Oracle Coherence: In-Memory-Computing für Einsteiger

Gabriele Jäger und Michael Fuhr, ORACLE Deutschland B.V. & Co. KG

Trotz Einsatz aktueller Hard- und Software-Komponenten, optimaler Datenstrukturen und Algorithmen, Tuning-Maßnahmen etc. zeigt eine Anwendung nach einiger Zeit nicht mehr die erwartete Performance. Häufig liegen dann die Ursachen in mangelnder Kapazität und zu hoher Latenzzeit.

Die speicherresidente Data-Grid-Lösung „Coherence" schafft Abhilfe bei Performance-Problemen und ermöglicht eine vorhersagbare Skalierung von Anwendungen für einen schnellen und zuverlässigen Zugriff auf häufig benötigte Daten. Hinzu kommen Datenanalysen in Echtzeit, Berechnungen im In-Memory Grid sowie eine parallele Transaktions- und Ereignisverarbeitung. Der Artikel zeigt neben den Grundprinzipien der Oracle In-Memory Data-Grid-Lösung den Mehrwert durch Anwendungsbeispiele wie RDBMS- und Mainframe-Offload, Anwendungen mit Zustands-Informationen in Web-Anwendungen, Http-Session-Management und Objekt-Interoperabilität (.NET, Java) auf.

Das In-Memory Data-Grid

Eine Anwendung wird entwickelt, getestet, produktiv geschaltet und läuft − die Anwender sind zufrieden. Aber über die Zeit wird die Anwendung langsamer und die Anwender beschweren sich über zu lange Antwortzeiten. In der Regel werden dann erst einmal ein Upgrade auf aktuellste Software-Komponenten, der Einsatz der aktuellsten Funktionalitäten, optimaler Datenstrukturen und Algorithmen, die Aktualisierung der Hardware und natürlich Tuning-Maßnahmen empfohlen. Diese Maßnahmen helfen meist nur kurzfristig, langfristig jedoch verschlechtert sich die Performance dieser Anwendung immer weiter. Es stellt sich die Frage nach dem wirklichen Problem.

Cameron Purdy, ehemaliger Gründer und CEO der Tangosol Inc., hat sich bereits im Jahr 2000 mit diesem Thema beschäftigt und mit Kollegen eine sehr erfolgreiche und gute Software-Lösung – das heutige Coherence – für diese Aufgabenstellung auf den Markt gebracht. Sieben Jahre nach Firmengründung wurde Tangosol von Oracle aufgekauft. Cameron Purdy ist mittlerweile Senior Vice President of Development der Oracle Corporation und verantwortet den Bereich „Cloud Application Foundation" mit dem WebLogic Server, Coherence und vielem mehr. Er beantwortet die Frage wie folgt: „Trust me, you have a capacity problem, not a performance problem." Wir haben also kein Performance-, sondern ein Kapazitätsproblem − ähnlich wie bei einer Warteschlange an einem Schalter. Der erste Kunde der Warteschlange fühlt eine gute Performance, da er sofort bedient wird, wohingegen der letzte Kunde der Warteschlange eine schlechte Performance erfährt, denn er muss warten, bis alle vor ihm bedient worden sind.

Eine kürzere Wartezeit lässt sich in diesem Fall durch den zeitgleichen parallelen Betrieb mehrerer Schalter erreichen. Diese sogenannte „horizontale Skalierung" ist auch die Lösung unseres Problems. Die Gerätekosten sind dank Commodity-Hardware abschätzbar, die Relation „Kosten zu Performance" ist nahezu linear und die physikalischen Limitierungen sind weitaus höher als bei einer vertikalen Skalierung.

Die System-Architektur einer horizontalen Skalierung benötigt etwas Aufmerksamkeit und sollte im Vorfeld betrachtet werden. Damit kommen wir zu unserer ersten Frage zurück: „Was ist ein In-Memory Data-Grid?" Im Internet gibt es mittlerweile viele Definitionen des Begriffs. So gibt der Java-Community-Weblog-Service eine Definition (siehe Abbildung 1).

Ein Data-Grid ist demnach ein System von mehreren Servern (Knoten), das Informationen in Form von Anwendungsobjekten auf diesen Knoten verwaltet und Operationen (wie Berechnungen) mit diesen Informationen ausführt. Ein In-Memory Data-Grid ist ein Data-Grid, das Informationen In-Memory speichert, um eine sehr hohe Performance zu erreichen.

