Die Oracle Datenbank in die Welt von Hadoop und NoSQL integrieren

Gunther Pippèrr
München

Schlüsselworte

NoSQL – Hadoop – Integration – Oracle Databank - Oracle SQL Connector for HDFS - Oracle Loader for Hadoop

Einleitung

Wie lassen sich die beiden Welten, Oracle RDBMS und der NoSQL Ansatz sinnvoll für die Archivierung und das Datensammeln einsetzen?

Ziel des Vortrags ist es aufzuzeigen, wie die Kombinationen aus den Vorteilen der beiden Welten für die Analyse und Archivierung von Daten eingesetzt werden kann.

Hadoop, eignet sich gut, um im ersten Schritt Daten zu sammeln und/oder im letzten Schritt Daten zu archivieren.

Die eigentliche Oracle RDBMS Datenbank kann dabei schnell und schlank gehalten werden, um Hardware und damit Lizenzkosten einzusparen.

Es werden Architekturansätze aufgezeigt, wie die Integration der Oracle RDBMS und NoSQL Datenbank in das Hadoop Ökosystem dabei erfolgen kann.

Mit den kostenpflichtigen Adaptern der Oracle RDBMS lässt sich zwar einfacher eine tiefe Integration mit Hadoop erreichen, aber auch mit den freien Lösungen kann bereits eine umfangreiche Lösung implementiert werden.

Intention und Ziel

Schon seit längeren setzt sich der Trend ununterbrochen fort, dass sich das zu verarbeitende Datenvolumen von Jahr zu Jahr massiv vergrößert. Das aktuelle Problem, wirklich sehr großen Datenmengen verarbeiten zu müssen, lässt sich aber einer gewissen Menge an Daten nicht mehr wirklich skalierbar mit den bestehenden Lösungen umsetzen.

Daten, die im ersten Schritt für das Tagesgeschäft bisher nicht so wichtige waren, müssen vermehrt ebenfalls gespeichert und verarbeitet werden. Meist sind diese, nur am Rand gesammelten Daten, aber nicht wirklich strukturiert und im ersten Schritt auch für die maschinelle Verarbeitung nicht wirklich geeignet. Ein Beispiel sind die reine Oracle Log Files, z.B. die klassische Alert.log Datei. Auch neigt diese Art von Daten stark dazu, sich von Release zu Release immer wieder zu verändern.

Auf der anderen Seite wachsen die DWH Lösungen und benötigen immer größere und kostenintensivere Umgebungen, um zum Teil nur Daten vorzuhalten, auf die nur mit geringer Wahrscheinlichkeit jemals wieder zugegriffen wird, die aber längerfristig für das Business durchaus noch wichtig sind. Zum Beispiel um bei Compliance Überprüfungen die Einhaltung von gesetzlichen Vorgaben nachweisen zu können.

Zusätzlich erschweren die kommerziellen Anbieter von Datenbanklösungen den weiteren Aufbau ihrer eigenen Umgebungen bzw. den Einstieg in ihre eigene Produktlandschaft mit oft hohen Kosten für Support.

Hier kann es dann nicht verwundern, dass neue und von dem Grundkonzept innovative offene Produktplattformen eine breite Basis finden. Zur Zeit halten auch diese neuen Produkte nicht alle ihre Versprechen ein. Oft fehlt auch noch so manches gewohntes Feature und beim Betrieb von diesen Lösungen wird noch viel Geduld mit der Softwarequalität abverlangt, aber es ist zu erwarten, dass sich hier in nächster Zeit noch viel innovativ verändern wird.

Als Softwarelösung für die Anforderungen an eine solche zentrale Plattform setzt sich zurzeit immer mehr das Apache Hadoop Ökosystem durch.

Apache Hadoop ermöglicht mit seinem zentralen Cluster Files System nach dem „Shared Nothing“ Prinzip und einem ausgeklügelten Batch Processing Framework den Aufbau sehr großer Umgebungen für die Verarbeitung von Massendaten mit Hilfe von vielen, im Prinzip preisgünstigen Servern.

Zahlreiche weitere Entwicklungen rund um Kern von Hadoop bieten inzwischen in den Grundzügen alles an, was für den Betrieb auch sehr großer Umgebungen notwendig ist. Und das meist mit auf den ersten Blick kostenarmen OpenSource Lösungen.

Damit ist um Hadoop eine starke Community entstanden und viele Hersteller unterstützen das Ökosystem mit immer mehr Werkzeugen und Schnittstellen. Durch den Open Source Ansatz wird der Einstieg in Hadoop stark erleichtert, ein schneller erster Einstieg scheitert nicht gleich an hohen Lizenzkosten. Durch die kommerziellen Distributionen, wie MAPR, Cloudera und HortonWorks, lassen sich aber auf der anderen Seite großen Umgebungen mit einen klassischen Supportvertrag aufbauen.

Ein konkretes Beispiel für das Sammeln von Daten im Unternehmen, die im Prinzip wenig mit dem Tagesgeschäft zu tun haben, ist die Anforderung, auf Grund von Compliancevorschriften alle Zugriffe in allen Datenbanken im Unternehmen möglichst fein granular zu protokollieren, aufzubewahren und zu auf verdächtiges Verhalten zu analysieren,

Eine zentrale Plattform für das Archivieren und Auswerten von diversen schwach strukturierten Daten kann dabei unterstützen, diese Anforderungen langfristig umzusetzen.

An diesem Beispiel wird im Folgenden die mögliche Integration einer bestehenden Oracle Datenbanklandschaft mit der Hadoop Welt aufgezeigt.

Integration oder alles neu entwickeln?

In den meisten Umgebungen mit hohen Datenaufkommen sind bereits ausgefeilte Datenbanksysteme und komplexe Data Warehouse Umgebungen im Einsatz. Viel Energie und erhebliche Kosten sind in den Aufbau dieser Landschaften geflossen, die zum aktuellen Zeitpunkt auch oft noch ausreichend skalieren und erfolgreich betrieben werden können.

Bei neuen Anforderungen kann es sich aber durchaus lohnen, in neue Technologien einzusteigen.

