ORACLE
INSURANCE

ORACLE
Coherence

An Oracle White Paper
July 2011

Oracle Insurance Policy Administration for Life and Annuity: Leveraging Oracle Coherence for Distributed Execution of High Volume Transactions

ORACLE

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.

Introduction

This technical white paper discusses how Oracle Insurance Policy Administration for Life and Annuity (OIPA) leverages the powerful capabilities of Oracle Coherence in-memory data grid middleware to provide predictable scalability, performance for high-volume batch cycle processing, and seamless fail-over support. It includes a technical discussion of how the Coherence Processing Pattern provides the grid computing infrastructure used by the Oracle Insurance Policy Administration cycle subsystem.

The first two sections of this paper provide an overview of Oracle Coherence and the Oracle Insurance Policy Administration products, respectively. The remaining sections of the paper describe how OIPA implements the Oracle Incubator Processing Pattern to enable distributed batch processing of work across a grid of interconnected Oracle Coherence powered compute nodes.

Oracle Coherence

Oracle Coherence is an in-memory data grid solution that enables organizations to predictably scale mission-critical applications by providing fast access to frequently used data. Data grid software is middleware that reliably manages data objects in memory across multiple servers. By automatically and dynamically partitioning data, Oracle Coherence ensures continuous data availability and transactional integrity even in the event of a server failure. It provides organizations with a robust scaled-out data abstraction layer that brokers the supply and demand of data between applications and data sources. Oracle Coherence enables the following benefits within today's enterprise applications:

- Performance – Oracle Coherence solves latency problems and drives dramatic increases in performance by moving data closer to applications for efficient access. In-memory performance alleviates bottlenecks and reduces data contention, improving application responsiveness.
- Reliability – Oracle Coherence is built on a fault-tolerant mesh that provides data reliability and accuracy. Organizations can meet data availability demands in mission-critical environments with Oracle Coherence support for data tolerance and continuous operation. The reliability of the data grid minimizes the need for applications to compensate for server and network failures, streamlining the development and deployment process.
- Scalability – Oracle Coherence enables applications to scale linearly and dynamically for predictable cost and improved resource utilization. For many applications, it offers a straightforward approach to increasing the effective capacity of shared data sources. Oracle Coherence handles continually growing application loads without risking data loss or interruption of service.

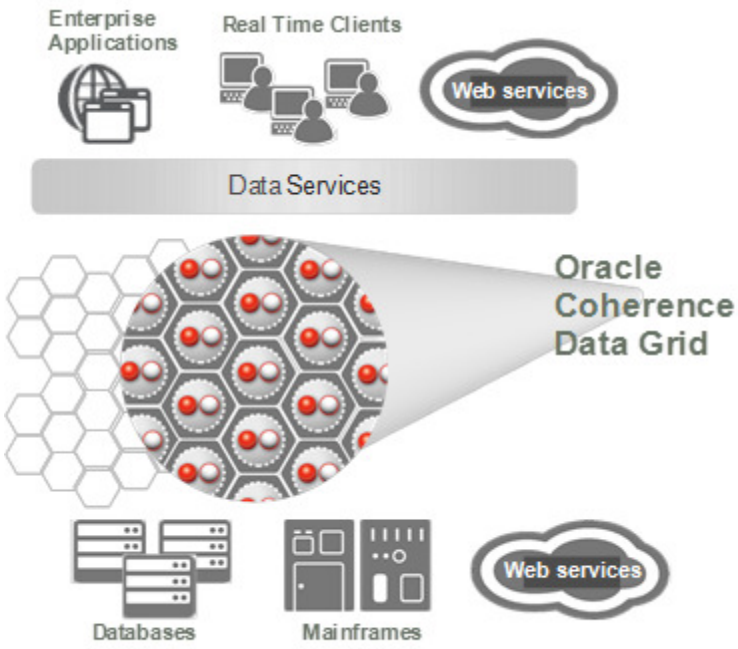

Figure 1. Oracle Coherence Overview

The information stored within the nodes of a data grid is evenly distributed among the cluster members. When applications access the data grid, Oracle Coherence ensures that the data is never more than one network hop away and thus allows for near in-memory latency to all data stored within the grid. This approach enables very large performance improvements in data access. Furthermore, since the data is distributed among all nodes of the data grid and each node does not have to hold all cached data, a data grid can store very large amounts of data (in the order of magnitude of hundreds of gigabytes) without a degradation in performance. In fact, the addition of nodes to a data grid leads to a linear increase in data capacity available to the clients of the grid without any degradation in performance. This latter property is what is known as a linear scalability. Finally, a data grid has the ability to store backup copies of its data objects in separate nodes and thus also provides for high availability.

Clustered Caching

One of the main capabilities of Oracle Coherence, and one that is used heavily by the features discussed within the rest of this document, is clustered caching. Clustered caching refers to the ability to maintain data in the application tier in such a way that the application can fulfill some portion of its data access requirements from the cache. This mitigates the application's load on the database without violating the application's requirements for data correctness if that data is being changed. Oracle Coherence provides two main types of clustered caching that are used by the Oracle Insurance Policy Administration solution: Replicated Caching and Partitioned Caching.

Replicated Caching

The best-known form of coherent clustered caching is the fully replicated cache. Replication is the ability to achieve guaranteed coherency of data across multiple nodes of a cluster by maintaining copies of the data within each node and synchronizing changes. If each server maintains a local copy of cached data, then the application logic running on each server can access local data without the need to communicate with any other servers. As a result, data access has no measurable latency. There are a few limitations to replicated caching. First, a change to the cache implies the need to communicate to the entire cluster. Such communication—often accomplished by use of group network protocols—cannot by its nature scale linearly. Second, the cache is severely limited in its in-memory size, because each cluster node is maintaining the entire cache within its process space.

