In veel bedrijfstakken is performance van steeds groter belang, omdat grotere datavolumes of meer transacties verwerkt moeten worden. Denk aan applicaties voor online gaming of voor het afhandelen van financiële transacties. Door het toenemende commerciële gebruik van het internet komen er steeds meer business cases met hoge performance eisen.

Oracle Coherence

High performance data grid computing in Java

Voor dit soort business cases worden meestal grootschalige data grid omgevingen opgetuigd. Oracle Coherence komt met een robuust en uiterst schaalbare in-memory data grid oplossing in Java om performance eisen te halen in een gedistribueerde omgeving waarin in korte tijd veel transacties en datavolumes moeten worden verwerkt.
Oracle Coherence positioneert zichzelf in een applicatielandschap tussen de datagebruikers en databeheerders om sneller en efficiënter data te lezen en te bewerken in een data grid.
De volgende lijst bevat typische gebieden waarin Oracle Coherence kan worden ingezet indien een bestaande implementatie niet de vereiste performance haalt en moeilijk uitschaalbaar is:

- Caching: om backend systemen te ontlasten en performance te verhogen, werken Oracle Coherence applicaties met data uit een in-memory cache in plaats van direct uit de database.
- Data analysis: Oracle Coherence ondersteunt parallelle query executie om interessante informatie uit het data grid te halen op basis van de data die het beheert.
- Transaction processing: de data grid dient als een in-memory transaction-engine, waarin zowel data als business logica opgeslagen is.
- Event processing: tegenwoordig moeten applicaties realtime kunnen reageren op events. In het data grid kunnen veel data veranderingen voorkomen die weer kunnen leiden tot veel events.

Wat is Oracle Coherence

Een Oracle Coherence in-memory data grid is een datamanagementsysteem voor applicatieobjecten opgeslagen in een cluster van JVM instanties. Oracle Coherence garandeert hierbij een hoge performance, voorspelbare schaalbaarheid, continue beschikbaarheid en een hoge betrouwbaarheid. Om JVM instanties te clusteren maakt Oracle Coherence gebruik van een peer-to-peer clusterprotocol. Bovenop dit clusterprotocol biedt Oracle Coherence verschillende replicerende en gedistribueerde datamanagement en caching services, waarmee eenvoudig een data grid kan worden opgezet en beheerd. Daarnaast bevat Oracle Coherence een uitgebreide Java API. De Java API zorgt ervoor dat de mogelijkheden van Oracle Coherence niet beperkt blijven tot alleen een cache voor applicatieobjecten. De Java API biedt functionaliteit waarmee een breed scala aan data grid services kan worden geschreven.

Caching & Clustering

In de basis is Oracle Coherence een cache waarin applicatieobjecten in hun eigen structuur opgeslagen worden. Objecten worden geserialiseerd opgeslagen. Een Oracle Coherence cache is een java.util.Map implementatie met toegevoegde functionaliteit om aan alle eisen te voldoen.
Oracle Coherence implementeert verschillende cache typen:

- Replicated: data wordt gerepliceerd naar iedere cluster node. Een replicated cache levert hoge performance bij lezen, maar minder performance bij schrijfacties. Daarnaast wordt de opslagcapaciteit van de cache begrensd door de kleinste JVM in het cluster;
- Distributed (Partioned): data wordt gepartitioneerd opgeslagen in het cluster, waardoor de opslagcapaciteit lineair toeneemt met de grootte van het cluster. Schrijf- en leesacties zijn constant en onafhankelijk van de grote van het cluster.
- Near: dit is een hybride cache waarbij gebruik wordt gemaakt van een combinatie van een local front cache (front-cache) en een distributed back cache. De local front cache wordt gesynchroniseerd met de distributed back-cache. Dit is een L1+L2 cache strategie. De front-cache bevat data