Die zu lösende Frage heißt dann aber wie die „neu“ und „alte“ Welt produktiv und sinnvoll verknüpft werden kann.

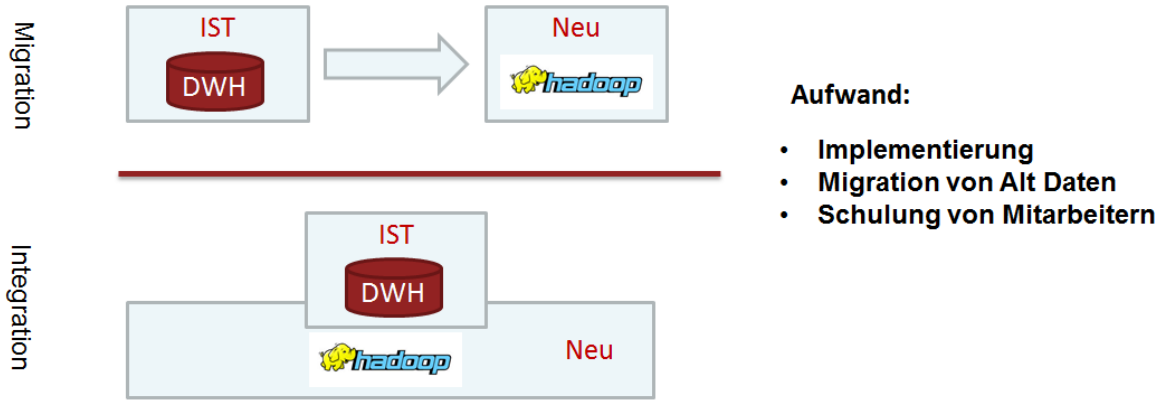

Abb.1 : Neuentwicklung oder Integration

Die Integration kann zum Beispiel über das Berichtwesen erfolgen, d.h. die Daten werden komplett voneinander getrennt verarbeitet und erst im Berichtwesen zusammengefügt.

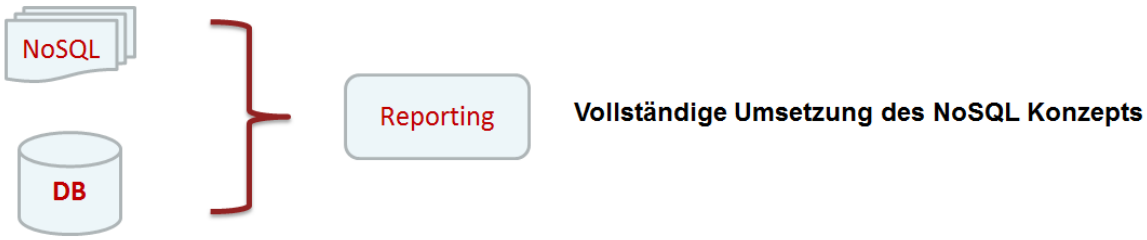

Abb. 2 : Integration über das Berichtswesen

Oder die Datenbanken werden eng mit Hadoop verwoben. Die eigentlichen Abfragen erfolgen weiterhin klassisch über eine zentrale Datenbankumgebung, die aber im Backend auf die Daten im Hadoop Cluster mit verschieden Methoden und adhoc Schnittstellen zugreift.

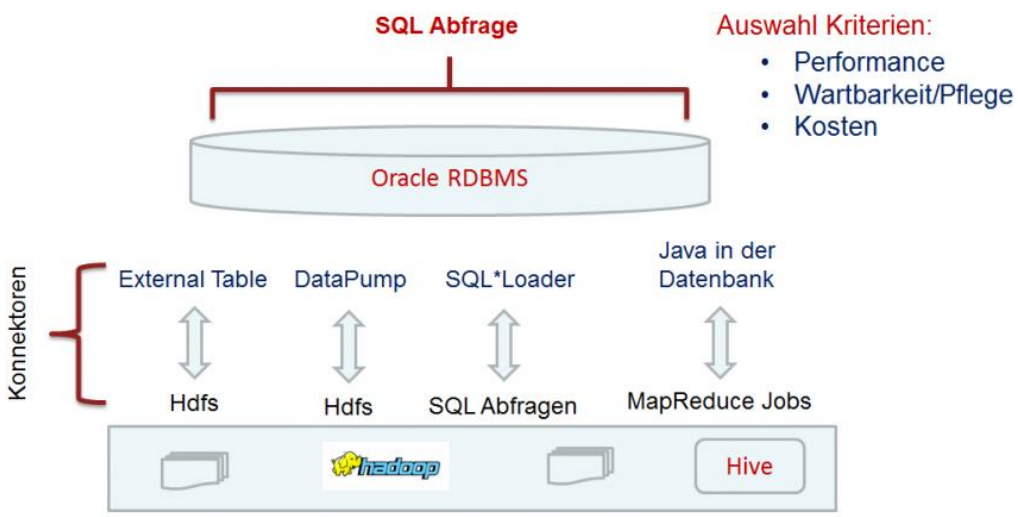

Abb.3 : Eine zentrale Datenbank Umgebung greift über verschiedene Konnektoren auf die Daten in Hadoop zu

Für die Integration stehen im Prinzip zwei Konzepte zur Verfügung:

- Das „External Table“ Prinzip – Daten bleiben im HDFS und werden über die Datenbank per SQL „on the fly“ gelesen
- SQL*Loader Prinzip – Daten werden in die Datenbank geladen

Für das „External Table“ Prinzip“, bei dem die Daten im HDFS gehalten werden, steht uns zur Verfügung:

- Oracle „External Table“ auf ein mit FUSE freigegebenes HDFS Verzeichnis
- Oracle „External Table“ mit dem kostenpflichtigen Oracle Adapter - Oracle SQL Connector for Hadoop - OSCH
- Oracle „External Table“ mit einem eigenen Adapter über die „PREPROCESSOR“ Anweisung des „External Table“ Kommandos
- Oracle heterogeneous Gateway mit einen JDBC Treiber auf z.B. eine Hive Datenbank
- Eigene Java Klassen implementieren das MapReduce Pattern in der Oracle Datenbank und lesen der Daten auf dem HDFS

Für das „SQL*Loader Prinzip“, bei dem die Daten in die DB geladen werden, steht uns zur Verfügung:

- Der Oracle Loader for Hadoop – OLH
- Der SQL Loader Apache Sqoop mit seinen Erweiterungen
- Eigene Java Klassen implementieren das MapReduce Pattern und lesen die Daten in die DB ein

Welche Lösung für den Hadoop Audit Vault wählen?

