An Oracle White Paper
July 2013

Oracle Coherence 12c
Planning a Successful Deployment

Disclaimer

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.

Executive Overview

Taking a Coherence Proof of Concept (PoC) application and making it ready for production involves a range of considerations and additional steps, which are not always obvious to architects and developers. The environment of a production application is quite different to that in development and steps to secure, monitor and manage the application will need to be taken. High Availability plans to meet expected SLA's will need to be put in place, which in turn will involve appropriate procedures to upgrade, patch and recover the system.

This white paper aims to address these issues and provide guidance about what to consider and how to plan the successful rollout of a Coherence application to a production environment.

Introduction

Planning the rollout of a Coherence application into a production environment can vary greatly in its complexity. For simple Coherence applications many of the considerations in this white paper will not be relevant, but each main section has recommendations that should be generally applicable.

The intention of this white paper is not to replace or replicate existing Coherence documentation, but to complement and extend it, and where possible any existing sources will be referenced. It includes guidance, tips and experiences from "field engineers", customers, Oracle Support and Coherence Engineers. With this in mind its structure is akin to a checklist, addressing each step of the process in sequence.

Design, Development and Testing

Best Practice

Save Time, Reuse, Don't Re-Write

Before you start writing code or architecting a new solution check out if what you want to do has already been implemented, either in the Coherence API or in the Coherence Incubator. The Coherence Incubator contains a number of production ready code templates to address a range of problems. If your problem cannot be solved directly by either of these sources then consider if either can be customized or extended. Take advantage of the testing, monitoring, tuning and documentation that has already been done.

Coherence is a Distributed Technology so Develop and Test in a Distributed Environment

Coherence is a distributed technology. It may be tempting to perform all your development and testing on a single machine, but Coherence will behave differently in a distributed environment. Communications etc. will take longer, so flush out any design and development issues early by regularly testing in a distributed environment.

Don't waste time identifying problems that have already been fixed

Begin you testing with the latest patch release of Coherence available from Oracle Support. This may seem obvious, but it is quite common for developers to spend time investigating Coherence problems that have already been fixed. If you wish to be updated when a new patch becomes available then subscribe to the RSS feed. Unfortunately patches are only available from Oracle Support, not OTN. So if you are evaluating Coherence and not yet a customer, please speak to you local Oracle contact.

Follow Best Practice

Many common problems can be avoided by following the Production Checklist, Performance Tuning guide and Best Practice for Coherence*Extend. Read and re-read these while you are testing your application or setting up a production environment. Also perform network tests, using the bundled datagram and multi-cast tests, before you start. These tools will identify networking problems before you even start to test your application.

For instance, if you are planning on using a virtualized environment then not using the latest network drivers can dramatically effect network communications. Running the datagram test can highlight such problems, by showing a throughput that is significantly less than the expected ~100MB p/s - over a 1GBE network. Failing to perform these simple tests can easily waste a lot of time diagnosing error messages in Coherence log files or looking at other symptoms, rather than focusing on the cause of the problem.

Avoid Anti-Patterns

These are design patterns that don't work well in a distributed architecture. Below are just some of these patterns and practices you should consider carefully before using;

- **Distributed transactions, which require multiple entries to be copied for read-consistency and rollback.** Alternative approaches are discussed later in this white paper.
- **Client locks, which increase contention and hinder scalability.** Executing operations where the data resides using entry processors, where local locks can be taken out, is usually a better alternative.
- **Client based processing.** A better alternative is to use entry processors or "Live Events", to process the data where it resides. Entry processors are like database stored procedures but also allow processing to be re-run, or failed-over, if the first processing node fails. The new "Live Events" feature in Coherence 12c completely transfers processing to the cluster. Event Interceptors are registered to fire when the cluster or cache entries change and are executed where the event is raised, inside the cluster. This enables Coherence to scale, failover and recover the processing interceptors perform. Interceptors can also be chained to execute more complex processing logic.
- **Un-indexed, un-targeted queries.** Although indexes consume memory, they can improve query performance by several orders of magnitude. Explore custom indexes to minimize their overhead or analyze index performance to balance storage and performance considerations. This is discussed in detail later in this white paper.
- **Data affinity, which would result in a highly unbalanced data distribution.** Consider alternative data models.

Need Help? – Search the Documentation and Check the Forum

It seems obvious but it's often not done. Most common tasks and features are explained in details in the documentation and the online documentation can be searched too.

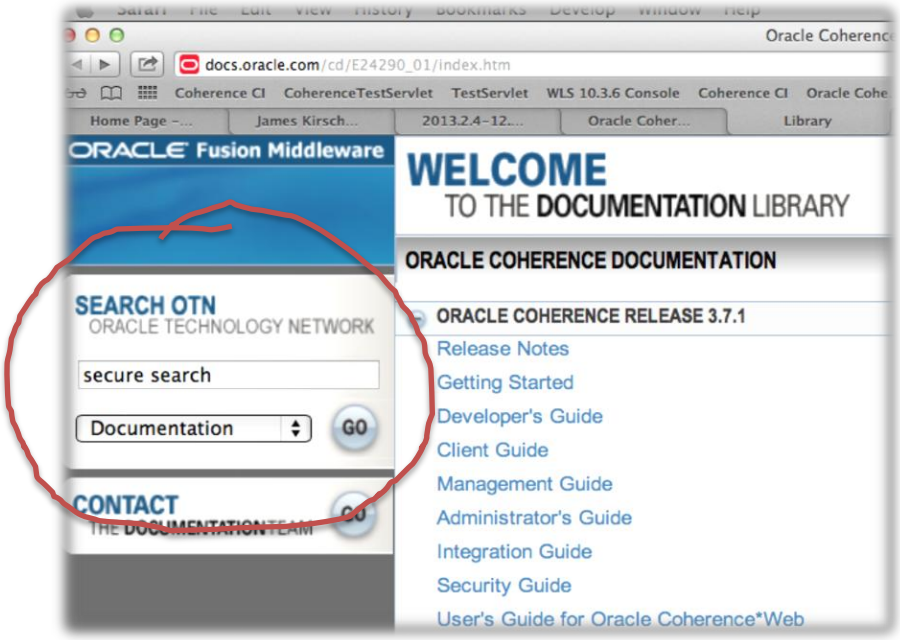

Figure 1. Oracle Documentation Search Screen

The Coherence Support Forum is also a great source of information and an easy way to reach out to experts in the Oracle Community – including the Coherence engineering team. The Oracle Technology Network (OTN) Coherence Home Page provides useful links to examples and online tutorials, and the Oracle A-Team (Architects Team) offer Coherence architectural guidance on their site. Virtual training in Coherence is available through the Oracle Learning Library and more formal in-class courses are provided by Oracle University.

For dedicated and on-site help, please contact Oracle Consulting or one of out Partners. Finally if you are already an Oracle Coherence customer and have a problem, raise a Support Request to get help or check the Oracle Support Knowledge Base for potential solutions.

The "Basics" - Design, Develop, Test (*repeat*)

Modeling Your Data

Just like a database, one of the first things to consider when planning to store your data in Coherence is what form it should take. Data modeling for Coherence is similar in many ways to modeling data for a database. When examining your entities (cache entries) and relationships between them you need to consider;

- How they are going to be accessed? For instance can an Order Item entity be part of an Order or will you need to access them separately?
- How they will be updated? For instance if Order Items are going to be stored in a separate cache from Order's, with the Order key embedded as an attribute, then moving an Order's Order Item's to another Order will require the Order key to be modified for each Order Item. The overhead and likelihood of such scenarios should be assessed when developing your data model to determine the consequences of different entity relationships.
- Will entities need to be modified atomically? If so then embedding Order Item's within an Order would make this easier.
- Will an Order and its Order Items need to be fetched together? If so then embedding the Order Item's within an Order will only require one network round trip to fetch all the entities.

Though planning your data model in Coherence is similar to relational data modeling, there remain some fundamental differences, the main one being that entities in Coherence are distributed. This difference means that adjustments need to be made to a relational model when moving it to a distributed environment. For instance, joins or transactions between entities can be expensive. Fortunately, data affinity can be used be used to address these limitations. Data affinity enables the co-location of related entities, for fast and atomic operations. But co-location can also cause un-balanced data distribution, leading to un-predictable response times. So care should be taken to ensure that the relationships selected (key associations) don't significantly distort the otherwise random distribution of entries.

Introducing design patterns, like the singleton pattern for managing data, should also be carefully considered, as they can significantly impact an applications performance and scalability. However,

strategies to mitigate contention, like issuing a range of sequence numbers or a portion of resources, can still make them relevant in a distributed environment.

In summary, consider your data model carefully, taking into account the points outlined above, and don't try to simply replicate a relational data model in Coherence. Further details on managing and developing a data model can be found in the Coherence Getting Started Guide.

Key vs. Filter Based Data Access

Once you have defined your data model, how do you access the entities/entries? It's tempting to just reach for Coherence filters to look up entries. However, to efficiently use filters, entries need to be indexed. If entries are not indexed queries can be several orders of magnitude slower. Furthermore, even if the attributes used by a filter are indexed, you should still check that the index is being used – and in the most efficient manner.

Key based access in contrast scales very well, uses the minimum amount of resources and provides predictable response times. This is because with key based access requests are sent only to the members that contain an entry or set of entries. With filter based access a request has to be sent to all members to determine which ones contain and entry or set of entries. So where possible you should try and use keys rather than a filter to access cache entries.

Where filters must be used then follow these steps to optimize their performance, minimize resources used and ensure your application scales.

- Use indexes wherever possible but remember that index structures take up memory too. Some simple tips are;
  - Monitor your query's through the JMX metrics that are made available in the *StorageManager* > <*Cache Service*> > <*Cache Name*> MBean, as shown below. The MaxQueryDescription and MaxQueryDurationMillis attributes give useful information about your worse query and how long it is taking to run. Note: that a query description is not captured unless it's duration exceeds the MaxQueryThresholdMillis.

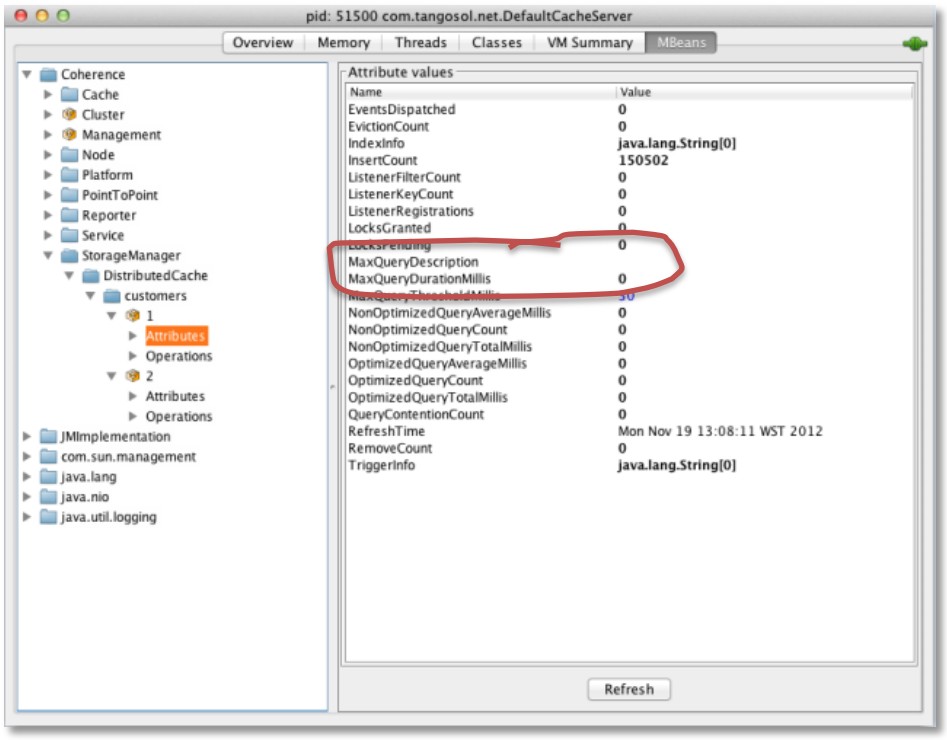

Figure 2. A cache's MaxQueryDurationMillis and MaxQueryDescription attributes can be found under its service in the *StorageManager* MBean

The difference in the metrics below illustrates just how much difference adding a couple of indexes can make, when using a filter. After adding the indexes the filter ran 20x faster.

Figure 3.The MaxQueryDurationMillis metric before the indexes have been added

Name	Value
EventsDispatched	0
EvictionCount	0
IndexInfo	java.lang.String[2]
InsertCount	150502
ListenerFilterCount	0
ListenerKeyCount	0
ListenerRegistrations	0
LocksGranted	0
LocksPending	0
MaxQueryDescription	Optimized query (78208 entries, 78208 matches for AlwaysFilter) - duration 51ms.
MaxQueryDurationMillis	51
MaxQueryThresholdMillis	30
NonOptimizedQueryAverageMillis	0
NonOptimizedQueryCount	0
NonOptimizedQueryTotalMillis	0
OptimizedQueryAverageMillis	43
OptimizedQueryCount	3
OptimizedQueryTotalMillis	131
QueryContentionCount	0
RefreshTime	Mon Nov 19 13:11:40 WST 2012
RemoveCount	0
TriggerInfo	java.lang.String[0]

Figure 4. The MaxQueryDurationMillis metric after the indexes have been added

– Be aware that the indexes for attributes with a high cardinality will typically consume more memory than those that have a low cardinality. The amount of memory consumed by and index can be measured by looking at the JMX Index metrics for a cache under the *StorageManager* MBean as shown below;

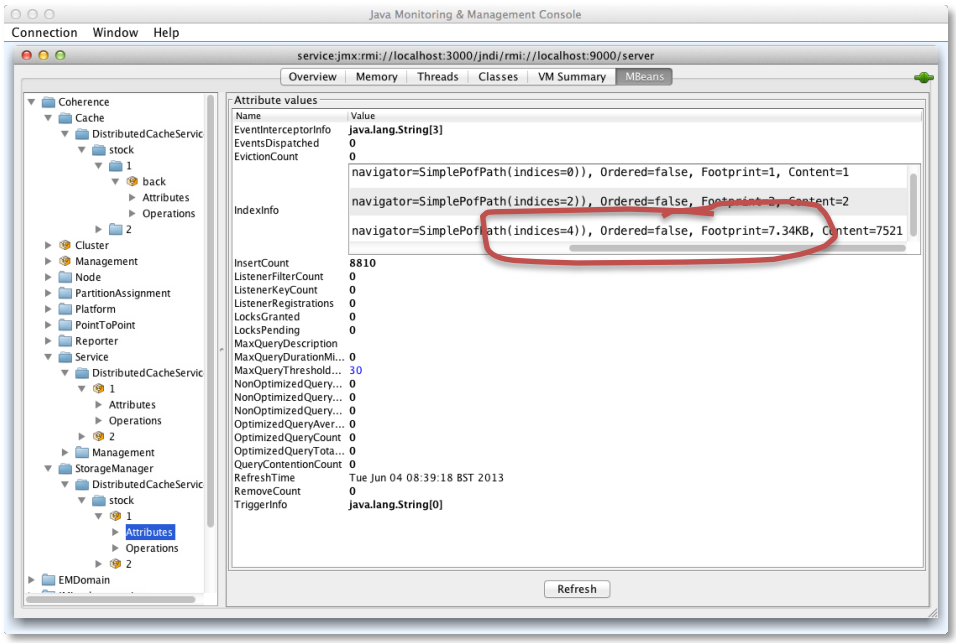

Figure 5. The IndexInfo attribute for a cache can be found under its service in the *StorageManager* MBean. It shows the memory consumed by the indexes

The IndexInfo attribute lists the memory footprint of an index along with its associated POF index. Here index 4 (the Id attribute) has a footprint of 34K. Although the footprint of this index is fairly small, compare it with the footprint of the other indexes for attributes it is high,

because its cardinality is much higher.

```
@Portable
public class Stock {
  public static final int PRICE = 0;
  @PortableProperty(PRICE)
  private double price;

  public static final int QUANTITY = 1;
  @PortableProperty(QUANTITY)
  private int quantity;

  public static final int SYMBOL = 2;
  @PortableProperty(SYMBOL)
  private String symbol;

  public static final int CREATED = 3;
  @PortableProperty(CREATED)
  private Date created;

  public static final int ID = 4;
  @PortableProperty(ID)
  private int id;

  public Stock() {
  }
```

Figure 6. The attribute that corresponds to POF index 4.

