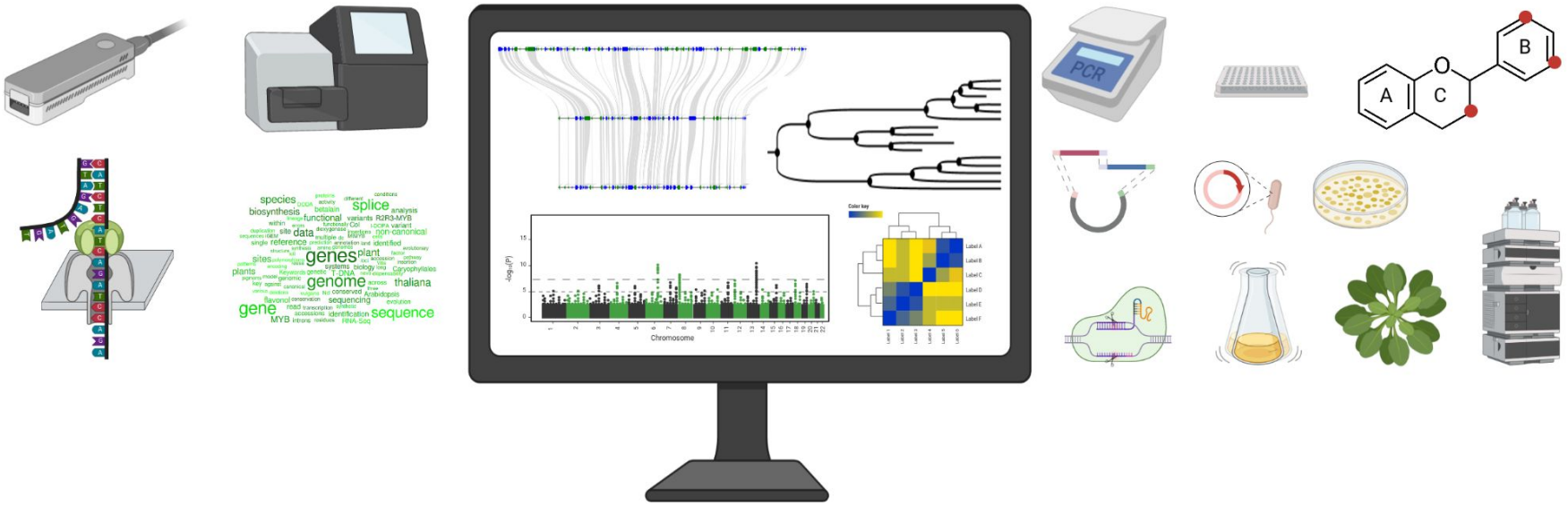


MapReduce

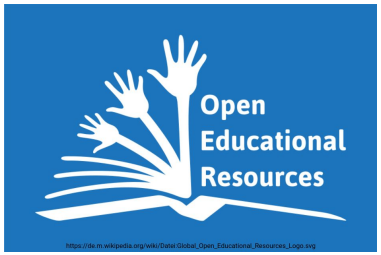
Prof. Dr. Boas Pucker

Organisation

PDF über GitHub frei verfügbar:

QR-CODE

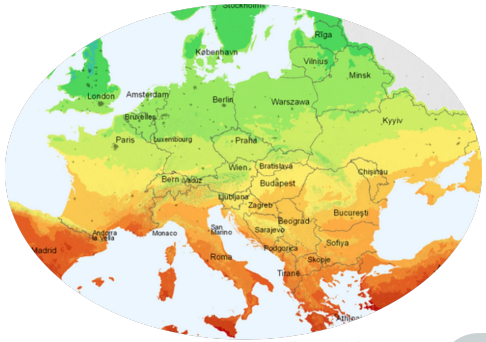
http://bxxxxxxE



Fragen, Feedback & mehr: b.pucker[a]tu-bs.de

“Big Data” Analyse-Herausforderungen

- Analyse von Wetterdaten (Temperaturminima und -maxima)