Für unseren Hadoop Audit Vault müssen nun zwei Aufgaben gelöst werden:

Auf der einen Seite sollen Stammdaten aus den Datenbanken und der Inhalt der AUD$ Tabellen im HDFS gespeichert werden, auf der anderen Seite wollen wir diese Daten dann wieder über eine Oracle Datenbank auswerten, z.B. mit Apex.

Neben dem gut strukturierten Daten aus der USER$ und der AUD$ sollen aber auch die weniger gut strukturierten Daten aus der Umgebung der Datenbank mitverarbeitet werden, wie der Listener.log, Alert.log und der SYS Log der Maschine in dem Vault..

Zum Glück liegen die Log-Daten der Datenbankumgebung mit dem Diag Mechanismus der Datenbank als XML Daten vor und lassen sich relativ leicht einlesen.

Vom SYS Log der Maschine interessieren uns nur User-relevante Daten, die entweder aus „/var/log/messages“ geparst oder gleich als separate Log-Ereignisse gelesen werden können.

Das richtige Werkzeug für jede von diesen Anforderungen zu wählen, ist leider auf Grund der vielen Möglichkeiten im Hadoop Ökosystem nicht so ganz trivial.

Auch ist hier noch viel Bewegung / Neuentwicklung im Markt. Die Werkzeuge ändern sich oft recht stark von Release zu Release. Als ein größerer Nachteil der hohen Agilität im NoSQL Markt ist hier auch die oft nur wenig gepflegte Rückwärtskompatibilität des ganzen Werkzeugs bei der Auswahl im Auge zu behalten. Schnell kann hier ein sorgsam entwickeltes Script nicht mehr funktionieren, weil sich Parameter mal eben von einer Release auf das andere komplett geändert haben.

Kann auf das kostenpflichte Oracle Feature „Oracle Big Data Connectors” zurückgegriffen werden, vereinfachen sich einige Aufgaben in einigen Punkten.

Der Connector muss allerdings auf Basis der Prozessoren in einem Hadoop Clusters lizensiert werden, d.h. für jeden Prozessor in einem Cluster muss der Kunde eine Lizenz erwerben.

Selbst wenn hier die Oracle Metrik für einen Prozessor zur Anwendung kommen kann, ist naturgemäß die Anzahl von Prozessoren in einen Hadoop Cluster nicht zu vernachlässigen. Wächst das Cluster mit der Zeit, wird der korrekte Nachweis der Lizenzen gegenüber Oracle zu einem nicht mehr trivialen Problem. Zumal heute Prozessoren von Tag zu Tag mehr Cores aufweisen, jeder neue Server kann so zu einer ungewollten Kostenexplosion führen.

So verständlich der Wunsch des Herstellers nach eine nachvollziehbaren Lizenzmetrik auch ist, passt das Modell nicht zum generellen Architekturkonzept eines Hadoop Clusters mit seiner gewünscht hohen Anzahl von einzelnen Einheiten (CPU’s), die noch dazu bei Bedarf dazu geschaltet werden können, um eine im Ansatz unbegrenzte Skalierung zu erreichen.

Haben wir aber diese Hürde mit unseren Kaufleuten überwunden kann eine erste Architektur für unsere Aufgaben so aussehen:

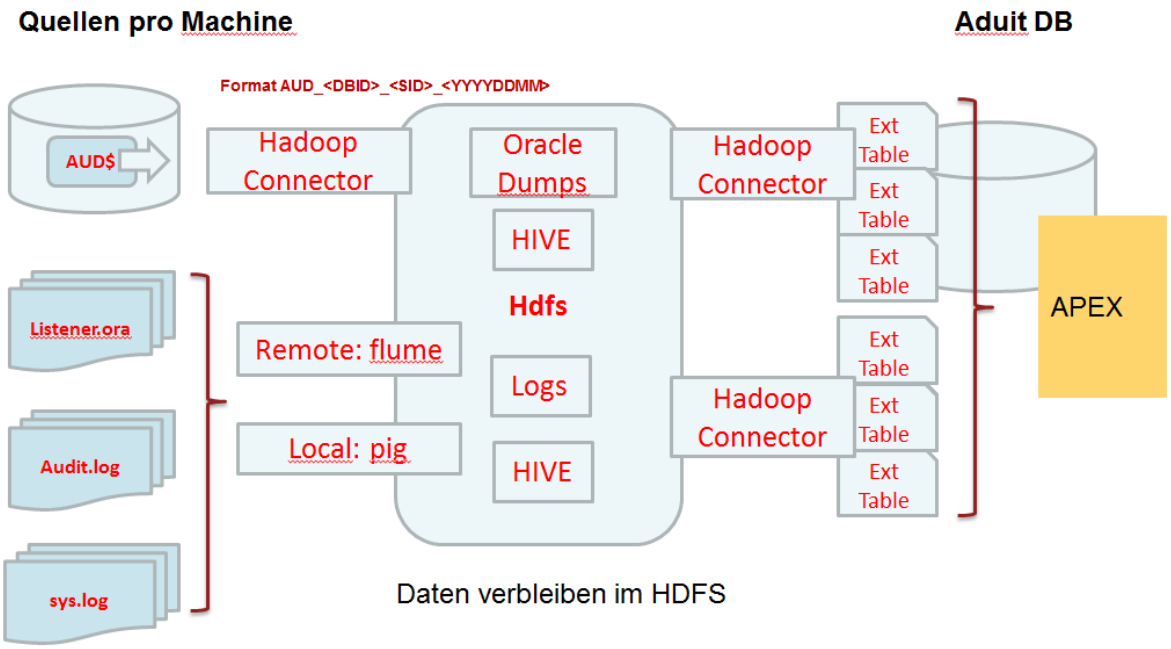

Abb. 4 : Eine Hadoop Audit Valut Lösung mit dem Oracle Big Data Konnektoren

Erste Erfahrungen mit dem „Oracle SQL Connector for HDFS“ - OSCH

Mit dem „Oracle SQL Connector for HDFS" wird eine „External Table“ in der Oracle Datenbank mit Daten auf einem HDFS Cluster verbunden. Die Daten werden als Datenstrom vom HDFS Filesystem gelesen und in der Oracle Datenbank wie gewohnt bei einer „External Table“ verarbeitet.