- If you are looking up a key from a value, a reverse index, then you can reduce space by specifying that a forward index (key to value) is not needed. Forward indexes are required for comparison filters, like the `GreaterFilter`, `ContainsFilter` etc. This can be done by setting the order flag parameter in the call to `addIndex()` to false, as shown below.

```
cache.addIndex(new PofExtractor(String.class, Stock.SYMBOL), false, null);
```

- Make sure your indexes are being used and used efficiently. For instance for the filter below;

```
AllFilter filter = new AllFilter(
    new Filter[] {
        new EqualsFilter(new PofExtractor(String.class, Stock.SYMBOL), "ORCL"),
        new EqualsFilter(new PofExtractor(Integer.class, Stock.ID), ENTRIES - 1),
        new EqualsFilter(new PofExtractor(Double.class, Stock.PRICE), 10.0)
        }
    );
```

Figure 7. A sample AllFilter

Its effectiveness and cost can be measured by running a query explain and trace plan. These provide detailed information about each step of a query. You can find out more about these features in the Coherence Developer's Guide, but an example is shown below;

```
Trace
Name                                    Index       Effectiveness          Duration
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 5012010(100%)        | 0
  EqualsFilter(PofExtractor(target= | 0          | 5012010(100%)        | 0

PartitionSet{128..256}

Trace
Name                                    Index       Effectiveness          Duration
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 4988011(99%)         | 70
  EqualsFilter(PofExtractor(target= | 0          | 4988025256(49%)      | 52
  EqualsFilter(PofExtractor(target= | 1          | 2525611(99%)         | 17
  EqualsFilter(PofExtractor(target= | 2          | 111(0%)              | 0

PartitionSet{0..127}

Index Lookups
Index  Description                             Extractor             Ordered
=============================================================================================
0      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
1      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
2      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false

Explain Plan
Name                                    Index       Cost
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 0
  EqualsFilter(PofExtractor(target= | 0          | 1
  EqualsFilter(PofExtractor(target= | 1          | 1
  EqualsFilter(PofExtractor(target= | 2          | 1

PartitionSet{128..256}

Explain Plan
Name                                    Index       Cost
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 0
  EqualsFilter(PofExtractor(target= | 0          | 1
  EqualsFilter(PofExtractor(target= | 3          | 1
  EqualsFilter(PofExtractor(target= | 2          | 1

PartitionSet{0..127}

Index Lookups
Index  Description                             Extractor             Ordered
=============================================================================================
0      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
1      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
2      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
3      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
```

Figure 8. Sample Trace and Explain Plan output for the above query. Note that there is one for each node – in this case 2

Here you can see that the first filter applied is not the most efficient as it only reduces the result set by 49%. In this case performance can be improved ten fold, simply by changing the order of the first and second filter to reduce the result set much faster, e.g. from 49% to 99%. This is illustrated below;

```
Trace
Name                                    Index       Effectiveness          Duration
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 50120|0(100%)        | 0
  EqualsFilter(PofExtractor(target= | 0          | 50120|0(100%)        | 0

PartitionSet{128..256}

Trace
Name                                    Index       Effectiveness          Duration
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 49880|1(99%)         | 6
  EqualsFilter(PofExtractor(target= | 1          | 49880|1(99%)         | 6
  EqualsFilter(PofExtractor(target= | 2          | 1|1(0%)              | 0
  EqualsFilter(PofExtractor(target= | 3          | 1|1(0%)              | 0

PartitionSet{0..127}

Index Lookups
Index  Description                              Extractor             Ordered
=============================================================================================
0      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
1      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
2      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
3      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false

Explain Plan
Name                                    Index       Cost
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 0
  EqualsFilter(PofExtractor(target= | 0          | 1
  EqualsFilter(PofExtractor(target= | 1          | 1
  EqualsFilter(PofExtractor(target= | 2          | 1

PartitionSet{128..256}

Explain Plan
Name                                    Index       Cost
=============================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 0
  EqualsFilter(PofExtractor(target= | 3          | 1
  EqualsFilter(PofExtractor(target= | 1          | 1
  EqualsFilter(PofExtractor(target= | 2          | 1

PartitionSet{0..127}

Index Lookups
Index  Description                              Extractor             Ordered
=============================================================================================
0      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
1      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
2      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
3      SimpleMapIndex: Extractor=PofExtractor(t  PofExtractor(target=  false
```

Figure 9. Sample Trace and Explain Plan output when the order of the first 2 filters is changed

Finally the trace and explain plans can also be generated without coding using the Coherence command line query tool.

- Apply the index before the entries are added to the cache. If the index is applied afterwards the operation can take a long time and even time-out if the entry set is very large.
- Consider using a `ConditionalExtractor` to create an index, for instance that excludes null values.
- For multi-value queries consider using a `MultiExtractor` for creating the index
- Consider a 2-phase query, fetching the keys first rather than the values and then iterating over the keys. This can reduce the memory footprint for a query on a client, as keys are generally smaller than values. It also allows a near-cache of values to be utilized.
- Where possible target filters by using a `KeyAssociatedFilter` to ensure a filter is only run on the node where the results set resides – if data affinity is being used.
- Consider batching results by fetching the results from the partitions of one node at a time using a `PartitionedFilter`. This can significantly reduce the memory required by the client and improve response times. Remember though that this must be the outer most filter, so that it is executed on the client.

Further details on using Filters can be found in the Coherence Developer's Guide.

Efficient and Scalable Transactions

Coherence provides a very fast and scalable mechanism for controlling concurrent access to one or more cache entries, and for performing atomic modifications. This is called "partition level transactions", and extends the functionality of an entry processor. An entry processor is similar to a database stored procedure, as it is executed where a cache entry or entries reside. They effectively queues concurrent requests to modify or access a cache entry. Partition Level Transactions extend them by enabling multiple cache entries in the same or other caches to be enrolled in an atomic operation. Any changes to multiple entries will be made in a "sandbox" and committed together at the end of the entry processor atomically – including the backup of any changes. To utilize them the following requirements should be met:

- Cache entries, which need to be accessed or modified together, need to be in caches managed by the same service and in the same partition, i.e. located in the same JVM. Therefore key association or data affinity must be implemented between the entries.
- Equal and opposite operations should not be performed simultaneously. For instance, modifying entry X in cache A and Y in cache B at the same time as modifying entry Y in cache B and entry X in cache A.

To gain exclusive access to another cache entry in an entry processor it needs to be accessed via the caches `BackingMapContext getBackingMapEntry()` method, as shown below;

```
/**
 * Add a new {@link Order} to a {@link Customer}. This method is called against the
 * {@link Order} object and the {@link Customer} object will be implicitly locked.
 *
 * @param entry The customer entry to run against
 *
 * @return null
 */
@Override
public Object process(Entry entry)
{
    // entry should not be present so lets set the value. E.g. create it
    entry.setValue(newOrder);

    BinaryEntry             orderEntry = (BinaryEntry) entry;
    BackingMapManagerContext ctx       = ((BinaryEntry) entry).getContext();

    // get the customer entry as a binary - this will be in the same partition
    // by using getBackingMapEntry you are adding the changes to the sandbox

    BinaryEntry customerEntry =
    (BinaryEntry) orderEntry.getContext().getBackingMapContext(Customer.CACHENAME)
    .getBackingMapEntry(ctx.getKeyToInternalConverter()
                        .convert(new Customer.Key(newOrder.getCustomerId())));

    // get normal Object Value
    Customer thisCustomer = ((Customer) customerEntry.getValue());

    // update the customer balance
    thisCustomer.setBalance(thisCustomer.getBalance() + newOrder.getOrderTotal());
    customerEntry.setValue(thisCustomer);

    return null;
}
```

Figure 10. An entry processor that locks and updates multiple entries as part of an atomic operation

Any changes to the customer object will be placed in a "sandbox" of changes to be applied atomically when the entry processor completes.

If exclusive access to cache entries is not required, because an entry is just being read, then access should be via the `BackingMapContext getBackingMap().get()` API, as shown below.

```
@Override
public Object process(Entry entry)
{
    ...

    BackingMapManagerContext ctx       = ((BinaryEntry) entry).getContext();

    // get the customer as a binary by using getBackingMap().get(...)
    // because we are not going to update it and so don't need to lock it

    Binary binCustomer =
    (BinaryEntry) orderEntry.getContext().getBackingMapContext(Customer.CACHENAME)
    .getBackingMap().get(y(ctx.getKeyToInternalConverter()
                        .convert(new Customer.Key(newOrder.getCustomerId())));


    // because we are getting a Binary instead of BinaryEntry, we need to
    // deserialize with the serlializer for the Service.
    Customer c = (Customer) ExternalizableHelper
    .fromBinary(binCustomer, ctx.getCacheService().getSerializer());

    ...
}
```

Figure 11. An entry processor that locks the entry it was targeted at and then accesses an entry in another cache – without locking it

Where transactional logic needs to be executed, "partition level transactions" should be explored first and a data model adjusted if necessary, rather than using the `ConcurrentMap` API (using `lock()` and `unlock()`) or transactional cache schemes. Some of the reasons for this are that client based locks can limit scalability, because of lock contention, and transactional schemes require additional logic to handle optimistic locking exceptions. It should also be noted that the different mechanisms should not be mixed, so select the one which best fits, your requirements.

Finally, with release 12c a "mirror" partition assignment strategy can also be used co-locate a service's partitions with those of another service. This strategy is used to increase the likelihood that key-associated; cross-service cache access remains local to a member – though this is not guaranteed. Using this strategy, in conjunction with an invocation service to perform cross-cache operations, may localize locks with cache entries to provide a scalable solution.

Ensuring Data Consistency Using the Golden Gate "HotCache"

In many use cases the data held in a cache is a copy of that in a database. Furthermore, the data in the database is often the master copy and updated outside of Coherence by third party applications. In the past to ensure that cache data is not stale a number of different strategies have been used;

- Setting a Time-To-Live (TTL) on cache entries and using a read-through cache store, so that they regularly expire from a cache and are reloaded from the database. This technique can also be used with the refresh-ahead feature, so that entries are reloaded asynchronously before the expiry time is reached. However, some cache data may still be stale and other entries that have not changed needlessly reloaded.
- Database table triggers can also be used to push changes to the database to Coherence, using messaging functionality like Oracle AQ. However, doing so requires changes to the database schema, additional development and may not provide the throughput needed. It can also add undesirable additional load to the database and the approach will differ depending on the underlying database.
- The Database Change Notification (DCN) feature of the Oracle JDBC driver can also be used to monitor an Oracle database for changes and update Coherence when they occur. Using the JDBC driver its possible to register SQL queries with the database and receive notifications in response to DML and DDL changes, without any need to modify the database schema. `DatabaseChangeEvents` containing values like the table changed, the operation performed and row ID are then returned when changes are committed as part of a transaction. But importantly not the old and new value of the row that has just been changed.

  Therefore, this mechanism is only really efficient when the data held in the database rarely changes because the database will need to be re-queried to find out the values of the row that has. Deleting an entry and using read-through also won't work unless the key is the row ID. This also raises the interesting questions of how you re-query the database for a deleted row?. Furthermore, certain database operations can even change a row ID, like the import and export operations.

A DCN solution would also have to be integrated with Coherence and implementing High Availability can be problematic. If a DCN listener fails and is re-started then any events that occurred when it was "offline" would be lost, and multiple DCN listeners run in parallel, to guard against this possibility, would need to be coordinated to prevent duplicate events being generated.

Release 12c of Coherence introduces the "HotCache" feature, which uses Toplink and Golden Gate to solve the problems raised by the above strategies. Golden Gate is a non-invasive technology for performing real-time Change Data Capture (CDC), and in conjunction with Toplink, can propagate database changes to Coherence through "HotCache". Applications already using the standard Java Persistence API (JPA) for Object Relational Mapping (ORM) in their cache store can simply configure their application to use it, without modifying the database or writing any additional code[1]. Golden Gate also writes a copy of database changes to an intermediate trail file and keeps track of changes applied using checkpoints, so any failure in the change capture process can be recovered from simply by restarting the capture or Java Client processes, without loosing any intermediate events. It can also capture and apply a high rate of database changes without significantly impacting the database.

Security

Coherence provides a range of security options to secure sensitive information. Below is a list of options available and considerations when using them.

COHERENCE SECURITY

SECURITY OPTION	SUPPORTED	NOTES
Data Encryption	Yes	Clients can use standard encryption libraries to encrypt keys and values or just part of them. These could further be incorporated as part of a custom serializer. However, inside a cluster these will then be opaque and so could not be used in a filter etc.
Secure Communications		
• Inside a cluster	Yes	TCMP over TCP or TCP Message Bus communications within a cluster can be secured using SSL.
• With extend clients	Yes	Communications between extend clients and proxy services can be secured using SSL. Furthermore, integration between Coherence and a hardware load-balancer, like F5, allows SSL termination processing to be offloaded.
Restricting Cluster Membership and Operations	Yes	Cluster access can be limited to new members using the "authorized host" mechanisms that require all the cluster members

[1] The initial release of the Golden Gate "HotCache" feature will only be supported with the Oracle Database.

		to be listed in a predefined pool. However, the list of members can be dynamically constructed if required using a custom hosts filter. An Access Controller can also be used to restrict the operations that cluster members can perform (e.g. create, join. destroy, all and none), the caches they can control and the services they can use.
Restricting Management Access	Yes	Connections to the JMX Management Node in a cluster can be secured using SSL and password based authentication. JMX access can also be configured to be read-only if required.
Restricting Client Access	Yes	Extend client access can also be limited through a range of host names/IP address or a custom hosts filter.
Rouge Clients	Yes	Extend clients that do not process their response messages fast enough can be disconnected to prevent a proxy service running out of memory.
Authentication	Yes	Extend clients in all technologies can pass security tokens to Coherence proxy services for validation. Authentication is performed using a standard JAAS login module or in Coherence
Authorization	Yes	Extend clients can use an interceptor class to wrap cache and invocation services that can then permit or disallow operations based upon the identity and permissions of a user.

Securing cluster or client operations inevitably introduces a processing overhead and it is recommended to add any necessary security measures at the beginning of any test cycles to incorporate this in any measurements. Also bear in mind that authentication only imposes an overhead at connection time where as authorization does so for each request.

Development Tooling

The free Oracle Enterprise Pack for Eclipse (OEPE) provides a set of project facets, configuration wizards and schema aware editors, to accelerate development and minimize configuration errors. If your development team use Eclipse its well worth investigating.

With release 12c of Coherence, OEPE has been updated to enable Coherence applications to be deployed as a GAR and the WLST editor and Weblogic Server runtime integration will help those new to this powerful scripting language to try it out in the same development environment.

Coherence 12c also introduces a Maven plug-in that synchronizes an Oracle home directory with a Maven repository and standardizes Maven usage and naming conventions. The Maven integration also

includes an archetype and packaging plug-in for a Coherence Grid Archive (GAR). A Coherence GAR is a module type that packages all the artifacts required to execute a Coherence Application; this includes any class dependencies and XML configuration files (though not an override file).

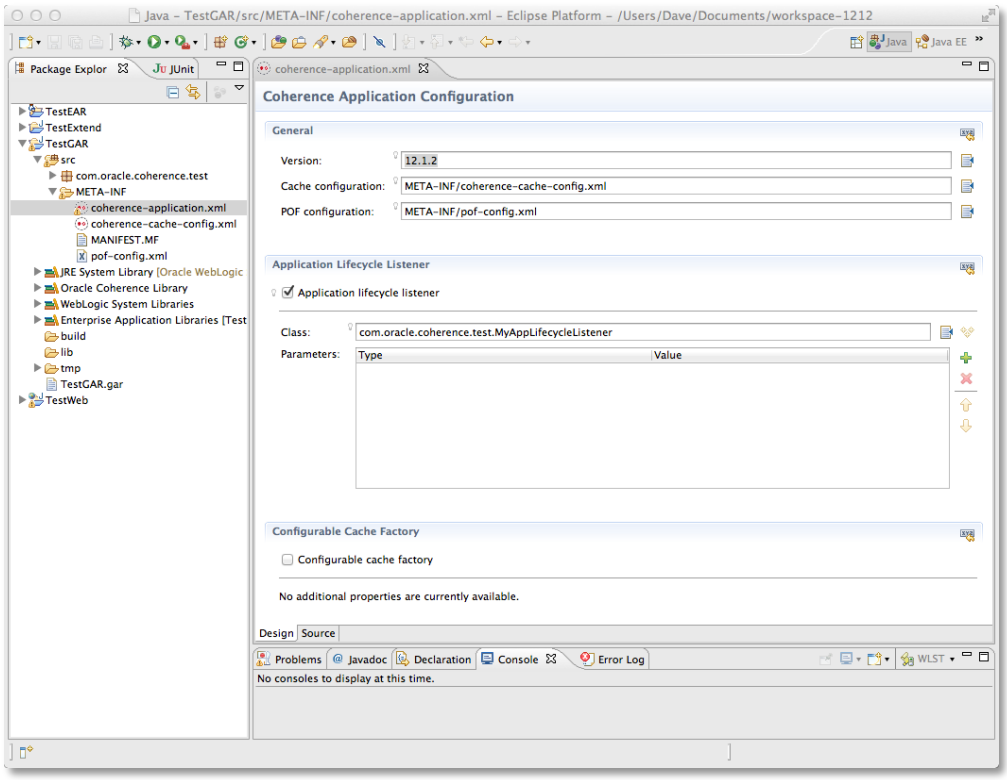

Figure 12. OEPE provides wizards for common development activities, like configuring and creating a Coherence GAR

In other development environments, like JDeveloper and Visual Studio, XSD validation of the Coherence configuration files can be used for auto-prompting and to ensure your application "fails-fast" if it's incorrect. For this last reason alone, its highly recommended schema validation is used.

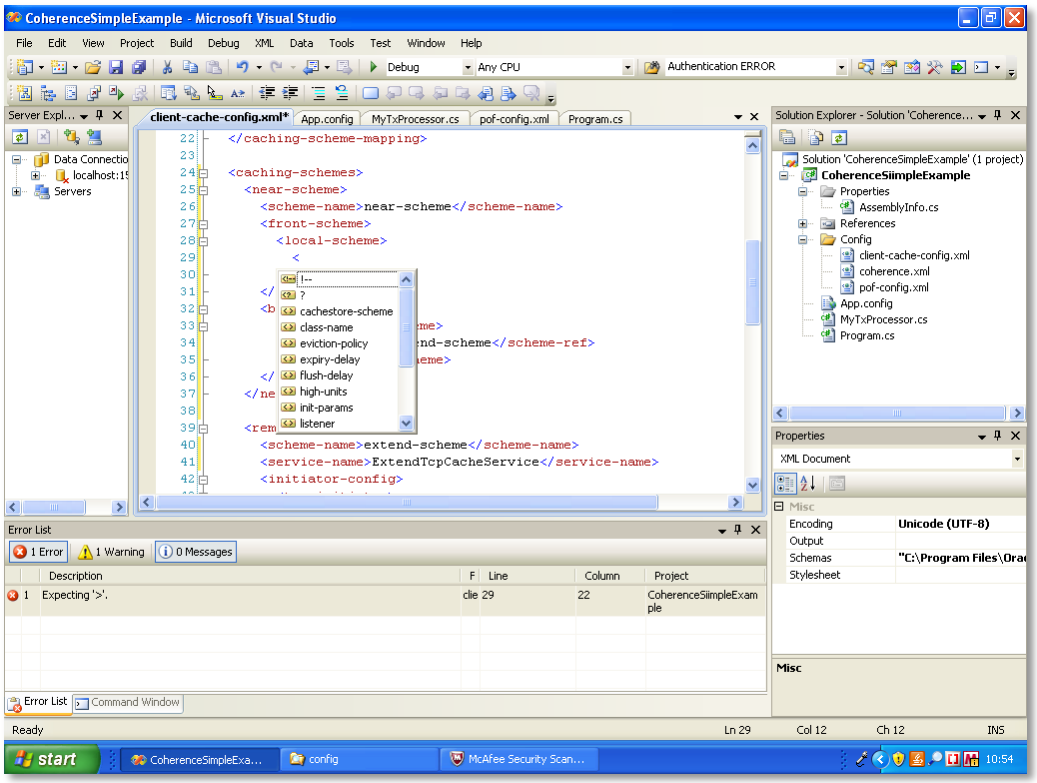

Figure 13. Schema validation can be used to prevent errors and make configuration easier.