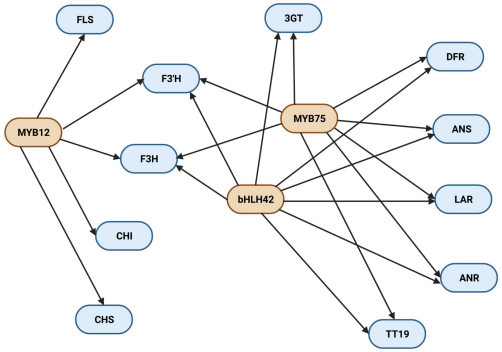


https://commons.wikimedia.org/wiki/File:SolarGIS-Solar-map-Europe-de.png, CC BY-SA 3.0

- Globale Verbreitung von Arten


https://www.gbif.org/occurrence/map?q=brassica%20rapa

- Genexpressionsanalysen (Transkriptionsfaktoren & Zielgene)



Herausforderungen der Parallelisierung

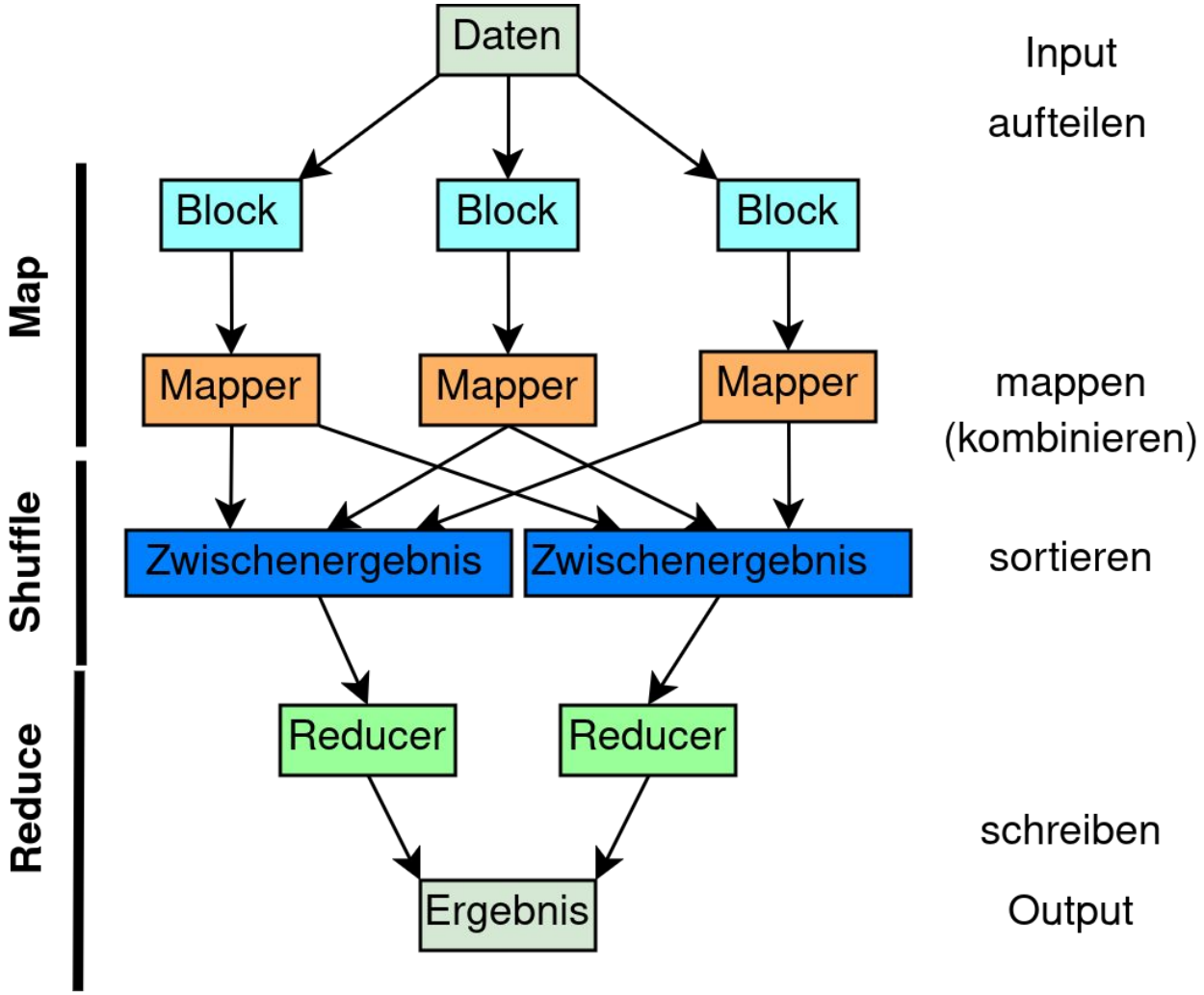
- Limitierender Faktor: langsamster Job bestimmt die Dauer
- Zuverlässigkeitsproblem: Was passiert wenn ein Job unvollständige Ergebnisse liefert?
- Umgang mit fehlenden Ergebnissen von einem Job
- Gleichmäßige Aufteilung der Daten
- Aggregation der Ergebnisse