Mit einem Kommandozeilenwerkzeug wird dazu die „External Table“ per SQL Befehlt in der Oracle Datenbank angelegt und pro Datendatei im HDFS ein Location File im Dateisystem im Verzeichnis des DIRECTORY der „External Table“ abgelegt. Es können DataPump „External Table“ Exports, Hive Tabellen und strukturierte Text Dateien ausgelesen werden.

Wird nun auf die „External Table“ zugegriffen, wird die PREPROCESSOR Anweisung der der Tabelle ausgeführt eine Java Klasse liest die Informationen im den Location Files und erzeugt eine Datenstrom auf die entsprechenden Dateien im HDFS.

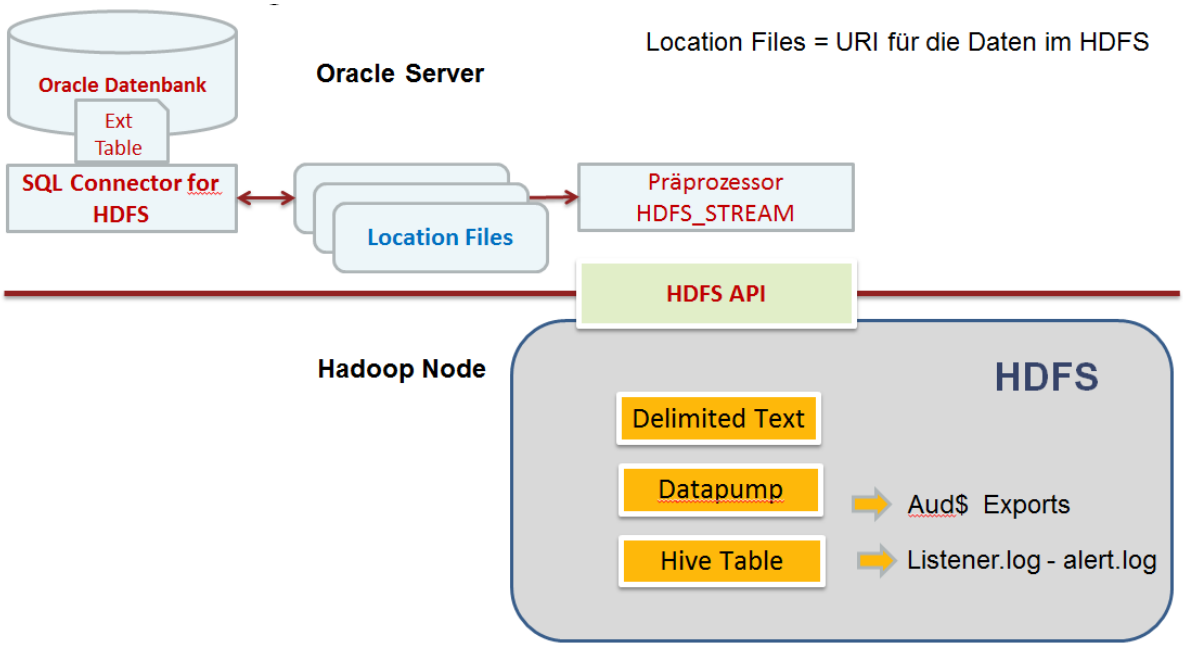

Abb.5 : Oracle SQL Connector for HDFS - OSCH

Beispiel für einen Befehlt, um eine solche „External Table“ anzulegen:

```
hadoop jar $OSCH_HOME/jlib/orahdfs.jar                          \
oracle.hadoop.exttab.ExternalTable                              \
-D oracle.hadoop.exttab.hive.tableName=ext_ora_audit_gpi        \
-D oracle.hadoop.exttab.hive.databaseName=default               \
-D oracle.hadoop.exttab.sourceType=hive                         \
-D oracle.hadoop.exttab.tableName=ext_ora_audit_gpi_hive        \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//localhost:1521/orcl     \
-D oracle.hadoop.connection.user=scott                          \
-D oracle.hadoop.exttab.defaultDirectory=HADOOP_EXT_TABS        \
-createTable
```

In der Datenbank wird die passende „External Table“ angelegt, der interessante Teil des DDL Statements für die „External Table“ ist die Präprozessor Anweisung und die Angabe der Location Dateien mit den Informationen (im XML) Format, wo auf dem HDFS die eigentlichen Daten zu finden sind.

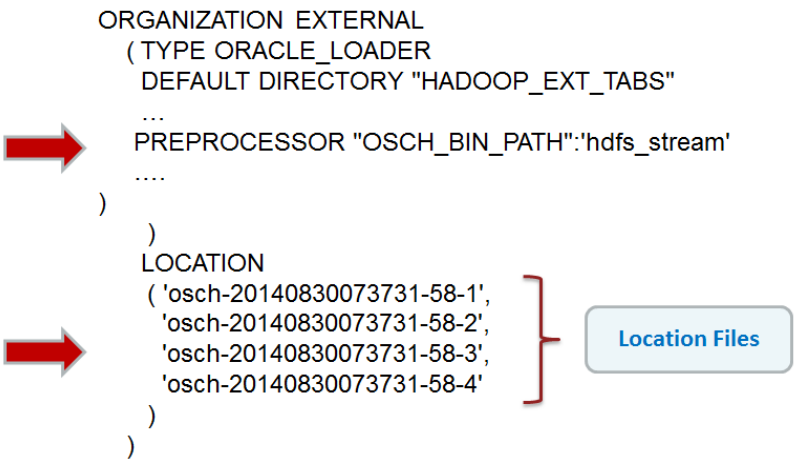

Abb.6 : Das erzeugt DDL Statement mit der Präprozessor Anwendung und den Verweis auf die Location Dateien

Die Performance hängt davon ab, wie schnell die Daten aus dem HDFS gelesen werden können. Eine performante Netzwerkanbindung an das Hadoop Cluster ist dabei unbedingte Voraussetzung, da die Daten zur Datenbank gestreamt und erst dort verarbeitet werden.