Die Informationen werden in Anwendungsobjekten (Domain Objects) gehalten.

Ein In-Memory Data-Grid ist ein **[In-Memory] Datenmanagment-System zum Verwalten von Anwendungsobjekten**, welche **verteilt** genutzt werden können und mit denen Operationen durchgeführt werden können.

Es zeichnet sich aus durch
- eine geringe **Latenz** bei Zugriffen
- einen hohen **Durchsatz**
- vorhersehbare **Skalierbarkeit**
- hohe **Verfügbarkeit** und
- ein **robustes** Verhalten.

Abbildung 1: Definition eines In-Memory Data-Grid

Kopien der Objekte sind redundant auf verschiedenen Knoten des Systems verteilt. Diese Eigenschaft bildet die Grundlage für die Elastizität des Systems und die Hochverfügbarkeit der Information im Falle eines Knotenausfalls. Die Struktur der Anwendungsobjekte wird mittels objektorientierter Sprachen wie Java (POJOs), C++ oder C# definiert. Es existieren Schnittstellen (APIs) auf Basis dieser Sprachen, um „Create"-, „Read"-, „Update"- und „Delete"-Operationen (CRUD), Abfragen und Berechnungen durchzuführen.

Ein In-Memory Data-Grid speichert jegliche Informationen im Hauptspeicher und führt dort auch alle Operationen aus. Dies ist die Grundlage für hoch performantes, verteiltes Rechnen, da der Zugriff auf persistent gehaltene Strukturen vermieden wird (keine Objekterzeugung aus relationalen Strukturen, kein File-I/O etc.). Anwendungsobjekte sind auf den verschiedenen Knoten des Systems gemeinsam nutzbar. Andere Systeme (wie Middleware-Anwendungen) können auf diese Objekte transparent zugreifen – das auf den In-Memory Data-Grid zugreifende System muss also nicht wissen, auf welchem Knoten sich die Objekte befinden. Die Objekte können Informationen aus anderen Systemen (RDBMS, Filesystem etc.) widerspiegeln und, falls notwendig, in diese Systeme geschrieben oder von dort gelesen werden. Das Schreiben kann sowohl synchron als auch asynchron erfolgen.

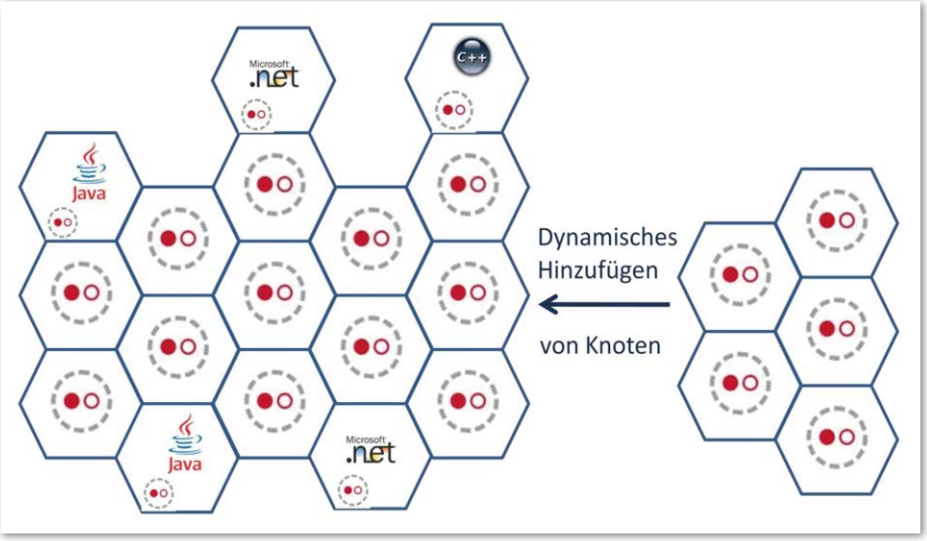

Abbildung 2: Verschiedene Coherence-Knoten formen ein Cluster, die Client-Knoten (Java, .NET, C++) sind selbst Teil des Clusters