MapReduce als mögliche Lösung

- MapReduce wurde von Google für Analyse von mehreren PB entwickelt
- Parallelisierung essentiell für Geschwindigkeit der Analysen
- Parallelisierung ermöglicht Analysen für Datensätze, die für ein System zu groß wären
- Resilienz gegenüber Fehlern im Cluster

Google Inc., 2004: US7650331B1

Konzept von MapReduce



Beispiel 1: Coexpressions-Netzwerk (Daten)

Welche Transkriptionsfaktoren haben gemeinsame Zielgene?

Formale Beschreibung von Coexpression:

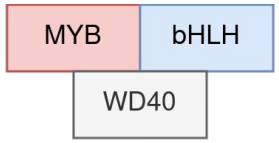
A -> B C D

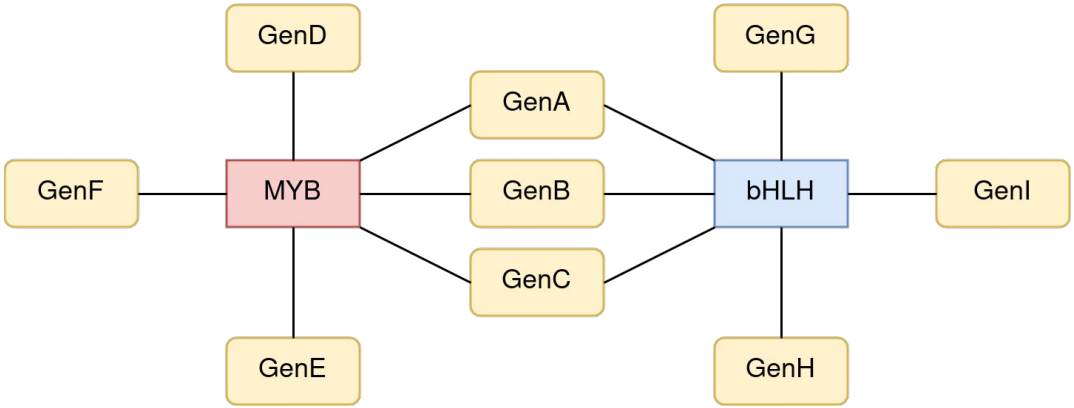
B -> A C D E

C -> A B D E

D -> A B C E

E -> B C D





Basierend auf https://stackoverflow.com/questions/12375761/good-mapreduce-examples

Beispiel 1: Coexpressions-Netzwerk (map)

map(A -> B C D) :
(A B) -> B C D
(A C) -> B C D
(A D) -> B C D

map(B -> A C D E) :
(A B) -> A C D E
(B C) -> A C D E
(B D) -> A C D E
(B E) -> A C D E

map(C -> A B D E) :
(A C) -> A B D E
(B C) -> A B D E
(C D) -> A B D E
(C E) -> A B D E

map(D -> A B C E) :
(A D) -> A B C E
(B D) -> A B C E
(C D) -> A B C E
(D E) -> A B C E

map(E -> B C D):
(B E) -> B C D
(C E) -> B C D
(D E) -> B C D

Map: K x V -> (L x W)*
(k,v) -> [(l_1,x_1), (l_2,x_2) … (l_m,x_m)]

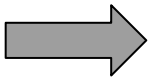
K, L: Menge der Schlüssel; V, W: Menge der Werte
Alle Elemente einer Menge müssen vom gleichen Typ sein (String, Integer, Float)

Basierend auf https://stackoverflow.com/questions/12375761/good-mapreduce-examples

Beispiel 1: Coexpressions-Netzwerk (sortieren & reduce)

Sortieren und zusammenfassen:

(A B) -> (A C D E) (B C D)
(A C) -> (A B D E) (B C D)
(A D) -> (A B C E) (B C D)
(B C) -> (A B D E) (A C D E)
(B D) -> (A B C E) (A C D E)
(B E) -> (A C D E) (B C D)
(C D) -> (A B C E) (A B D E)
(C E) -> (A B D E) (B C D)
(D E) -> (A B C E) (B C D)



Reducer

Ergebnis:

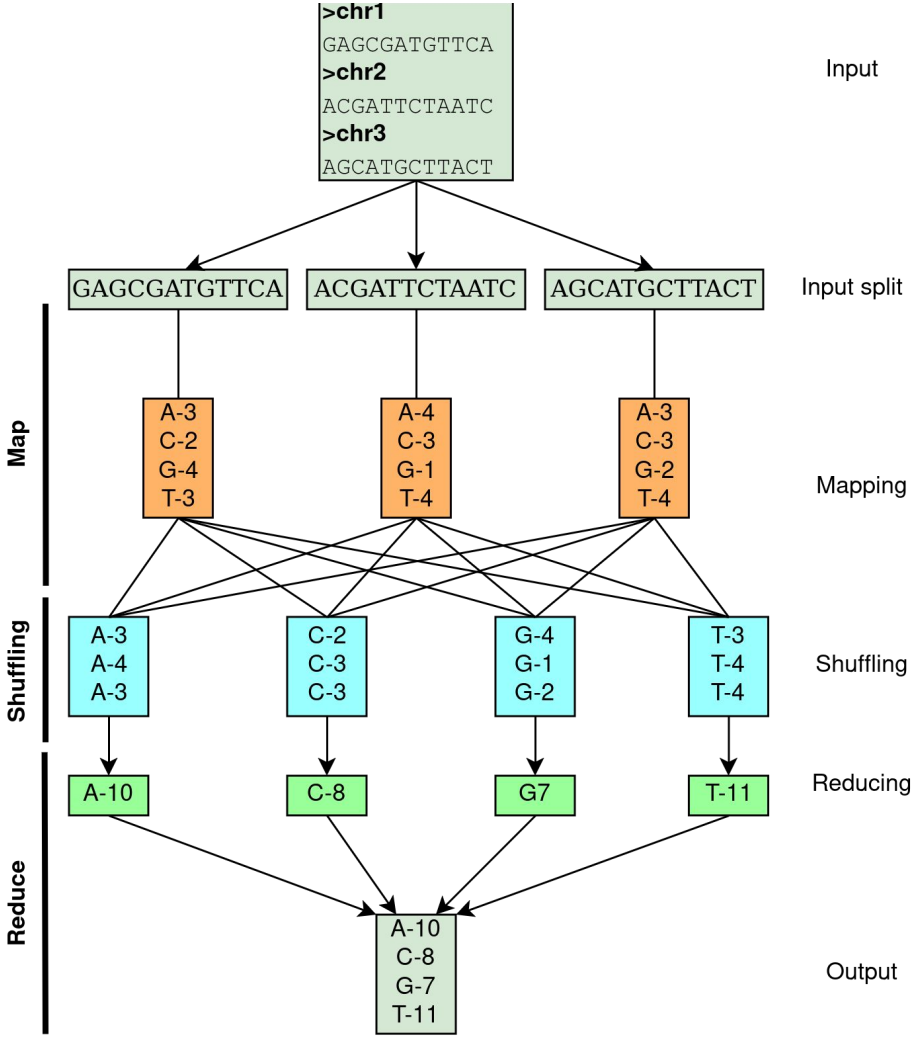
(A B) -> (C D)
(A C) -> (B D)
(A D) -> (B C)
(B C) -> (A D E)
(B D) -> (A C E)
(B E) -> (C D)
(C D) -> (A B E)
(C E) -> (B D)
(D E) -> (B C)

Map: L x W* -> X*
$(l[y_1,y_2 \ldots y_m]) \rightarrow [w_1,w_2 \ldots w_n]$

L: Menge der Schlüssel; W,X: Menge der Werte
Alle Elemente einer Menge müssen vom gleichen Typ sein (String, Integer, Float)