Partitioned Caching

To solve the limitations of a replicated cache model without sacrificing either the high-availability (HA) benefits of redundancy or the coherency guarantees provided by the clustered cache, Oracle invented the concept of a partitioned cache. With the data partitioned, the cache capacity grows linearly with the size of the cluster, as does the processing capacity available for managing the cache. Further, in a shared-nothing architecture, each piece of data in the cache has exactly one owner within the cluster who is responsible for managing that data. All network communication can be point-to-point in a partitioned model, allowing the cache throughput to scale linearly on a switched network. In order to recover from a possible node failure, a partitioned cache needs to store, at a minimum, one redundant copy of the cache data on a different physical node. The partitioned caching capability of Oracle Coherence provides the foundation for distributed processing in OIPA.

Oracle Insurance Policy Administration for Life and Annuity

Oracle Insurance Policy Administration is a highly-flexible, rules-based policy administration solution that provides full record keeping and support for all policy lifecycle transactions (e.g. policy issue, billing, collections, policy processing, and claims). With OIPA, insurers rapidly adapt to changing business needs and regulatory requirements while supporting straight-through processing throughout the policy lifecycle.

OIPA is used by leading insurers to accelerate product development and speed time to market for differentiated life, health, and annuity products. The system enables insurers to provide real-time policy servicing of customers and sales channels throughout the policy lifecycle for increased retention and loyalty. It also helps insurers reduce risk and support compliance while better managing the business to optimize performance through the use of a single system for life and annuities.

Insurers require the ability to quickly bring to market innovative products that stand out from the competition, capture more market share, and ultimately maximize profitability. OIPA is an industry leading, fully configurable system allowing insurance companies to outpace competitors by getting to market faster. In addition, it enables insurers to drive transactions using business rules. It also provides real-time access to policy data, enhancing self-service capabilities for customers and distribution channels.

Rule-based Configuration

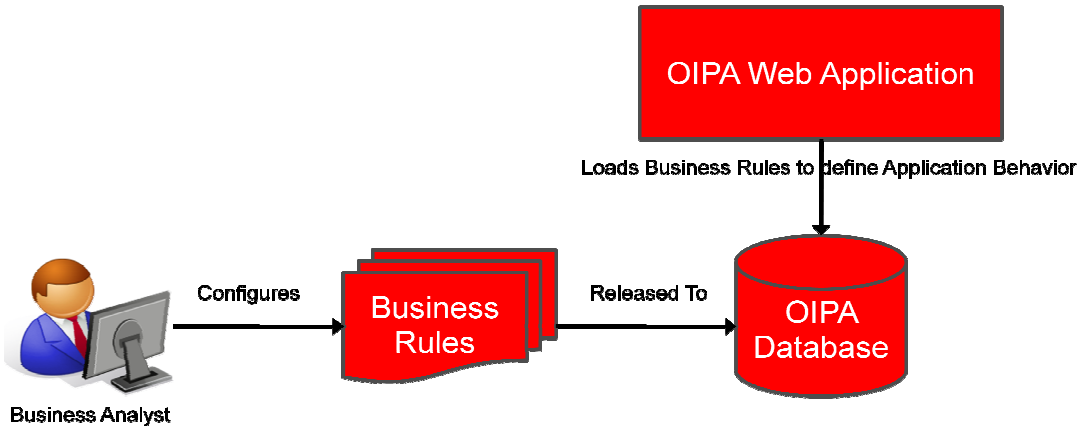

Figure 2. OIPA enables insurers to accelerate the development and launch of life, health, and annuity products and update existing products with new features or riders through flexible, rules-based configuration.

The rules-based architecture of OIPA enables rapid new product development, freeing insurers from the limitations of legacy policy administration systems. The rules-driven capabilities of OIPA are unmatched in the industry; almost all changes to the system—including products, fields, screens, languages, and currencies—can be made without ever touching the core code or recompiling the application. The system does the heavy lifting through pre-configured templates, rules reusability and product cloning, and a visual Rules Palette with easy to use, drag-and-drop functionality.

One of the most important elements configured by business analysts within an OIPA system is the concept of a transaction. Transactions define how the system updates, modifies, or deletes data in OIPA. As an example, a premium transaction takes a payment submitted by an insured and applies that money to a policy. A premium transaction performs a number of actions including validating inputs, removing value from suspense, applying fees and charges, issuing commission, applying the money to one or more funds on the policy, and adjusting the cash value of the policy. OIPA allows insurers to configure the sequence of steps, business rules, and calculations that are performed, as well as what data is updated as a result of the transaction processing.

An instance of a transaction is known as an activity. The execution of an activity involves the execution of all processing associated with its underlying transaction within a single atomic unit of work. The execution of an activity also involves the auditing of all of the changes so that they can easily be reversed or undone.

In most cases, activities that are entered into the system are scheduled for execution at a later time. In a typical OIPA system, Customer Service Representatives (CSRs) may create hundreds or even thousands of pending activities during the day. These pending activities are subsequently executed during a scheduled batch process.

OIPA Cycle Processing

Cycle is the component of an OIPA system that is responsible for the scheduled execution of pending activities. The activities executed by cycle are classified into different levels, where each Cycle Level determines the type of work item to be processed and the sequence of execution:

- Pre-Company – process all pending company level activities that are configured to be executed *before* any policy activities are processed.
- Pre-Plan – process all pending plan level activities that are configured to be executed *before* any policy activities are processed.
- Pre-Client – process all pending client level activities that are configured to be executed *before* any policy activities are processed.
- Policy – process all pending policy activities in the insurance database
- Post-Client – process all pending client level activities that are configured to be executed *after* any policy activities are processed.
- Post-Plan – process all pending plan level activities that are configured to be executed *after* any policy activities are processed.
- Post-Company – process all company level activities that are configured to be executed *after* any policy activities are processed.

