Bring the process to the cached data in Oracle Coherence

Nicolás Fonnegra Martínez
esentri AG
Ettlingen

Keywords
Coherence, Coherence agents, Distributed processing, Data Grid

Introduction

Oracle Coherence is an in-memory data grid solution that takes part in the Oracle Cloud Application Foundation. A Coherence cluster is comprised of several cache nodes that maintain and mange the data in a distributed and collaborative way. Its main objective is to provide a reliable, scalable and high-performance data grid that can be used by several kinds of clients.

Querying the data is one of the more important aspects of such topologies and Coherence provides several options to access and filter the cached entries. One option is to use the powerful FilterBuilderAPI to write queries that can filter the cache keys and their respective values. Another option is to use agents (or entry processors), which can be more efficient depending the scenario. These entry processors pursue two main objectives. First they follow the principle of sending the process to the data rather than bringing the whole cached data to process it locally. The second objective of the entry processors is to provide a better way to lock entries in the cache by sending an agent to acquire the lock where the data resides.

This document is divided in four parts. The first one will describe the Coherence entry processors. The second and the third part will describe two scenarios where the agents provide a very good alternative and high-performance alternative. The last part will describe the anatomy of the agents.

Coherence Entry Processors (aka Agents)

Coherence agents (also known as Coherence entry processors) provide a different alternative to process and query the data in a Coherence Data Grid. A Coherence Agent is piece of code that resides in every cache node that gets activated when the “invoke” or the “invokeAll” method gets invoked in the NamedCache. It can be used to alter values in the cache or to retrieve values from it. The main benefit of this approach is that the resources being used in this process are the ones of the cache node where the data resides and not the ones from the part originating the query; in other words, the whole processing effort (and therefore the resource usage that comes along such as CPU, RAM) is being delegated to the node(s) owning the data.

There are two sample scenarios where the use of agents is best seen:

Scenario 1: Dedicated Storage Nodes

The dedicated storage node is a very popular and particular implementation of the Coherence distributed cache. In it, dedicated cache nodes are configured to store the data while other nodes only act as proxies. These proxy nodes are in the same JVM with the applications that use Coherence such as ADF applications, SOA composites, Webcenter portals or JEE applications. The main benefit of this approach is that these applications can use the Coherence cluster without having the overhead of storing cache objects in their heap or using their CPU to process distributed queries.

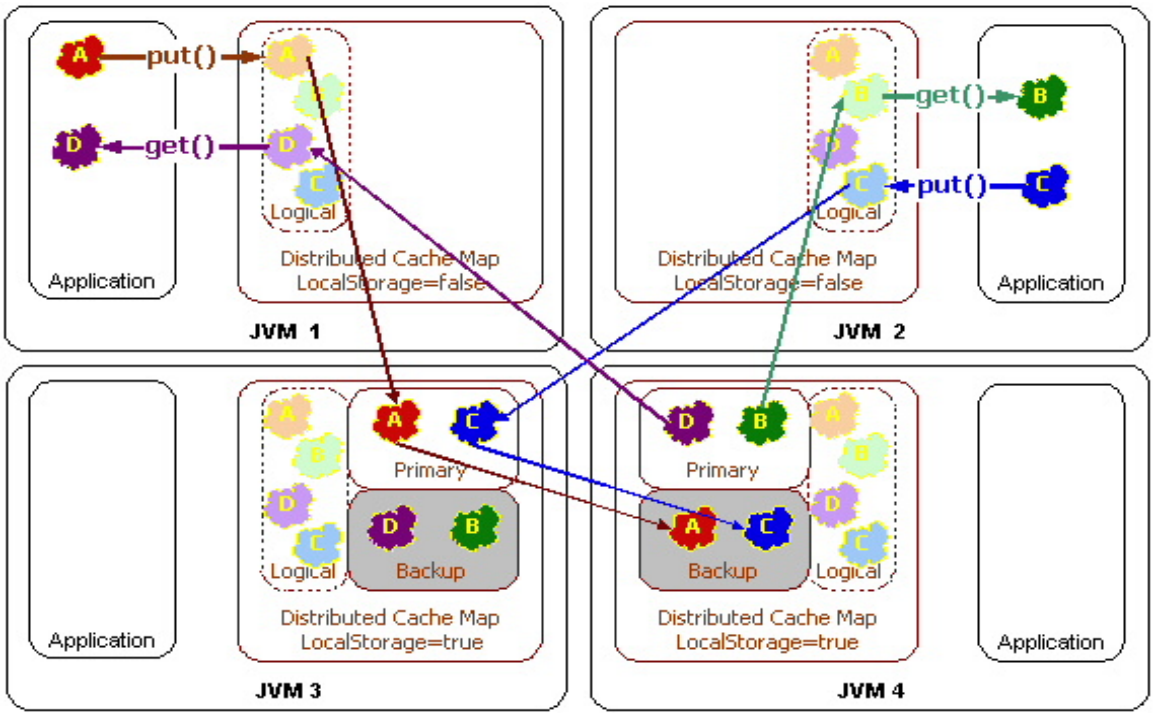

Figure. 1: Agents querying distributed data

Normally these applications can use the FilterBuilderAPI to access the data in the cache but in some occasions this is not possible. For example, the Coherence SOA Suite adapter allows the storing of XML documents rather than Java objects. The content of this XML documents cannot be filtered out with the FilterBuilderAPI. There are other cases where some post- processing is needed after performing the query, which cannot be done with the FilterBuilder API.