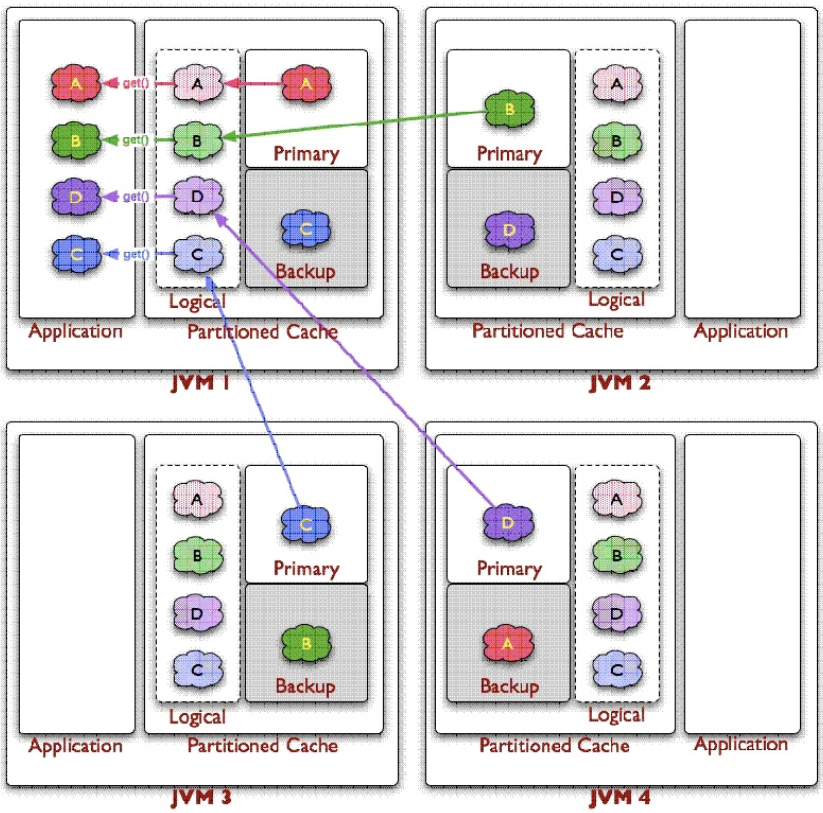
Tom Höfte
is Senior SOA/Java Consultant bij Oracle Netherlands.

die lokaal veel geraadpleegd wordt (L1-cache). De back-cache is hierbij de geclusterde cache (L2-cache) waarin data van alle nodes beschikbaar is. Door een juiste invalidatiestrategie te kiezen in de front-cache kan men voorkomen dat data 'out-of-sync' raakt met de data in de back-cache.

Oracle Coherence is gebouwd bovenop een collectie van services, zoals de clusterservice en de cacheservice. De services draaien als een deamon in een Oracle Coherence server. De clusterservice is verantwoordelijk voor het clusterbeheer, zoals het detecteren van nieuwe of afgestorven clusterleden. Er draait altijd een clusterservice per server. Een cacheservice is verantwoordelijk voor het beheer van de data in de cache, zoals datapartitioning en fail-over als een clusterserver wegvalt in een distributed cache. Per cachetype is er een cacheservice. De services kunnen automatisch worden gestart tijdens het starten van een Oracle Coherence applicatie door dit te configureren of de services kunnen expliciet in de code gestart worden door gebruik te maken van de Oracle Coherence API.

Oracle Coherence servers communiceren door middel van een peer-to-peer clusterprotocol. Dit clusterprotocol is ontwikkeld ten behoeve van de performance en robuustheid van het cluster. In dit protocol bestaat er geen masterserver die alles binnen het cluster coördineert met als voordeel dat een Oracle Coherence cluster geen single-point of failure bevat; alle Oracle Coherence servers in het cluster dragen hiermee zelf de verantwoordelijkheid voor datamanagement, gezondheid van het cluster en uitvoeren van taken.

In een distributed cache topologie is de data opgedeeld in partities. De data wordt gedistribueerd over de partities in het cluster, waarbij de partities weer verdeeld zijn over de clusternodes. Data in het cluster wordt standaard twee keer opgeslagen in een primaire kopie en een back-up kopie om betrouwbaarheid en beschikbaarheid van data te waarborgen. De primaire kopie en de back-up kopie worden opgeslagen in verschillende Oracle Coherence servers. Het is daarnaast mogelijk dat er meerdere back-up kopieën bewaard worden en dat de back-up kopieën fysiek gescheiden worden van de primaire kopie en eventuele andere back-up kopieën van dezelfde primaire back-up kopie. Het clusterprotocol zorgt er ook voor dat iedere server weet waar de data (zowel de primaire kopie als de back-up kopie) opgeslagen ligt in het cluster.
De figuur toont een voorbeeld van een Oracle Coherence cluster bestaande uit vier Oracle Coherence servers die ieder in een eigen JVM draaien. Iedere Oracle Coherence server heeft een logische view van alle data in de cache. Iedere server weet waar het data kan vinden: lokaal dan wel op een andere server. De



Figuur 1. Distributed Cache read.

primaire kopie en back-up kopie zijn opgeslagen in verschillende Oracle Coherence servers.