The association of levels with activities allows business analysts to define the sequence within which policy level activities execute. An example is the importing of the latest fund unit values into the system so that policy level activities that affect cash value can be executed properly. Fund unit values would

have to be imported before policy level cycle can execute, because many policy level activities require the latest fund unit values in order to process successfully. Other examples are the execution of calculations for downstream reporting or exporting of data into a data warehouse. This should happen after policies are processed, as policy level processing results in the modification of a significant amount of operational data, which should be current before it is imported into a data warehouse.

Challenges of Cycle Processing

The requirements of the OIPA Cycle system lead to inherent time and volume challenges as described in this section. The next section will describe how Oracle Coherence's capabilities enable Cycle to meet these challenges.

Time Constraints

OIPA Cycle processing places a large demand on database and system resources and can slow down the online OIPA application. Cycle is typically scheduled to execute off business hours in order to minimize the impact to the OIPA application users. The increasing globalization of insurance companies expands the OIPA application availability and therefore shortens the time intervals that Cycle has to complete processing.

High Volume Processing

Cycle activities must be able to meet varying processing volume demands with specific time constraints. There are peak periods when the volume of pending activities can be significantly greater than normal. Example peak periods include end-of-month, end-of-quarter, and end-of-year processing when many, if not all, active policies in the system have pending activities to process. During these peak periods when the number of pending activities possibly number hundreds of thousands or even millions, Cycle must still be able to meet time constraints.

Powering Cycle with Oracle Coherence

A consideration of the challenges presented by Cycle requirements makes it clear that the use of a distributed computing infrastructure that allows for the parallel execution of process intensive tasks across a number of processing nodes is needed. Furthermore, the infrastructure would also need to adhere to the following additional requirements:

- Fault Tolerance – The pending tasks from a failed Cycle processing node should fail over to another node and be scheduled for processing.
- Scalability – The sizing requirements for the grid are different based on the customer. While some customers may be relatively small in terms of processing requirements, other customers need the ability to execute many millions of pending insurance activities in parallel.
- Simplicity – The need to achieve distributed computing capability without the system being overly cumbersome and difficult to understand and setup.
- Extensible – The ability to execute custom tasks

- Proven – The underlying architecture must be proven at many existing production installations.

After careful consideration of these requirements, the OIPA team concluded that the best approach in addressing Cycle's demanding needs would be through the use of Oracle Coherence.

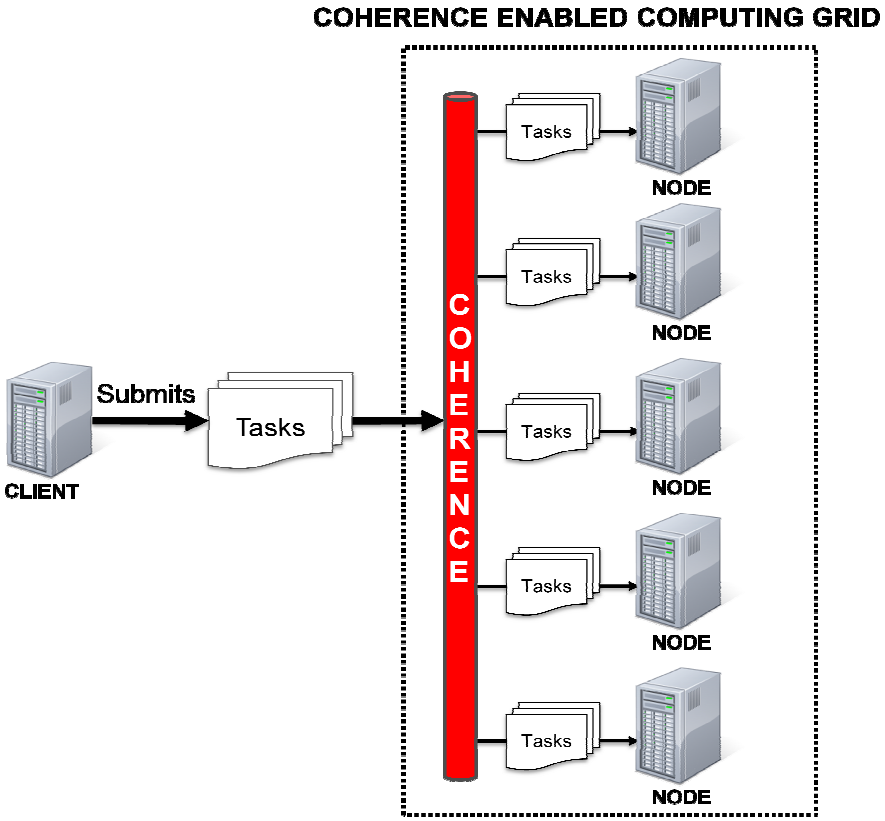

Figure 3. The use of Oracle Coherence allows OIPA to achieve the challenging performance and scalability needs driven by insurers' demand for high-volume, nightly batch processing.

Oracle Coherence provides a repository of projects, known as the Coherence Incubator, that represent best practices for common solutions. The Processing Pattern is a Coherence Incubator project that provides a generalized approach for the submission and asynchronous processing of work within a Coherence cluster. This makes the Processing Pattern a good fit as a foundation for a distributed computing platform. This fit, combined with OIPA's existing and successful use of Oracle Coherence as a distributed cache, provided the basis for choosing the Processing Pattern as the foundation for the implementation for OIPA's distributed computing needs.