Testing

A popular approach for developing Coherence applications (and others) is through Test Driven Development (TDD). As a result a number of tools and frameworks have emerged to help developers perform continual unit testing throughout their development process. They can also help simulate complex "edge case" scenarios that involve multiple components, by running a whole cluster in a single JVM. Some examples are the littlegrid framework and Oracle Tools.

Plan for Change and the "Worse Case Scenario's"

Preventing Failures, Data Loss and Data Corruption

Coherence already has many features to minimize the chance of data loss, due to a software or hardware failure. For instance multiple copies of a cache entry can be kept and with 12c you have as many copies as your like.

Figure 14. Guarantee data availability during multiple simultaneous machine loss

By default all updates are synchronous, so on receipt of a response Coherence guarantees that the change has been replicated to all the copies. Coherence also ensures that the primary copy of an entry and the backup(s) are on different physical machines - if possible.

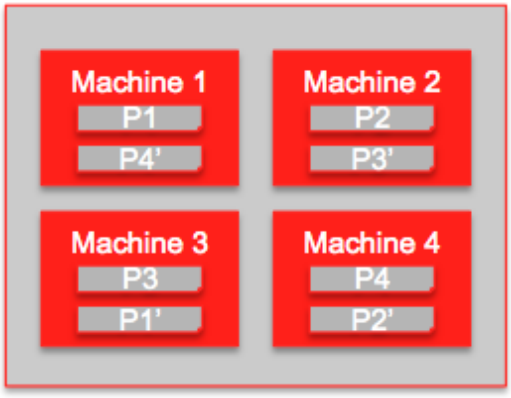

Figure 15. By default, Backup Partitions Always on Separate Machines

However, it's also possible to configure rack and site safety, as well as machine safety. These additional options inform Coherence that a primary and backup copy of an entry should be on different racks or even a different site.

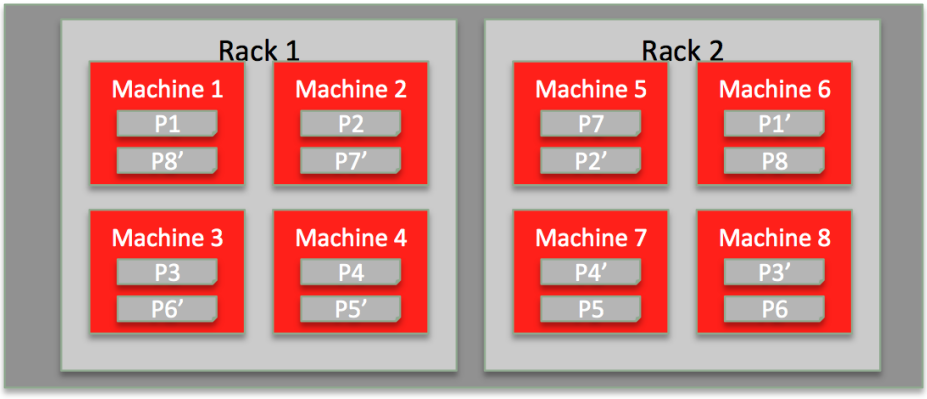

Figure 16. Configured with Rack Safety. If a Rack is lost the cluster falls back to Machine Safety.

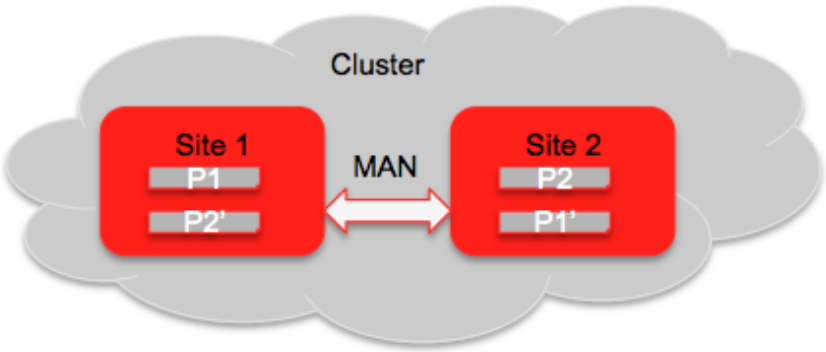

Figure 17. Configured with Site Safety. If a site is lost the cluster falls back to Rack Safety.

When considering a site-safe configuration there are a number of implications. For instance, 50% of read and all write operations will traverse the inter-site link (and 50% of all write operations will do so twice). While some customers have successfully used this architecture, the synchronous nature of communications will affect performance, scalability, and stability.

Imagine a 20 node cluster on a single LAN switch with 1GBE links, the cluster could serve up around 20Gbs. Now imagine those same 20 machines split evenly over two sites separated by a fast reliable 1GBE WAN link. Sounds pretty good for a WAN, but what is the cluster throughput? Well since at least half of all requests will traverse the WAN and that shared 1GBE link, the effective cluster bandwidth is gated by that link, and the cluster bandwidth is going to be closer to 2Gbs rather then 20Gbs. This will restrict the scalability, and thus performance of the cluster. The same argument applies to the request latency as well. As for stability you are obviously more likely to loose connectivity and thus face a "split brain" scenario, requiring preventative action (see below for further details). Even if you have better link than this it still isn't scalable, i.e. adding more nodes on either end will not (cannot) improve performance. This is the reality of synchronous replication across a WAN link.

As a result, this configuration is very sensitive to the throughput and latency of the inter-site link and should only be considered if it provides reasonable throughput (e.g. 1GBE) and relatively low latency (for instance <10ms). Use cases where this may be suitable are:

- Caches have a lot of reads, but few updates
- Caches where throughput is relatively low

There are some options for improving the performance of a site-safe configuration, like using near caching - to reduce network operations – and even asynchronous backups, a new feature in release 12c. The Coherence quorum feature, introduced in release 3.7, can also be used to prevent data corruption and help Coherence overcome environmental issues. For instance, if the nodes in a cluster cannot communicate with each other, perhaps because of intermittent network issues, the cluster quorum policy can be used to maintain a minimum number of members and prevent a cluster breaking apart.

The partitioned quorum policy can also be used to prevent recovery and rebalancing if a temporary cluster split occurs. Updates to cluster islands can also be prevented to ensure that data doesn't become corrupted due to a "split brain" scenario[2].

Coherence 12c also introduces a new partitioned quorum policy option, to manage failover access. This moderates client request load during a failover event, to give cache servers more time to recover and rebalance partition backups. It can be particularly useful where a heavy load of high-latency requests may prevent, or significantly delay, cache servers from successfully acquiring exclusive access to partitions that need to be transferred or backed up.

```
<!-- Distributed caching scheme. -->
<distributed-scheme>
    <scheme-name>example-distributed</scheme-name>
    <service-name>DistributedCacheService</service-name>

    <backing-map-scheme>
        <local-scheme>
            <unit-calculator>BINARY</unit-calculator>
        </local-scheme>
    </backing-map-scheme>

    <partitioned-quorum-policy-scheme>
        <class-name>com.tangosol.net.partition.FailoverAccessPolicy</class-name>
        <init-params>
            <!--
                Specifies the delay before the policy should start holding requests
                (after becoming endangered).
            -->
            <init-param>
                <param-name>cThresholdMillis</param-name>
                <param-value>7000</param-value>
            </init-param>
            <!--
                Specifies the delay before the policy makes a maximum effort to hold requests
                (after becoming endangered).
            -->
            <init-param>
                <param-name>cLimitMillis</param-name>
                <param-value>30000</param-value>
            </init-param>
            <!--
                Specifies the maximum amount of time to hold a request.
            -->
            <init-param>
                <param-name>cMaxDelayMillis</param-name>
                <param-value>2000</param-value>
            </init-param>
        </init-params>
    </partitioned-quorum-policy-scheme>

    <autostart>true</autostart>
</distributed-scheme>
```

Figure 18. A partition quorum policy to manage failover access to partitions and improve recovery performance

[2] A "split brain" scenario arises when a Coherence cluster splits into separate clusters, perhaps because of communicate issues, and external clients then separately update identical entries in each clusters, causing the data to diverge.

Persistence Options

Coherence supports both synchronous (write-through) and asynchronous (write-behind) persistence. Which provides the required level of performance scalability and recovery guarantees, involves weighing up the benefits and drawbacks of each strategy.

SYNCHEOUNOUS VS. ASYNCHRONOUS PERSISTENCE

FEATURE	SYNCHROUNOUS	ASYNCHROUNOUS
High Availability		
• **Recovery**. Guaranteed recovery from external data store.	Yes	No
• **Data Store Availability Errors**. Ability to handle availability errors and re-try later. Note, additional memory should be allocated to hold queued updates that need to be re-tried.	No	Yes
• **Integrity Constraint Errors**. The ability to handle integrity constraint errors.	Yes	No
Performance		
• Write Performance.	No	Yes
• Read Performance.	Yes	Yes
Scalability	Yes[3]	Yes

When using either approach for persisting cache data to a database, a JDBC connection pool should be used and an appropriate timeout set for database operations that is less than the cache store timeout which in turn should be less than the Service Guardian timeout. If using and Oracle database then the JDBC Statement class has a `setQueryTimeout()` method that can used to set an overall timeout on the execution of the statement[4]. With JPA 2 this can also be set using the `javax.persistence.query.timeout` property in the `persistence.xml` file. This will

[3] A synchronous cache store cannot batch writes like an asynchronous cache store. However, a feature called operation bundling can provide pseudo write batching. It works by capturing the separate write operations across the worker threads of a service during a pre-defined time interval, grouping them into a batch. For use cases where there are frequent write operations it can improve scalability, but since write operations will be paused during the pre-set time window to create the batch, performance can be impacted. Another side effect is that any persistence exceptions will affect all the writes in a batch.

[4] For further details see the Oracle Support Note 1531408.1. However, this option is not available on Windows.

prevent cache store threads being terminated and left in an inconsistent state because the Service Guardian had not received a heartbeat during a specified interval.

There are also a number of limitations with each persistence approach. These are listed below;

Asynchronous Persistence

- Operations need to be idempotent, i.e. produce the same result if repeated, as they may be re-run if a node failure occurs.
- Operations should not fail because of errors like a referential integrity violation, as it's not possible to communicate these types of errors problems back to the client.
- If a re-try queue (`<write-requeue-threshold>`) is used care should be taken to size it large enough to handle any failed persistence entries – for instance during a period when a database goes offline – but not so high that a node may run out of memory.
- Remove operations are always synchronous

Synchronous Persistence

- Ensure that a sufficient number of service threads are allocated, as one will be used for the duration of not only the cache put but also the database operation when using write-through. Check the *Service* MBean's ThreadIdleCount JMX metric to monitor thread utilization.
- Ensure the a read-write-backing-map-scheme has the `<rollback-cachestore-failures>` element set to true, to pass back exceptions to the client

Detecting and Reacting to Failures

Coherence has a number of failure detection mechanisms. The Service Guardian detects thread deadlock. Guarded Threads, like the worker threads of a service, issue a regular heartbeat to the Service Guardian indicating they are still active. If the Service Guardian doesn't receive a heartbeat for a set period of time it will issue a soft timeout (causing a thread dump) followed by a hard timeout (resulting in a thread shutdown), if it still gets no response[5]. The failure of a cluster node can be detected in a number of ways, including the closing of a TCP socket connection that forms a ring around the cluster. It will usually only take a few milliseconds to detect this and for the recovery process to begin. Machine failure is detected using an IP Monitor daemon. Periodically the machine senior IpMonitor daemon (i.e. only one per machine) will pick the address of a cluster member and determine if the machine it is running on is alive. This is done via the `java.net.InetAddress.isReachable` call with a default timeout setting of 5s. If the machine cannot be contacted after 3 attempts the machine will be

[5] The problem can also just be logged if a thread re-start is not desired

deemed "not reachable" and its members removed from the cluster, prompting the recovery process to begin. Reducing the number or intervals of heartbeats is possible, but may result in false positives.

The implementation of "isReachable" is platform specific. As of Java SE 7 (and older), it will attempt to either send ICMP requests or (if ICMP requests are not allowed by the operating system) attempt to connect to TCP port 7 on the remote host. Port 7 is the default port for the Echo Protocol. This service is disabled by default on most operating systems, but the connection exception that results from a machine that rejects this connection is used to determine that the machine is running.

If a firewall that prevents ICMP packets and/or connections to port 7 is in place, this may prevent the formation of a cluster since Coherence cannot verify the reachability of the machine. Therefore if a firewall is required, it is recommended that port 7 be opened to allow IpMonitor to function.

When a node or machine failure occurs and the recovery and re-distribution process is initiated Coherence will throttle the movement of partitions at a rate determined by the transfer-threshold (which by default is 512KB p/s per service per node, i.e. if a single partition exceeds this size only 1 will be transferred by this service each second). This is to balance rebalancing with normal operations. For a 1GBE network this should be adequate, but on faster networks it can be increased.

Now lets calculate approximate Mean Time to Rebalancing (MTR), after what is sometimes called a "cluster shock", can be calculated. First lets look at what is possible assuming there is no throttling.

Assumptions:

- Each machine is on a fully switched 1 GBE network and can transmit and receive ~100 MB of data p/s. With on going processing less than this will inevitably be available.
- During recovery the re-creation of backups is almost instantaneous (lost primaries will be created from promoted backups) and for the most part it will be backups that will be moved during rebalancing.
- The rebalancing process is network bound and re-creating indexes and firing Backing Map Listeners (triggers when entries are created) usually happen very quickly.

Calculation:

- Let M be the number of machines in a cluster
- Let D be the size of the serialize backup data
- When a machine fails the fraction of data that needs to be rebalanced is D / M
- This needs to be shared amongst M - 1 machines
- As rebalancing can be done in parallel – every machine can transmit backup entries - each machine can transmit 1 / (M - 1) of the data at the same time
- Therefore, the MTR = ((D / M) * (1 / (M – 1)) * 100 MB p/s or

*MTR = (D / (M (M – 1) * 100)) seconds*

Worked Example:

- 24 GB of primary data (with 1 backup) or 48 GB in total
- 3 machines, each with 64 GB memory and 2 x 4 cores
- 1 GBE network
- 1 machine fails

MTR = 24,000 MB / (3 * 2 * 100 MB) = 40 seconds

This accuracy of this formula will be affected processing activity during recovery, the number of partitions etc., so any estimate should be validated through testing. But, the MTR will go up linearly as more data is added and down linearly as network throughput is increased, for instance in the above example the MTR would be ~4s if a 10 GBE network were used.

Now compare this with the MTR when allowances are made for throttling during rebalancing.

Assumptions;

- C is the number of cache servers per machine, 8 in this case
- N is the number of partitions, 2039 in this case
- Here we assume there is only 1 service
- The average partition size is 24,000 MB / 2039 =11.77 MB
- Default transfer threshold = 0.5 MB

Calculation;

- As the partition size is > transfer threshold then the rebalancing time, adjusted for throttling, will be;

N / (M * C * (M – 1)) = 2039 / (3 * 8 * 2) = ~42 seconds

During recovery Coherence will throttle partition transfers so that each node can only send 1 partition p/s – as the partition size > transfer threshold. Therefore, each machine is limited to sending 8 * 11.77 MB = ~94.6 MB of data p/s. Here throttling will not limit rebalancing as the threshold is near the limit of the network capacity anyway, but it can be used to limit the network bandwidth utilized for rebalancing so that other operations can continue.

To summarize, the complete recovery time involves detecting node or machine failure, regenerating lost copies of data and finally re-distributing/re-balancing these new copies. The first phase can take anything up to 15s (to detect a machine failure), the second should be almost instantaneous and the

third, the MTR, will depend on the network bandwidth, cluster load and parameters effecting throttling.

Managing Planned Changes

Inevitably at some point your production environment will need to be changed. For instance Coherence itself will need to be upgraded or patched, or an enhancement made to your application. These are changes you can plan for and part of the natural evolution and optimization of your application and environment. The rest of this section focuses on online changes, as offline changes tend to be much easier. An exception is data loading after a planned outage, for instance after a complete cluster shutdown. Strategies for loading data into caches from a database and other sources are covered in the Coherence Developer's Guide, so will not be discussed here. Cluster persistence is planned for a future release, allowing an image of the cluster to be persisted and re-loaded from disk.

Rolling re-starts

This is the process of serially re-starting each node in a Coherence cluster. It allows a complete cluster re-start, perhaps to introduce some code changes, without processing or availability being interrupted – though it will typically have some impact on performance and throughput. The pseudo code for this operation is as follows;

```
Upgrade database schema
Update Coherence nodes CLASSPATH

For each server
  For each node except management node
    For each distributed cache service
      While JMX value of StatusHA not MACHINE_SAFE
        Pause for 5s
      End while
    End For
    // Node loop
    Stop / Start node
  End For
End for

Restart management node
Introduce new version of client
```

Figure 19. Pseudo code for the rolling re-start logic

Note: it is a prerequisite for a rolling re-start that there is sufficient capacity on N-1 nodes in the cluster to store all the cache data (where N is the number of cluster nodes – or Managed Coherence Containers). Remote JMX management should also be enabled on all the cluster nodes.

Usually this has been done using a custom script but Coherence 12c can now automate this process through a bundled Weblogic Server Scripting Template (WLST), `rolling_restart.py`.

Extend Client Compatibility

Forward compatibility is maintained from client to server, i.e. from extend client to the cluster. For example a 3.7.1 client can connect to a 12c proxy. However a 12c client cannot connect to a 3.7.1 proxy.

The Coherence binaries/JAR files

Minor upgrades to the Coherence binaries can usually be done online in a rolling fashion, for instance an upgrade from release 3.7.1.7 to 3.7.1.8 (*Major.Major.Pack:Patch*). However, the release notes should be checked first, as very occasionally this is not possible. With major and pack release upgrades a complete cluster shutdown will be required. This is because a major or pack release will likely not be compatible on the internal protocol boundary.