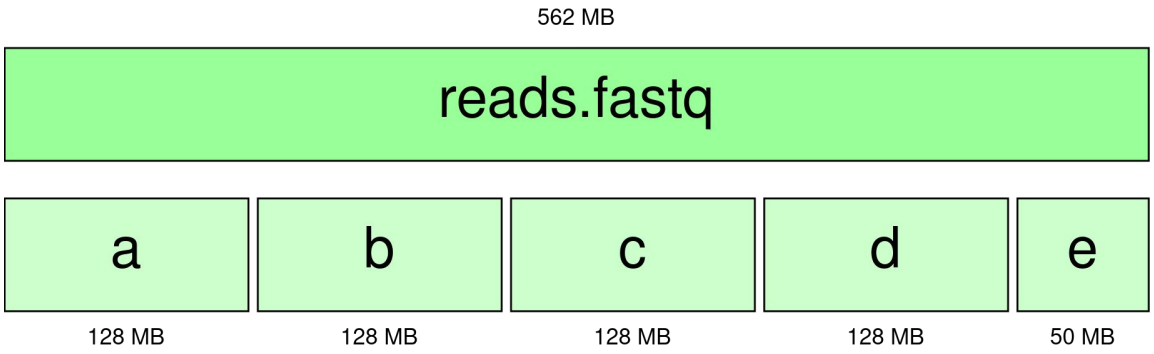
Basierend auf https://stackoverflow.com/questions/12375761/good-mapreduce-examples

Beispiel 2: GC-Gehalt-Berechnung



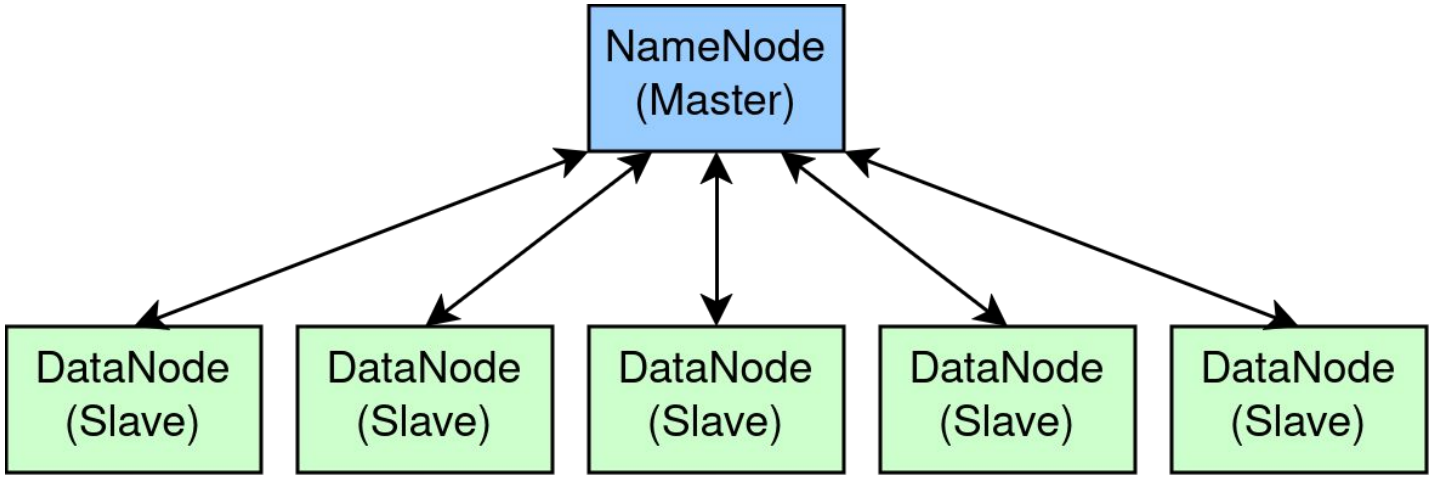
Aufteilung der Daten am Beispiel Hadoop

- Dateien werden normalerweise als Sammlungen von Blöcken gespeichert
- Apache Hadoop verwendet 128MB Blöcke
- Blockgröße bestimmt Menge an notwendigen Blöcken und damit Metadaten