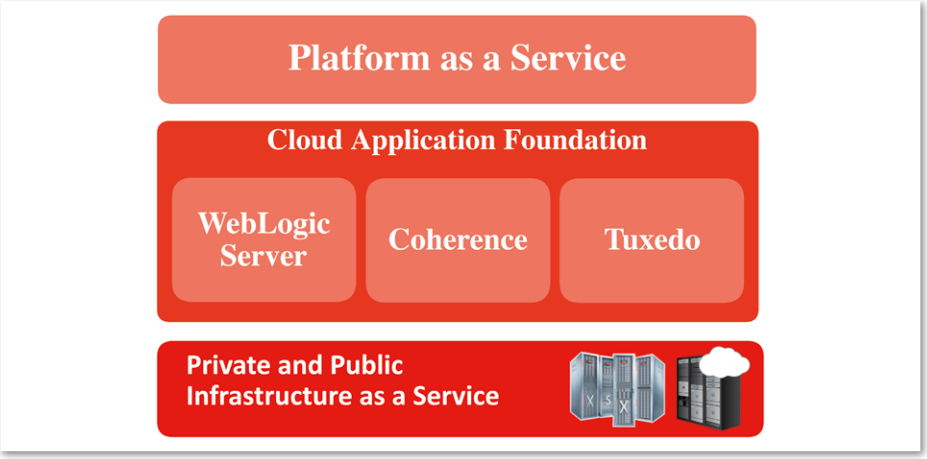

Abbildung 3: Coherence als Bestandteil der Oracle Cloud Application Foundation

Oracle Coherence

Ein Coherence-Knoten ist nichts anderes als ein Java-Prozess (JVM-Prozess), der mit den geforderten Coherence-Java-Bibliotheken und gewissen Coherence-Konfigurations-Informationen gestartet wurde. Lädt man Coherence vom Oracle Technology Network (OTN) herunter, erhält man ein 86 MB großes „.zip"-File, das im Wesentlichen eine Handvoll Java-Bibliotheken enthält. Diese haben keine Abhängigkeiten zu Drittpartei-Bibliotheken; es wird nur eine JDK-Installation vorausgesetzt.

Ein Coherence Cluster ist eine Menge von solchen Java-Prozessen (auf einem Rechner beziehungsweise verschiedenen physikalischen Rechnern) mit ähnlicher Konfiguration, die miteinander kommunizieren. Ein Client, geschrieben in Java, C++ oder auch .NET, kann sich mit dem Cluster verbinden, um per API gewünschte CRUD-Operationen, Abfragen und Berechnungen auf den Storage-Knoten (Knoten, auf denen sich die Anwendungsobjekte befinden) ausführen zu lassen (siehe Abbildung 2).

Ist zur Laufzeit zusätzliche Kapazität notwendig, werden einfach neue Storage-Knoten gestartet und dem Cluster hinzugefügt. Nach Spitzenzeiten können Knoten den Cluster auch wieder verlassen. In beiden Fällen sind die Anwendungsobjekte automatisch im Cluster rebalanciert – sie werden also auf den im Cluster befindlichen Knoten gleichmäßig neu aufgeteilt. Coherence sieht verschiedene Cache-Topologien vor, so zum Beispiel Local Cache, replizierter Cache, partitionierter Cache und Near Cache.

Es können verschiedene Backing-Map-Schemata zum eigentlichen Ablegen der Objekte im Cache zum Einsatz kommen. Je nach Anwendungsfall wird das gewünschte Szenario flexibel umgesetzt. Die Konfiguration erfolgt durch XML und/oder Java-System-Eigenschaften. Es gilt das Prinzip „Configuration by Default"; nur Verhalten, das vom Default abweicht, muss per Konfiguration überschrieben werden. Für das Anlegen und Manipulieren der XML-Konfigurationsdateien besteht grafische Unterstützung im Oracle Enterprise Pack for Eclipse ab Version 11.1.1.6.