To perform a major upgrade to the Coherence binaries, e.g. from 3.7.1 to 12c, a parallel and identical cluster must be available. Usually this will be a Disaster Recovery (DR) site or another cluster setup in an active-active configuration using the Push Replication pattern, so that the data in each cluster is synchronized.

Configuration Changes

Coherence cache and cluster configuration changes can be made either in the XML configuration files or through JMX. JMX enables a number of changes to be made at runtime, such as changing the logging level, the high-units of a cache etc. However, these changes are made at a node or JVM level and not persisted. Therefore, changes made via JMX will also need to be made to the start-up configuration files to preserve them between node re-starts. With Managed Coherence Containers WLST can be used to perform online cluster wide cache (and cluster configuration changes). Also by applying the changes to an external cache configuration file (referenced in a GAR file through a JNDI name) these changes can be re-applied after an application re-starts.

JMX cluster and cache configuration parameters that are read-only[6] will need to be modified through a rolling restart of relevant Coherence cluster nodes.

Code Changes

Changes to custom classes deployed to Coherence, such and entry processers, classes used to represent keys or values, custom cache stores, eviction policies, event interceptors etc., can be modified online with a rolling restart of a cluster. The steps for initiating such code changes are as follows;

[6] For a full list of Coherence parameters exposed through JMX, including which are read-only, please see the Coherence Management Guide

Prerequisites

- Classes that need to be sent over the network or called remotely, like keys, values, custom entry processors, filters, aggregators or invocable agents, must implement the Java `Serializable` or Portable Object Format (POF) interface
- If multiple client versions need to be supported then value classes should also support the Evolvable [7] interface and any changes to them must be "additive", that is they must add attributes not remove or change existing attributes.
- Changes to custom entry processors, event interceptors, custom filters etc., should be backwardly compatible, i.e. they should take account of Evolvable objects
- Any database changes must be made first, i.e. if a database cache store is used, and these changes must be "additive" too
- Finally the rolling restart prerequisites should also be met

Steps

1. Apply any database changes
2. Deploy your new application code changes to your cache servers or Managed Coherence Containers and modify the POF configuration file to add any new POF types
3. Perform a rolling restart of your cluster. With Managed Coherence Containers the bundled WLST script can be used to perform step 2 and 3.
4. Apply the code changes to any Extend clients that will use the new functionality

Setting up the Production Environment

Capacity Planning

Capacity planning involves not only estimating and validating the capacity of a cluster but also ensuring that if limits are set they are not exceeded. Estimates should also include spare capacity to accommodate failures, peaks in demand and growth. But before performing any calculations it's worth answering some simple questions to save both time and resources.

[7] Please see the Coherence documentation for a full description of how the `Evolvable` interface enables Coherence to support multiple versions of a cache data and cache clients. Note the Evolvable objects also need to be POF objects

- Does the whole data set need to be held in memory?

  If the answer is no, then just caching the frequently used data will save memory and make capacity planning easier. A cache store to load data from a database when there is a cache miss, could load entries on-demand. To size limit the cache a high-units threshold can be configured, along with an eviction policy to determine which entries are evicted when it's reached. However, remember that when configuring the high-units for a cache it is per cache per node, not for the whole cache across the cluster.

  Finally when only the frequently accessed data is cached it will not be possible to query the entries. This is because when a query is performed missing data will not be loaded on demand[8].

- Is a backup copy of every cache entry required?

  When using the cache aside pattern the answer is no. But even when cache data will be changed so that a backup is required, it is usually only needed until the change is persisted. For instance when using write-behind, the backup can be removed after the change has been persisted[9]. This will reduce the memory requirements for a cache but again assumes access is only key based, so that any missing entries will be re-loaded.

- Can POF serialization be used?

  The Coherence Portable Object Format (POF) is a very compact and highly optimized object serialization format, which can provide up to 5x greater compression than the standard Java serialization format – though this will depend on the structure of an object graph. So using POF serialization should yield a much higher density of cache data.

[8] This is also true if read-through to a cache store is configured.

[9] This can be configured by setting the `<backup-count-after-writebehind>` element to 0.

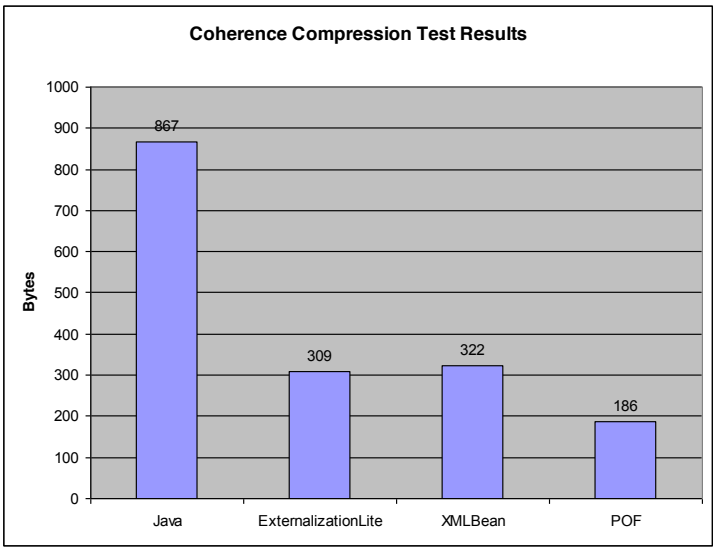

Figure 20. 5x better compression of a sample object graph using POF

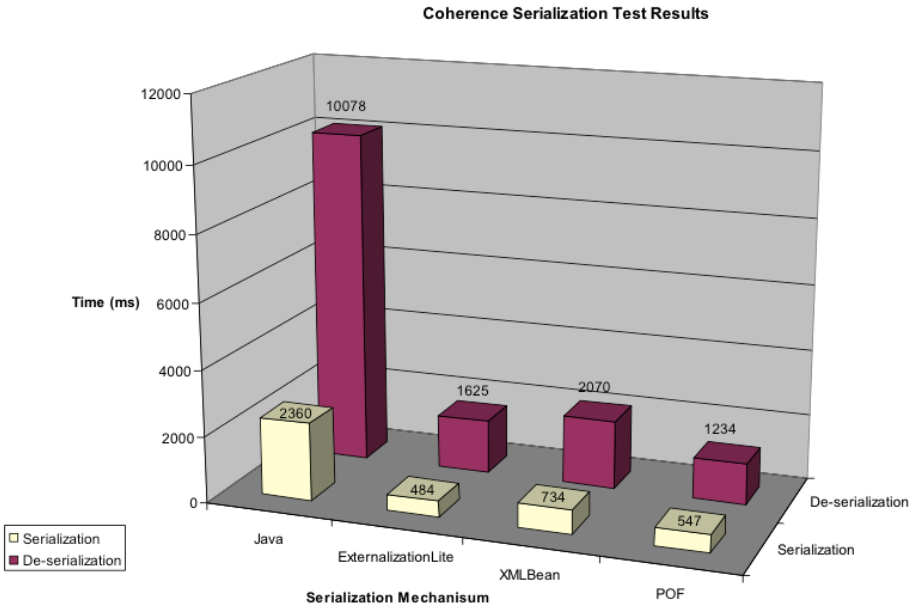

Figure 21. 10x faster de-serialization of a sample object graph using POF

- Can fast disk storage, like an SSD or even a local disk, be used?

  If fast storage is available then the Elastic Data feature can increase storage capacity. A good starting point for exploring the benefits of Elastic Data is to consider using if for initially for storing backups.

- Can the Concurrent Mark Sweep (CMS) Garbage Collector *NewRatio* be increased?

  The *NewRatio* is the ratio of the young generation to the old. The default setting for the *NewRatio* is 8 if the JVM is running in server mode[10], i.e. Old:Young is 8:1. This can sometimes be too low if an application, like Coherence, retains a large number of relatively long-lived objects, especially if a large heap is specified. Testing and monitoring GC pauses will guide you to the optimum setting, but applications that perform a lot of write operations or in-grid processing will likely need a larger young generation than those that have a relatively static data set. On an x86 platform

- Can the G1 Garbage Collector (GC) help?

  G1 will typically enable greater heap utilization without the significant drop in performance exhibited by the Concurrent Mark Sweep (CMS) GC. If you are considering very large heaps, for instance 32 GB, then the G1 may provide better performance and require less tuning than the CMS GC.

Estimating capacity requirements is not an exact science. Assumptions can be made and calculations performed, but these should be seen as a starting point for testing. Therefore, the following should simply be seen as a guide;

- How much memory does my cache use?

  The easiest way to answer this question is to perform some simple tests. Store some representative cache entries, measure their size and extrapolate the results to estimate the size of your planned cache population. JConsole can measure the heap of a cache server JVM, before and after the entries have been loaded and any indexes created, but remember to run multiple cache servers so that backups are created. The Coherence JMX metrics on the *Cache* MBean can also be used, as shown below, by configuring a binary `unit-calculator` and the `unit-factor` in the cache configuration file.

[10] This can be because the *–server* mode is specified or because the JVM defaults to running in this mode.

```
<caching-schemes>
  <distributed-scheme>
    <scheme-name>DistributedTestCacheScheme</scheme-name>
    <service-name>DistributedTestCacheService</service-name>
    <backup-count>1</backup-count>
    <backing-map-scheme>
      <local-scheme>
      <unit-calculator>BINARY</unit-calculator>
      <unit-factor>1048576</unit-factor>
      </local-scheme>
    </backing-map-scheme>
    <autostart>true</autostart>
  </distributed-scheme>
</caching-schemes>
```

Figure 22. Configuring a BINARY unit-calculator and unit-factor

The total size of cache data held in memory (excluding indexes) can be calculated as follows;

*Total memory for cache data = number of storage nodes * units * unit-factor * (backup count + 1)*

Since data distribution should be balanced, a sample storage node can be selected to determine average `units` and `units-factor`

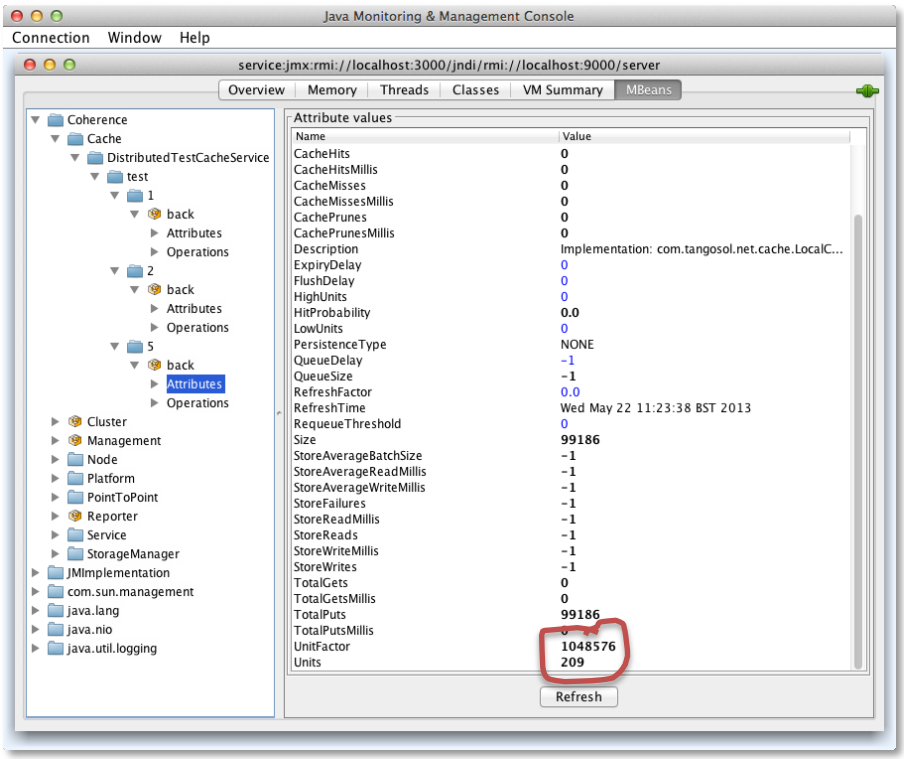

Figure 23. The Units and UnitFactor attributes on the *Cache* MBean show the size of a cache contents on a node.

This calculated figure include keys but not indexes, and will usually be slightly smaller than any heap based calculation, which will also include an overhead for cache and cache entry data structures.

If a replicated or near cache is being used then a different approach is required. Because these entries[11] are stored in a deserialized format its not possible to calculate their binary size using JMX metrics. Instead a tool like JConsole is required to measure the memory overhead of a sample data set from which the target cache populations can be extrapolated.

- How many JVM's will I need to store my cache data?

When using the CMS GC you should generally use no more than 2/3 of the JVM's heap, otherwise the JVM may be susceptible to longer GC pauses. This includes primary data, backup data, indexes, and application data. Lets take an example to illustrate how to calculate the number of cache servers required to store some cache data.

Assumptions:

- The total size of cache data, including backups and indexes, is 48 GB
- The size of application data is negligible, i.e. its stored in Coherence caches
- Each cache server has a maximum heap size of 6 GB
- Only 2/3 of heap will be available for cache data, or 4 GB
- The CMS GC being used.

Total number of cache servers required = Cache Data / Available Memory

Or for our example;

Total number of cache servers = 48 GB / 4 GB = 12 cache servers

To ensure is there is sufficient memory to hold all the cache data, should a cache server fail, an additional cache server should be provisioned. However, as this is also true for physical servers, this requirement will be met by subsequent recommendations.

[11] Strictly speaking replicated cache entries are stored in a serialized format until they are accessed for the first time, after which they are then held in a deserialized format.

Note: Detailed JVM tuning guidance is beyond the scope of this white paper, as it's usually dependent on the memory and CPU utilization profile of an application, it's desired performance characteristics and other factors.

- How many servers will I need?

  Although the resource requirements for different Coherence applications can vary significantly, some general recommendations can be offered.

  - At least 3 machines should be used to ensure that the data is evenly balanced and if one should fail your data configuration is still resilient, i.e. "machine-safe"
  - As a general rule start with 1 core per cluster node, be it a proxy node, cache or management node.
  - Apportion resources according to the requirements of your application, not the resources of your machines. Having machines with balanced resources that align with the requirements of your applications workload will ensure they are fully utilized. For instance, if your servers have 512 GB of memory, but only 16 cores and sit on a 1 GBE network, your application may not be able to full utilize that much memory if its workload is compute or IO intensive. Furthermore, the MTR after a machine failure will be a lot longer in an un-balanced configuration like this, because a large amount of data needs to be rebalanced over a network with limited bandwidth.
  - Finally, just as $n + 1$ cache servers should be provisioned (where n is the number of cache servers to support your application under normal conditions), $n + 1$ machines should also provided, so if one fails or is shutdown there is still be sufficient capacity to store all the data.

  Using these recommendations, the example below illustrates how to calculate the number of machines required to host this Coherence cluster.

  Assumptions:

  - Oracle HotSpot 1.7 64 Bit JVM
  - JVM binaries overhead for 64 bit Hotspot JVM ~280 MB
  - OS memory overhead ~4 GB
  - 12 cache servers to hold 24 GB of primary data, each with a 6 GB heap (from previous example)
  - Assign approximately 1 core per cache server
  - 1 Management Node with a heap of 4 GB. However, this will not be included in sizing.
  - Server Specification of 8 cores, 64 GB memory and 1 GBE network

  Memory footprint for each cache server = 280 MB + 6 GB = 6.28 GB

Memory requirement for 8 cache servers = 8 * 6.28 GB = ~50 GB

Total number of servers required = (total cache servers / cache servers per machine) + 1 additional machine for resilience = (12 cache servers / 8 cache servers per machine) + 1 = 3 machines

The additional machine is required to ensure that if a one fails there are still sufficient resources (memory) available to store the complete data set.

Cache server processing requirements can vary significantly. For instance an application processing a lot of events, queries or performing in-grid processing will typically use more CPU resources than one handling simple key based access operations – put/get/remove. However, as a rule of thumb allocating 1 core per JVM is a good starting point for testing.

- How many proxy servers will I need, if I have extend/external clients?

  This will depend on a number of factors;

  - The number of concurrent client connections
  - The network capacity available to the proxy server JVM's
  - The number of requests and events per client p/s
  - The size of the events and requests / responses

  To estimate how many requests or clients a single machine could support lets take an example.

  Assumptions;

  - Proxy servers are more likely to be network rather than CPU bound.
  - Additional cluster traffic will be ignored
  - It is assumed that each machine has a 1GBE Network Interface Card (NIC), providing ~100 MB p/s of inbound and outbound through-put
  - The client workload is a mixture of read and write requests (80:20)
  - Each entry is approximately 1 K with a small key (which will be ignored).
  - Each read may incur an additional network call to another machine and a write operation certainly will – if backups are used. So a read may involve a 1 K response from a primary and

onwards to the client. A write may involve a 1 K request from the client, a 1 K request on to the primary, a 1 K response from the primary[12] and a 1 K response to the client.

This means that 80 reads will result in ~80 K of both inbound and outbound traffic, and 20 writes will result in 2 x ~20 K = 40 K of both inbound and outbound traffic. So 100 requests will result in 80 K + 40 K = 122 K of network traffic, both in and out of the NIC

Therefore, a single server could support 100 MB / 120 K = 833 clients sending 100 requests p/s, or 833 clients * 100 requests = 83 K requests p/s per machine

If there are a large number of extend clients then you may want to consider;

- Dedicating servers for proxies
- Dedicating one network interface for external network traffic and another for cluster network traffic

As usual theoretical sizing calculations should be validate through testing and the above estimates assume that events are not being sent to clients. If events are being generated then the additional network bandwidth and processing this will required will also need to be incorporated.

Finally, a management node will need to be hosted within the cluster. This should be a storage disabled cluster member. Its memory requirements will vary depending on the size of the cluster, but here a JVM with a 4GB heap would be a reasonable starting point.

Cluster Topology

The topology of a Coherence cluster can have a significant impact on its performance, reliability and scalability. This section discusses the various components of a Coherence cluster, introduces some new features in the 12c release and complements the documentation with some suggestions around best practice.