Distributed Computing with the Coherence Processing Pattern

The Processing Pattern leverages Oracle Coherence as the connection, communication, and management mechanism to enable distribution and execution of work across a network of computers, also known as a cluster. It allows clients to submit work to the cluster, ensures that the work gets processed, and reports the results of the execution back to the client. The Processing Pattern provides a

framework that allows implementers to define their own types of work to be processed and customizes how work is dispatched and processed.

The Processing Pattern provides a simple interface so that clients do not have to be concerned with how work is distributed and processed. Clients of the Processing Pattern see the cluster as a single process, and are not aware of the number and types of computers that are being used.

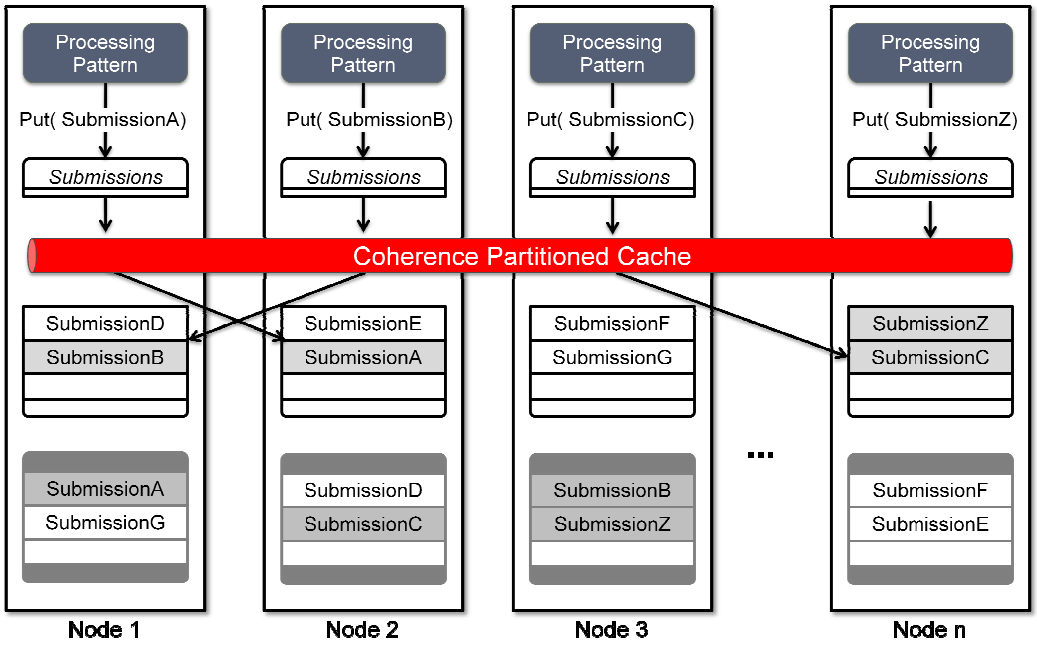

Figure 4. The Processing Pattern distributes work using a submissions cache that is partitioned across the cluster.

The Processing Pattern is built on top of the clustered caching capability of Oracle Coherence. When work is submitted using the Processing Pattern, it is stored in a partitioned cache where Oracle Coherence assigns the storage for the submission to one of the nodes connected to the cluster. When the submission is stored on the node, Oracle Coherence also makes a backup copy of the submission and stores the copy on a different node in the cluster to provide failover capability in the event that one of the nodes fails. When a node fails, the submissions in the backup storage are automatically distributed among the remaining operational nodes in the cluster. This automatic failover feature of Oracle Coherence is used by the Processing Pattern to support re-dispatching of queued tasks in the event of node failure.

Distributing Tasks in the Processing Pattern

There are different methods of distributing work using the Processing Pattern. The method used by OIPA Cycle includes two phases: dispatching and processing. The sequence of steps for this two phase submission process is as follows:

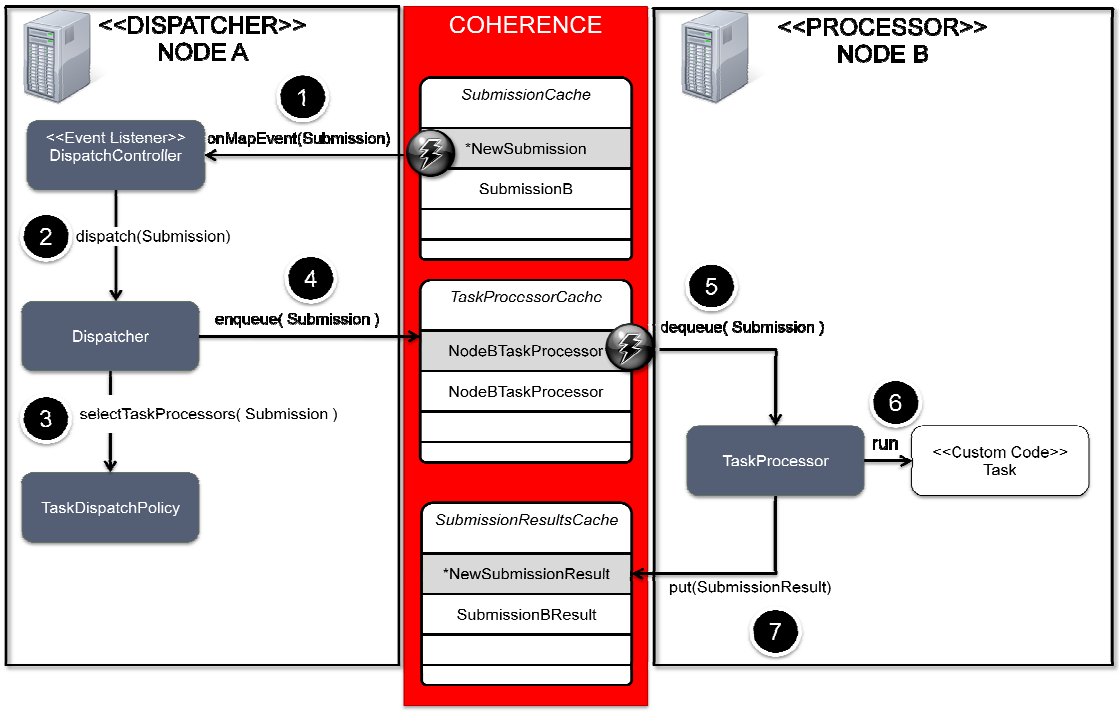

Figure 5. The Processing Pattern distributes work in two phases: dispatching and processing.

- When a task is submitted to the Processing Pattern, it is bundled with a submission and stored in the submission partitioned cache. When the submission arrives at a node, an event is fired notifying registered listeners that a new submission has arrived. The Processing Pattern has a listener (the DispatchController) that is registered to receive new submission events.
- The DispatchController dispatches the new submission through a set of registered Dispatchers. The registered dispatchers form a chain of dispatchers, each one capable of performing some processing on part of the incoming submission.
- The DefaultTaskDispatcher is the most important type of Dispatcher used by OIPA Cycle. It consults one or more TaskDispatchPolicy instances to determine which TaskProcessors are available to process the new submission. A TaskProcessor maps to a node in the cluster that is capable of processing the Submission. One type of TaskDispatchPolicy is the attribute-matching policy, which maps a submission to a TaskProcessor based on attributes on the submission. Another type is the round robin policy, which guarantees uniform dispatching of submissions across all nodes in the cluster. OIPA Cycle uses both of these policies to route submissions to the appropriate TaskProcessors and perform round robin load balancing of work.
- Once the DefaultTaskDispatcher has selected a TaskProcessor, the DefaultTaskDispatcher assigns the submission to the TaskProcessor's work queue using a Coherence partitioned cache.
- The new submission entry is received by a TaskProcessor using another registered listener on the TaskProcessor partitioned cache. The TaskProcessor can be a different node than the node that

dispatched the submission. The TaskProcessor accepts the submission and schedules it for processing by unbundling the task associated with the submission and placing the task in a thread pool.

- At some point, the task is picked up by a thread from the thread pool and processed. The code that is executed is application specific, so the task can do anything that the application requires. As an example, OIPA Cycle has a type of task called the CycleTask, which is responsible for processing all pending activities on a single unit of work such as a policy.
- Once the task is completed, the TaskProcessor stores the result of the task in a SubmissionResult partitioned cache. The result of processing gets transmitted to the originating client through additional event listeners.

Handling Long Running Tasks with Resumable Tasks

A Resumable Task is a special type of task provided by the Processing Pattern that provides additional features of reporting progress, yielding, resuming, and returning a result. These capabilities make Resumable Tasks well-suited for executing long running tasks such as asynchronously executing a Web service or governing a process that takes a long time to complete like an OIPA Cycle Level. That task can then go through a sleep/wake cycle checking on the status of processing until it is complete.

When a Resumable Task is executed, it can store the progress made by saving information using a checkpoint. Thus, if the Task needs to be restarted, it can start from the latest checkpoint rather than starting from the beginning. For clients that want to be informed of the progress of a Task, the progress is reported back by listening for events on that task. It can also return a special object called a Yield that indicates the task is not yet finished and needs to be re-scheduled for execution. The Yield object is used to store the intermediate state of the task, as well as specify a delay before running the task again.

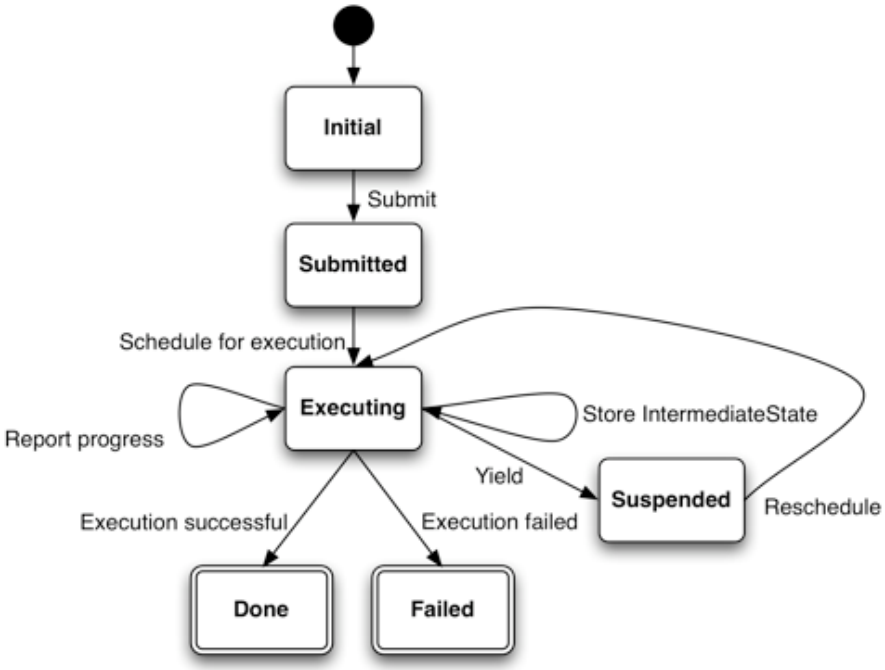

Figure 6. The above diagram shows the states of a Resumable Task.