Zudem kann Coherence out of the box für das Management von Http-Session-Zuständen in Application Servern mittels Coherence*Web oder für das Ersetzen der Standard-Caching-Mechanismen in verschiedenen Persistenz-Frameworks (Hibernate L2 Cache, EclipseLink JPA Shared Cache) genutzt werden. Der Zugriff auf einen Coherence Cache kann per API mit

Java, .NET oder auch C++ erfolgen. Das API umfasst unter anderem:

- Zugriffssteuerung auf Cache
- Unterstützung von speziellen, sehr effizienten Serialisierungsmechanismen für die Domain-Objekte
- Abfragen durch Filter und Value Extractors
- Erstellen von Indizes
- Benutzen von Aggregatoren
- Benutzen von Entry Processors zur effizienten parallelen Verarbeitung im Cache-Verbund
- Definition von Agenten und Benutzen des WorkManager-API
- Definition und Abgreifen von Data-Grid-Events
- Implementieren von CacheLoader und CacheStore

Oracle Coherence lässt sich mittels JMX administrieren und überwachen. Es gibt eine Integration in den Oracle Enterprise Manager, zudem stehen Monitoring-Werkzeuge anderer Hersteller zur Verfügung. Coherence bietet ferner eine komplette Integration im WebLogic- und GlassFish-Server, falls Coherence in Verbindung mit einem dieser Java-Applikationsserver laufen soll.

Coherence setzt auf eine komplett geclusterte Architektur auf. Analog zu einem Telefonkonferenz-Modell ist Coherence Consensus ein Vertrag zwischen einer Menge von Prozessen über die Mitgliedschaft innerhalb des Verbunds zu einer bestimmten Zeit. Dies ermöglicht transparentes, dynamisches und automatisches Failover/Failback von Services und Daten, zuverlässiges Partitionieren von Daten und Services sowie In-Memory-Funktionalitäten.

Die Software benutzt das Tangosol Coherence Management Protocol (TCMP), ein spezielles proprietäres Peer-to-Peer Unicast-basiertes Protokoll. Es arbeitet asynchron, die Kommunikation ist also nie blockiert, auch nicht wenn viele Threads eines Servers gleichzeitig kommunizieren, und gewährleistet damit eine echte Skalierbarkeit. Darüber hinaus bedeutet die Asynchronität auch, dass die Netzwerk-Latenz den Cluster Throughput nicht beeinträchtigt, obwohl sie die Geschwindigkeit von bestimmten Operationen beeinflussen kann. TCMP ist ein geclustertes, IP-basiertes Protokoll für Server Discovery, Cluster Management, Service Provisioning sowie Datenübertragung und kann optional Multicast unterstützen.

Coherence wird vor allem eingesetzt als Puffer für stark frequentierte Daten und zur Reduzierung der Latenz von Backend-Systemen, dadurch werden In-Memory Datenabfragen fünf bis zwanzig Mal schneller beantwortet als aus dem Backend. Es ist einer der wichtigsten Bestandteile der Oracle Cloud Foundation, einem Platform-as-a-Service-Angebot von Oracle, kann aber auch standalone betrieben werden (siehe Abbildung 3).

Grid Computing kombiniert Server-Virtualisierung und Clustering über den gesamten Stack und liefert damit eine dynamische, verteilte Infrastruktur als Basis für Cloud Computing. Cloud Computing teilt damit viele Eigenschaften und technologische Anforderungen mit Grid Computing.

Abbildung 4: RDBMS-Offload

Abbildung 5: Mainframe-Offload

Abbildung 6: Zustands-Informationen in Web-Anwendungen

Der Oracle-Schwerpunkt bei Grid-Computing-Funktionalitäten, wie dynamisches Resource Provisioning, dynamisches Resource Scheduling und hochautomatisiertes Management von Clustern und virtuellen Maschinen, passt exakt zu den Anforderungen aus dem Cloud Computing. Nachfolgend wird auf einige am häufigsten anzutreffende Coherence-Anwendungsfälle und ihren Nutzen eingegangen.

RDBMS-Offload

Wiederholtes Lesen von sich nicht verändernden Daten (wie Preislisten, Fahrplänen etc.) aus Datenbanken ist ineffizient, da massive Redundanz durch die Datenbank-Zugriffe auf dieselben Inhalte erzeugt wird und es somit zu korrespondierenden Objekten in allen JVMs führt, in denen Abfragen gestartet wurden *(siehe Abbildung 4)*.

Ebenso latenzträchtig ist die Zwischenspeicherung von großvolumigen, transient verwendeten Daten mit gewisser Lebensdauer in der Datenbank, bei der sich die Datenbank-Zugriffe als Flaschenhals erweisen. Auch die Bereitstellung transienter Daten (Daten und Berechnungen) über die Anwendungsgrenzen hinaus durch Zwischenspeicherung in der Datenbank führt in der Regel zu Engpässen.

