A Scotas white paper
September 2013

Scotas OLS

scotas
real time search solutions

Introduction

When you have to perform searches over big data, you need specialized solutions that can deal with the velocity, variety and volume of this valuable information which analysis allows you to implement better solutions and to delineate appropriate business strategies. An important challenge in this area is how to handle the company's big data and normalized data in an efficient, on line and integrated way.

Oracle®database has the ability to implement user's Java applications inside the engine in a transparently way. Following this approach, Scotas OLS is a product that has all the previous characteristics, through the implementation of the market standard solution Solr/Lucene inside Oracle®engine adding transactional and security features.

Technical overview

Scotas OLS is a tight integration of the popular, blazing fast open source enterprise search platform from the Apache Lucene project Solr with Oracle® 11g Enterprise Edition.

Our implementation provides to the Oracle RDBMS the power of the Solr searching facilities such as faceting, geospatial and highlighting among others natively into the SQL.

Also by running Solr into the Oracle® RDBMS your SQL data is automatically indexed and updated in NRT (Near Real Time) way without any programming code.

For Solr users who want other features such as transactional storage, encryption, compression or row level filtering, for example, this integration is the perfect solution.

Architecture

OLS is tightly integrated into the Oracle® RDBMS by using the Oracle® Data Cartridge API (ODCI) adding a new Domain Index such as Oracle® Text or interMedia -a domain index is like any other Oracle® index attached to some column of an specific table-.

This integration provides to Solr Engine the ability of running as another Oracle® server process receiving notifications when rows are added, changed or deleted. Also, it is a bi-directional integration provisioning to the SQL a new relational operator (*scontains*) and several new functions and procedures which can be used on SQL constructions or procedural code.

Following picture depicts the architecture of Scotas OLS.

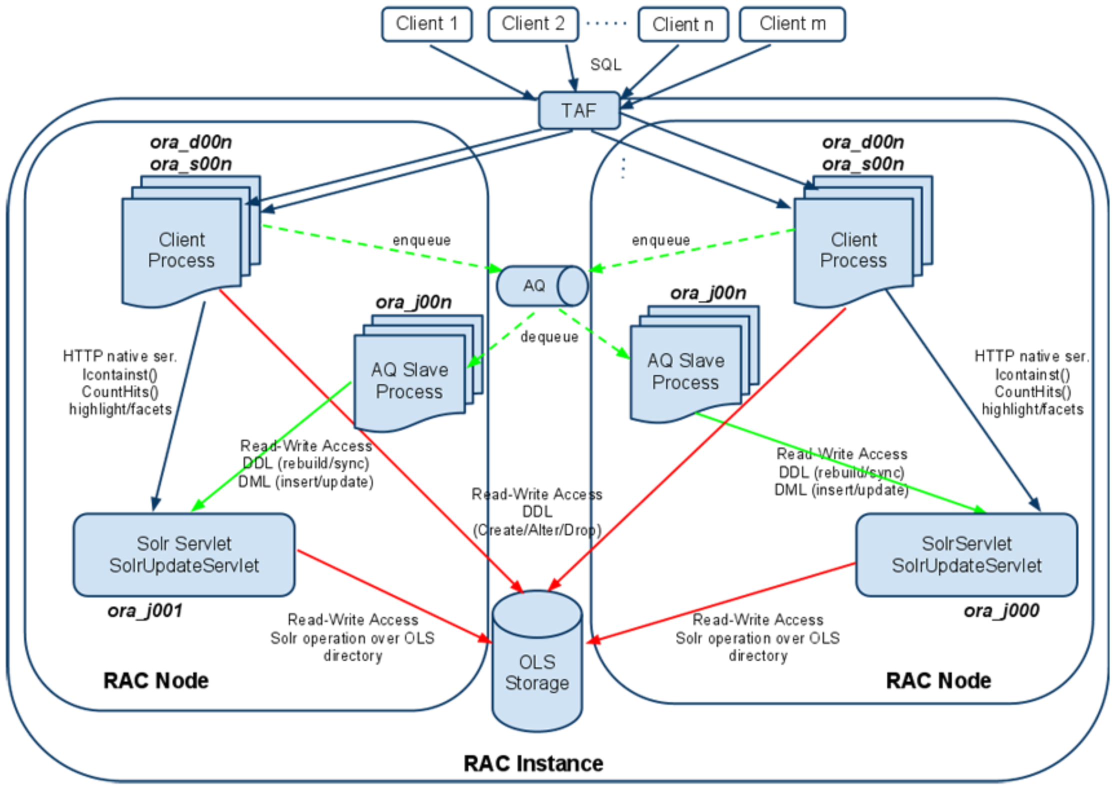

Figure 1: Diagram of Scotas OLS architecture.

Above picture shows how many processes interacts during OLS operation in an Oracle® RAC installation.