In JVM 1 draait zowel een Oracle Coherence server als een Oracle Coherence client applicatie. In de praktijk is het vaak het geval dat de client en de server een eigen JVM hebben. Het is namelijk niet efficiënt (CPU resources, opslagcapaciteit) om een JVM zowel voor de client als voor de server te gebruiken.

Data Source integratie

Een in-memory data cache is nuttig om data voor tijdelijk gebruik op te slaan, vooral waarneer deze data onderhevig is aan veel lees- en schrijfacties in korte tijd. Het is in sommige gevallen echter noodzakelijk dat data in de cache gepersisteerd wordt in bijvoorbeeld een database. Oracle Coherence biedt mogelijkheden om data in de cache te persisteren. In de configuratie van een cache kan een CacheStore gedefinieerd worden. In de CacheStore configuratie is gedefinieerd hoe data gepersisteerd moet worden, bijvoorbeeld een database, welke CacheStore implementatie klasse gebruikt moet worden en welke strategie gehanteerd moet worden.

Er kan uit vier strategieën gekozen worden:

- Write-Through-Caching: wijzigingen in de cache worden direct gepersisteerd na een wijziging in de cache.
- Write-Behind-Caching: wijzigingen worden na een bepaalde tijd gepersisteerd. Op deze manier

Een cache laag bovenop bestaande persistentie lagen

Oracle Coherence is gebouwd bovenop een collectie van services

is het mogelijk om op een minder druk moment, data uit de cache te persisteren.
- Read-Through-Caching: wanneer een object x uitgelezen wordt uit de cache door een applicatie en object x is niet aanwezig in de cache, wordt object x uit de onderliggende data source geladen in de cache en vanuit de cache aangeboden aan de applicatie.
- Refresh-Ahead-Caching: data in de cache wordt automatisch periodiek gesynchroniseerd met data in de data source. De synchronisatie frequentie is configureerbaar.

De ontwikkelaar kan zelf een CacheStore implementatie schrijven door een specifieke interface te implementeren en hierbij bijvoorbeeld JDBC API te gebruiken om data weg te schrijven naar de database. Als alternatief voor het schrijven van eigen JDBC code bevat Oracle Coherence CacheStore implementaties voor Oracle TopLink Essentials, JPA en Hibernate. Hiermee kan Oracle Coherence als cache laag dienen bovenop bestaande persistentie lagen die geïmplementeerd zijn met de meest gangbare O/RM frameworks.

Installatie en configuratie

De Oracle Coherence distributie bestaat uit een set van JAR- files, documentatie, voorbeelden en testscripts om een default Oracle Coherence server te starten.
Om een Oracle Coherence applicatie te bouwen, heb je minimaal de coherence.jar nodig. Deze JAR-file bevat alle benodigde functionaliteit en een aantal standaard configuratiefiles om een Oracle Coherence server te starten en data grid services te bouwen. De andere JAR-files bevatten functionaliteit voor bijvoorbeeld integratie met Hibernate of Oracle TopLink.
Een Oracle Coherence applicatie bestaat enerzijds uit code implementatie en anderzijds uit een deel XML configuratie. Er zijn twee belangrijke configuratie bestanden in Oracle Coherence om de cluster eigenschappen te configureren en de cache typen te configureren. In de coherence.jar file zitten hiervoor twee standaard XML-bestanden:
- tangosol-coherence.xml
- coherence-cache-config.xml

De tangosol-coherence.xml file bevat configuratiesettings die nodig zijn om een cluster op te zetten. Bijvoorbeeld het multicast adres dat er voor zorgt dat meerdere Oracle Coherence servers elkaar kunnen 'vinden' en dus een cluster kunnen vormen. In de tangosol-coherence.xml worden ook de cache en cluster services geconfigureerd.
In de coherence-cache-config.xml file staan een aantal standaard cache configuraties. Een cache configuratie kan gekoppeld aan een naam door middel van een cache-mapping. De naam wordt

gebruikt in de applicatie om een specifieke cache instantie te creëren. In de naam kunnen wild-cards voorkomen. In de onderstaande code listing staat een voorbeeld van de configuratie van een distributed cache en de cache-mapping.