Diese Engpässe werden durch einmaliges, hochfrequentes Laden der sich nicht verändernden Daten im Coherence Grid vermieden. Die Verwaltung transienter Daten findet nur im In-Memory Data-Grid statt. Eine Policy steuert die Lebensdauer von Objekten. Wenn diese Zeitlimits erreicht sind, werden die entsprechenden Objekte aus Coherence entfernt und entweder eliminiert oder in den Langzeitspeicher für eventuell später noch durchgeführte Langzeit-Aktionen verlagert. Coherence ermöglicht demzufolge die schnelle Daten-Analyse auf Basis transienter Daten. Der Einsatz von Coherence ermöglicht Einsparpotenziale durch Ressourcen-Schonung, deutlich höhere Performance und mehr Skalierbarkeit.

Mainframe-Offload

Der Zugriff und Operationen auf Mainframes (MOPS, million operations per second) sind teuer. Wiederholtes Lesen von sich nicht verändernden Daten (wie Kundenstammdaten) in Anwendungen, die Nutzung von transienten Daten über Anwendungsgrenzen hinaus (Daten und Berechnungen) und die Zwischenspeicherung von großvolumigen, transient verwendeten Daten mit gewisser Lebensdauer im Mainframe erzeugen viele MOPS und somit Kosten. Zudem dauert die Ergebnisbereitstellung bei einem synchronen Zugriff auf einen Mainframe zu lange *(siehe Abbildung 5)*.

Die Lösung dieser Latenzprobleme besteht im Herauslesen von benötigten Daten aus dem Mainframe und aus der einmaligen Ablage in Coherence: Die Kalkulation findet nun im Java-Adressraum statt, anschließend kann das Ergebnis zurückgeschrieben werden. Der Zugriff auf bestimmte Anwendungsobjekte erfolgt über den Coherence Grid. Zugriffszeiten lassen sich somit optimieren. Die Skalierbarkeit erfolgt durch Hinzunahme von kostengünstiger Commodity-Hardware. Somit können transiente Daten in Coherence über Anwendungsgrenzen hinaus bereitgestellt werden. Diese Ressourcen-Schonung führt zu deutlichen Mainframe-Kosteneinsparungen und schneller Ergebnisbereitstellung bei synchronen Zugriffen.

Zustands-Informationen in Web-Anwendungen

Bei der „Zustandslose"-Programmierung ist der Zustand innerhalb von Request-/Response-Paaren in der Datenbank gespeichert. Der Zustand muss also immer wieder geschrieben und geladen werden. Bei Web-Anwendungen mit Megabyte-großen Zustands-Informationen oder großem Volu-

men statischer Information (wie Benutzerprofile) stellt dann das Speichern in der Datenbank einen Engpass dar. Darüber hinaus stellt sich die Frage, was mit der Anwendung geschieht, wenn die Datenbank nicht verfügbar ist. In solch einem Fall ist die Anwendung dann auch nicht verfügbar *(siehe Abbildung 6)*.

Die Daten und der Application Server sind in der JVM kollokiert. Dies bedeutet, dass viele beziehungsweise große Anwendungsdaten zu Garbage-Collection-Herausforderungen in JVMs führen, was Performance-Einbußen durch große Heaps zur Folge hat.

Betrachtet man das Rolling Upgrade von Produktionsservern (Infrastruktursoftware, Anwendungssoftware), also die Frage, was mit Daten geschieht, die nur In-Memory gehalten und von Anwendungen auf Application Servern verwendet werden (wie Benutzerprofile, die kollokiert mit der entsprechenden Anwendung im Application Server in einer JVM vorliegen). Auch hier gilt, dass das Herunterfahren eines oder mehrerer Application Server nicht ohne Weiteres möglich ist, da Daten quasi verloren gehen können.

Die Lösung für die beschriebenen Probleme besteht in der Einführung einer separaten Datenschicht durch ein Coherence Cluster. Cache Loader und Cache Store werden zum Laden und Speichern von zu persistenzierenden Zustands-Informationen (zum Beispiel Benutzerprofile) eingesetzt. Damit können Lastspitzen ausgeglichen, Service-Level-Agreements wieder eingehalten, die Applikationsschicht-Kapazität erhöht und Backend-Last stark reduziert werden. Dieser Anwendungsfall erzielt neben einer Ressourcen-Schonung auch deutlich bessere Ergebnisse im Hinblick auf Skalierbarkeit, Verfügbarkeit und Zuverlässigkeit.