External applications connects to the Oracle instance using transparent fail-over configuration (TAF) which routes the SQL sentences to some of the nodes of the installation.

Each connection to the RDBMS has associated an Oracle® process ora_d00n if is connected using a dedicated connection or an ora_s00n if is configured in shared mode.

Once a SQL query is interpreted by the engine several actions are started depending on if it is a DML operation or a query.

For DML operation once is committed the changes are propagated by the Oracle® AQ sub-system and the Solr instance running as separated server process receives the rowid(s) that was(were) changed updating their index structure, note that only rowid values are transmitted between processes, not the row data.

A parallel slave server process which is started during database startup runs a Solr server instance for one or all OLS index declared, the communication between this Solr instance and the client process which is associated to the client connection is using HTTP binary format. For DML operations Solr receives the rowid that need to be updated and if it is necessary gets the row values from the table(s) involved (these are processed using the internal OJVM drivers which have directly access to SGA areas of the RDBMS).

During SQL select operations the client associated process (ora_d00n or ora_s00n) talks to the Solr server instance using the start-fetch-close semantic, sending Solr query arguments, sorts, etc, during fetch stage, consuming rowid(s) that match in the fetch stage and cleaning up any temporary structures at close.

Working modes

OLS works in two different modes depending on the kind of application's behaviour.

For applications which have a high rate of DML operations by second on the table which the domain index is defined, a Deferred mode is better. In this scenario changes on Solr index are apply at an application specific point in time or periodically using the scheduler.

On the other hand, OnLine mode works in a Near Real Time mode (NRT), once the changes are committed they are applied into Solrindex resulting in a positive hit if it was an insert, for example. In case of delete operations they are working in Real Time (RT) mode, similar to the SQL transaction behavior, once the row is deleted it will result in negative hit immediately for the current transaction and at commit point for others concurrent transactions.

The following trace diagram shows how the NRT/RT mode works in terms of data consistence in a concurrent multi-user environment.

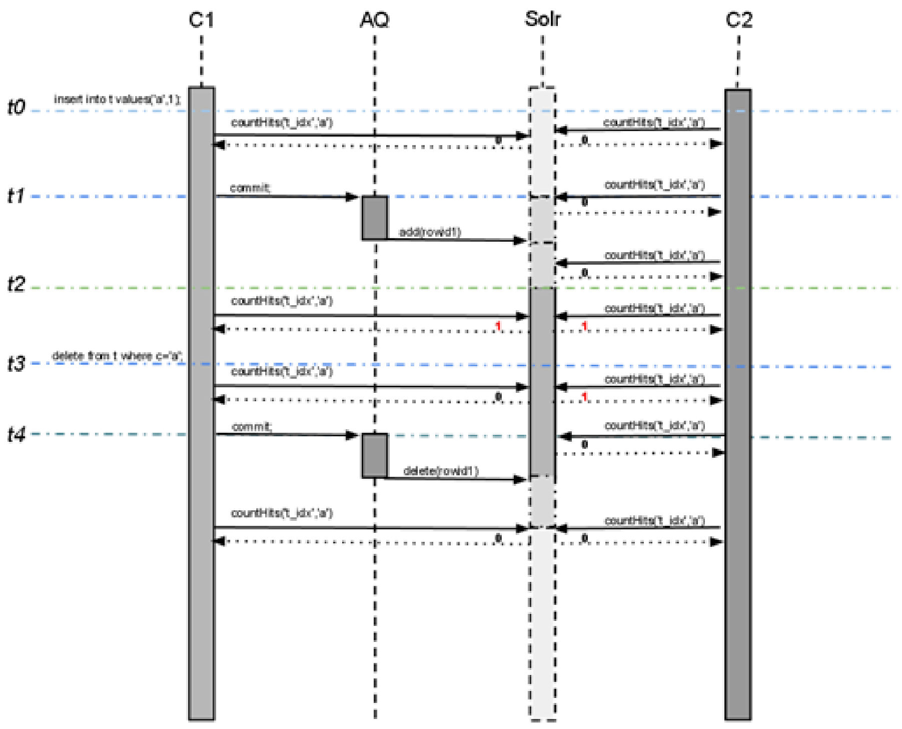

Figure 2: Diagram of NRT/RT mode work in terms of data consistence.

For DML insert (update) there is time frame t0-t2 where a negative hit is returned and differs from traditional SQL consistence, note that even if the C1 transaction inserts a row during this period (t0-t2) a call to *countHits()* function results in a negative hit.

When C1 commits their changes these are processed by the AQ sub-system which sends a set of rowid(s) for processing to Solr instance, during t1-t2 time frame (which is very small) also C2 connection gets a negative hit, this is why the mode is called Near Real Time to make a difference from SQL consistence mode, but the implementations results in high performance search sub-system for free text queries.