Ideally all the nodes in a Coherence cluster should be located on the same network segment and connected via the same switch, to ensure fast and reliable cluster communication. Also each machine should preferably have the same hardware and software specification, as a heterogeneous environment will be more complex to configure and harder to mange. If an environment is too unbalanced then a cluster may not be able to recover from the failure of a large machine. Even running multiple Operating Systems and JVM's can introduce problems, as their different performance characteristics can create areas of contention.

Coherence 12c enables a complete separation between the logical and physical topology of a cluster, allowing developers and administrators to work separately on their respective areas of concern. The

[12] If a "blind `put()`" is not being used. That is `putAll()`, which does not return the previous entry value.

physical elements of a proxy and invocation service, like their network addresses and ports, no longer need be part of the logical cache definitions. To introduce this separation;

1. In the `<tcp-acceptor>` element in the Coherence cache configuration file an empty `<tcp-acceptor/>` element is specified, instead of one with nested socket information. |

```
<!--
Proxy Service scheme that allows remote clients to connect to the
cluster over TCP/IP.
-->
<proxy-scheme>
  <scheme-name>examples-proxy</scheme-name>
  <service-name>ExtendTcpProxyService</service-name>
  <thread-count system-property="tangosol.coherence.extend.threads">2</thread-count>
  <acceptor-config>
    <!-- empty tcp-acceptor means to use the name service -->
    <tcp-acceptor/>
  </acceptor-config>
  <proxy-config>
    <cache-service-proxy>
      <lock-enabled>true</lock-enabled>
    </cache-service-proxy>
  </proxy-config>
  <load-balancer>proxy</load-balancer>
  <autostart>true</autostart>
</proxy-scheme>
</caching-schemes>
```

Figure 24. Showing blank <tcp-acceptor> configuration when using the 12c name service

This starts the Coherence name service on ephemeral sub-port 3[13] of the unicast listener port (`-Dtangosol.coherence.localport`). The following output indicates this has taken effect;

```
Oracle Coherence GE 12.1.2.0 <Info> (thread=NameService:TcpAcceptor,
member=4): TcpAcceptor now listening for connections on
192.168.1.100:9020.3>
```

If Managed Coherence Servers are being used the Proxy specific cache configuration file can be specified in the Admin Console as shown below.

[13] To simplify its configuration Coherence multiplexes connection on a single TCP port so that a request for TCPRing will go to TCPRing and a request for the naming service will go to the naming service. This is all done via sub-porting (ephemeral port) to designate what service to speak to. This is possible because Coherence mediates both sides of the connect and can add a protocol header to requests that indicate the sub-port destination.

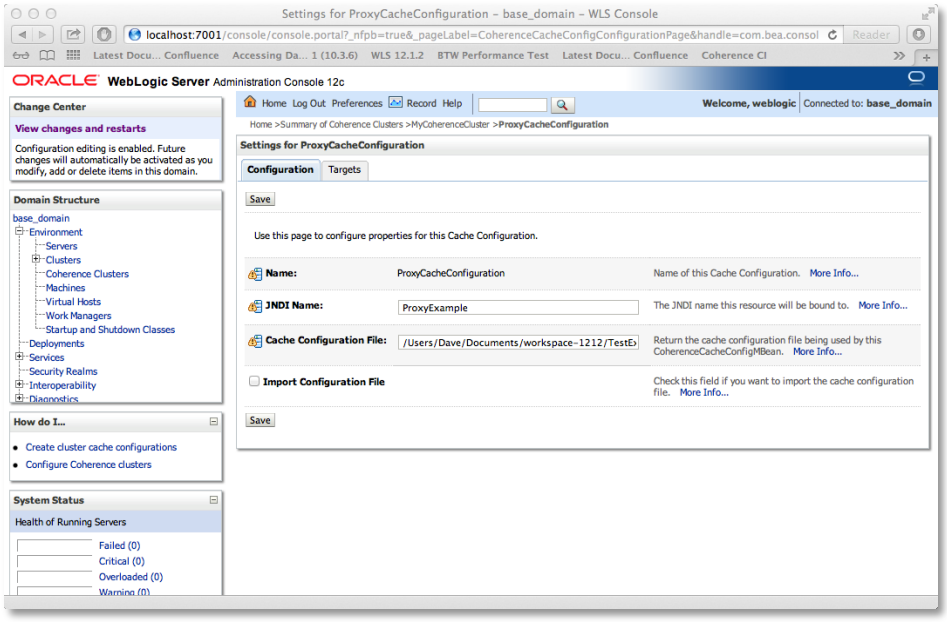

Figure 25. With Coherence 12c Managed Servers a custom cache configuration file can be specified

The default cache configuration in the GAR file will be overridden by the custom cache configuration that include Proxy service settings, because it has the same JND name. Here the "ProxyCacheConfiguration", setup in the Admin Console, is targeted at the Proxy Managed Servers.

```
<?xml version="1.0"?>
<coherence-application
    xmlns="http://xmlns.oracle.com/coherence/coherence-application">
    <cache-configuration-ref override-property="cache-config/ProxyExample">META-INF/coherence-cache-config.xml
    </cache-configuration-ref>
    <pof-configuration-ref>META-INF/pof-config.xml</pof-configuration-ref>
    <application-lifecycle-listener>
        <class-name>com.oracle.coherence.test.MyAppLifecycleListener
        </class-name>
    </application-lifecycle-listener>
</coherence-application>
```

Figure 26. Use the "ProxyExample" JNDI name to reference an alternative cache configuration

2. The proxy service starts, registers its **service-name** with the name service and listens on a random ephemeral sub-port of the unicast listen port. For instance;

```
Oracle Coherence GE 12.1.2.0 <Info>
(thread=Proxy:ExtendTcpProxyService:TcpAcceptor, member=4): TcpAcceptor
now listening for connections on 192.168.1.100:9020.60563>
```

3. An address-provider is specified in the `tangosol-coherence-override.xml` file for the extend client which points to the unicast-listen ports for the name service

```
<?xml version='1.0'?>
<coherence
    xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-operational-config
        http://xmlns.oracle.com/coherence/coherence-operational-config/1.2/coherence-operational-config.xsd">
    <cluster-config>
        <address-providers>
            <address-provider id="extend-address-provider">
                <socket-address>
                    <address>localhost</address>
                    <port>9020</port>
                </socket-address>
            </address-provider>
        </address-providers>
    </cluster-config>
    <logging-config>
        <severity-level system-property="tangosol.coherence.log.level">9</severity-level>
    </logging-config>
</coherence>
```

Figure 27. Address provider used by the name service is added to the override file

4. A client then uses the address provider configuration to contact the name services and specifies the **service-name** of the proxy service it wishes to contact, which in this case is "ExtendTcpProxyService".

5. The name service returns the location (sub-port) of the proxy service to the client, which it then uses for further communications.

The relationship between the different configuration files is shown below;

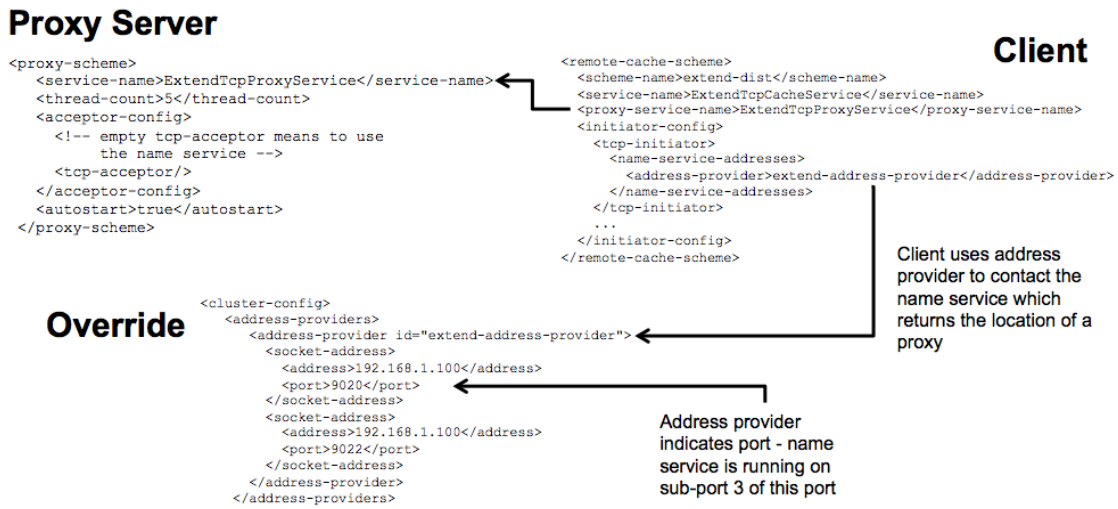

Figure 28. The relationship between the configuration files when using the name service

This rest of this section outlines some guidelines for configuring and deploying the various components of a Coherence cluster.

Proxy Servers

As part of a product wide initiative to make Coherence more "self managing", 12c introduced a dynamic thread pool for proxy services. This allows a proxy service to grow and shrink its thread pool to handle extend connections as needed. By default the minimum value is 2 times the number of cores and the maximum 8 times the number of cores. This should be a good starting point for many applications, but if the maximum is reached a log alert will be raised and you can increase the pool size. Proxy based load balancing should also be considered to ensure the client load is evenly spread across the proxy servers.

Extend Clients

Extend clients have nearly all the functionality of those in a cluster, but can utilize a range of technologies (Java, .Net, C++ and others like JavaScript, using the REST API). Since they access cache data via a proxy service and are not directly involved in cluster communications, they can reside on slow or unreliable networks and don't impact the cluster when they startup or shutdown. Therefore, Coherence clients, that are desktop application or only run for a short period of time - for instance to load some data or to perform a query - should be configured as extend clients.

To ensure extend communications are reliable and resilient, extend clients can be configured to use an out-going heartbeat – with a timeout – to make sure the proxy it is connected to is still responsive. Where extend clients are listening for events they can register a "member left" event listener, to detect connection errors, and re-connect by issuing a simple out-going request - for instance by calling the cache `size()` method.

The Management Server

The Management Server should run as a separate cluster node and need only start a "dummy" cluster service to join the cluster. An example configuration could look something like this;

```
<?xml version="1.0"?>

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
  xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
  <caching-schemes>
    <invocation-scheme>
      <service-name>Dummy</service-name>
      <autostart>true</autostart>
    </invocation-scheme>
  </caching-schemes>

</cache-config>
```

Figure 29. Sample Management Node cache configuration

However, when using Managed Coherence Containers in the 12c release, it is the Weblogic Domain Administration Server that collects JMX metrics for the Coherence cluster. So a separate Management node is not required.

Cache Servers

Coherence cache servers should typically run in a separate JVM from a client application, to enable both the client and data-grid tier to be tuned and scaled independently from each other. However, for simplistic caching use cases or for small amounts data, running clients and cache services in the same JVM can make sense.

Caches

Near Cache

A near cache can improve application performance and reduce network traffic by caching a local copy of a previously requested entry. However, in deciding whether a near cache is appropriate a number of factors need to be considered. For instance;

- How often the same data is re-requested by clients?

  If entries are not requested more than once by a client then a near cache will not improve performance

- What is the distributed cache update rate?

  If entries are updated so frequently that a client does not have a chance to re-use their local copy a near cache won't improve performance

Coherence JMX metrics can be used to gather a lot of this information and if the data access profile is not well understood an example near cache can be configured to measure its effectiveness by monitoring the HitProbability attribute of the *Cache* MBean. As a rule of thumb, ensure that near caches are yielding a hit probability of at least 80%. A hit rate lower than this probably means there is not much benefit from having a near cache.

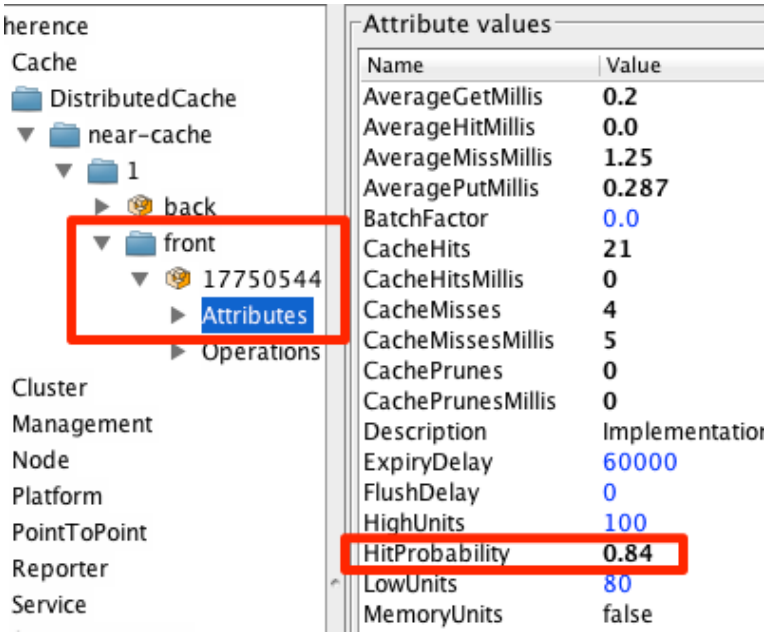

Figure 30. JConsole view of a near-cache HitProbability attribute

The flow diagram below illustrates the decision process for deciding if your client applications will benefit from a near cache, and if so what configuration will provide the best performance. As always validate your configuration through testing.

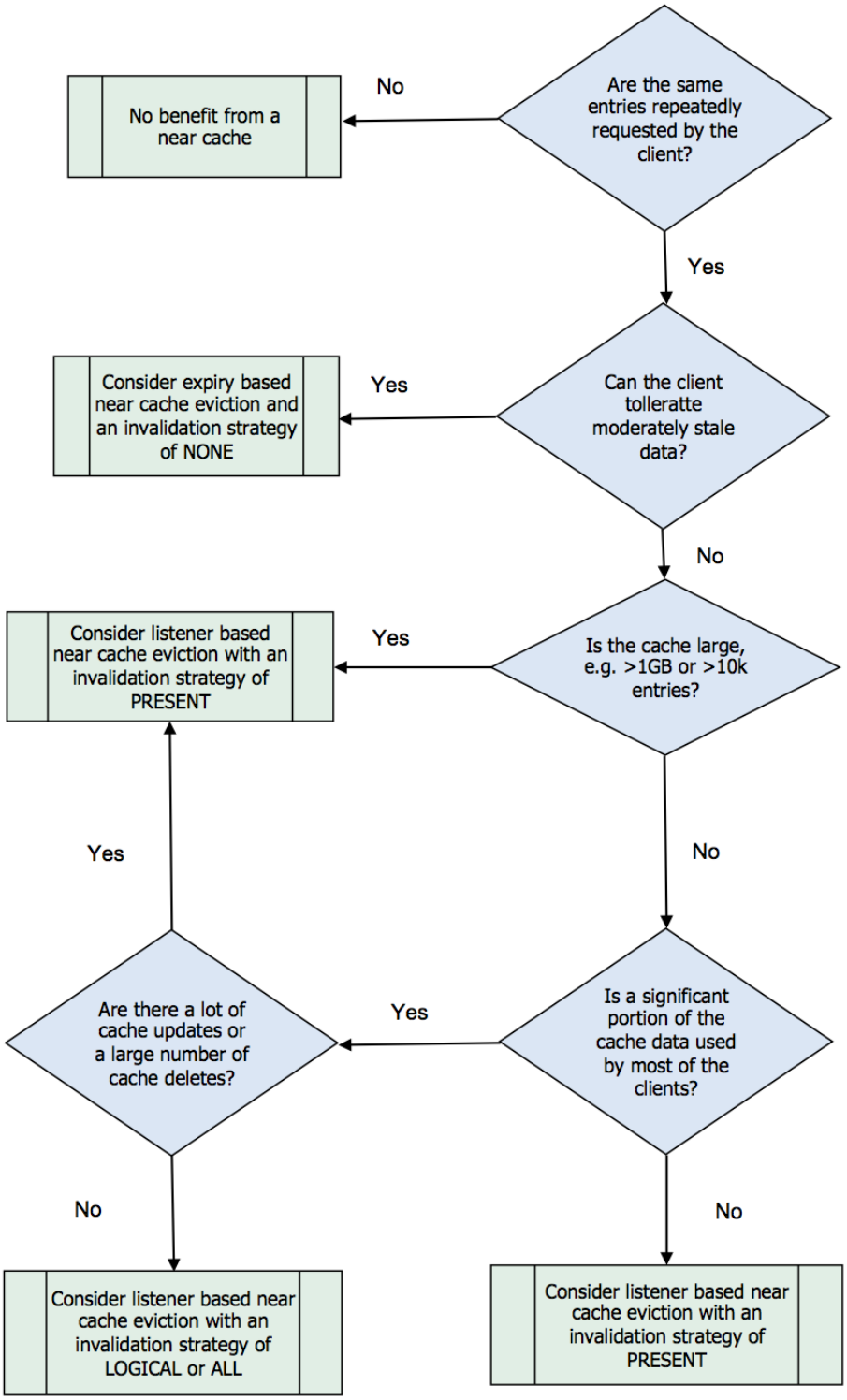

Figure 31. Decision tree for selecting the best near-cache configuration

Finally when configuring a near cache;

- Always size limit your near cache and be conservative. An optimum near cache size cane be found by setting a small near cache size and then gradually increasing its size to find a good "hit rate" that consumes the minimum amount of memory.
- Do not use the ALL invalidation strategy if you intend to call `cache.clear()` or perform a bulk update/delete of you distributed cache, because of the large number of events this can generate..
- Be aware that the new default invalidation strategy for a near cache (if none is specified) in Coherence 12c is now PRESENT.
- Coherence 12c introduces a new invalidation strategy of LOGICAL. This is like the ALL strategy, except synthetic events – from operations like cache eviction – are not sent to clients.
- If the entries in the distributed cache are periodically updated in bulk consider using a NONE invalidation strategy, to prevent a large number of events being sent to clients, in conjunction with one the following techniques;
  - After the bulk update use an invocation service to clear the clients near cache, so that they will re-read the new entries
  - Notify clients before the bulk update, so that they can remove their near cache, and afterwards so that they can re-create it. This could be accomplished using an event listener on a control cache to communicate the state changes
- The cache configuration for a proxy member should never contain a near-scheme. When a cache server (storage-enabled member) reads its cache configuration, it will skip the creation of the near cache (since it's not a client) and instead create the back-scheme as defined by the configuration. When a storage-disabled member reads its configuration it will do the opposite, it will create the front of the near cache and skip the creation of the back-scheme.

