SAS, HADOOP & YARN - BEST PRACTICES

HANS-JOACHIM EDERT, SAS SOFTWARE ARCHITEKT

AGENDA

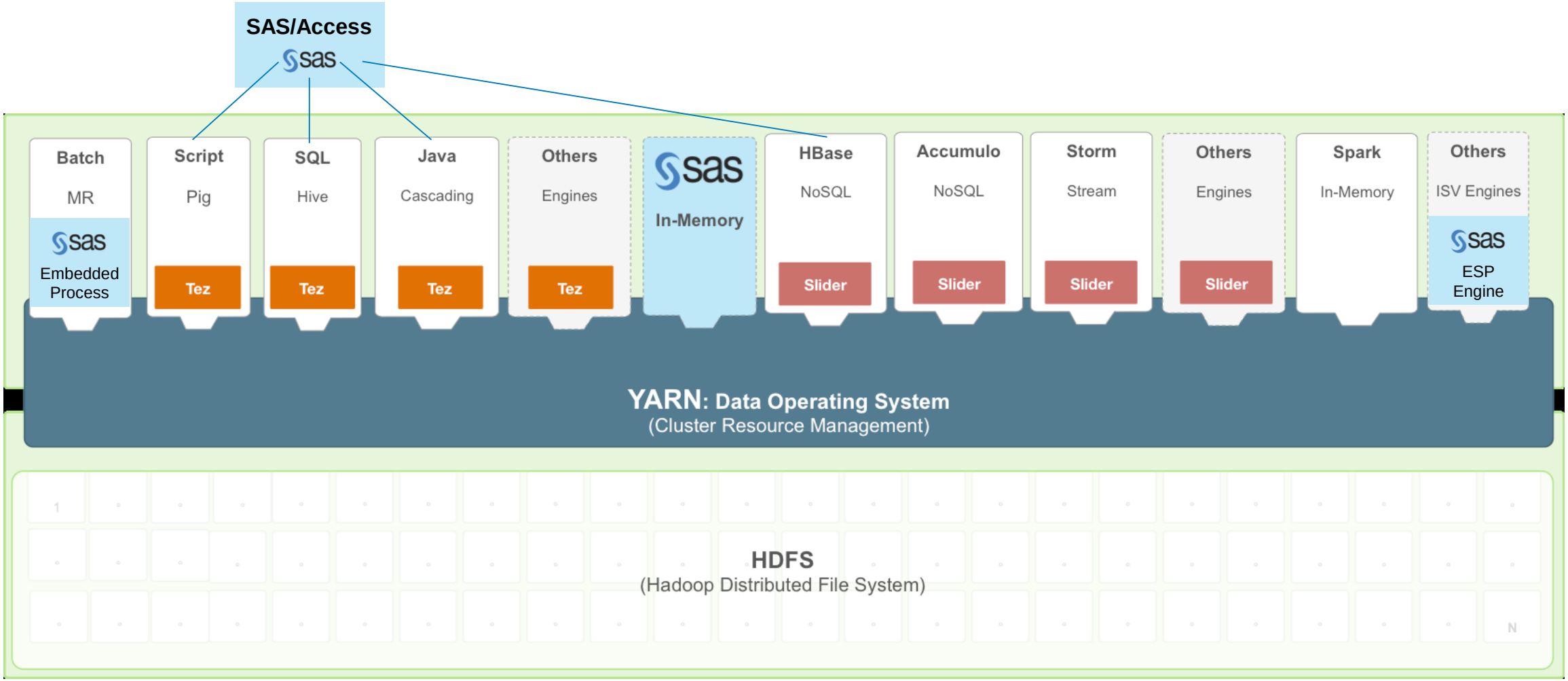
Resource-Management in Hadoop für SAS – Yarn

SQL-Abfragen optimieren – Hive

DAS HADOOP ÖKOSYSTEM

- Hadoop: Kein geschlossenes Projekt, sondern ein offenes **Ökosystem.**
  - **Dienste**: Dateisystem (HDFS), Datenbanken (Hive), Scheduler (Oozie)…
  - **Programmier-Frameworks**: MapReduce, Spark…
- Keine zentrale Steuerung (weder technisch noch organisatorisch).

SAS & HADOOP: INTEGRATIONSPUNKTE



RESOURCE-MANAGEMENT IN HADOOP FÜR SAS - YARN

YARN - YET ANOTHER RESOURCE NEGOTIATOR

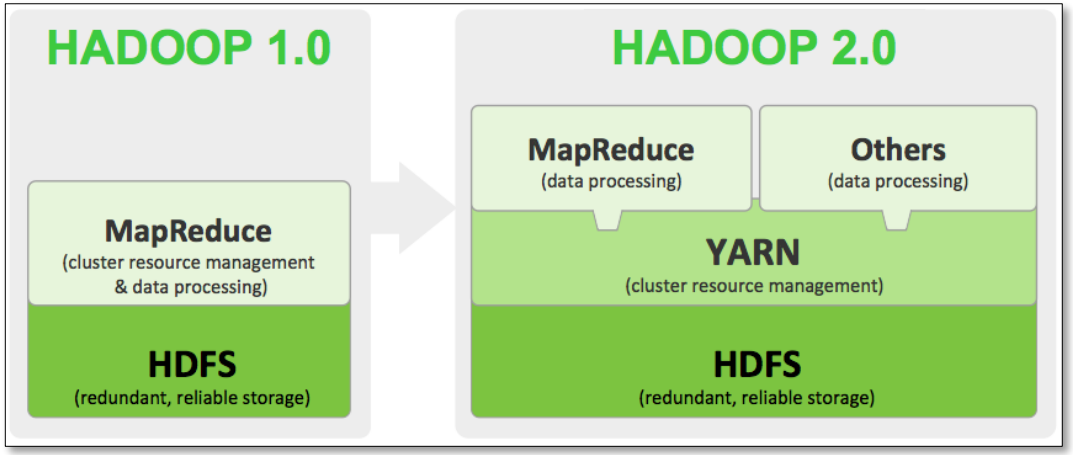
- Motivation:
  - Evolution der Hadoop Architektur von **single-purpose** Clustern zu **mixed-workload** Szenarien (batch, interaktive *und* real-time Engines).
  - **Multi-Tenancy** Umgebungen erfordern garantierte Verarbeitungskapazitäten.

- Architektur:
  - Orchestrierung durch zentralen *ResourceManager.*
  - *NodeManager* auf den Worker Nodes überwachen die Ausführung einzelner Jobs.
  - Kapazitäten werden einem Job als *Container* (kombinierte CPU Shares und RAM) zugewiesen.