After t2 point in time the main transaction and others concurrent connections see a positive hit of the row just inserted.

Following the timeline a delete DML operation is shown, unlike inserts or updates, deletes can't be deferred and always works consistence in pair with traditional consistence level, this is because of the transaction C1 OLS domain index must no return a rowidwhich was deleted as a positive hit (time frame t3-tn)
but C2 connection is still sees the row as positive hit until C1 commit theirs changes (t3-t4), once C1 do a commit *countHits()* function immediately returns a negative hit on C2 connection too.

Index storage

OLS replaces the Lucene inverted index storage, which by default is stored on the OS file-system, by Oracle® Secure File BLOBs, resulting in high scalable, secure and transactional storage, see Secure Files whitepaper for more details and performance comparison againstNFS or ext3 file-system.

A summarizes advantages of this approach are:

- Transactional storage, a parallel process can do insert or optimize operations and if they fail simply do a rollback and nothing happens to other concurrent sessions.
- Compression and encryption using Secure File functionality, applicable to Lucene Inverted Index storage and Solr configuration files.
- Shared storage for Lucene Inverted Index, on RAC installations several processes across nodes can use the storage transparently.

Examples of Use

From the DBA or application developer's perspective the integration of Scotas OLS with your current system is very simple.Once you define all the fields in the schema.xml file as you normally do on Solr, just need to start working with OLS using SQL sentences.

Creating an Index

Suppose that you have a MOVIES table with a large volume of data. Once Scotas OLS is installed on your Oracle® RDBMS, you need to create an index into the table and fields you want to run queries.

The next SQL sentence is very similar to the way to normally create indexes except for a couple of parameters that OLS requires.

```
CREATE INDEX "MOVIES_SIDX" ON "MOVIES" ("TEXT") INDEXTYPE IS
"LUCENE"."SOLRINDEX" PARAMETERS
('BatchCount:4000;LockMasterTable:false;LogLevel:ALL;SyncMode:OnLine;Hi
ghlightColumn:HL;DefaultColumn:text;ExtraCols:GENRE,LANGUAGE,COUNTRY,DI
RECTED BY,substr(text,1,200) HL');
```

Code 1: Example of index creation with Scotas OLS.

The first difference you can find is the `INDEXTYPE`that OLS uses (`LUCENE.SOLRINDEX`). The `PARAMETERS`list can be adjusted to your current environment. In the example of Code 1, the BatchCount:4000 means that Oracle®database will be indexing documents in a batch of this numbers. Then, the `DefaultColumn:text`means that OLS will use the column text as the default one in case that the query doesn't specify any. The `ExtraCols:GENRE,LANGUAGE,COUNTRY`… are the other columns that will be indexed too. Finally, the `substr(text,1,200) HL`means that the highlight will be done on the first 200 characters of the text's column.

The index creation is all that you need to start indexing documents with Scotas OLS.

Running Queries

Depending on the type of results you want to run there are different classes of SQL queries.For example, to get the first 25 documents that match with the text "terminator", run this query:

```
SELECT sscore(1) SSCORE, TEXT, DIRECTED_BY, LANGUAGE, COUNTRY FROM
MOVIES WHERE scontains(text,'rownum:[1 TO 25] AND terminator',1)>0
```

Code 2: SQL query to get documents that match with "terminator" string.

In case that you also want to highlight the results, just add the sentence `shighlight(1) as HL`in the place you list the columns to be retrieved.

To get the total number of documents that match with a particular query, OLS provides another function called *SolrDomainIndex.countHits*. For example:

```
SELECT SolrDomainIndex.countHits('MOVIES_SIDX','terminator') as count
FROM dual
```

Code 3: SQL query to get the count of documents that match with “terminator” string.

To retrieve facets and number of documents in each one, the SQL query is the following (in this case it’s filtering using both facets GENRE and COUNTRY):

```
SELECT * FROM TABLE(SELECT T.FACETS FROM
TABLE(SFACETS('SCOTT.MOVIES_SIDX','text:terminator','facet.field=GENRE&
facet.field=COUNTRY')) T
```

Code 4: SQL query to get the facets.

Finally, you can mix queries to get not only the results but also filtered by specific facets. For example, the next case is similar to Code 1 but getting only the movies with genre Action (`GENRE:”Action”`):

```
SELECT shighlight(1) as HL, sscore(1) SSCORE, TEXT, DIRECTED_BY,
LANGUAGE, COUNTRY FROM MOVIES WHERE scontains(text,'rownum:[1 TO 25]
AND terminator AND GENRE:"Action"',1)>0
```

Code 5: SQL query to get the documents refined with facets.

To get more examples you can visit the Scotas OLS’ demo site at http://demo.scotas.com

scotas
real time search solutions