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-*</cache-name>
      <scheme-name>example-distributed</scheme-name>
      ...
    </cache-mapping>
...
    <!--
    Distributed caching scheme.
    -->
    <distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>

      <!-- To use POF serialization for this partitioned
service,
           uncomment the following section -->
      <!--
      <serializer>
                    <class-name>com.tangosol.io.pof.
ConfigurablePofContext</class-name>
      </serializer>
      -->

      <backing-map-scheme>
        <local-scheme>
         <scheme-ref>example-binary-backing-map</scheme-
ref>
        </local-scheme>
      </backing-map-scheme>

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

De distributed cache maakt gebruik van de DistributedCache service. Deze naam is een referentie naar de service definitie in de tangosol-coherence.xml file. In commentaar staat tevens een POF serializer gedefinieerd. POF staat voor Portable Object Format en is een uiterst efficiënt object serialisatie methode om de overhead van de standaard Java serialisatie methode te vermijden. In het distributed-scheme element wordt gerefereerd naar een backing-map-scheme: example-binary-backing-map. Het example-binary-backing-map scheme definieert de eigenschappen van de cache bijvoorbeeld welke specifieke java.util.Map implementatie gebruikt wordt.
De standaard tangosol-coherence.xml en coherence-cache-config.xml worden in de praktijk vaak aangepast vanwege de specifieke eisen van een Oracle Coherence implementatie. Een goed start punt om in de configuratie mogelijkheden te duiken is de Oracle Coherence wiki.

Simpel Hello Grid! voorbeeld

Na uitleg over de configuratie van Oracle Coherence is het nu tijd om een simpel voorbeeld te geven van een Java applicatie die eerst een Oracle Coherence server start en vervolgens een Hello Grid! bericht in de cache zet en deze weer uitleest. Door de coherence.jar file in het classpath te plaatsen, kan gebruik gemaakt worden van de Oracle Coherence API en standaard configuratie bestanden.

De onderstaande code-listing toont de code die het Hello Grid! voorbeeld implementeert.

```
package nl.oracle.com.coherence.hellogrid;

import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;
import com.tangosol.util.UUID;

public class HelloGrid {
    public HelloGrid() {

    }

    public static void main(String[] args) {
        UUID Key = new UUID();
        CacheFactory.ensureCluster();
        System.out.println("Cluster Service started");
         NamedCache cache = CacheFactory.getCache("dist-
hellogrid-cache");
        cache.put(Key,"Hello Grid!");

        System.out.println(cache.get(Key));

 //Slaap zacht
 try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            // TODO
        }

    }
}
```

De methode CacheFactory.ensureCluster() initialiseert het nieuwe clusterlid en door de clusterservice te starten. In principe is deze methode optioneel, omdat de CacheFactory.getCache methode dit impliciet ook doet.
De methode CacheFactory.getCache()geeft een NamedCache object terug. Op basis van de meegeven naam wordt in de coherence-cache-config.xml de juiste cache configuratie bepaald. Vervolgens worden die eigenschappen gekoppeld aan het NamedCache object. Een NamedCache object is een logische view van het cache cluster en is het toegangspunt om objecten te lezen en te schrijven. In het bovenstaande voorbeeld wordt een NamedCache object terug gegeven die gekoppeld is met een gedistribueerde cache service. De cache service wordt gestart door de aanroep van CacheFactory.getCache() methode. De sleep aan het einde van de main method zorgt ervoor dat de cluster node in leven blijft zodat een cluster van Hello Grid! applicaties gevormd kan worden.

Het testen van dit Hello Grid! voorbeeld is eenvoudig: executeer de HelloGrid class file 1 of meerdere keren en zorg daarbij dat de coherence.jar op het classpath staat. Door de applicatie meerdere keren op te starten wordt een cluster gevormd aangezien er een slaaptijd in de applicatie zit die voorkomt dat de Oracle Coherence server in leven blijft. In de output log kan afgelezen worden wat Oracle Coherence doet (distributie van data etc.) in het geval een nieuwe node aan het cluster wordt toegevoegd of als er een bestaande node wordt verwijderd als de slaap tijd verstreken is.

Aanpassing van data wordt ook doorgevoerd op de back-up kopie

In het HelloGrid! voorbeeld wordt Oracle Coherence puur en alleen gebruikt als object cache. Oracle Coherence is meer dan een cache. Oracle Coherence bevat een Java API met een breed palet aan gereedschappen met een veelvoud aan mogelijkheden. Dit hoofdstuk gaat kort in op een aantal interessante functionaliteiten.