  This presents an interesting scenario when proxy members are added into the cluster. The best practice for proxy members is to disable storage in order to preserve their heap for handling client requests. This means that the proxy will create a near-scheme, just like any other storage-disabled member, but won't use the deserialized data in it as the main function of the proxy is to forward binary entries to Extend clients. This not only leads to more memory consumption, but also more CPU cycles for deserializing cache entries. Also if the proxy is serving more than one client, it is likely that the near cache will experience a lot of turnover, resulting in greater GC overhead.

Replicated Cache

A replicated cache will replicate its contents wherever it is running. Although its configuration is similar to that of a distributed cache, some key differences are;

- Replicated cache services are always started when they appear in a cache configuration, even on a storage-disabled node - which is usually undesirable. The auto-start and storage-enabled options are only for distributed cache services.

- Entries are held in a deserialized format – after they have been accessed for the first time.

Replicated caches can be useful, but in most cases a distributed cache combined with a near cache will provide greater flexibility.

Distributed Cache

A distributed cache offers flexibility, scalability, performance and reliability. The cache data managed by a distributed cache service is split into partitions. Cache entries are randomly assigned to these partitions, unless data affinity is used to store related cache entries together in the same partition. A partition assignment strategy is then used to distribute the partitions amongst the cluster members that run the same service.

The number of partitions that should be configured for a distributed cache service will depend on the amount of cache data being stored and some suggested sizes are outlined in the Coherence Developers Guide.

Cluster Replication

Where Coherence clusters in different locations need to share data for resilience, or simply to improve performance, a number of replication options are available. Each has its own benefits and drawbacks, but between them they are able to meet a wide variety of requirements. The options are:

- Using the extend mechanism, so that one cluster is a client of another. The benefit of this approach is that it's very simple. A custom cache store can be used to use propagate changes from one cluster to another. This can be configured as a write-through (synchronous) or write-behind (asynchronous) cache store. One drawback is that bi-directional or more complex replication topologies may require a lot of additional development.
- The Coherence Push Replication pattern in the Coherence Incubator provides asynchronous cluster replication. It has a rich set of features that can be utilized through configuration rather than coding. These include;
  - Support for a wide range of replication topologies, like active-active, active-passive, hub-spoke, multi-master etc.
  - JMX monitoring support (integrated with Oracle Enterprise Manager)
  - Examples, a tutorial and full source code, available through the Open Source CDDL model and integrated with Maven and GitHub
  - An architecture designed to be extensible and customizable

- Facilities for batching, conflict resolution etc.

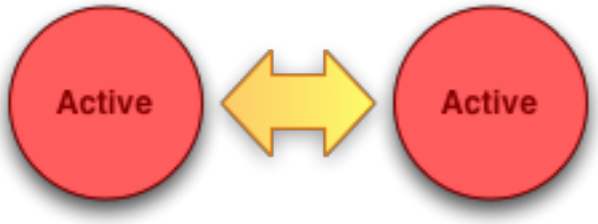

Figure 32. An active-active cluster topology

- Replication through the Coherence 12c "HotCache" feature. In many use cases a database is the primary data source, so when cache entries held in Coherence are changed they are written back to the database. Furthermore, the database is often already being replicated to the locations where other Coherence clusters reside. In these scenarios "HotCache" can be used for propagating database changes, applied through database replication, into a Coherence cluster at the remote location. In fact if Golden Gate is being used for the database replication it can be configured to replicate to both the database and the Coherence cluster at each location through "HotCache".

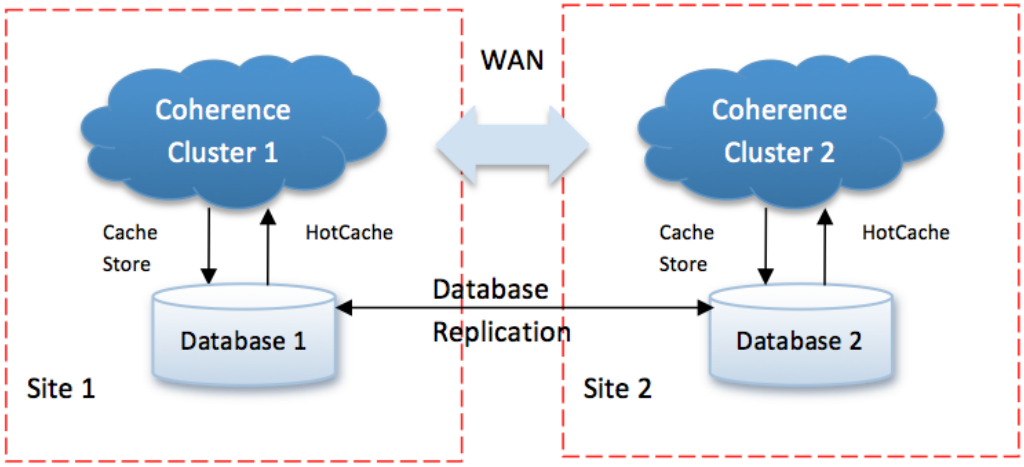

Figure 33. Inter-site replication using HotCache and Golden Gate

Hardware Considerations

It is tempting to size your hardware resources based on the normal requirements of your application. However, if failures occur Coherence will need to perform more processing, i.e. the recovery and rebalancing of lost data, with fewer resources. So make sure you have sufficient hardware to handle potential failures, within your SLA's. Platform specific considerations are outlined in the Coherence Administration Guide.

Software Considerations

Coherence has few additional software requirements. Cluster members are all based upon Java and so certified against Java, not the underlying Operating System. However, when setting up a test or production environment for Coherence you should consider the following;

- **JVM**. Some platforms provide their own JVM implementation. However, if there is a choice of JVM's then select the one that is most widely used– everything else being equal
- **Operating System**. Although Coherence runs on a range of operating systems, and they each have their strengths and weaknesses, Linux is probably the most popular platform - and the OS used for internal testing.
- **Monitoring**. See the next section.

Monitoring

A range of monitoring tools exists for Coherence, provided by Oracle and 3rd party vendors like SL Corporation, LogScape, Splunk, Evident Software and ITRS. Monitoring Coherence is critical in a production environment and the overhead of doing so should be factored in to any performance testing.

Metrics available through JMX are usually gathered through the Coherence Management Node. The interval at which JMX metrics are gathered will usually need to be extended (the default interval is 1s). For small clusters an interval of 10s may be suitable but for larger clusters this should be increased to 30s – or even higher. The quantity of MBean metrics collected will depend on the number of nodes, caches and services in a cluster. The size of each MBean will also vary, depending on its attributes. Below is a list of the main Coherence MBean's and the multiplier to calculate how many will be collected at each interval.

MBEAN TYPE	MULTIPLER
Cache	Service Count * Cache Count (for service) * Node Count
Node	Node Count
Partition Assignment	Service Count * Node Count
Point to Point	Node Count
Reporter	Node Count
Service	(Service Count * Node Count) + (Management Service * Node Count)
Storage Manager	Service Count * Cache Count (for service) * Node Count

To reduce the number of Coherence and platform MBean's collected, filtering can be used to prevent them being accessed remotely through the Coherence management framework. Filtering can be useful

if platform metrics are being collected through a different mechanism, for instance using Oracle JVM Diagnostics, or to exclude metrics from ancillary or temporary caches.

Larger clusters will need to configure a larger heap size for the Management Node. Where as a 2 GB heap for a small cluster may be fine a 4 GB heap may be required for a larger cluster. If the Management Node is performing excessive processing, using a lot of memory or metrics have collection gaps, then the collection interval may be too short or fewer MBean's may need to be collected at each interval. Further details on configuring the Coherence Management Node can be found in the Coherence Management Guide.

Although Coherence makes a wide range of metrics available through JMX it's also important to monitor its eco-system, including the JVM, CPU, memory, OS, network, logs and disk utilization. Below is a list of some of the key metrics that should be monitored, along with some suggested thresholds for raising alerts. Note this is not a definitive list of metrics to monitor and thresholds may vary between applications and environments.

1. **Data Loss**. This is the most important metric to monitor, for obvious reasons. In Coherence 12c there is a new partition MBean notification on the *SimpleStrategy* MBean raised when a partition is lost. This event is also logged as a "orphaned partition" log message. The JMX "partition-lost" notification event is shown below[14].

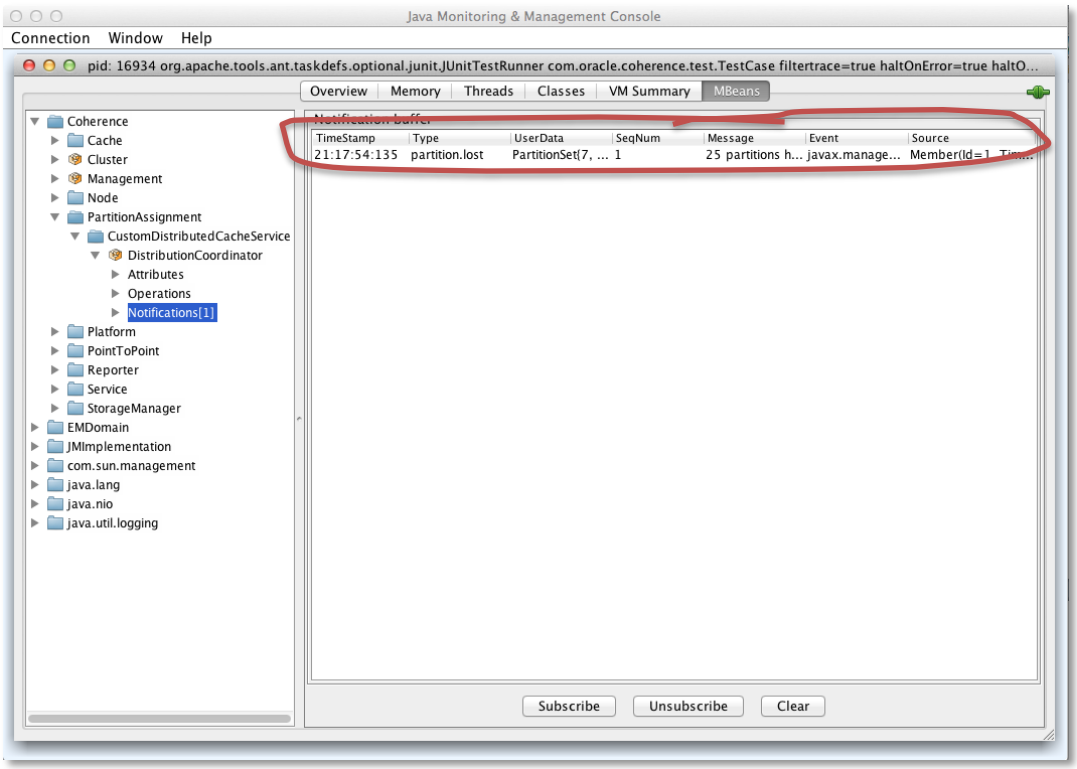

Figure 34. A partition-lost JMX notification event raised when a partition and data is lost

Partitions can be lost because 2 machines have failed simultaneously. An error alert should be raised if this JMX notification event is received or an "orphaned partition" error message appears in the Coherence log file. Configuring multiple backups or implementing the other HA strategies outlined earlier can mitigate the risk of this occurring.

2. **Data At Risk**. The vulnerability of cache data is highlighted through the *Service* MBean StatusHA attribute. It indicates how safe your data is. If the HA (High Availability) status of one of the distributed cache services changes to NODE_SAFE[15] or even worse ENDANGERD[16] then a warning alert should be raised. This can happen if a storage node leaves a cluster. If a node remains in this state for a prolonged period, for instance >2 min's, then an error alert should be raised. The HA status of a distributed service may legitimately be in one of these states for a short period of time when re-balancing during a rolling restart. However, if this happens for a prolonged period then

[15] The NODE_SAFE status indicates that the primaries and backups of some cache entries are only held on different nodes, not different machines

[16] The ENDANGERED status means that data is only held in one node

there may be an issue with a service thread on a node, for instance a partition maybe "pinned" to a node because of an error state.

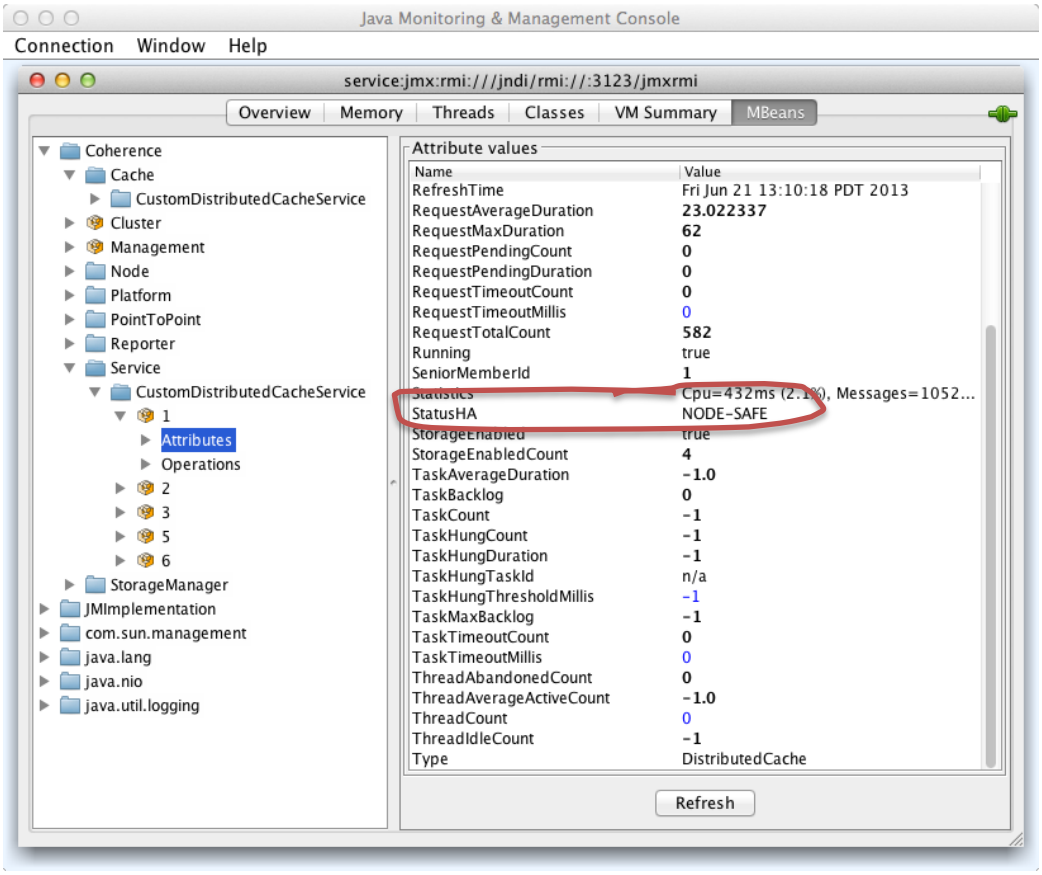

Figure 35. The *Service* MBean StatusHA attribute indicates here that cache data is only held on another node, not another machine

3. **Cluster Change**. A change to the number of cluster nodes can be significant. If the node that leaves holds cache data, is a proxy or runs some important processing, then an alert should be raised. The MembersDeparted and MembersDepartedCount attributes on the *Cluster* MBean capture changes in cluster membership. Using the "role" of a node is one way to determine if this is a significant event.

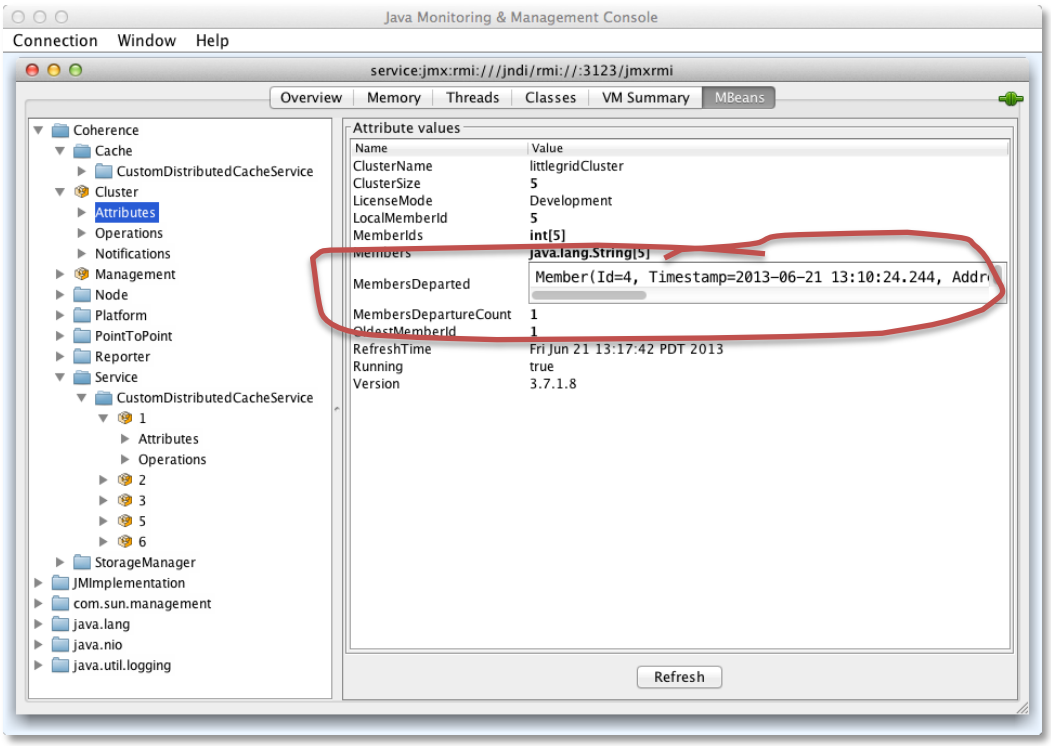

Figure 36. The *Cluster* MBean MembersDeparted and MembersDepartedCount attributes record metrics about members that have left the cluster

A service re-start also indicates that a problem has occurred, for instance because a service thread on a node has hung and had to be re-started by the service guardian. To capture this Coherence 12c adds the service JoinTime as a new *Service* MBean attribute. If it changes between JMX collections then the service has been re-started.