Grafik entnommen: http://blog.cloudera.com/blog/2015/09/untangling-apache-hadoop-yarn-part-1/

YARN - KEIN OPTIONALES „NICE TO HAVE“ FEATURE

- Alle SAS Technologien für Hadoop integrieren sich in Yarn:
  - Zumeist „frei Haus“, da sie in native Hadoop Frameworks eingebettet sind:
    - Access to Hadoop / Impala, SAS Embedded Process (Accelerators).



- Konfigurationsbedarf für die SAS In-Memory Analytics Infrastruktur:
  - SAS koordiniert den Resourcenbedarf mit dem Yarn ResourceManager („book-keeping“), startet aber keine Yarn-Container, sondern eigene (TKGrid-)Prozesse.

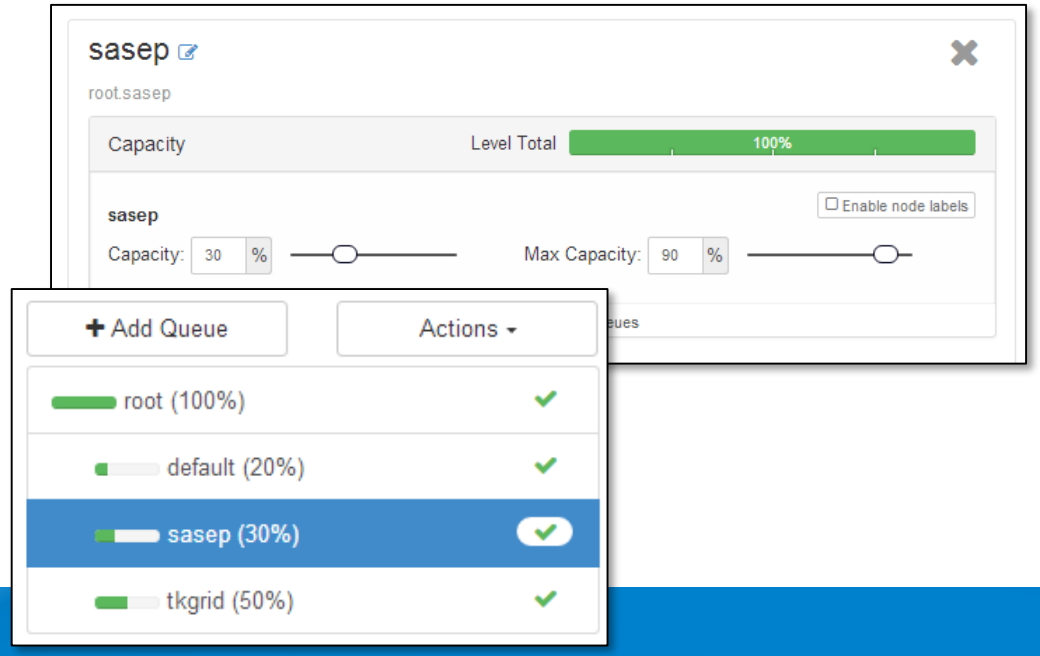
YARN - KEIN OPTIONALES „NICE TO HAVE“ FEATURE

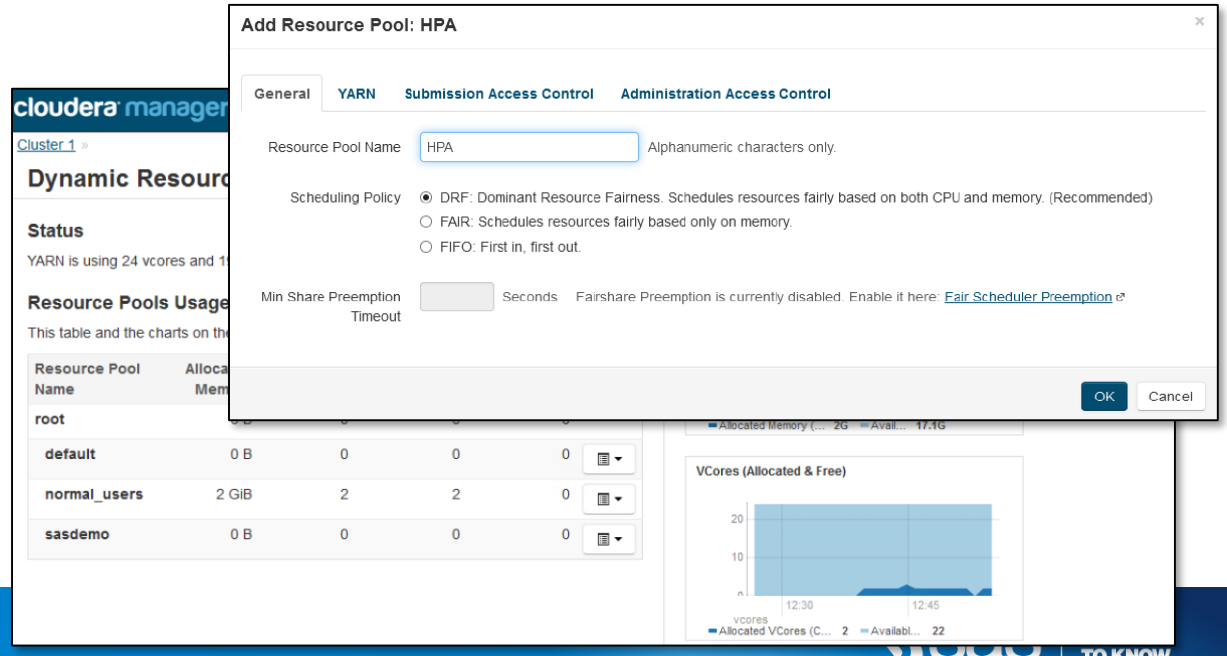
- Ohne Yarn-Integration konkurrieren Hadoop-Jobs und -Dienste ungesteuert um Resourcen („first-come-first-serve“ - „winner takes all“).

- Unerwünschte Ergebnisse:
  - Abbrüche, Fehler, nicht vorhersehbare Laufzeiten (unerklärliches Verhalten aus Anwendersicht).
  - SAS Technologien bilden komplexe Abhängigkeitsketten (z.B. Hive-Daten mit HPA Prozedur analysieren):
    - Abhängige Prozesse „verhungern“, da Vorläufer-Prozesse „steckenbleiben“.