In unserem Beispiel wollen wir die Daten aus den Hive Tabellen lesen, in die zuvor per FLUME die Logfiles der DB Umgebung geschrieben wurden.

In den ersten Schritten ergaben sich einige kleinere Problem bei der Umsetzung.

Für die Verwendung von Hive Tabellen müssen zum Beispiel die Hive Klassen mit dem Klassenpfad der Umgebung aufgenommen werden, das ist mir nur über einen fest kodierten Eintrag in der „hadoop.sh“ Datei gelungen.

Auch darf die Hive Tabelle darf keine Spalte von Typ Binary enthalten.

Auch konnte das Kommandotool die Darstellung der Null Spalte nicht richtig interpretieren, in meine zuvor mit Sqoop auf das HDFS importieren Daten wird null als 'null' dargestellt, d.h. der im SQL Statement für die externe Tabelle erzeugte Format Anweisung „TIMESTAMP" CHAR DATE_FORMAT DATE MASK 'YYYY-MM-DD' NULLIF "TIMESTAMP"=0X'5C4E'“ (/t) muss dann auf „TIMESTAMP" CHAR DATE_FORMAT DATE MASK 'YYYY-MM-DD' NULLIF "TIMESTAMP"=0X'276E756C6C27'“ angepasst werden.

Erste Erfahrungen mit den Oracle Loader for Hadoop - OLH

Mit dem Oracle Loader for Hadoop - OLH werden Daten aus dem Hadoop Cluster in die Oracle Datenbank geladen.

Dabei wird das MapReduce Pattern eingesetzt, d.h. zuerst wird ein Hadoop Job konfiguriert, der die auf dem Hadoop Cluster hinterlegten Daten auswertet und je nach Bedarf in ein für Oracle lesbares Format transferiert (zum Beispiel in des Oracle External DataPump Format). Diese Daten können offline zur Verfügung gestellt oder gleich direkt in die Datenbank geladen werden.

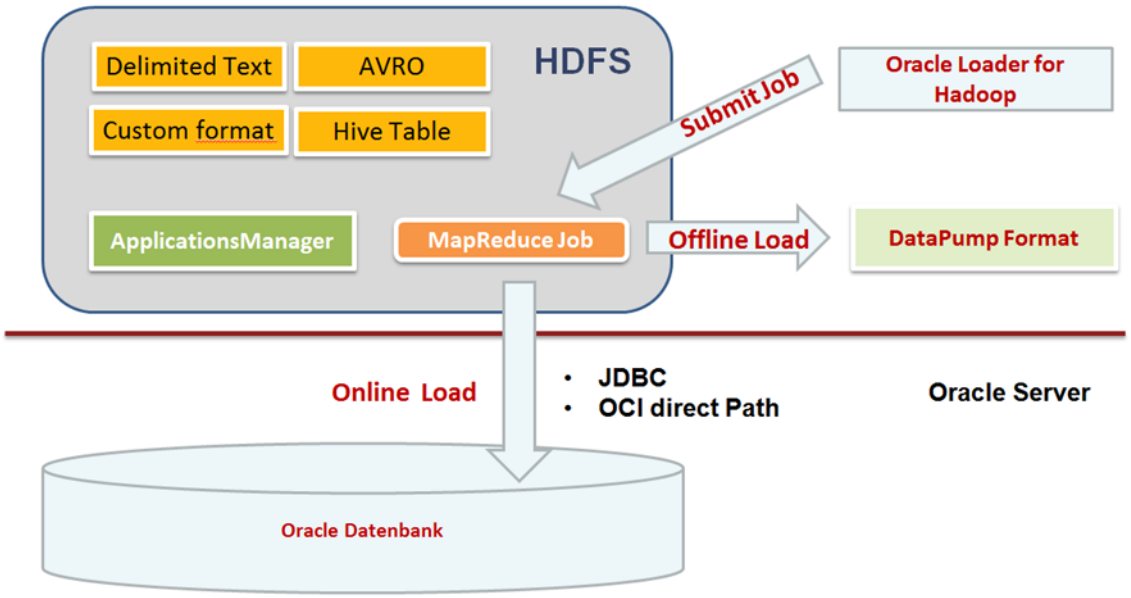

Abb.7 : Übersicht über den „Oracle Loader for Hadoop" – OLH

Um einen solchen Job zu erzeugen, werden zuvor in einer XML Datei alle Parameter definiert und dann der Job gestartet.

Einsatz von Open Source Werkzeugen

Auch ohne den Oracle Adapter kann eine Lösung auf Basis der Werkzeuge rund um das Hadoop Cluster umgesetzten werden:

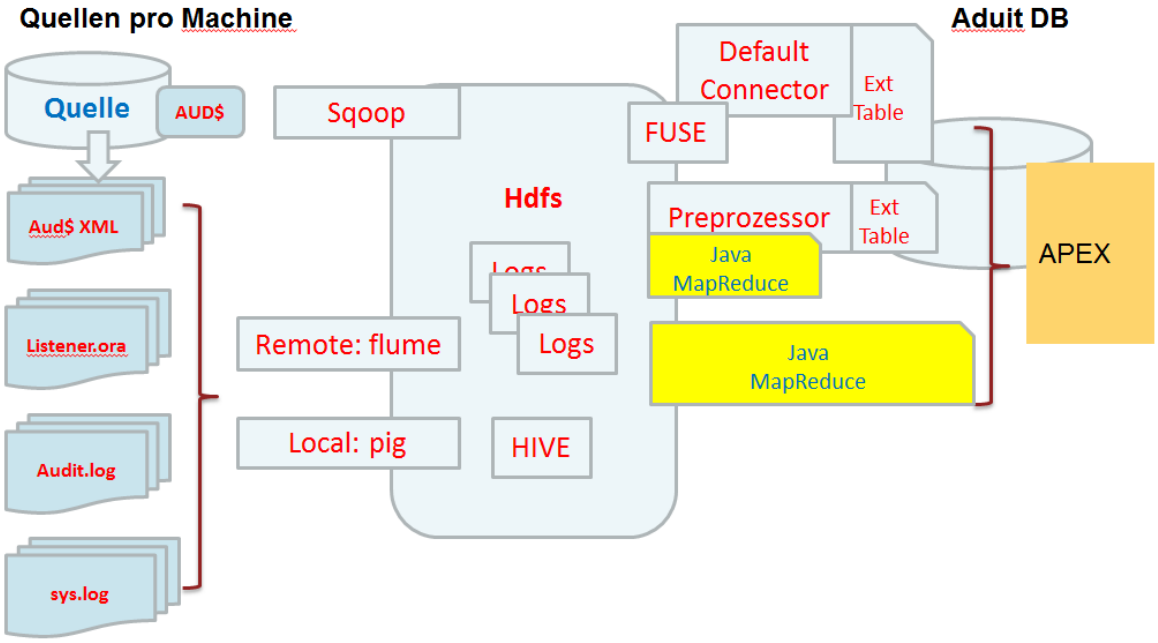

Abb. 8 : Eine Hadoop Audit Valut Lösung mit nur mit OpenSource Komponenten

Mit dem SQL*Loader Werkzeug Apache Sqoop kann ähnlich wie mit dem Oracle Loader gearbeitet werden, mit Sqoop können Daten in beiden Richtungen, auf das HDFS oder in die Datenbank transferiert werden.

Natürlich bietet sich auch noch die Lösung an, über XML in das Filesystem der Oracle Datenbank die Audit Log Daten zu schreiben und dann komplett die Input Verarbeitung über Apache Flume durchzuführen.

Apache Flume kann als eine Art syslog Dämon verstanden werden, der Daten über eine Pipe-Architektur transformieren und in das HDFS schreiben kann.

Apache Scoop 2 als SQL*Loader einsetzen

Sqoop 2 besteht

- aus einem eigentlichen Server, der die Daten nach oder vom HDFS transferiert,
- der Metadatenbank und
- einen Client, um die Jobs zu konfigurieren.

Per JDBC werden die Daten aus der relationale Datenbank gelesen und dann auf das HDFS geschrieben.

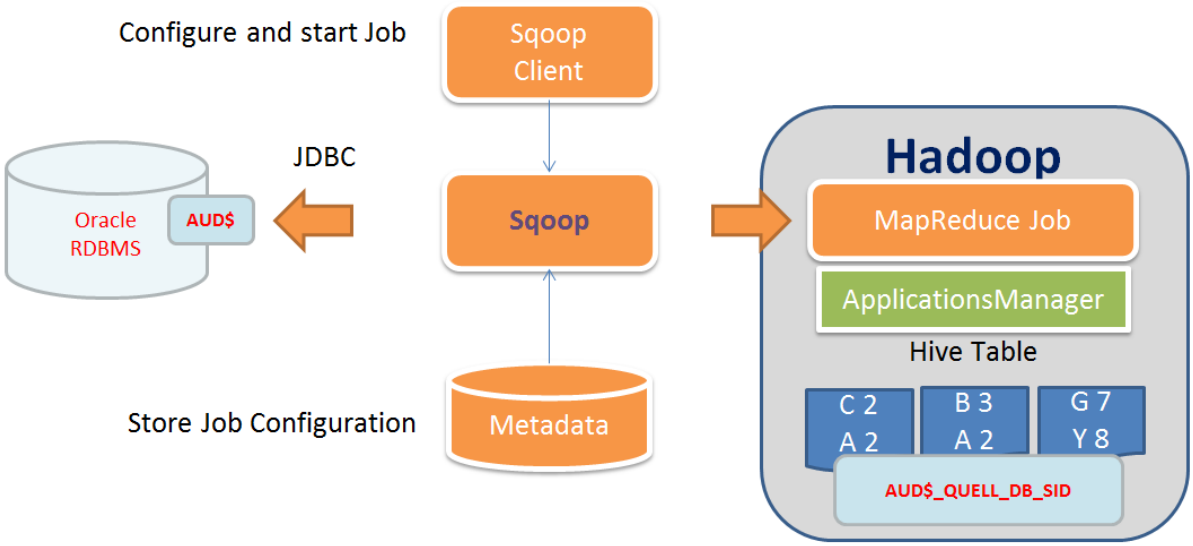

Abb. 9 : Apache Sqoop Übersicht

Leider stellt sich bei den ersten Tests heraus, dass Sqoop nicht ganz „SQL Injection“ save ist.

Sqoop versucht, die Daten zu partitionieren. Dazu kann eine Spalte angegeben werden, über die dann per SQL Statements Teile der Tabelle gelesen werden. Allerdings prüft Sqoop nicht, ob in den Daten Zeichen sind, die bei der Ausführung mit interpretiert werden.