Scalability and High Availability in the Processing Pattern

The Processing Pattern relies on the scalability and high availability capabilities provided by Oracle Coherence to scale and provide resiliency of failure to the tasks managed by the Processing Pattern. If a node crashes during execution of a task, that task is automatically restarted on another node in the cluster. Similarly, capacity can be added to the cluster without the need for stopping or reconfiguring existing nodes. As soon as a node joins the cluster, the Processing Pattern takes advantage of the additional capacity.

OIPA Cycle Execution using the Coherence Processing Pattern

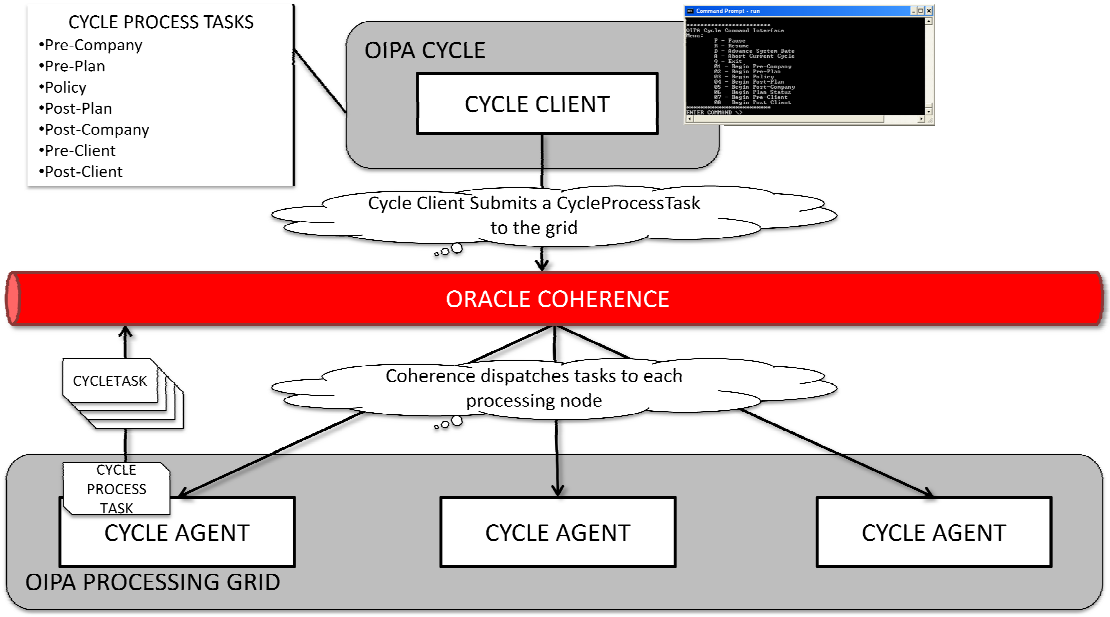

Figure 7. OIPA Cycle executes catch processing using Oracle Coherence and the Processing Pattern

OIPA Cycle is built on top of the Oracle Coherence Processing Pattern to enable the distributed and parallel processing of activities during the batch processing of pending activities in the OIPA system. OIPA Cycle implements the Processing Pattern framework and provides custom tasks that execute the Cycle process. There are two components to Cycle, the Cycle Client and the Cycle Agent. The Cycle Client is a simple console application that requests a Cycle Level be completed by submitting a task to the grid for processing. The Cycle Agent is a processing node that is connected to a cluster of Cycle Agents and provides the distributed computing capability for OIPA and Cycle.

Cycle acts as a workload manager; the actual work of processing activities is done by the OIPA run-time. Each Cycle Agent is bundled with the same OIPA libraries that the OIPA Web Application uses and simply delegates to these libraries to process the pending activities in the OIPA system. In this sense, Cycle provides the plumbing that connects the distributed computing capability of the Coherence Processing Pattern with the activity processing capability of OIPA.

Cycle uses its own queue that exists in the underlying OIPA database in order to direct the execution of a Cycle Level. The Cycle table in the database holds a record for every task that is to be processed during the execution of a single Cycle Level. For example, when running the Policy Cycle Level, each record in the table contains an identifier for a policy that has pending activities that need to be processed. Cycle maintains its own queue instead of using the Processing Pattern's internal queue for a two reasons. First, the number of tasks that need to be executed may number in the millions; submitting millions of tasks at one time would overwhelm the grid. Second, Cycle needs to audit each task individually to support downstream reporting on the nightly batch process. Keeping the queue in the database allows Cycle to distribute manageable chunks of work to the grid for processing and persist the results of each task.

Extending the Processing Pattern to Support Cycle

The Processing Pattern is a purpose-built platform that enables grid computing. It leverages Oracle Coherence as the clustering technology and builds on top of it the dispatching, assignment, scheduling, execution, and management of tasks in the cluster. One of the benefits of using the Processing Pattern is the simplicity of the framework that you need to extend in order to leverage the grid computing infrastructure that it provides. The following diagram illustrates two custom task implementations:

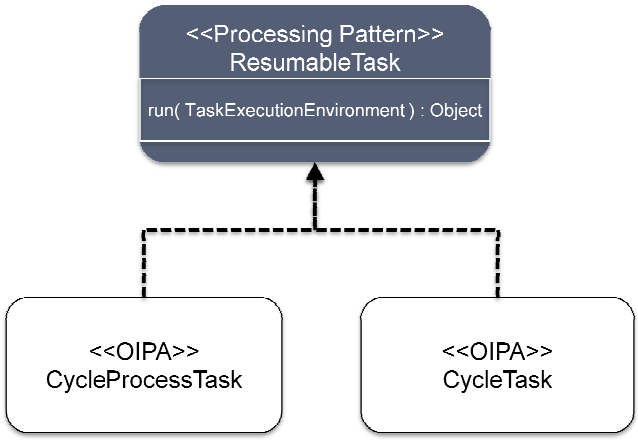

Figure 8. Cycle classes that extend the Processing Pattern are illustrated above.