YARN QUEUES / RESOURCE POOLS

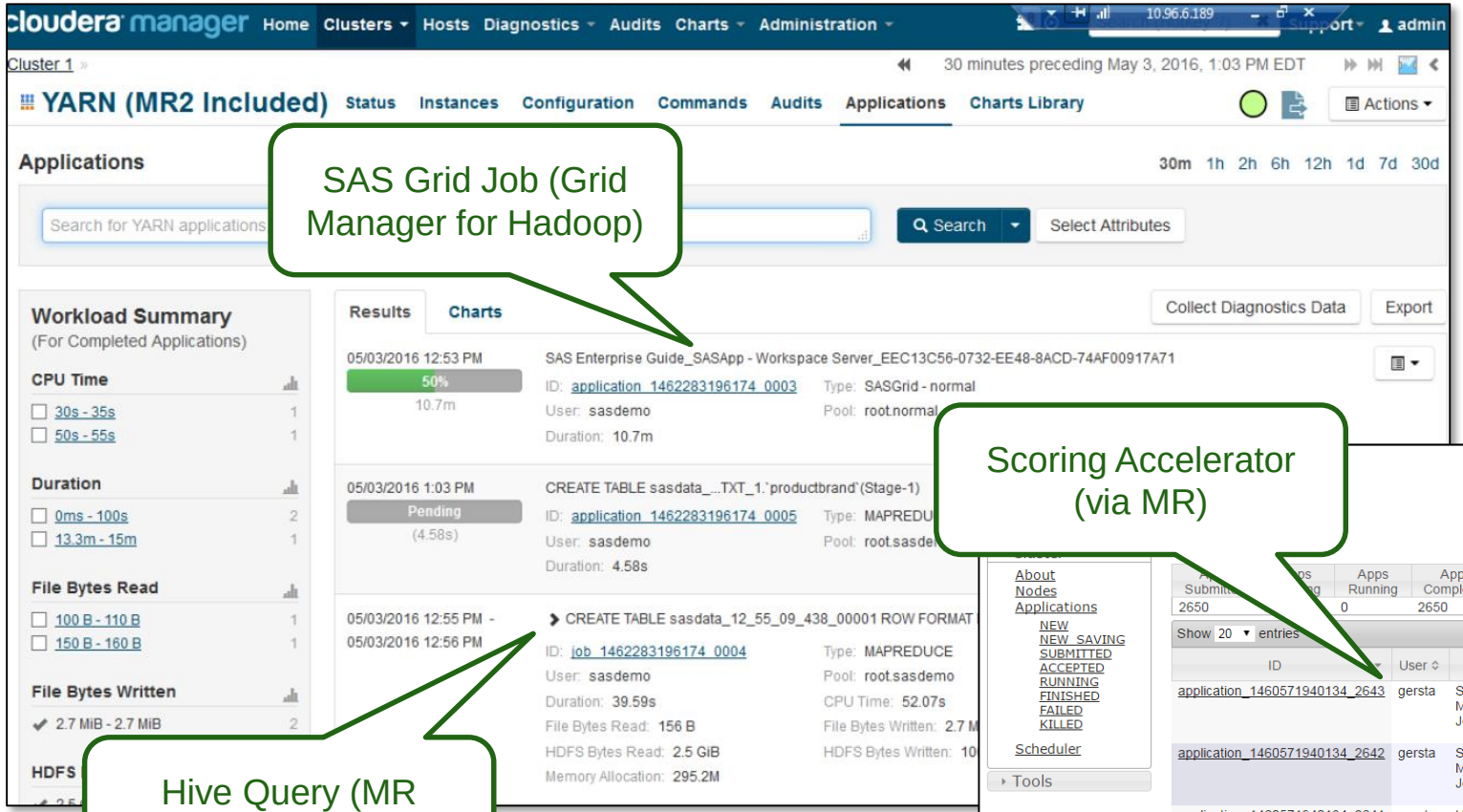
- Empfehlung: Zuweisung von garantierten Verarbeitungskapazitäten über Queues (Hortonworks) bzw. Resource Pools (Cloudera):
  - Idealerweise: separate Queues je nach SAS Workload.
  - Queues können dynamisch angelegt werden, um Hardware optimal auszunutzen:
    - D.h.: min capacity != max. capacity, Preemption aktiviert.
    - TKGrid Container unterstützen kein Preemption (→ min. capacity ist entscheidend).





RESOURCE MANAGEMENT

YARN QUEUES / RESOURCE POOLS



SAS Grid Job (Grid Manager for Hadoop)

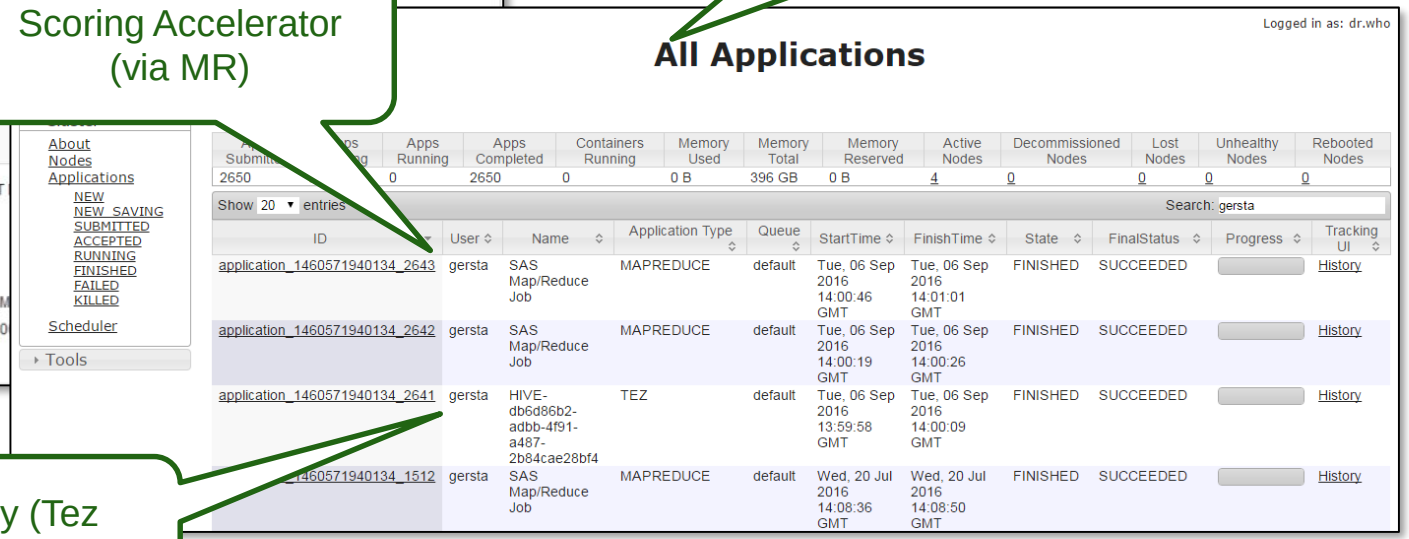
Überwachen des SAS Workloads in Cloudera …