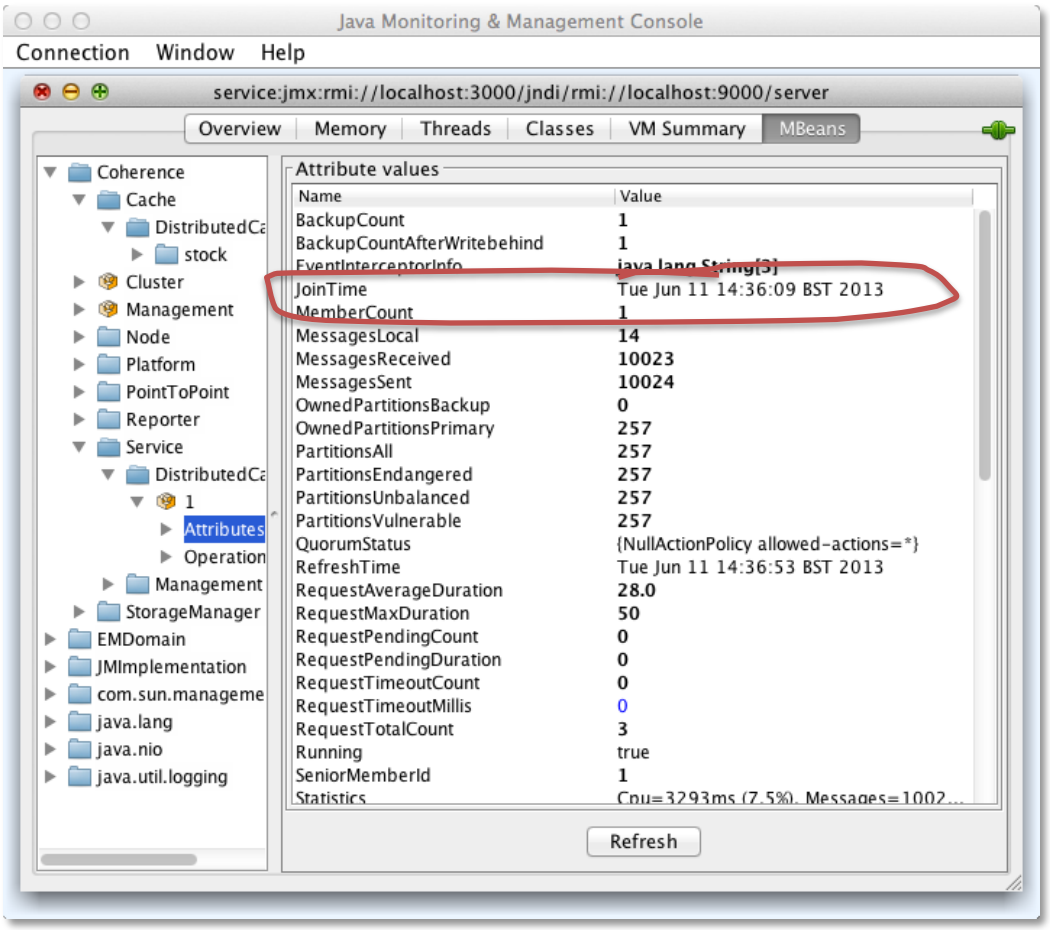

Figure 37. The new *ServiceMBean* JoinTime attribute

4. **Low Memory**. Memory consumption needs to be closely monitored so that if data volumes grow beyond expected levels, additional capacity can be provisioned and the unexpected growth investigated. Although careful capacity planning should ensure that there are sufficient resources to withstand the loss of a machine, processing spikes or incremental growth may not have been predictable. The threshold at which this growth becomes a risk to the cluster can be calculated as follows;

   Assumptions:

   - Let N be the number machines each with an overall capacity of C
   - The maximum safe utilization level is 70%
   - If one machine is lost then those remaining (N-1) must accommodate 1/N of the overall data

Therefore, the following should be true: 0.7 > C + (C / (N-1))

To illustrate this with an example, lets assume that there are 3 machines in a cluster. If an alert is raised when the total memory utilization exceeds 50%, the limit of 70% will be exceeded if a machine fails, as 0.5 + (0.5 / (3 – 1)) = 0.75 or 75%. The alert threshold needs to be lower, for instance 45%, as 0.45 + (0.45 / (3 – 1)) is 0.675 or 67.5%. So for a 3 machine cluster a critical alert should be raised if the total memory used is greater than 45% and perhaps a warning alert if it exceeds 40%.

As well as identifying suitable memory utilization thresholds, its also important to make sure memory usage is accurately measured. The only reliable way to do this is to measure the heap space available after the last GC, which is available on the following MBean *Coherence:type=Platform,Domain=java.lang,subType=GarbageCollector,name=PS MarkSweep,nodeId=<node id>* as the used value. However, between collections the best approximation of used memory is the *Coherence:type=Platform,Domain=java.lang,subType=GarbageCollector,name=PS Scavange,nodeId=<node id>* used value, as shown below.

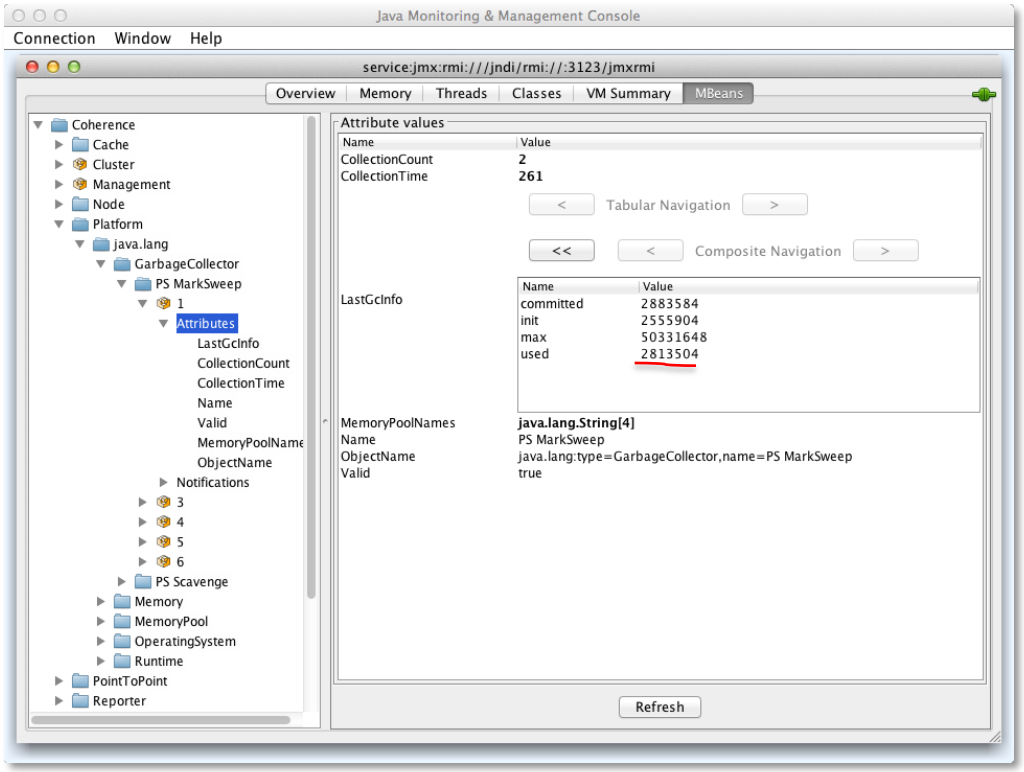

Figure 38. The heap used after last GC can be found under the "*PS MarkSweep*" MBean

5. **Long Garbage Collection (GC) pauses**. Long GC pauses can impact an applications performance, throughput and scalability. If a cluster node is taking a long time to perform a garbage collection

then an alert should be raised. What the threshold for an alert should be can vary, but a suggested threshold for a warning alert is >1s and >5s for a critical alert. This information can be accessed on the *"PS MarkSweep"* MBean outlined below. Possible reasons for these alerts can be members leaving the cluster (putting additional memory pressures on the remaining members), CPU starvation (because the machine has become overloaded), swapping etc.

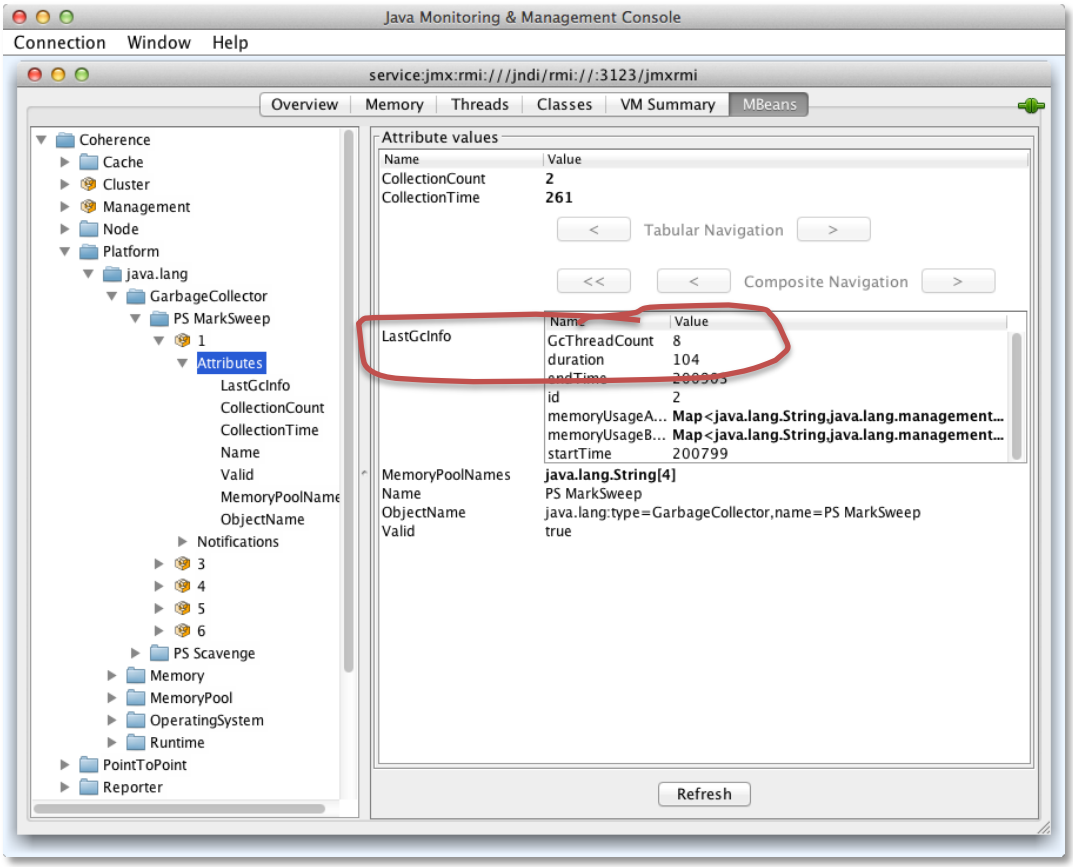

Figure 39. The last GC pause duration can be found in the in *PS MarkSweep* MBean GC metrics

6. **Insufficient Processing Resources**. If the Coherence service threads cannot keep up with the number of requests being made then the task backlog queue will start to grow, impacting an applications performance, throughput and scalability. The TaskBacklog attribute can be found under the *Service* MBean for each service. A suggested threshold for raising a warning alert is if the queue length >5 and if >20 a critical alert. Corrective action here could be to start more service threads. However, if too many service threads are allocated the ThreadIdleCount attribute under the *Service* MBean will be consistently >0, indicating that the thread count for a service is too high.

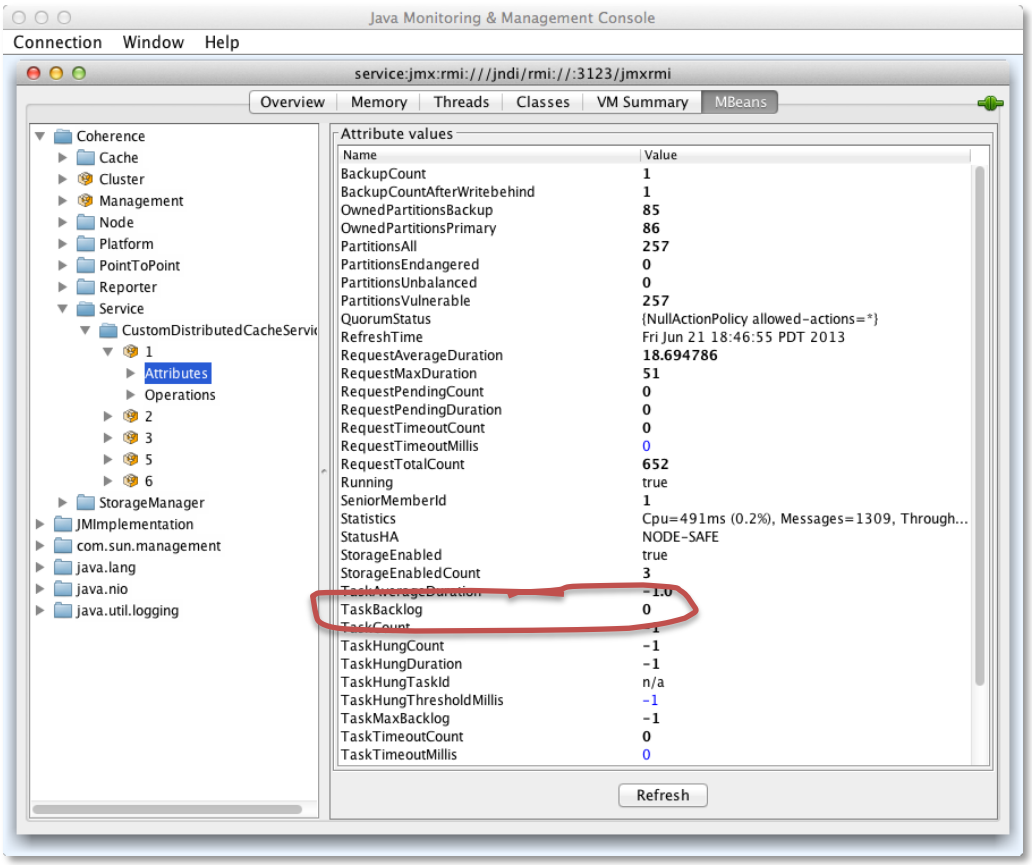

Figure 40. The TaskBacklog attribute for a service indicates if task are being queued before being processed

7. **High CPU Utilization**. Overloading a machine hosting Coherence may lead to unpredictable behavior and prevent recovery within the target SLA's in the event of a failure. Therefore, it's important to monitor CPU utilization. If overall CPU utilization is high this could be symptomatic of other processing being performed on a Coherence server, like a system backup, or that there are insufficient processing resources to meet the demands of the application. Alternatively if load is high on just one or more cores, a Coherence service be overloaded and increasing the service `<thread-count>` may help – see the previous section. Some suggested CPU utilization thresholds are >80% for a warning alert and >95% for a critical alert – or lower if you prefer to be more cautions.

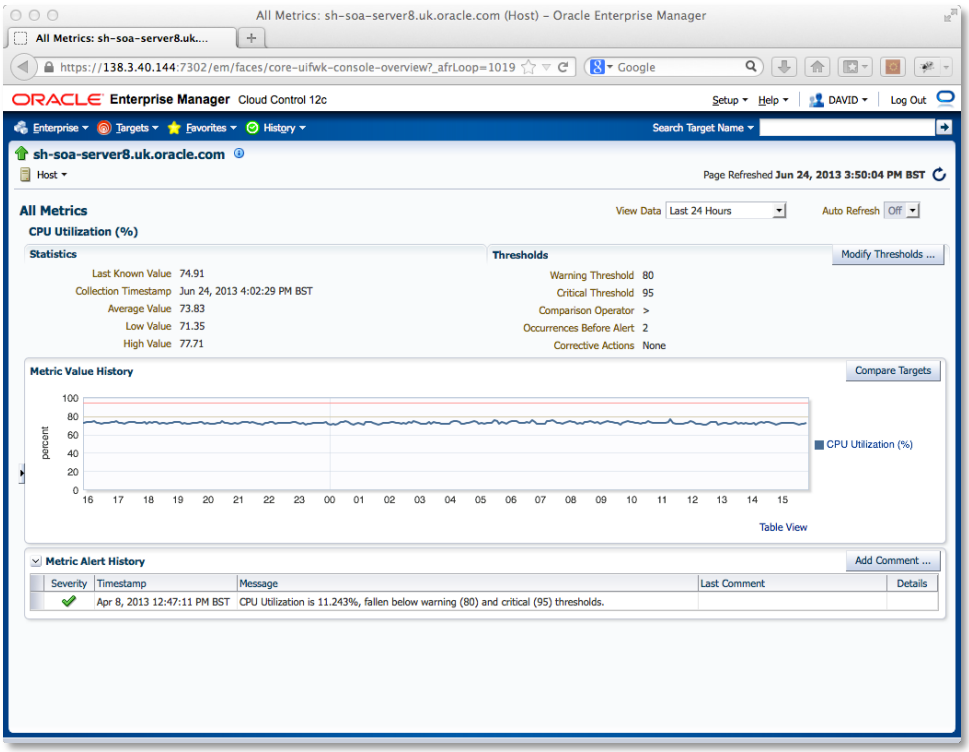

Figure 41. Oracle Cloud Control 12c can monitor and alert on host metrics, like CPU utilization.

8. **Communication Issues**. Coherence is a network centric application and sensitive to communication issues. The *PointToPoint* MBean can highlight these in its PublisherSuccessRate and ReceiverSuccessRate attributes. If either is lower than <95% a warning alert should be raised, and if lower than <90% an critical alert. A possible cause could be incorrectly configured network equipment or Operation System parameters, so check that the Production Checklist etc. has been followed.