The only requirement to provide custom task processing for Cycle is the implementation of the ResumableTask interface. The ResumableTask interface has a single **run** method that passes in a TaskEnvironment and returns an Object. The TaskEnvironment object provides access to intermediate state and gives the custom task the ability to checkpoint information and report progress. The return parameter gives the ability for the custom task to return a Yield object, which moves the task into a sleep state and schedule it for future execution.

The CycleProcessTask class is the Cycle task that governs the execution of an entire Cycle Level, such as Policy Level Cycle or Pre-Company Level Cycle. The CycleProcessTask class is discussed below; it is responsible for loading the Cycle queue in the database and replenishing the grid with work until the Cycle Level is complete.

The CycleTask class is responsible for processing all of the pending activities on a unit of work, (for example, a single policy). When a CycleTask object completes processing, it updates a record in the Cycle table with the results.

Using the Processing Pattern to Submit Tasks

The Processing Pattern provides a simple service facade called the ProcessingSession that is used to submit tasks to the grid for processing. The ProcessingSession supports different ways of waiting for the results of a task to complete, including blocking for the task to be finished (synchronous) or receiving event notifications (asynchronous).

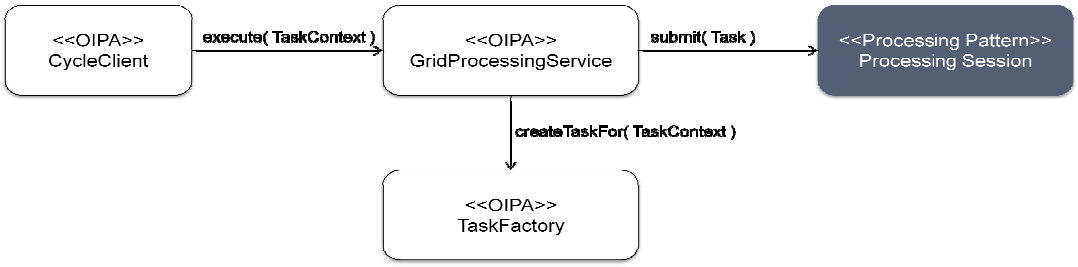

Figure 9. Cycle uses the ProcessingSession provided by the Processing Pattern for Task Submission.

OIPA encapsulates interaction with the Processing Pattern through a service façade called the GridProcessingService. When the Cycle Client submits a task to process, the Cycle Client submits a TaskContext object, which is an OIPA class that contains the definition of the task to process. The GridProcessingService handles the creation of the actual task object using a TaskFactory, and submits the task along with the Submission configuration to the ProcessingSession for processing. The Cycle Client blocks pending the result of the task completing.

Besides task submission, the ProcessingSession provides the following capabilities:

- Check that a task already exists.
- Attach to an existing task and receive its result. This is particularly useful to handle failure scenarios and allows a new client to attach to an existing task that was submitted by a failed client.
- Cancel the execution of an already running task.

Executing a Cycle Level using ResumableTask

The Resumable Task provided by the Processing Pattern gives Cycle the ability to submit and monitor the long running nightly batch process, which could take several hours to complete. When the Cycle Client runs a Cycle Level, it calls the execute method on the GridProcessingService, which blocks the Cycle Client until the Cycle Level is complete. The Processing Pattern creates a Submission for the task and stores it in the Submissions Partitioned Cache where it is delivered by Coherence to one of the Cycle Agents. This section describes the custom processing that is done in the CycleProcessTask object when it is executed in the grid.