… und Hortonworks

Scoring Accelerator (via MR)

Hive Query (MR engine)

Hive Query (Tez engine)

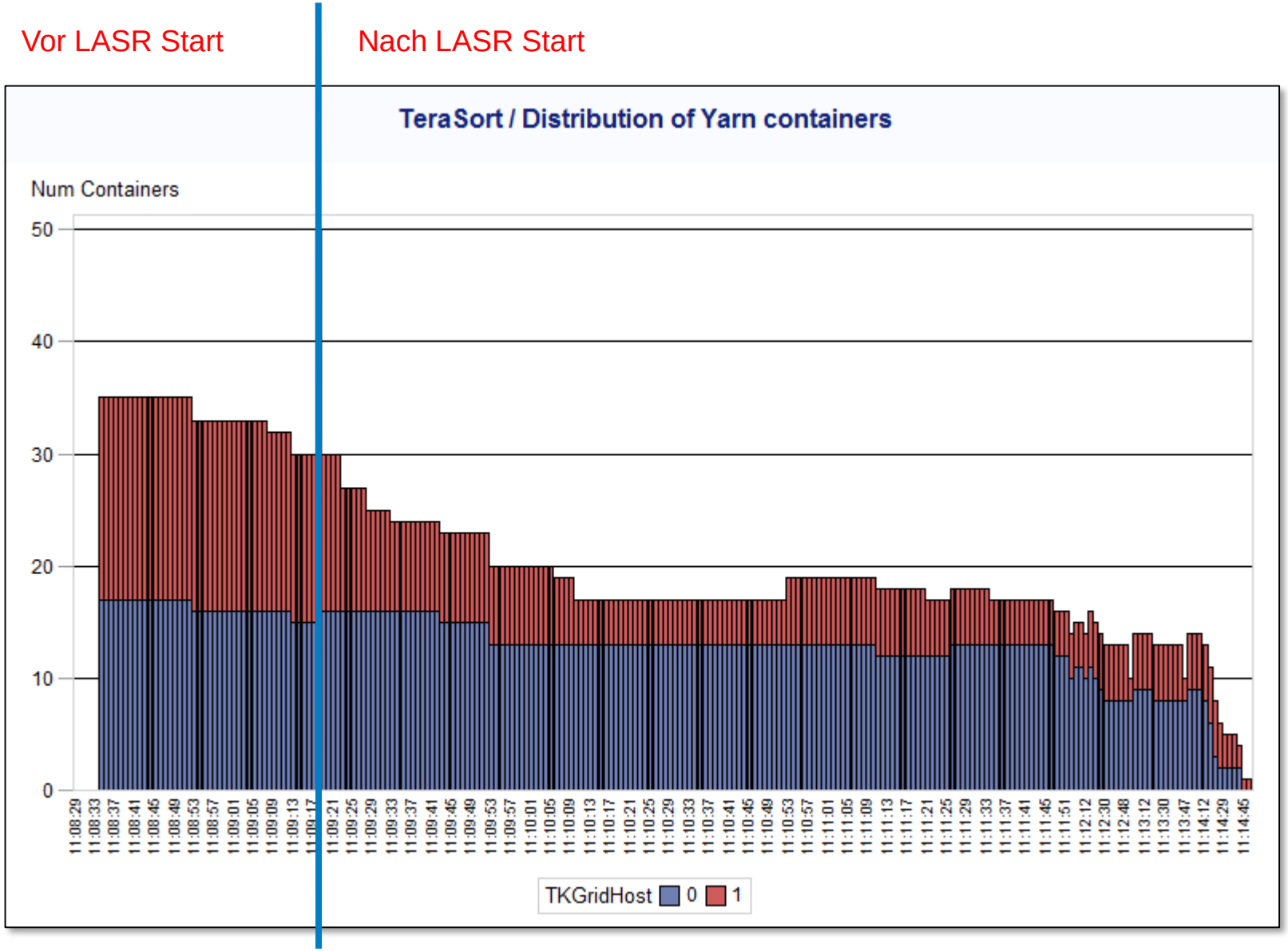


SAS TKGRID UND YARN QUEUES - TERASORT + LASR

- Diagramm zeigt den Resourcenverbrauch eines TeraSort M/R Jobs.
- Kurz nach Start des TeraSorts wurde ein LASR Server gestartet.

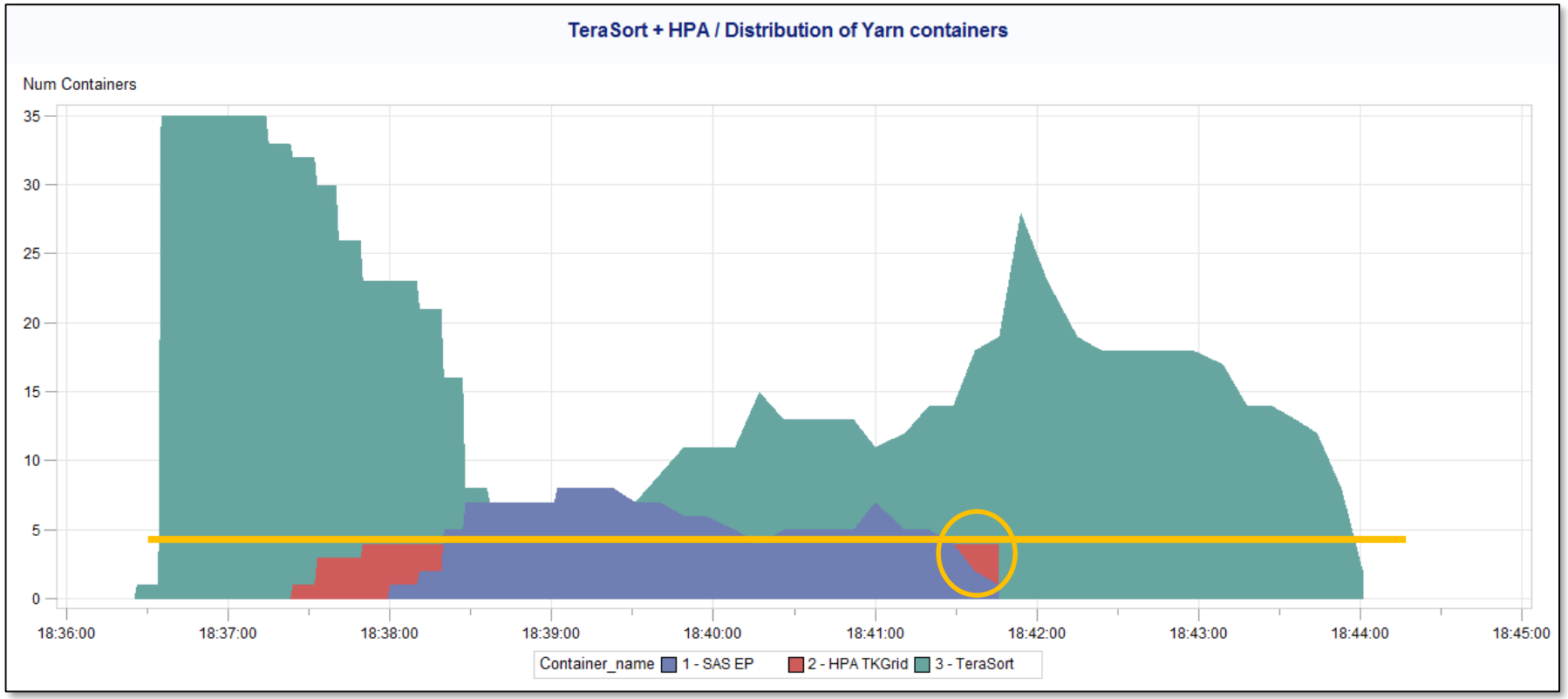
- Initial vergibt Yarn Container gleichverteilt sowohl auf TKGrid- wie auch auf Nicht-TKGrid Knoten
  - (default queue ist elastisch).