Agents

Een agent is een stuk code dat een specifieke taak implementeert (e.g. update van een cache entry). Agents kunnen in een data grid uitgevoerd worden, waarbij het data grid bepaald waar de agent uitgevoerd wordt. Een agent wordt uitgevoerd op de node die de data bevat waartegen het uitgevoerd moet worden. Op deze manier is het alleen noodzakelijk om datatoegang, concurrencycontrol en queuing van agents op een enkel node te regelen in plaats van gedistribueerd wat veel complexer en kostbaarder in tijd is. Oracle Coherence zorgt er ook voor dat data-aanpassingen ook doorgevoerd worden op back-up kopie.
Agents kunnen zowel op een cache object (bijvoorbeeld een update actie) of een clusternode (bijvoorbeeld het uitlezen van het geheugen gebruik) uitgevoerd worden.
De NamedCache implementeert de InvocableMap interface waarmee het de methoden bevat:

- invoke(key, agent)
- invokeAll(key, filter | key collection)

De invoke(key, agent) executeert een agent tegen een enkel object. De invokeAll(filter | keyCollection, agent) executeert een agent parallel tegen een set van objecten die gedefinieerd zijn met een key set of een filter query. Een agent wordt geïmplementeerd door de EntryProcessor interface te implementeren. Standaard zitten er in Oracle Coherence een aantal handige implementaties van de EntryProcessor.

Map Listeners

Met Oracle Coherence kan data opgeslagen worden in een in-memory data grid. Vanuit een databeheer perspectief kan het belangrijk zijn om events die optreden in het data grid te kunnen monitoren en indien noodzakelijk actie te kunnen ondernemen. Een voorbeeld van een event is het toevoegen of verwijderen van een object. Oracle Coherence gebruikt het JavaBean Event model om naar events op objecten in de cache te kunnen luisteren.
De Java API bevat een MapListener interface die de EventListener interface uit het JavaBean Event model implementeert. De MapListener interface bevat drie methoden, waarbij het MapEvent het ObjectEvent uit JavaBean event model is:

- entryInserted(MapEvent evt)
- entryDeleted(MapEvent evt)
- entryUpdated(MapEvent evt)

Door de MapListener interface te implementeren, kan een eigen map listener geïmplementeerd worden. Een map listener wordt geregistreerd bij het NamedCache object door de methode addMapListener() te gebruiken of door de MapListener in de cache configuratie file te koppelen aan de cache. Doordat de NamedCache de ObservableMap interface is een cache in staat om naar events te luisteren met behulp van de map listener. De MapListener functionaliteit in Oracle Coherence ondersteunt tevens filter mogelijkheden op zowel event type als op objecten die de events veroorzaken.

Incubator Patterns: Command & Functor

Het Oracle Coherence Incubator project in het najaar van 2008 gestart en is een project waarin een aantal nuttige design patterns geïmplementeerd zijn. De keuze voor de patterns is gebaseerd op feedback van klanten met real-life Oracle Coherence implementaties. Twee interessante patterns uit het Incubator project zijn:

- Command pattern
- Functor pattern

Het Command pattern implementeert het klassieke Command pattern. Het Command pattern maakt het mogelijk om asynchroon taken in het data grid uit te voeren, waardoor de initiator dus niet wacht op een resultaat en door kan gaan met andere taken. Het Functor pattern implementeert het klassieke Functor pattern en is een extensie van het Command pattern. Het Command pattern biedt geen mogelijkheid om resultaten terug te geven aan de initiator. Het Functor pattern biedt deze mogelijkheid door het gebruik van de Java 5 Futures functionaliteit.

Conclusie

Oracle Coherence biedt een robuuste en schaalbare cluster oplossing om applicatie objecten in een object structuur in-memory op te slaan. Oracle Coherence maakt daarmee grid computing in de Java wereld mogelijk. Door de opzet en uitgebreide documentatie van Oracle Coherence is het vrij eenvoudig om in korte tijd een Oracle Coherence applicatie op te zetten. Daarnaast zijn de patterns in het Incubator project een echte meerwaarde.
Een aantal Oracle producten zijn inmiddels geïntegreerd met Oracle Coherence om ze 'grid-aware' te maken. Een voorbeeld hiervan is Oracle TopLink Grid, waar Oracle Coherence gebruikt wordt als queryable cache tussen de Oracle TopLink JPA object laag en de database. Oracle Coherence biedt ook ondersteuning voor Oracle Weblogic Server and Oracle Weblogic Portal om het HttpSession object in een Oracle Weblogic Server cluster efficiënt in een cache op te slaan. Daarnaast biedt Oracle Coherence ondersteuning om als L2 cache of query cache provider te dienen in Hibernate. «

Het is vrij eenvoudig om snel een Oracle Coherence applicatie op te zetten.