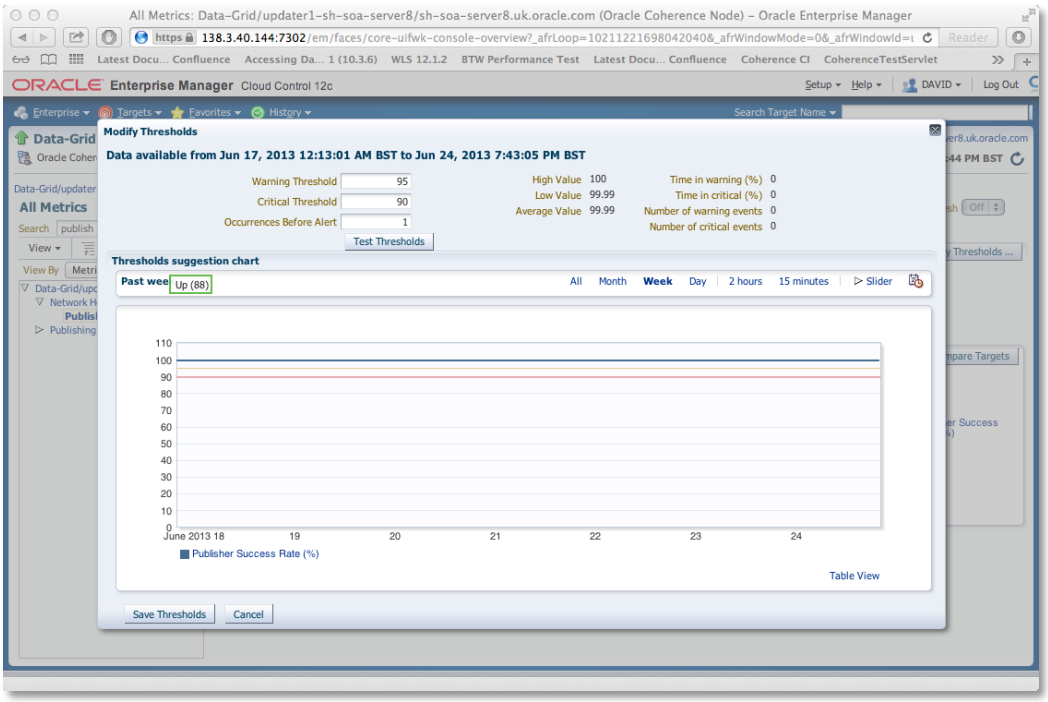

Figure 42. Thresholds can be set for both warning and critical alerts on the *PointToPoint* MBean PublisherSuccessRate and RecieverSuccessRate attributes in Oracle Cloud Control 12c

9. **Network Saturation** The Network Interface Card's (NIC's) being used by Coherence should be monitored. Some suggested thresholds for alerts on a 1GBE network are >60 MB/s for a warning and >90MB/s for a critical alert. Network saturation may not always be visible from the metrics gathered from a local NIC, for instance if Coherence shares a switch or router with other applications that over utilize the network, the problem may only be visible from the local NIC's.

10. **Error Messages** Error and Warning messages in log files should be monitored and appropriate alerts raised. These messages are listed in the Administrator's Guide and can be monitored using a number of tools, like Splunk, LogScape and Oracle Enterprise Manager. These tools perform pattern matching to detect their occurrence.

11. **Event Monitoring** When a large number of cache entries are removed or updated a lot of events can be generated, especially if a large number of clients use a near cache invalidation strategy of ALL or are listening for changes in a large number of entries. To monitor the number of events being generated look at the MBean attribute *StorageManager* -> *Service* -> * -> EventsDispatched and define alerts if the number of events over a collection interval exceed appropriate thresholds. Note this value is the total number of events received by a node since the statistic was last reset. So if you expect 10k events p/min over a 10 node cluster, then >2k could raise a warning alert and >5k a critical alert. To determine sensible thresholds it's usually necessary to monitor a test environment first, to see how an application behaves under normal conditions.

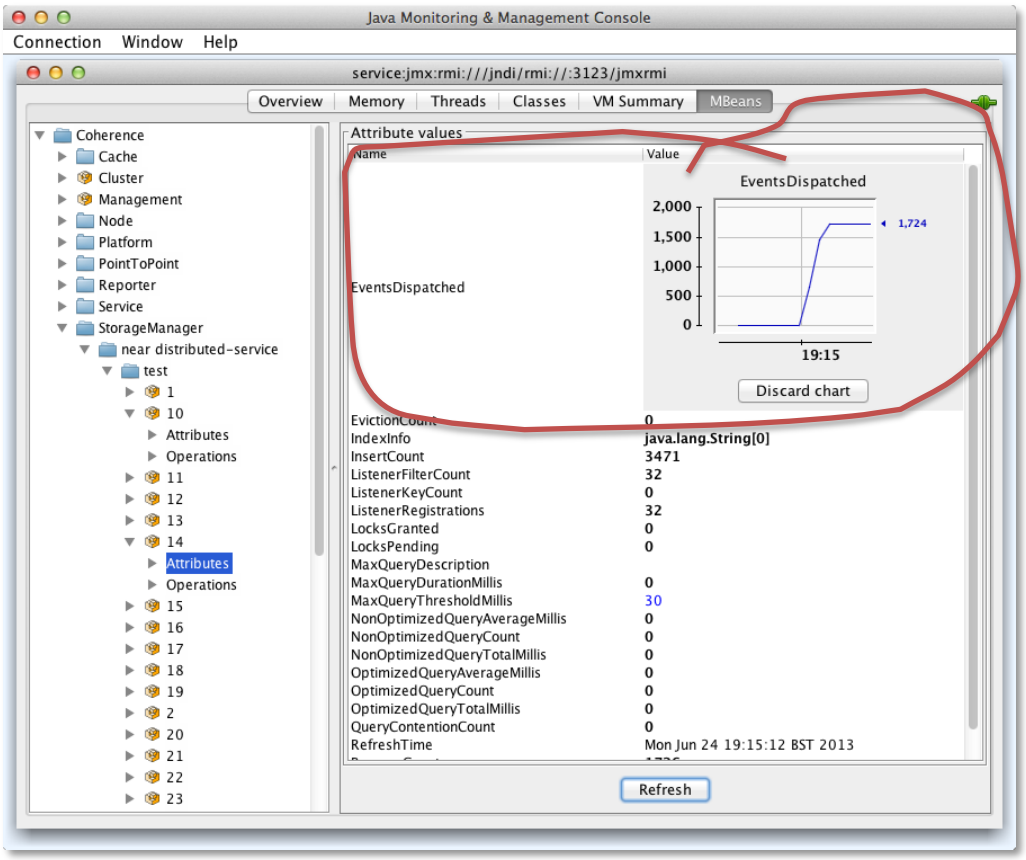

Figure 43. The EventsDispatched attribute on the *StorageManager* service MBean shows the events since statistics were last reset

Logging

Another source of monitoring information are the Coherence and Java GC log files. Java GC logging parameters are covered in the Coherence Administrator's Guide, and so will this not be discussed any further here[17]. To ensure the maximum amount of information is captured in the Coherence log files the highest log level for Coherence, level 9, should be used. This should not create a lot of additional logging under normal conditions, but will provide invaluable information if problems arise, making it easier and quicker to diagnose their cause. Since Log4J is one of the most popular Java frameworks for logging, the following guidance is illustrated using its components. However, they equally apply to

[17] Java 6 Update 37 and Java 7 have introduced the ability to now roll GC log files. For example: -Xloggc:./gc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=50M

other logging frameworks and Coherence 12c also adds support for the Simple Logging Facade for Java (SLF4J).

A large Coherence cluster can generate a significant amount of logging information, by virtue of the number of processes writing log files. Capturing and managing this effectively is critical to ensure that disks don't become full and application performance isn't adversely impacted. Regularly rolling and archiving log files, using a Log4J *rolling file appender*, should prevent local disks filling up, and a non-blocking Log4J *asynchronous appender* can be used to discard messages in case it does. Care should be taken tune both mechanisms to optimize the frequency that log-files are rolled, the duration archives are kept for, as well as the number of messages that are buffered by an Log4J *asynchronous appender*.

If Log4J is being used then the Coherence Log4J *logger* level needs to be set to "debug" and the logging limited by the specific Coherence log level in its override file. This is because the Coherence log levels are more fine grained than those in Log4J, for instance Log4j INFO is the same as Coherence log level 4 (INFO), but the Log4J DEBUG level includes Coherence log level 4-9.

Management

A major new feature introduced in Coherence 12c is "Coherence Managed Servers". They allow Coherence applications to be deployed and managed inside Weblogic Server. Each Coherence application is packaged as a Grid Archive (GAR), much like a JEE application, and like JEE applications, Weblogic completely isolates each application using separate class-loaders and uniquely identified cache services. Each Managed Coherence Server is a separate node in a Coherence cluster, with the Weblogic Administration server acting as the Management Node. Below is a diagram that shows a logical view of a Coherence deployment using Coherence Managed Servers.

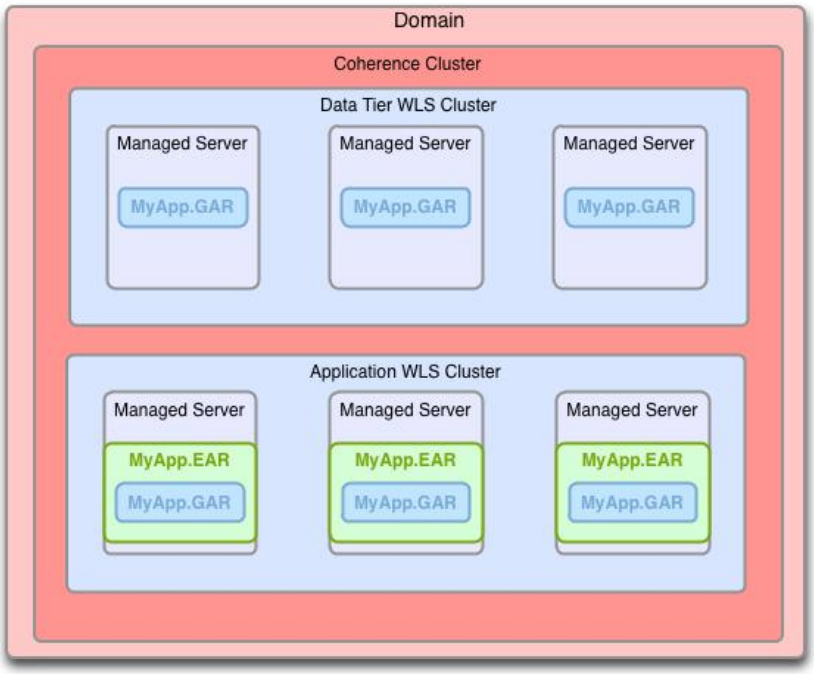

Figure 44. Logical architecture of Coherence Managed Servers in a Weblogic Domain

The overhead from running Coherence inside a Weblogic Managed server is minimal. For instance Weblogic only starts 20-30 pooled threads and only adds ~80MB to the JVM heap, or just 2% of a 4GB heap (Coherence standalone uses ~40MB).

Although optional some of the key benefits of Coherence Managed Servers are:

- A centralized, mature and proven Management Console for controlling the complete lifecycle of Coherence applications.
- A rich scripting framework, WLST, to setup, start-up, deploy and manage Coherence clusters and applications, simplifying operations.
- The ability to clone a Managed Coherence Server
- More intuitive bulk management capabilities. Managed Coherence Servers can be grouped into Weblogic Clusters (as distinct from Coherence clusters) to manage then as a group. Typical groupings could be client applications, proxies or storage nodes. This enables operations to be performed at a group level (or Weblogic Cluster level), like;
  - Deploying or re-deploying a Coherence GAR
  - Making configuration changes, like increasing the high units of a near cache
  - Deploying a shared library

To effectively manage your Coherence clusters its also good practice to give each a unique network address (using a separate multicast address and port or Well Known Address), a meaningful cluster name (like "PROD_APPC_CLUSTER") and explicitly label all the other facets, like role etc. This will make management and monitoring much easier and also prevent a QA or Test cluster inadvertently joining a Production cluster.

Production Testing

Before you start make sure that your environment is setup according to the recommendations outlined in the documentation. This includes;

- Production Checklist (Coherence Administrator's Guide)
- Performance Tuning (Coherence Administrator's Guide)
- Platform Specific Deployment Considerations (Coherence Administrator's Guide)
- Best Practice For Coherence*Extend (Coherence Client Guide)

Also use the bundled tools, like the datagram and multi-cast utilities, to validate that many of the above recommendations have been implemented.

This section will focus on non-functional testing. Functional unit and end-to-end testing will not be covered here.

Soak Testing

It's often overlooked, but running your application tests for a prolonged period of time – hours and even days - will usually highlight problems that short tests won't detect. For instance memory leaks often only manifest themselves after an application has been running for some time. Inadequate processes to handle log files or capacity-planning errors can also show up in soak tests.

Load and Stress Testing

Before deploying a Coherence application into production you should simulate (and if possible exceed) the load and stress that it will need to support. These tests should also include simulated failures in the environment, to ensure there is sufficient spare processing, network and memory capacity to handle these and still meet your target SLA's.

Hardware Failures

Test that in the production environment your hardware provides the desired level of resilience. For instance under load remove a network cable, fill-up a local disk, shut down a server etc. When simulating these failures monitor application correctness, throughput, latency and recovery time (MTR), to ensure you still meet your target SLA's.

Software Failures

As with hardware failure testing, under load try shutting down a Cache Server, the Management or Administration Server (if you are using Managed Servers) etc., to ensure your application meets your target SLA's. Also try disabling connections to any external resources, like a database or even a DNS service.

Recovery

As well as starting your application for the first time you may need to re-start it later, after maintenance or because of an un-planned outage. Ensure that you can recover if necessary from a complete system failure – and in a reasonable time frame – in case you need to.

Resolving Problems

Make sure operational staff are familiar with your application infrastructure, monitoring dashboards and alerts. If issues arise which seem related to Coherence, then a Support Request (SR) ticket should be raised with Oracle. If your operations team is not familiar with this process, it is worth raising a test SR beforehand, to make sure they are ready to do so if needed.

The typical information requested by an engineer looking into a Coherence issue is:

- The Coherence configuration files
- Heap and/or thread dumps. In Coherence 12c one of the bundled WLST scripts for performing these operations can be used – if Coherence Managed Servers are being used. To perform a remote thread dump, another option is to use the new Coherence 12c *Cluster* `logClusterState(roleName)` and *ClusterNode* `logNodeSate(nodeId)` MBean operations.

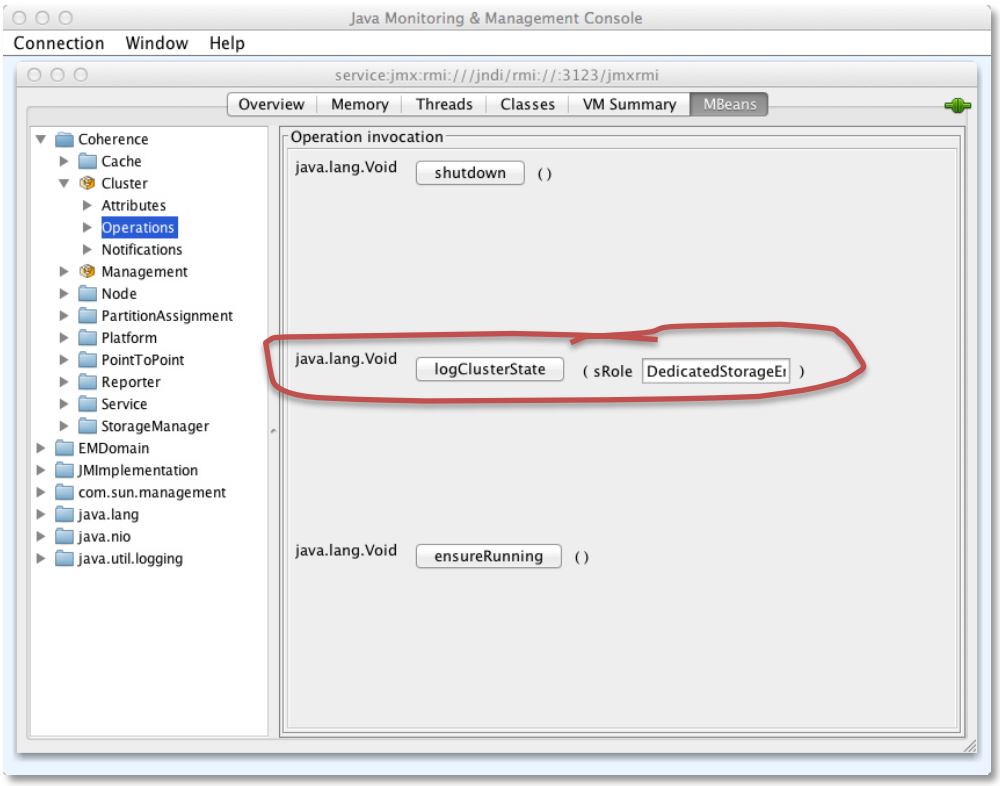

Figure 45. In Coherence 12c a thread dump of cluster nodes can be performed through JMX.

- The Coherence log files. Make sure your application logging goes to a different file from the Coherence logging. This makes them easier to read and share with Oracle Support - because they won't contain any confidential information. Also try not to change the Coherence log file format so that its easier for support engineers to examine.
- JVM startup information and parameters
- JMX Reporter log files, if you are using it. The JMX reporter is a great way to capture historically JMX metrics.
- JVM GC logging information. This can be in the Coherence log files or in a separate GC log file.
- Environmental metrics, like CPU and network statistics. A simple tool called OSWatcher is freely available from Oracle Support that contains standard Operating System scripts to capture this kind of information in log files.

Make sure that you can easily capture this information, so any issues can be promptly investigated.

Conclusion

Following these guidelines, recommendations and suggestions will help you have a more successful Coherence deployment. But read the Coherence documentation too and use this white paper as an additional checklist or as input to your planning process. The formula's outlined should give you a feel for your resource requirements and whether your applications SLA's can be me, but these should be validated through testing. Finally try the new installation and management features in Coherence 12c; they should make the setup and configuration of Coherence a lot easier.

Coherence 12c – Planning a Successful Deployment
July 2013
Author: David Felcey
Contributing Authors: Craig Blitz, Tim Middleton, Jason Howes, Mark Falco, Harvey Raja, Randy Stafford, Jon Purdy and Patrick Peralta

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together