- Nach dem Start des LASR Servers werden die M/R Containers *auf den TKGrid Knoten* zurückgefahren
  - (da LASR die garantierten Resourcenzusagen seiner Queue beansprucht).



SAS TKGRID UND YARN QUEUES - TERASORT + HPA

- Diagramm zeigt den Resourcenverbrauch eines TeraSort M/R Jobs sowie einer parallel ausgeführten HPA Prozedur
- HPA Prozedur wurde kurz nach dem Terasort gestartet.
- HPA nutzt den EP um Daten aus Hadoop zu laden.



- HPA (rot) startet vor dem EP und reduziert die Anzahl Containers des TeraSort Jobs.
- EP (blau) startet danach und reduziert die TeraSort Containers noch weiter.
- HPA nutzt konstant 4 Containers (1 AM + 3 worker).
- HPA Verarbeitung startet erst, nachdem der EP beendet wurde (Oval).
- Nach Beendigung des HPA Jobs nutzt TeraSort die wieder freigewordene Kapazität.

BEISPIELHAFTE UMSETZUNG EINES MULTI-TENANCY KONZEPTS

- Jede **LASR Start / Stop / Data Load** Aktion (PROC LASR) sowie jeder Aufruf einer **HPA Prozedur** (PROC HP…) durchläuft ein Shell Skript:
  - `/opt/sas/TKGrid/resource.settings` (auf allen TKGrid Nodes).
  - Initial: alle Einträge auskommentiert, d.h. keine Yarn-Integration von LASR/HPA.
  - Verarbeitet Umgebungsvariablen, die von SAS aus gesetzt werden können.

- TKGrid-Architektur bedingt eine statische Allokation der benötigten Resourcen:
  - Reservierter Speicher ist geblockt solange LASR/HPA laufen.

- Yarn-Integration wird aktiviert durch den Export einer Umgebungsvariablen mit Angabe der benötigten Resourcen (Reservierungsanfrage):
  - `export TKMPI_RESOURCEMANAGER=„--memory <X GB> --queue <Y>“`

RESOURCE MANAGEMENT

BEISPIELHAFTE UMSETZUNG EINES MULTI-TENANCY KONZEPTS

- Jedem Mandant ist 1 LASR Server zugewiesen, zudem die Möglichkeit zum Ausführen von HPA Prozeduren.

- Mandantenbezogene SAS AppServer Kontexte definieren zulässige Resourcen-Obergrenzen:
  - `.../Lev1/<Project Application Server>/appservercontext_env.sh`

```
export TKMPI_LASR_YARN_QUEUE=tkgrid
export TKMPI_LASR_MEMSIZE_MB=10240
export TKMPI_HPA_YARN_QUEUE=tkgrid
export TKMPI_HPA_MEMSIZE_MB=10240
```

- Anpassungen an SSH Konfiguration:

```
/etc/ssh_config
SendEnv TKMPI_LASR_YARN_QUEUE
…
```

```
/etc/sshd_config
AcceptEnv TKMPI_LASR_YARN_QUEUE
…
```

BEISPIELHAFTE UMSETZUNG EINES MULTI-TENANCY KONZEPTS

- Skript nutzt dynamisch hereingereichte sowie automatisch gesetzte Umgebungsvariablen zur Reservierungsanfrage an Yarn:
  - Von SAS automatisch gesetzte Variablen:
    - TKMPI_APPNAME: Name der HP Prozedur (z.B. „tkhpsum“) oder „lasr“
    - TKMPI_INFO: Name der angeforderten LASR Aktion (Start / Stop …)

- Entscheidungslogik im Skript verwendet beide Informationsquellen:

```
if [[ "$TKMPI_APPNAME" =~ "tkhp" ]];
  export TKMPI_RESOURCEMANAGER=" --memory $TKMPI_HPA_MEMSIZE_MB --queue $TKMPI_HPA_YARN_QUEUE"
fi

if [ "$TKMPI_APPNAME" = "lasr" ]  && [ "$TKMPI_INFO" = "LASR_CREATE" ];
  export TKMPI_RESOURCEMANAGER=" --memory $TKMPI_LASR_MEMSIZE_MB --queue $TKMPI_LASR_YARN_QUEUE"
fi
```

SQL-ABFRAGEN OPTIMIEREN - HIVE

SQL-ON-HADOOP FRAMEWORKS

„Strength in Specialization“

IBM BigSQL

HIVE OPTIMIERUNG

APACHE HIVE – DAS DATA WAREHOUSE SYSTEM FÜR HADOOP

- Von Facebook entwickelt um das Arbeiten mit Daten in Hadoop für Anwender ohne Java Programmierungs-Skills zu vereinfachen.
- Kernmodul des Hadoop Ökosystems (in jeder Distribution enthalten).
- Verwendet HiveQL (SQL-Dialekt) als Abfragesprache.