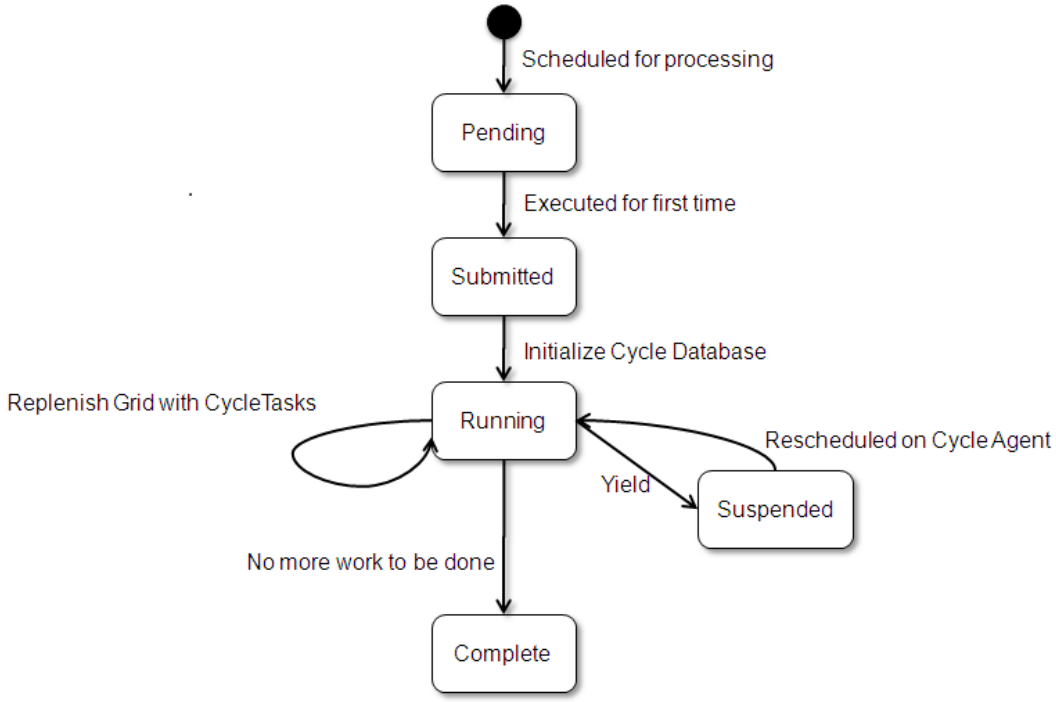

Figure 10. Custom processing that can be completed in CycleProcessTask object when it is executed in the grid.

- The Cycle Client submits a CycleProcessTask object to the grid for processing. The Cycle Client blocks and waits for the CycleProcessTask object to complete. The CycleProcessTask object is initially in a Pending state, awaiting execution.
- The Processing Pattern assigns the CycleProcessTask object to a Cycle Agent node for processing. The first time the CycleProcessTask object is executed, it changes its state to Submitted.
- In the Submitted state, the CycleProcessTask object prepares the cycle process, which calls a stored procedure to populate the Cycle table with data. Each entry in the Cycle table represents a unit of work to be processed (i.e. a Policy, Plan, Company, or Client). Once the cycle level is prepared, the CycleProcessTask object moves into a Running state.
- In the Running state, the CycleProcessTask object replenishes the OIPA Processing Grid with work. It selects a batch of work out of the Cycle table in the database, and submits each work item in the batch to the grid as a CycleTask object. The CycleTask class is another implementation of the ResumableTask interface. The CycleTask object is submitted using the GridProcessingService, which is the same method that the CycleProcessTask object was submitted. Each CycleTask object is dispatched and executed on a Cycle Agent by the Processing Pattern. Once complete, each CycleTask object updates its corresponding record in the Cycle database.
- After replenishing the grid, the cycle goes into a Suspended state by returning a Yield object, and it is rescheduled for processing. The CycleProcessTask object goes through this sleep/wake process, replenishing the processing grid until all of the work items in the Cycle table are complete.

- The CycleProcessTask object moves to a "Complete" state when there is no more work left to do and all work items have completed processing. When the state is updated to Complete, instead of returning a Yield object, the CycleProcessTask object returns the result of the Cycle Level which is further returned to the Cycle Client. The Cycle Client stops blocking, and exits with a normal status. If the CycleProcessTask object failed, the Cycle Client terminates with an error status.

Benchmark Results for Cycle Processing

In January of 2011, Oracle Insurance conducted a benchmark lab to test the performance capability of OIPA Cycle. The benchmark test was on OIPA Version 9.3 running on Oracle Exadata Database Machine X2-2 and Sun SPARC Enterprise M9000 application servers. The test used a representative sample of a full nightly batch cycle for a large-scale system supporting 100 million active policies (split evenly between term life and variable annuities). The system executed the batch cycle in less than two and a half hours.

The findings show that OIPA, through its use of Oracle Coherence as described in this paper, enables insurers to improve scalability, availability, and performance. These improvements allow them to exceed customer service needs while lowering total cost of ownership.

The technology combination of OIPA and Oracle Coherence provides a solution that allows insurers to drastically improve the performance of their batch processing, expand the time that systems are available for processing policies, and deliver quality service to customers. This solution also enables insurers to reduce risk and cost by providing them with an expanded window of time to perform other vital nightly processes such as gain/loss and trade reports necessary for variable annuity processing.

As previously noted, insurers face unprecedented pressure to deliver higher levels of customer service, reduce risk, and improve speed to market. The extreme performance of Oracle Insurance Policy Administration, leveraging Oracle Coherence and running on Oracle Exadata Database Machine X2-2, supports these requirements for even the largest Tier One carriers. It enables faster batch runs that provide an extended window for system availability to service customers and conduct essential transactions, while reducing server costs and opening up new opportunities for hardware consolidation.

Conclusion

Insurers realize that they can no longer continue to rely on aging and inflexible policy administration systems that lock them into legacy business practices and impede their ability to sustain, scale, and fuel growth. Modernizing their critical core platform with an adaptive policy administration system—one with powerful transaction processing capability that allows for an optimized use of their existing computing resources—helps them achieve this objective. Insurers who leverage the powerful processing capabilities of Oracle Insurance Policy Administration powered by Oracle Coherence are well positioned to support their evolving needs for scalability, performance and long-term growth for high-volume transaction, and batch cycle processing.

About Oracle Insurance

Oracle believes that insurers should be able to leverage technology to help transform their business. Oracle Insurance provides adaptive, rules-drive systems that enable insurance companies to change business processes as their business needs change. This positions insurers to readily respond to dynamic market conditions and take advantage of new opportunities as they arise.

For more information about Oracle Insurance, please visit www.oracle.com/insurance, contact us by email at insurance_ww.oracle.com or call 1.800.735.6620 to speak to an Oracle Insurance representative.

For more information about Oracle Coherence, please visit http://www.oracle.com/us/products/middleware/coherence/index.html.

Oracle Insurance Policy Administration for Life and Annuity: Leveraging Oracle Coherence for Distributed Execution of High Volume Transactions
July 2011
Author: Paul Cleary
Contributing Authors: Reza Shafii, Christer Fahlgren

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