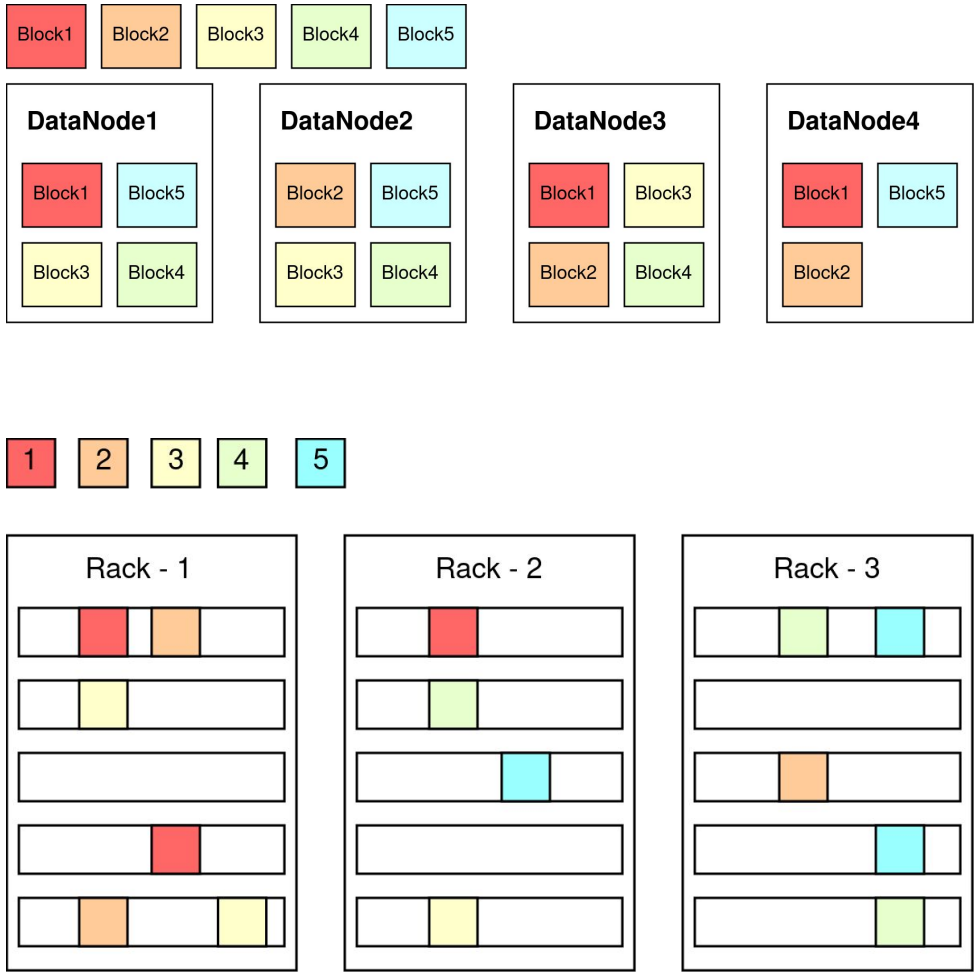
Hadoop Distributed File System (HDFS)

- Speicherung über verschiedene Maschinen im Cluster verteilt (Data Nodes)
- Master/Slave-Architektur mit einem “NameNode” und vielen “DataNodes”
- Master speichert nur Metadaten (Größe, Ort, Berechtigungen, …)
- Master kontrolliert Replikate und gleichmäßige Nutzung des Speichers