Data Definition (DDL) Operations:

- CREATE, DROP, TRUNCATE, ALTER, SHOW, DESCRIBE
- VIEWS and INDEXES
- TBLPROPERTIES, PARTITIONS, FUNCTIONS

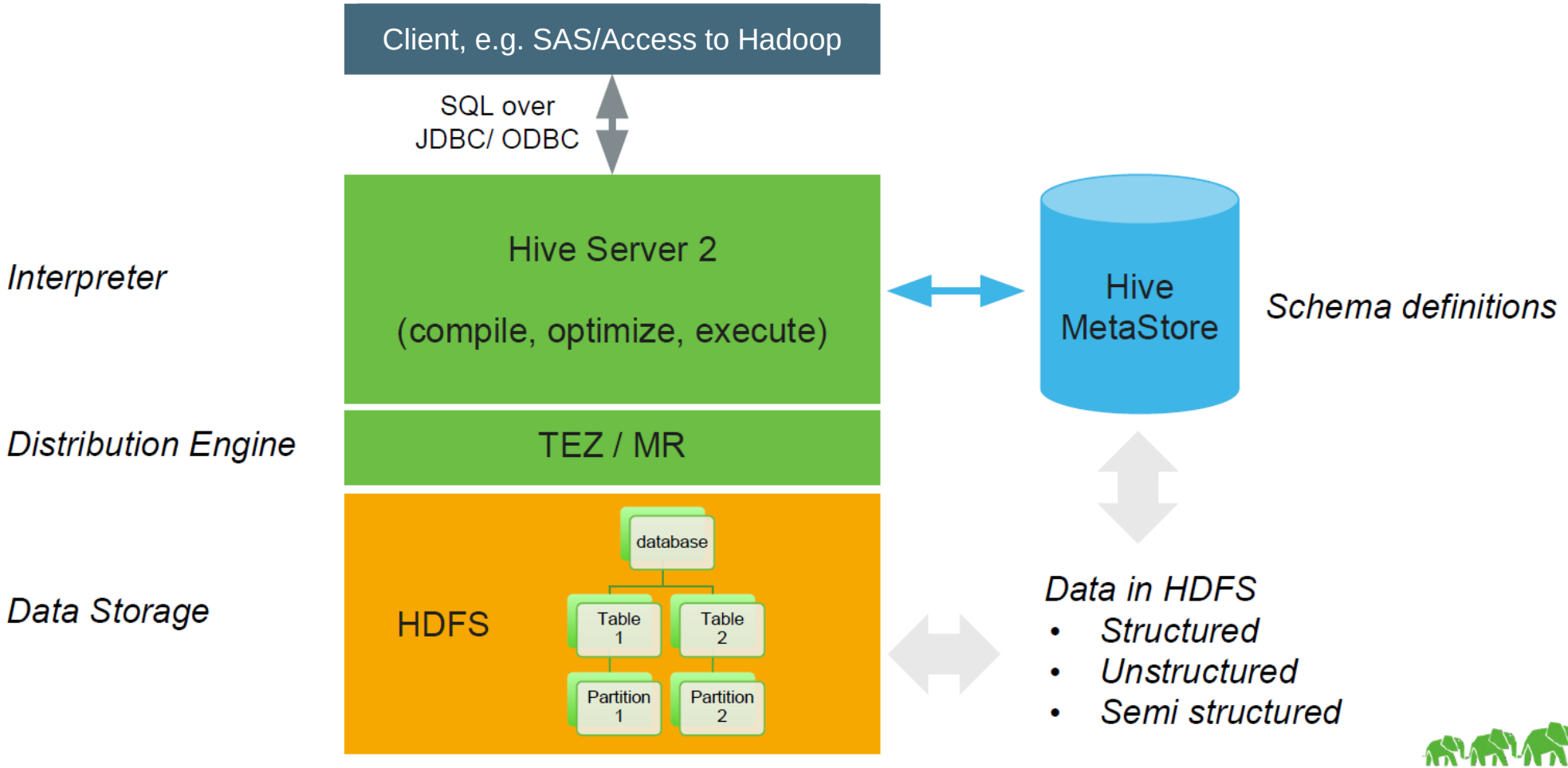
Data Manipulation (DML) Operations:

- Modify data: LOAD, INSERT, UPDATE(*), DELETE(*)
- Query data: SELECT (GROUP BY, SORT BY, JOIN, UNION, Subqueries)

(*) starting with Hive 0.14, limited functionality

SAS/Access to Hadoop könnte umbenannt werden in „SAS/Access to *Hive*“

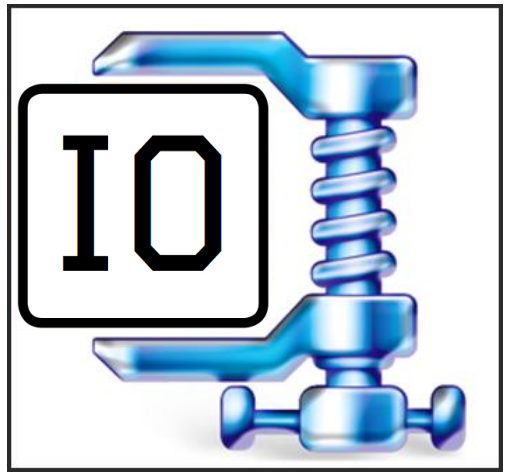
Hive Architecture



Hortonworks

Quelle: Hortonworks SQL Masterclass

HIVE OPTIMIERUNG | ANSATZPUNKTE



- Reduzieren des I/O für Hive Queries

… durch geschickte Wahl des Storage Formats:

- Hive basiert auf MapReduce → I/O-lastig.
- Storage Formate wie ORC verkleinern die Datenmengen aufgrund von spaltenorientierter Speicherung und Kompression.

… durch Partitioning und Bucketing:

- „Dynamic Partition Pruning“ - Hive führt keine Table Scans auf nicht benötigten Partitionen aus.

… durch Einsatz des Cost Based Optimizers (CBO):

- Kostenbasierte Heuristiken für Abfrage-Optimierungen auf Basis von Statistiken.

HIVE OPTIMIERUNG | HIVE STORAGE FORMATE

- Hive unterstützt zahlreiche Storage Formate:
  - ~ das Datenformat, in dem Hive-Tabellen im HDFS gespeichert sind
    - Z.B. SEQUENCEFILE, RCFILE, ORC, PARQUET, AVRO…
  - Manche Storage Formate sind spaltenbasierend und verwenden Kompression.
  - Zugriff auf Daten erfolgt über Java-basierte „SerDes“ (Reader- und Writer-Klassen).