Ein Versuch, die AUD$ - partitioniert über den User - zu lesen, führte zu einem Fehler:

```
SELECT * FROM SYSTEM.AUD$ WHERE ''TESTER' <= USERID AND
USERID < '+����'
```

```
014-08-30 16:46:51,537 INFO [main] org.apache.sqoop.connector.jdbc.GenericJdbcImportExtractor: Using query: SELECT * FROM SYSTEM.AUD$ WHERE ''TESTER' <= USERID AND USERID < '+6666'
014-08-30 16:46:53,650 INFO [main] org.apache.sqoop.job.mr.SqoopMapper: Stopping progress service
014-08-30 16:46:53,650 INFO [main] org.apache.sqoop.job.mr.SqoopOutputFormatLoadExecutor: SqoopOutputFormatLoadExecutor::SqoopRecordWriter is about to be closed
014-08-30 16:46:53,659 INFO [OutputFormatLoader-consumer] org.apache.sqoop.job.mr.SqoopOutputFormatLoadExecutor: Loader has finished
014-08-30 16:46:53,660 INFO [main] org.apache.sqoop.job.mr.SqoopOutputFormatLoadExecutor: SqoopOutputFormatLoadExecutor::SqoopRecordWriter is closed
014-08-30 16:46:53,661 WARN [main] org.apache.hadoop.mapred.YarnChild: Exception running child : org.apache.sqoop.common.SqoopException: MAPRED_EXEC_0017:Error occurs during extrac
        at org.apache.sqoop.job.mr.SqoopMapper.run(SqoopMapper.java:101)
        at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:764)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:340)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:168)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1548)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:163)
aused by: org.apache.sqoop.common.SqoopException: GENERIC_JDBC_CONNECTOR_0002:Unable to execute the SQL statement
        at org.apache.sqoop.connector.jdbc.GenericJdbcExecutor.executeQuery(GenericJdbcExecutor.java:63)
        at org.apache.sqoop.connector.jdbc.GenericJdbcImportExtractor.extract(GenericJdbcImportExtractor.java:50)
        at org.apache.sqoop.connector.jdbc.GenericJdbcImportExtractor.extract(GenericJdbcImportExtractor.java:31)
        at org.apache.sqoop.job.mr.SqoopMapper.run(SqoopMapper.java:98)
        ... 7 more
aused by: java.sql.SQLSyntaxErrorException: ORA-00920: invalid relational operator
```

Abb.10 : Apache Sqoop “SQL Injection” Fehler

Oracle „External Table“ über mit Hilfe der „PREPROCESSOR“ Anweisung anbinden

Über die PREPROCESSOR Anweisung beim Anlegen einer Oracle „External Table“ kann eine Shell Datei aufgerufen werden.

Als Aufrufparameter erhält die Shell Datei dabei die LOCATION Angabe der „External Table“.

In der Datei wiederum ist der Entwickler frei, alles zu entwickeln und aufzurufen, was notwendig ist. Das Ergebnis wird auf STANDARD OUT ausgegeben und von dort von der DB verarbeitet. Pro Datei in der LOCATION Angabe wird die Shell Datei dabei einmal aufgerufen.

Bei der Entwicklung ist zu beachten, dass die Shell Datei im Scope eines DB Server Prozess aufgerufen wird und damit nur die in diesem Scope gültigen Rechte und Umgebungsvariablen zur Verfügung stehen. Entsprechend genau müssen Pfade und Umgebungsvariablen bei Bedarf gesetzt werden.

Ein einfaches Beispiel für das Lesen einer Datei auf dem HDFS:

Das Schell Kommando:

```
vi readHadoopTab.sh

#!/bin/sh
#Laufzeitumgebung setzen
export PATH=$PATH:/bin:/sbin/:/usr/bin
export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
export HADOOP_CONF_DIR=/etc/hadoop/conf.bigdatalite
# Übergabe Parameter Datei auslesen und Inhalt als Parameter verwerten
FILENAME=`/bin/cat $1`
#Mit hdfs Bordmitteln die Datei auswerten
/usr/lib/hadoop-hdfs/bin/hdfs dfs -cat $FILENAME
```

„External Table“ anlegen:

```
CREATE TABLE MY_HADOOP_EXT(
 wert1 VARCHAR2(2000)
 ,wert2 VARCHAR2(2000)
 ,wert3 VARCHAR2(2000)
 )
 ORGANIZATION EXTERNAL
 ( TYPE ORACLE_LOADER
  DEFAULT DIRECTORY HADOOP_EXT_TABS
  ACCESS PARAMETERS
  ( RECORDS DELIMITED BY NEWLINE
   PREPROCESSOR HADOOP_EXT_TABS:'readHadoopTab.sh'
   SKIP 0
   LOGFILE HADOOP_EXT_TABS:'data_load.log'
   BADFILE HADOOP_EXT_TABS:'data_load.bad'
```

```
   NODISCARDFILE
   FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
   MISSING FIELD VALUES ARE NULL
  )
  LOCATION (HADOOP_EXT_TABS:'hadoop_locator.dat')
 )
 REJECT LIMIT 0
 NOPARALLEL
 NOMONITORING
 /
```

In der `hadoop_locator.dat` steht der Pfad zur Datei im HDFS:
`/tmp/test_ext_data.dat`

Wird nun mit einem „`SELECT * FROM scott.MY_HADOOP_EXT`" die Tabelle abgefragt, wird die Shell Datei „`readHadoopTab.sh`" mit dem Parameter „`hadoop_locator.dat`" aufgerufen, liest den Inhalt der Datei aus und ruft dann mit „`/usr/lib/hadoop-hdfs/bin/hdfs dfs –cat /tmp/test_ext_data.dat`" die Daten vom HDFS ab.

Wichtig ist es natürlich, dass dann auf dem DB Server ein vollständiger Hadoop Client mit konfiguriertem Zugriff auf das HDFS zur Verfügung steht.

Oracle „External Table“ über FUSE anbinden

Über das FUSE Projekt kann das Hadoop HDFS als Filesystem an den Datenbank Host direkt angebunden und damit mit normalen I/O Operation und allen Linux Tools angesprochen werden.

Damit steht die klassischen Oracle „External Table“ Methode zur Verfügung ohne eine extra Lizenz zur Verfügung.

Allerdings muss mit deutlich höheren CPU Bedarf für diese Lösung gerechnet werden.

Apache Flume um die Logfiles einzulesen

Mit Apache Flume besteht die Möglichkeit, Log Events einzusammeln und zentral auf dem HDFS abzulegen.