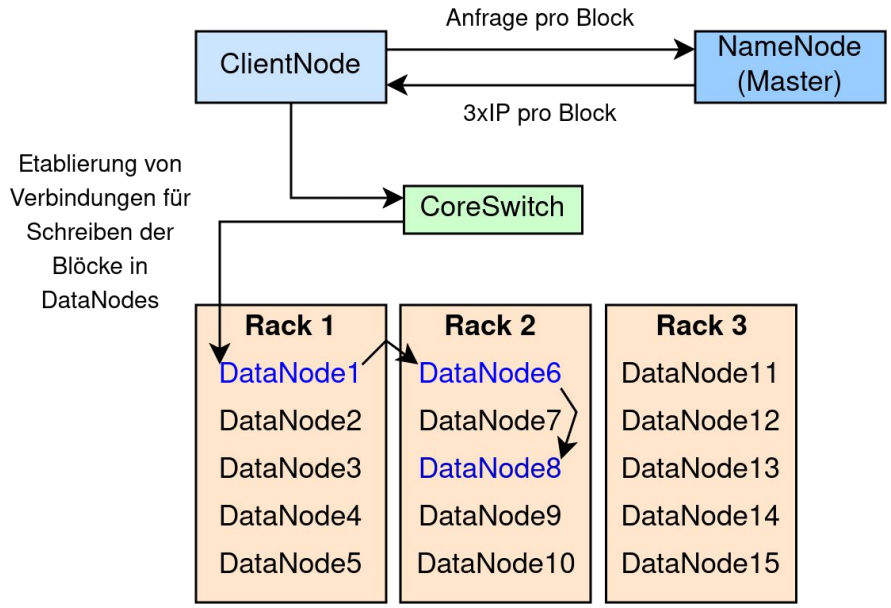
Verwaltung von Replikaten

- Jeder Block wird mehrfach gespeichert (default: 3x)
- Speicherung auf verschiedenen DataNodes
- NameNode (Master) kontrolliert Replikate
- Replikate werden über Racks verteilt (bessere Performance, kein Datenverlust)



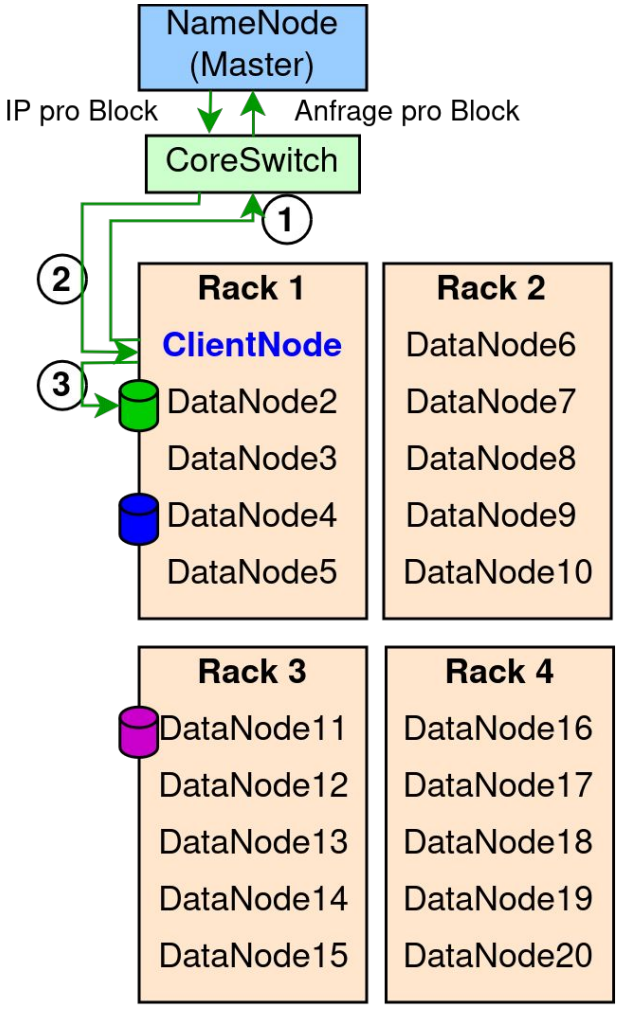
HDFS - Schreiben von Dateien

- Master gibt HDFS-Client zufällige IP-Adressen von DataNodes
- IP-Adressen sind für jeden Block einer Datei anders
- Schreiben in drei Schritten:
  - 1) Pipeline-Setup
  - 2) Data-Streaming
  - 3) Pipeline-Abschluss
- Transfer von einem DataNode zum nächsten
- Alle Blöcke werden gleichzeitig (auf verschiedenen DataNodes) geschrieben



