VerteilteWebInf Hausaufgabe 12

Gruppe 6

January 17, 2015

Aufgabe 1

Impala: open-source, massively parallel processing (MPP) SQL-Query-Engine

- geringe Latenz: verteilte Architektur auf Daemon-Prozessen aufgebaut, die für Anfrageausführung zuständig sind
- hohe Nebenläufigkeit
- komplett neu geschriebenes System in C++/Java. Die meisten anderen Engines basieren auf Postgres
- Benutzt Hadoop Standard Komponenten
- HDFS als darunterliegender Speichermanager optimal
- sehr schnell:
  single-user queries: bis zu 13-mal schneller als alternative Systeme; durchschnittlich 6.7-mal schneller
  multi-user queries: durchschnittlich bis zu 18xschneller
- läuft auf hunderten Rechnern in Hadoop-Cluster; unabhängig von darunterliegender Speicherarchitektur
- drei Services: Query Planner, Query Coordinator, Query Executor (alle Daemons können in allen Rollen auftreten)
- Datenlokalität: jeder Knoten akzeptiert Anfragen und führt sie aus; Synchronisierung parallel (keine RPCs; Push der wichtigen Informationen an alle Interessenten (Subscribers)
- unterstützt Standards: z.B. JDBC/ODBC, Authentikation über Kerberos/LDAP, SQL
- Spalten können mit "PARTITIONED" partitioniert werden und mit LOCATION kann der hdfs path angegeben werden.
- als Dateiformat wird RCFile, Avro (binäres Format) und Parquet unterstützt. Weiters können Partitionen innerhalb einer Tabelle verschiedene Formate haben.

Frontend (Java)

- Standart SQL SELECT Syntax (SQL-92, SQL-2003)
- SQL → Query Plan: Parsing, semantische Analyse, Optimierung
- Query Planner: Ein-Rechner-System-Planung → Parallelisierung (maximale Lokalität, minimaler Datentransfer): Broadcast- oder partitionierter Join; lokale Prä-Aggregation + Merge

Backend (C++)

- Codegenerierung zur Laufzeit (mit LLVM; für Code, der oft ausgeführt wird), Pipelining
- I/O mittels HDFS-Feature short-circuit local reads (lesen mit fast komplett Bandbreite)
- Speicherformate: meist Apache Parquet (hohe Kompression, hohe Effizienz): Encoding zur Laufzeit, Statistiken zur Optimierung

Resource/Workload Management

- Apache YARN: zentraler Ressourcen-Manager, volles Wissen über Lastverteilung
- eigene Implementierung: verteilte Kontrolle + Llama (für caching, Scheduling) + YARN
- Llama/YARN: alle Ressourcenanfragen an Llama; falls Ressourcen in Cache verfügbar, dann Zuteilung; ansonsten Weiterleitung zu YARN

Aufgabe 2

```
SELECT ?actor WHERE
{ ?actor <actedIn> ?movie.
  ?actor <directed> ?movie2.
}
```

bzw.

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpprop: <http://dbpedia.org/property/>

SELECT ?actor ?name
WHERE {
?actor dbpedia-owl:occupation dbpedia:Actor.
?actor dbpedia-owl:occupation dbpedia:Film_director.
OPTIONAL {?actor dbpprop:name ?name.}
} ORDER BY ?actor
```