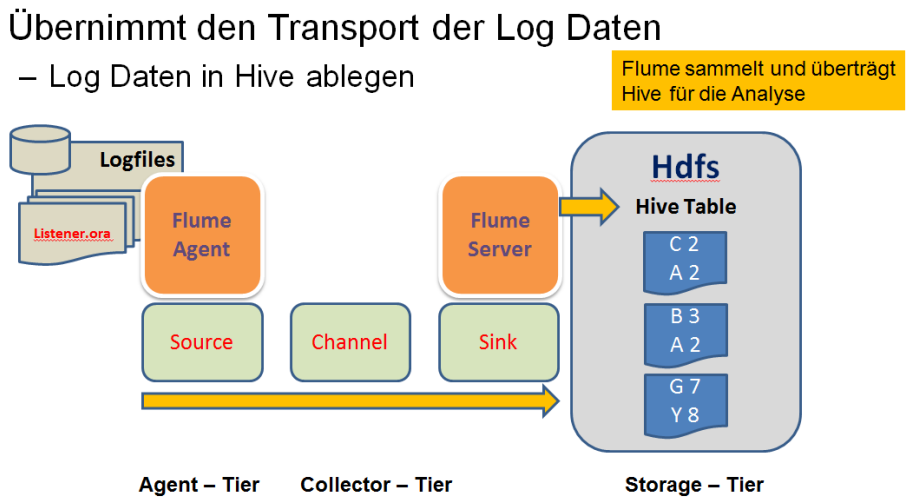

Abb. 11 : Apache Flume Übersicht

Eine „Source“ liest die Quelle ein, ein „Channel“ puffert zwischen und hilft bei der Flusskontrolle und der „Sink“ liefert die Daten dann am Ende ab, schreibt zum Beispiel die Daten auf das HDFS.

Um zum Beispiel das Listener.log einer Datenbank auszuwerten, wird auf dem DB Server ein Flume Agent für das Einlesen der Daten und auf einem der HDFS Knoten ein weiterer Agent für das Empfangen und Schreiben der Daten installiert.

Der Agent auf dem DB Server liest als Source die XML Log Datei des Listeners ein. Ein Channel im Memory sorgt für die Flusssteuerung, ein Avro Skink serialisert die Daten und versendet diese Daten zu dem Agenten auf dem HDFS Knoten. Der Agent auf dem HDFS Knoten hat eine Source, die wiederum die serialisierten Daten im Avro Format empfangen kann, einen Channel im Memory und einen Skink vom Typ HDFS. Damit werden dann die Daten direkt in das HDFS bzw. in Hive Tabelle geschrieben.

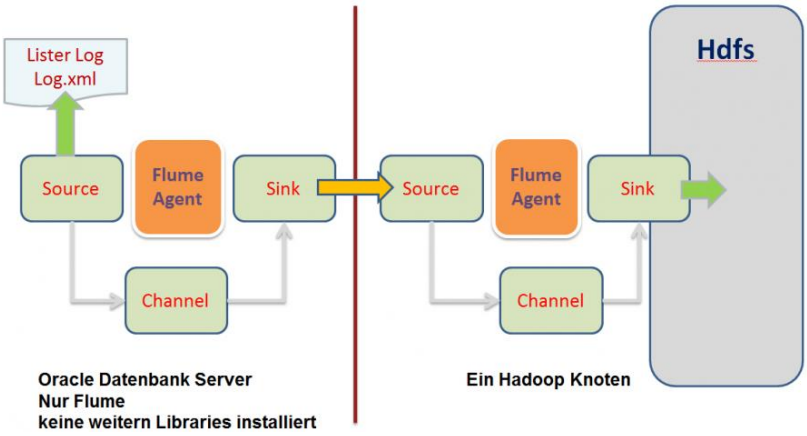

Abb.12 : Lesen des Listener Logs mit Flume, übertragen auf einen Empfänger auf dem HDFS Knoten und schreiben in das HDFS

Auswertung der Daten

Ziel ist es, über die zentrale Oracle Datenbank des Vaults das Berichtwesen zu organisieren. Jedoch sind dem bei sehr großen externen Tabellen Grenzen gesetzt.

Da per JDBC direkt auf die Hive-Tabellen zugegriffen werden kann, können auch vielen bekannte Werkzeuge verwendet werden, um die Daten direkt auszuwerten

Als Alternative bietet sich eine für Hadoop entwickelte Lösung, wie Hue, an. Neben den reinen Abfragen kann das Werkzeug auch die Verwaltung der Daten vereinfachen.

Fazit

Ein Apache Hadoop Cluster bietet sich als eine gute Lösung für die Anforderung eines zentralen Audit Vaults für Log Daten an.

Die notwendigen Werkzeuge für die Umsetzung der Anforderung sind verfügbar, die meisten Fragen relativ gut dokumentiert. Funktional steht alles zur Verfügung, was benötigt wird. Die Orchestrierung des Ganzen ist jedoch die große Herausforderung bei einem solchem Projekt.

Der damit verbunden Aufwand sollte nicht unterschätzt werden. Neben der benötigten Hardware ist der Betrieb einer größeren Cluster-Umgebung schnell eine große Herausforderung. Professionelle Lösung wie Splunk© bieten „Out of the box“ bereits alle notwendigen Features und es muss genau im Detail geklärt werden, ob sich der hohe Zeitaufwand einer Eigenentwicklung am Ende auszahlt.

Der große Vorteil der Eigenentwicklung einer Audit Valut Lösung liegt darin, mit diesem Projekt einen guten Einstieg in diese neue Technologie ohne großes Business Risiko umsetzen zu können. Die damit gemachten Erfahrungen können Eins zu Eins dazu dienen, die bestehenden DWH Umgebung zu erweitern und zu ergänzen, um auch in der Zukunft auch mit großen Datenmengen wettbewerbsfähig zu bleiben.

Kontaktadresse:

Gunther Pippèrr
GPI Consult
Schwanthalerstr. 82
D-80336 München

Telefon: +49 (0)89 53 026 418
Mobil: +49(0)171 8065113
E-Mail gunther@pipperr.de
Internet: http://www.pipperr.de/dokuwiki/doku.php?id=nosql:hadoop_integration