOs in Cluster!

• Master / Slave pattern at the Entry level!
  • Not a sequential JVM-based one-to-one recovery pattern

  • Cache still operational during recovery!

Distributed Scheme
Clients & Servers

• Sometime Members should not store Data
  • Members lifetime in the cluster is short
  • Members join and leave frequently

• Each time Membership changes, partitioning and distribution needs to be re-assessed

• To reduce impact, Members may be ‘storage disabled’

Distributed Scheme
Clients & Servers

• Cache Client
  • Member has storage disabled for Partitioned Topologies

• Cache Server
  • Member has storage enabled for Partitioned Topologies

• Same Cache API
  • Transparent to developer
  • Storage is (re)configured outside of code
Distributed Scheme Clients & Servers

Scheme Composition

- Schemes may be ‘composed’ to address system requirements and SLAs.
  - Keep recently used data in-memory, the rest on disk

- Base Schemes
  - Class, Local, Replicated, Distributed, Extend*

- Composite Schemes:
  - Near, Overflow (to disk)
  - Allow other schemes to be ‘plugged in’

Coherence Schemes

The Near Scheme
The Near Scheme

- A composition of pluggable Front and Back schemes
  - Provides L1 and L2 caching (cache of a cache)
- Why:
  - Partitioned Topology may always go across the wire
  - Need a local cache (L1) over the distributed scheme (L2)
  - Best option for scalable performance!
- How:
  - Configure ‘front’ and ‘back’ topologies
  - Configurable Expiration Policies:
    - LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable

The Near Scheme Coherency Options

- Local Cache Coherency Options
  - Seppuku: Event-Based ‘Kill Yourself’ Invalidation
  - Standard Expiry: LFU, LRU, Hybrid, Custom
- No messaging system required for invalidation!
  - Built into infrastructure
  - High-performance
Coherence Schemes

Data Source Integration
Coherence Persistence Integration

- **Coherence integrates tightly with databases**
  - Read through – pass query through to database
  - Write through – persist cache data to database
  - Refresh Ahead – asynchronously refresh cache from database
  - Write Behind – asynchronously persist data to database

- **Persistence solution integration**
  - Out of box with TopLink
  - Out of box with Hibernate
  - Simple JDBC
  - Other

---

How Much Effort?

---

Accessing & Updating data in a cache

- To create a named cache:
  ```java
  CacheFactory.ensureCluster();
  NamedCache myCache = CacheFactory.getCache("employees");
  ```

- The NamedCache interface implements java.util.Map, so you can use the standard map methods such as:
  ```java
  get, put, putAll, size, clear, lock, unlock...
  ```
- Also implements JCache

---

Accessing & Updating data in a cache

- To put data into the cache use:
  ```java
  myCache.put("Name","Paolo");
  ```

- To retrieve data use:
  ```java
  String name = (String)myCache.get("Name");
  ```
**QueryMap Interface**

- A set containing all of the open trades
  ```java
  Set openTrades = trades.entrySet(new EqualsFilter("isOpen", BOOLEAN.TRUE));
  ```

- A set containing people with a last name beginning with "Mac"
  ```java
  Set macPeople = people.entrySet( new LikeFilter("getLastName", "Mac%"));
  ```

- A set of keys of people with a last name beginning with "Mac"
  or "Mc"
  ```java
  Set macPeopleKeys = people.keySet(
    new OrFilter(
      new LikeFilter("getLastName", "Mac%"),
      new LikeFilter("getLastName", "Mc%")));
  ```

**Features: InvocableMap Interface**

- The total value of the open orders
  ```java
  BigDecimal result = orders.aggregate(
    new EqualsFilter("isOpen", Boolean.True),
    new BigDecimalSum("getValue"));
  ```

- The categories of books on sale over $25
  ```java
  Set categories = stock.aggregate(
    new AndFilter(
      new EqualsFilter("isOnSale", Boolean.True),
      new GreaterThenFilter("getPrice", 25)),
    new DistinctValue("getCategory"));
  ```

**QueryMap Interface - Examples**
Features: Observable Interface

Entry Processors

- `com.tangosol.util.InvocableMap.EntryProcessors` are agents that perform processing against Entries directly where they are being managed
  - Requests are sent directly to owners to do work
  - Equivalent to “agents” executing services in parallel on the data in the cluster
  - Processing...
    - may mutate cache entries, including creating, updating or removing, or
    - just perform calculations, or anything else!

Example

- Usually you create your own custom implementations
- Simply sub-class `com.tangosol.util.processors.AbstractProcessor`

```java
class StockSplitProcessor extends AbstractProcessor {
  ...
  Object process(Entry entry) {
    Position position = (Position)entry.getValue();
    position.setAmount(position.getAmount() * factor);
    entry.setValue(position);
    return null;
  }
}
```

- Now to run this on all entries

```java
stocks.invokeAll(AlwaysFilter.INSTANCE, new StockSplitProcessor());
```
Fusion Middleware Integration

Coherence Grid with Fusion Middleware

- Deployed as separate tier
  - Provides shared in-memory data grid to all FMW
- Embedded in middleware applications
  - Provides in-memory data grid for application layer
- Deploy both in middleware and separate tier
  - Common scenario

Coherence as a Separate Tier with FMW

Oracle Fusion Middleware Coherence Integration Points

- Session Sharing and Data Caching
- Business Intelligence
  - CTS, CSS, OAM, Reporting, Search, BI, Time
- Integration & Process Management
  - Business Process Management
- Application Server
  - J2EE, WebSphere, Oracle
- Grid Infrastructure
  - Coherence Grid and Clusters
- Data Caching, Extended
  - State Replication, Shared
- In-Memory Infrastructure
- Shared Service for Java, .NET
- Coherence Data Grid Service
- Systems Management
  - Systems Application Management
- Identity Management
  - Identity Federation
  - Single Sign-On
  - Identity Administration
Coherence Embedded and Separate Tier

SOA Application
Java Application
Portal Application

Oracle Application Server

In Memory Coherence Data Grid

Oracle Coherence: SOA Integration

Custom Applications
Workflow
Rules
Data Grid Service
Packaged Applications

Accelerating BPEL Performance with Coherence

• Extreme BPEL performance using in memory clustered Coherence for dehydration store

Oracle Web Center Portlet Session Sharing

WSRP Producer Server
WSRP Portlet 1 Producer
WSRP Portlet 2 Producer

Web Center
Portlet 1
Portlet 2
Portlet 3
Portlet 4

WSRP Producer Server
WSRP Portlet 3 Producer
WSRP Portlet 4 Producer

In Memory Coherence Data Grid for WSRP Producer/Consumer Session Sharing
Maximum Availability Architecture

Asymmetric Active/Passive

Web Tier

Global Router

WAN

WebCenter

SOA

J2EE

IDM

Firewall

RAC

Coherence Data Grid Service

Average Latency and Bandwidth

Swing-Demo and Code Labs

ORACLE