Querying and retrieving the whole data in order to process it locally is not such a good idea. It will damage the whole concept of having light Coherence nodes besides the application to save resources and it will generate a lot of network traffic. Instead agents can be sent directly to where the cache entries are stored in order to query and process the data remotely. This approach reduces significantly the network traffic used while saving valuable resources in the nodes that most need them. Figure 2 shows the above-described scenario

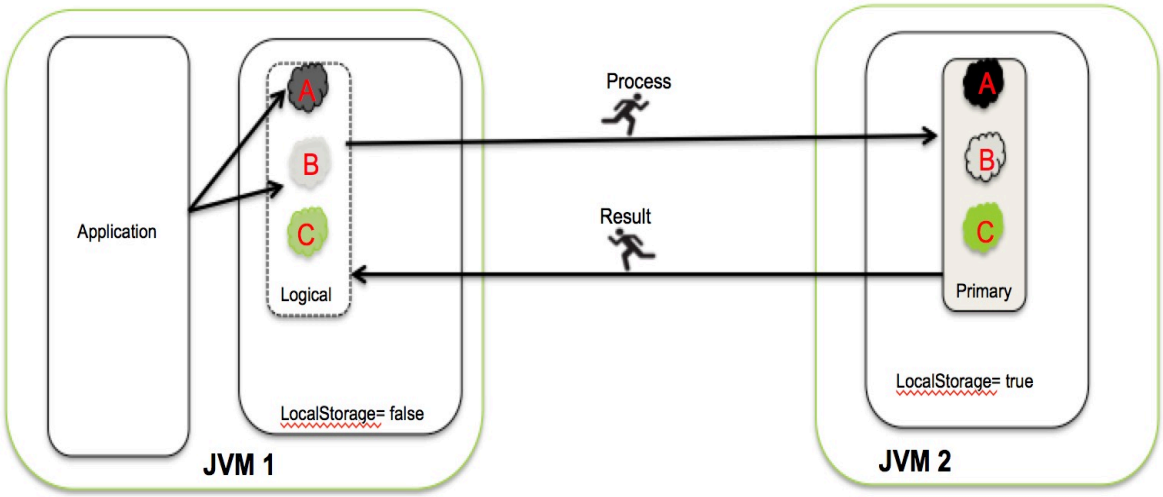

Figure. 2: Agents querying distributed data

Scenario 2: Distributed locking

Many of the Coherence examples show how to query and alter simple cache entries. But Coherence is more than an in-memory map; it is a data grid and therefore it should support many requirements that other data containers such as databases provide. In many cases, concurrency may be an issue and it is desirable to lock some entries in order to process them atomically.

Coherence provides the option to acquire locks for a determine entry. Blocking an entry and executing the operations on the entry requires network access, one connection for each operation. So a simple operation involving querying and updating an entry will require four network access, one for querying, one for updating, one for locking and one for unlocking. Instead, an agent could be used to process the data where it resides involving only one network access. This approach is much more efficient and more fault tolerant.

In some cases, more than one entry needs to be updated. Using the traditional approach, a lock will have to be acquired for all entries involved, multiplying the needed network connections for the operations and the possibility that a network failure happens. A better alternative will be to use the invokeAll method from the NamedCache to send an agent with the corresponding key to the entries that have to be updated.

The anatomy of an agent

Coherence agents (or entry processors) are normal Java classes that extend from com.tangosol.util.processor.AbstractProcessor implement Serializable. These agents use weak migration, meaning that their code has to reside in both source and destination. In other words, the agent source code should reside in each node's classpath. It is important to mention that Coherence needs that the same version of the agent's source code exits in each node, which is guaranteed by the serialVersionUID attribute.

The agent has to implement the process method. This method takes as parameter and entry of the cached map and returns a java.lang.Object. It is also possible to override the processAll method, which takes a Set as a parameter and returns a Map with the multiple responses. The process and processAll

methods are called when the methods invoke and invokedAll are used respectively. The agent has to be serializable in order for its state to be transferred between the source and the destination.

As mentioned above, agents have to be distributed among each node of the cluster. To achieve this, GAR archives can be used to deploy the agents, along with the respective cluster information. GAR (Grid archives) files are very similar to JAR files. The main difference is that GAR files contain a file called “coherence-application.xml” in the META-INF directory in order to help the application server to identify them properly

Conclusion

Coherence provides a very powerful way to distribute the storage of data by providing a data grid. Distributing data has many benefits like linear scalability and fault tolerance, but it also comes with several challenges inherent to this architectures. Querying and modifying the data has now the complexity of the distribution of the information and in some cases the traditional methods are not enough in terms of performance and efficiency.

In some cases a different approach is needed and Coherence agents (o entry processors) may be the ideal solution. The main idea behind this paradigm is to send the process to the data rather than bringing the data to the process it locally. This alternative can be very powerful because it takes the performance overhead to where the data is stored, letting the consumer of the operation to focus its resources on its core business functions. Also, it can save a lot of network traffic since agents can filter and return only the needed data remotely instead Another benefit achieved with this solution is the reduction the network connections needed to lock and alter multiple entries, that can escalate dramatically if multiple entries need to be modified.

Kontaktadresse:
Nicolas Fonnegra Martinez
esentri AG
Pforzheimerstr 132
D-76275 Ettlingen

Telefon: +49 (0) 15118733911
Fax: +49 (0) 72433549099
E-Mail Nicolas.fonnegra@esentri.com
Internet: www.esentri.com