- Je kompakter das Storage Format, desto geringer die I/O Last.

- Im Standard schreibt SAS Hive Daten im **TEXTFILE** Format (plain text):
  - Maximale Portabilität, langsame Verarbeitung.
  - Über Libname- bzw. Dataset-Optionen sollte das Storage Format geändert werden (`DBCREATE_TABLE_OPTS=„stored as ORC“` usw.).

SAS/ACCESS TO HADOOP

Vergleich zwischen verschiedenen Storage Formaten.

PROC MEANS ist eine der SAS Prozeduren, die in Hadoop ausgeführt werden.

HIVE STORAGE FORMATS

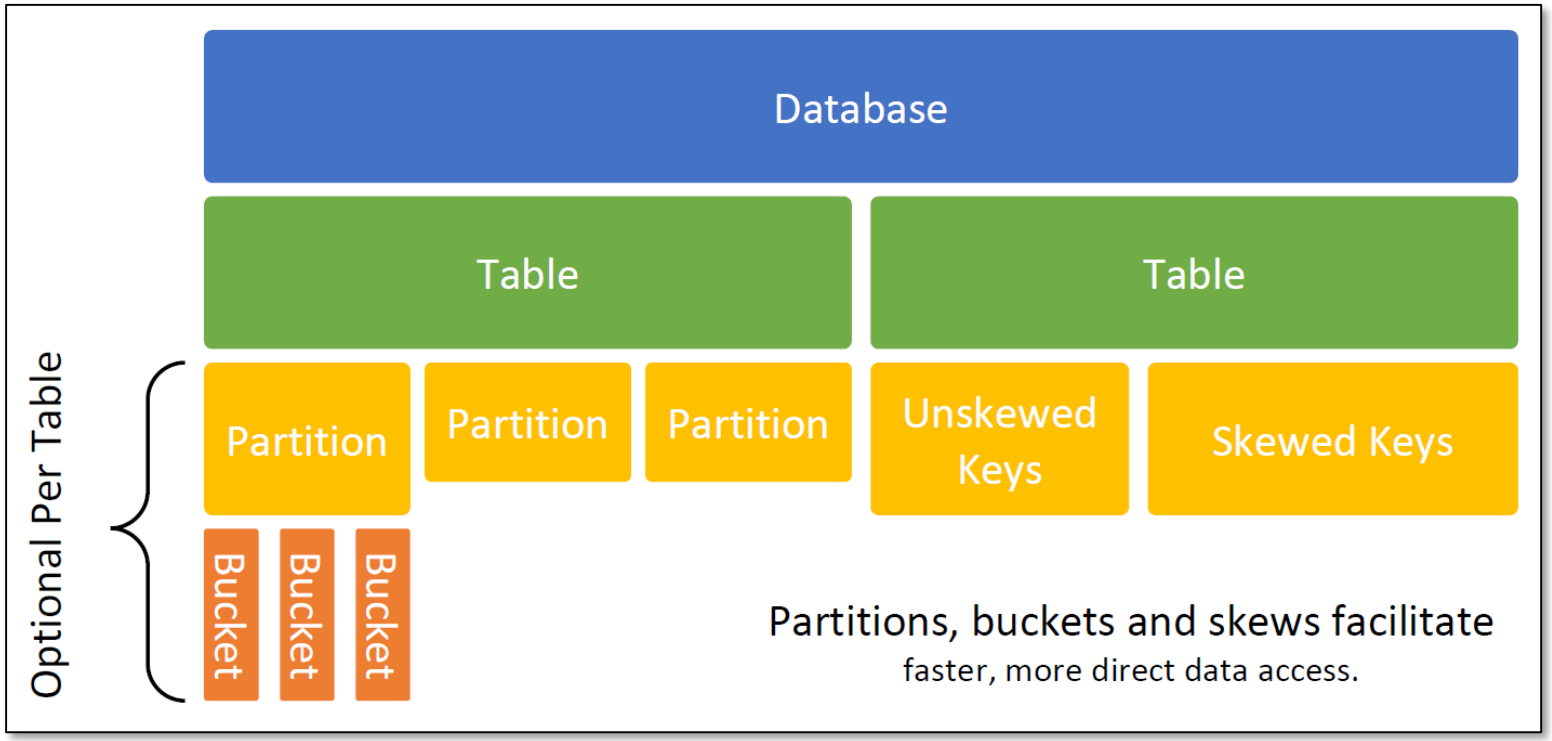
```
PROC MEANS DATA=mybase.bayarea_sas
        FW=12 PRINTALLTYPES CHARTYPE NWAY
        VARDEF=DF N NMISS MIN MAX RANGE SUM MEAN STD STDERR;
        VAR max_temperature_f mean_temperature_f min_temperaturef
                max_dew_point_f meandew_point_f ... ;
        CLASS start_station / ORDER=UNFORMATTED ASCENDING;
RUN;
PROC MEANS DATA=myhive.bayarea_abt_orc  ... RUN;
PROC MEANS DATA=myhive.bayarea_abt_text ... RUN;
```

	SAS	TXT (Hive)	ORC (Hive)
File Size	420MB	230MB	17MB
Create Table		8 sec.	40 sec.
Select	1.2 sec.	18 sec.	12 sec.
Means	1.4 sec.	24 sec.	15 sec.

HIVE OPTIMIERUNG | STORAGE-OPTIMIERUNG FÜR TABELLEN

- Hive Partitionen sind Ordner im HDFS:
  - Hive vermeidet „unnötige“ Lese-Operationen wenn möglich.

- 3 Tuning-Optionen:
  - Partitioning:
    - Split nach Datenwerten.
  - Bucketing:
    - Split nach Hashwerten.
  - Skewing (Hive >= .14):
    - Split nach (benannten) Datenwerten, die sehr häufig auftreten (heavy skew).