Http-Session Management

Werden Http-Session-Zustände von verschiedenen Web-Anwendungen gleichzeitig genutzt, dann führen viele beziehungsweise große Http-Sessions zu Garbage-Collection-Herausforderungen. Für die Ablage dieser Information in einer Datenbank gilt analog zum vorhergehenden Beispiel: Ist die Datenbank nicht verfügbar, dann ist die Anwendung auch nicht verfügbar. Zudem wird die Datenbank bei Skalierbarkeitsproblemen schnell zum Flaschenhals und vorhandene In-Memory-Replikationsmechanismen von Anwendungsservern sind zum Teil zu ineffizient. Auch das Rolling Upgrade der Produktionsserver

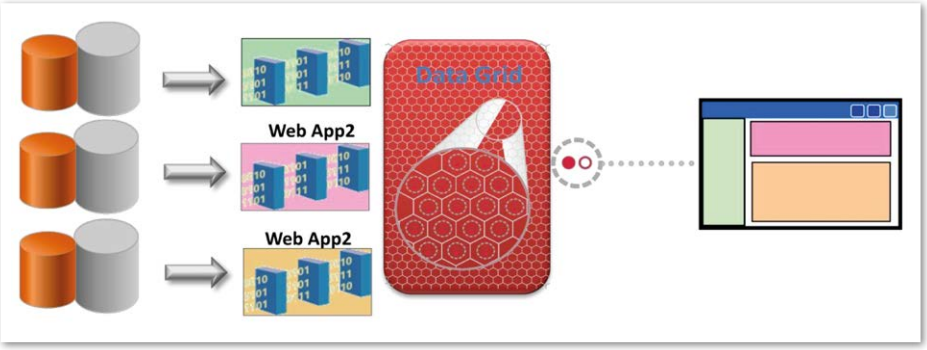

Abbildung 7: Http-Session-Management (unter anderem über Anwendungsgrenzen hinaus)

Abbildung 8: Objektoperabilität (Java, C++, .Net)

gestaltet sich schwierig, wie bereits zuvor beschrieben *(siehe Abbildung 7)*.

Eine separate Datenschicht durch Coherence Cluster mit Coherence*Web als Out-of-the-Box-Lösung für verschiedene Anwendungsserver löst hierbei den Engpass. Coherence*Web ist ein Http-Session-Management-Modul für das Managen/Verwalten des Session State beziehungsweise des Http-Session-Zustands in geclusterten Umgebungen. Coherence*Web unterstützt den WebLogic Server, GlassFish und alle andere Mainstream Application Server.

Bei Performance-Problemen ist es die ideale Lösung für sehr umfangreiche, webbasierte Umgebungen mit einer Composite Application (zusammengesetzte Anwendung), die von unterschiedlichen darunterliegenden Inhalten abhängt. Diese Lösung führt zu einer massiven Verbesserung der Gesamt-Performance und Skalierbarkeit der Infrastruktur, zu höherer Site-Performance durch Session Sharing und zu guter Benutzererfahrung durch stabile Sessions.

Objekt-Operabilität

In diesem Anwendungsfall geht es um die gemeinsame Benutzung von Objekten aus verschiedenen Programmiersprachen heraus. Coherence agiert hierbei als eine Art „Objektmediator" und ersetzt an dieser Stelle langsamere, Webservice-basierte Serialisierungs-/Deserialisierungs- und Standardserialisierungs-Mechanismen. Dieser Ansatz erzielt sehr hohe Verfügbarkeit und bietet eine schnelle, zuverlässige und hochperformante Bereitstellung von High-End-Produktions-Umgebungen *(siehe Abbildung 8)*.

Gründe für Oracle Coherence

Mit Coherence können Unternehmen die Skalierung erfolgskritischer Anwendungen voraussehen, sodass ein schneller Zugriff auf häufig benötigte Daten gegeben ist. Anwendungen lassen sich mithilfe von Coherence linear und dynamisch skalieren, was eine bessere Vorhersehbarkeit der Kosten und eine bessere Ressourcen-Nutzung ermöglicht.