Quelle: Hortonworks SQL Masterclass

HIVE OPTIMIERUNG | COST BASED OPTIMIZER

- Der CBO optimiert Hive Abfragen vor Ausführung (Umwandlung in MapReduce)
- Ziel: Reduktion der Ausführungszeiten und des Resourcenbedarfs
  - Erzeugt effizientere Ausführungspläne basierend auf Kostenheuristiken (Tabellenstatistiken und Evaluierung der erwarteten Selektivität der Filterkonditionen).
  - Verfügbar seit Hive Version 0.14 (a.k.a. Apache Calcite project).
- Vorgehensweise:
  - CBO erstellt alternative Ausführungspläne und weist den Alternativen erwartete Kosten zu

HIVE OPTIMIERUNG | COST BASED OPTIMIZER - FEATURES

- **Join Order Optimization** – die Reihenfolge der Joins wird optimiert:
  - Joins mit einer höheren Selektivität (= die erwartete Ergebnismenge ist kleiner) werden vorgezogen
  - Benötigt Tabellenstatistiken für Kosten-Heuristiken.
- **“Bushy Join” Support** – die Joins werden soweit als möglich parallelisiert
  - Default Query Planner generiert einen linearen “left-deep” Ausführungsbaum.
- **Join Predicate simplification** – identische Filterbedingungen in mehreren Joins identifizieren:
  - Reduktion des Ausführungsbaums für Joins durch Umwandlung mehrerer “WHERE Klauseln” zu einer einzelnen Klausel.

HIVE OPTIMIERUNG COST BASED OPTIMIZER

- Der CBO sollte zentral aktiviert werden:

```
set hive.cbo.enable = true;
set hive.compute.query.using.stats = true;
set hive.stats.fetch.column.stats = true;
set hive.stats.fetch.partition.stats = true;
```

- Tabellenstatistiken müssen verfügbar sein:

```
analyze table XYZ compute statistics;
analyze table XYZ compute statistics for
  columns;

analyze table XYZ partition(pk=1) compute
  statistics for columns;
```

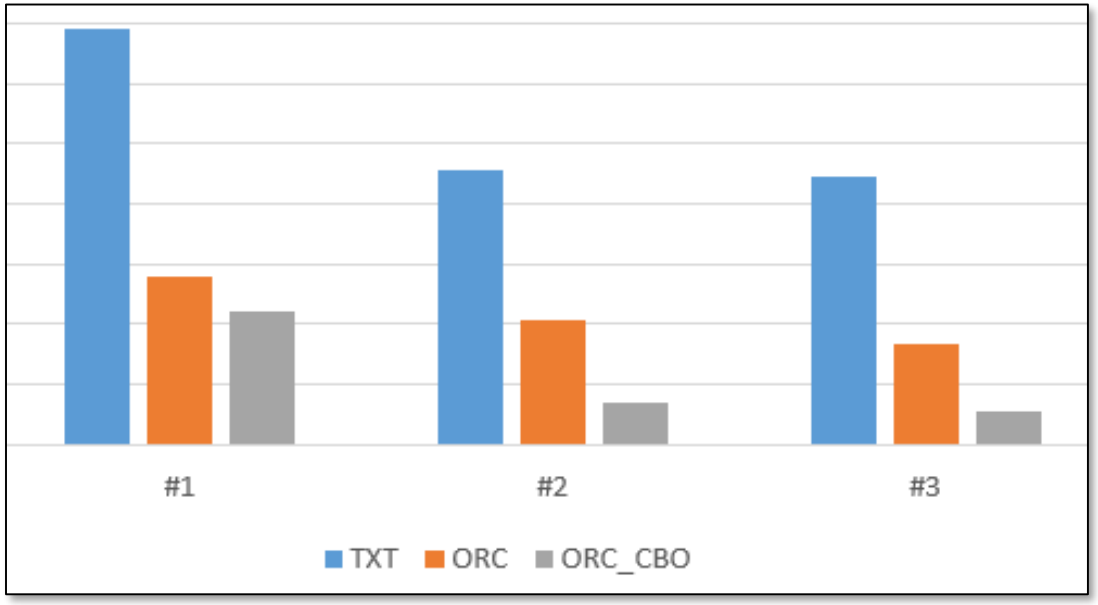
Tabellenstatistiken in Hive

- Statistiken werden auf Ebene von Tabellen, Spalten oder Partitionen ermittelt.
- Werden im Hive Metastore gespeichert.
- Automatisiertes Sammeln aktivieren durch Property: `hive.stats.autogather`
- DESCRIBE Statement zeigt, ob Statistiken für eine Tabelle verfügbar sind.

HIVE OPTIMIERUNG | EIN (NICHT REPRÄSENTATIVER) TEST …

- Identische Hive-Abfrage (komplexer Join) ausgeführt auf:
  - Daten (nicht partitioniert) im Textformat.
  - Daten im ORC Format (part.) ohne CBO.
  - Daten im ORC Format (part.) mit CBO.

  - In absoluten Zahlen: Optimierung reduzierte die Ausführungszeit von ~70 auf ~5 Sekunden.



Hinweis: Laufzeiten für #1 erhöht, da Tez Session noch nicht angelegt

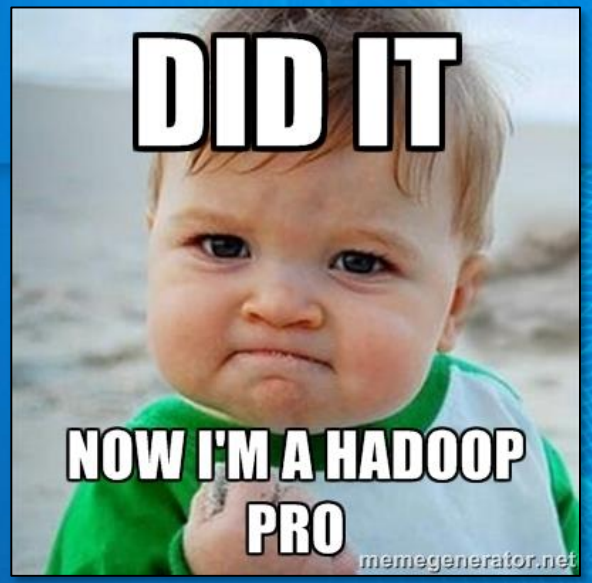
ZUSAMMENFASSUNG

Resource-Management in Hadoop für SAS - Yarn

- Yarn-Integration ist kein „nice-to-have“ Feature, sondern essentiell
- Alle SAS Technologien für Hadoop unterstützen Yarn
- Access Engines und Accelerators sind „out-of-the-box“ integriert
- TKGrid (HPA/LASR) benötigt Konfiguration (resource.settings)

SQL-Abfragen optimieren - Hive

- Hive Optimierung == I/O vermeiden
- 3 Strategien
- Geeignetes Storage Format wählen (spaltenorientiert und Kompression)
- Partitioning und Bucketing nutzen („Dynamic Partition Pruning“)
- Cost Based Optimizer nutzen (Tabellen- und Spaltenstatistiken vorhalten!)



VIELEN DANK !!

sas | THE POWER TO KNOW®