Zudem ist eine signifikante Steigerung der Transaktionsraten bei Verkürzung der Antwortzeiten von Anwendungen zu verzeichnen. Coherence sorgt selbst bei einem Serverausfall für eine kontinuierliche Datenverfügbarkeit und Transaktionsintegrität. Organisationen kommen ebenfalls die extremen Transaction-Processing-Fähigkeiten zugute: Durch die Verteilung der Daten, kombiniert mit der Leistungsstärke des Grid Com-

puting, ergibt sich eine schnelle und zuverlässige High-End-Transaktionsverarbeitung.

Fazit

Zusammenfassend lässt sich sagen:

- Coherence ist eine elegante und praktikable Lösung für Caching
- Coherence ist keine klassische persistierende Datenbank, sondern ein Zwischenspeicher (Cache)
- Es gibt umfangreiche Möglichkeiten, an Coherence anzudocken
- Coherence entlastet das Backend und beschleunigt die Zugriffe
- Dies führt zu weniger Wartezeit durch höhere Kapazitäten

Gabriele Jäger
gabriele.jaeger@oracle.com

Michael Fuhr
michael.fuhr@oracle.com

OWB – Wie finde ich den richtigen Nachfolger?

Michael Klose, CGI Deutschland Ltd. & Co. KG

Der Warehouse Builder wird von Oracle nicht weiterentwickelt und der Standard Support endet in naher Zukunft. Gerade Kunden, die die Datenbank-Version 12*c* nicht installieren wollen, sind gezwungen, sich nach Alternativen umzusehen.

Oracle stellt mit dem Data Integrator zwar einen würdigen Nachfolger bereit; dieser ist allerdings mit zusätzlichen Lizenzkosten verbunden. Der Artikel zeigt verschiedene Möglichkeiten des Umstiegs auf ein anderes ETL-Tool auf. Sie werden anhand von Praxiserfahrungen aus einer ETL-Toolauswahl dargestellt. Darüber hinaus werden die Vor- und Nachteile der verschiedenen Tools sowie grundsätzliche Veränderungen in der Entwicklungstätigkeit und Architektur aufgezeigt. Die Schwerpunkte liegen bei Oracle Data Integrator, Informatica Powercenter, Talend, PL/SQL und einer Empfehlung für die Vorgehensweise bei der Tool-Auswahl.

Grundsätzliches

Bei den ETL-Tools gibt es durch die große Anzahl von Entwicklungswerkzeugen gerade aus Architektur-Sicht unterschiedliche Ansätze der Hersteller. Grundsätzlich unterscheidet man zwischen ETL (Extract – Transform – Load) und ELT (Extract – Load – Transform). Beim ETL-Prozess werden die Daten aus den Quellsystemen beispielsweise als Flat Files extrahiert, im Anschluss in einer ETL-Engine transformiert, um letztendlich final in die Zieldatenbank (Data Warehouse) geladen zu werden. Im Gegensatz dazu laden ELT-basierte Werkzeuge die Daten zuerst in die Zieldatenbank, um dort dann die Transformationen durchzuführen. Ein weiterer Unterschied liegt in der Programmiersprache des generierten Codes der Transformationsprozesse.

OWB – Stärken und Schwächen

Gerade für die in der Oracle-Programmierung kompetenten Entwickler stellt die Generierung von PL/SQL-Code eine klare Stärke dar. Der Code ist komplett nachvollziehbar und für die jeweilige Datenbank-Version optimiert. Nahezu alle Datenbank-Features können verwendet werden und die Ausführung erfolgt direkt auf der Datenbank (Push-Down). Die hohe Integration in die Oracle-Produktpalette und die einfache Installation innerhalb der Datenbank bedingen keine zusätzlichen Backup-Aufwände. Das vollständige, teilweise sogar Produkt-übergreifende Metadaten-Repository (OBI) ermöglicht detaillierte Data-Lineage-Analysen und unterstützt das Deployment auf andere Umgebungen. Der Funktionsumfang im ETL-Bereich umfasst alle wichtigen Komponenten wie Mappings, Workflows, Fehlerprotokollierung etc. Eine nicht zu vernachlässigende Stärke ist auch, dass der Oracle Warehouse