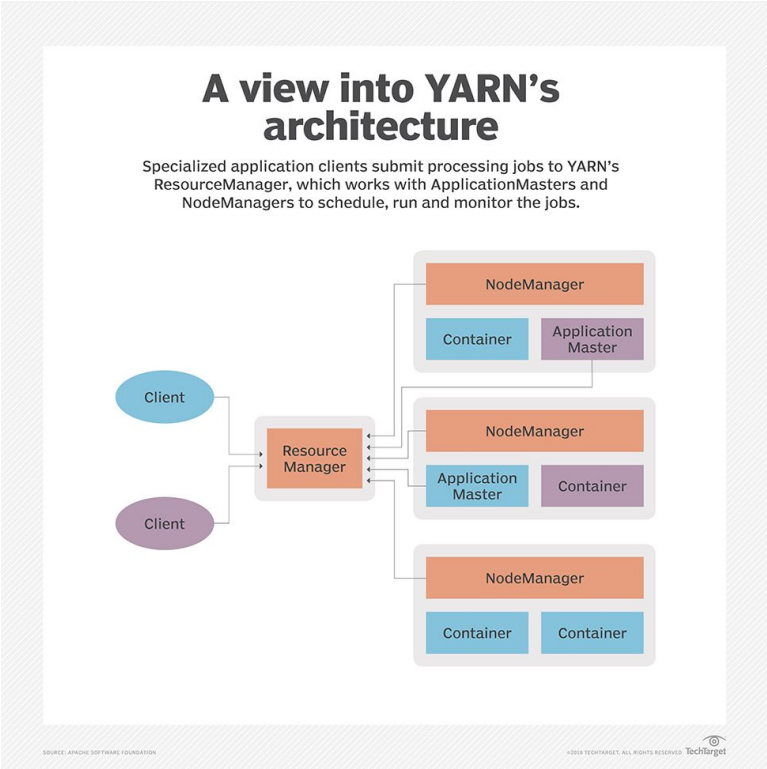
HDFS - Lesen von Dateien

- Master liefert Speicherorte (IP) der Blöcke einer Datei
- Speicherorte nahe am Client werden bevorzugt
- Client kann alle Blöcke gleichzeitig von verschiedenen DataNodes lesen



Yet Another Resource Negotiator (YARN)

- Ressourcen Management und Job Scheduling für Hadoop
- “MapReduce 2” oder “NextGen MapReduce”
- Zuweisung von Ressourcen mit verschiedenen Prioritäten und Reservierungsfunktion
- Über Subcluster können Zehntausende Knoten verknüpft werden



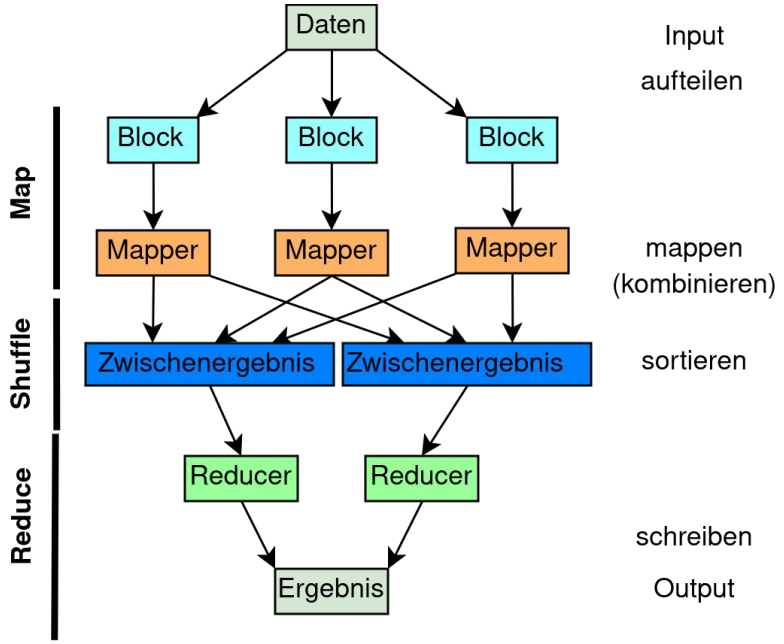
https://www.computerweekly.com/de/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator

Verschiedene Frameworks: Hadoop vs. Spark

	Hadoop	**Spark**
Datenspeicherung	Festplatte	RAM
Prozessierung	batches	Echtzeit
Sicherheit	Datenverschlüsselung & Zugangskontrolle	Umgebung muss gesichert sein
Skalierbarkeit	Exzellent	Erfordert mehr RAM
Kosten	Günstige Infrastruktur	Hoher RAM-Bedarf ist teuer

Zusammenfassung

- Map-Phase: Aufteilen der Daten auf Mapper; Bildung von Schlüssel-Wert-Paaren
- Shuffle-Phase: Gruppierung der Werte mit gleichem Schlüssel
- Reduce-Phase: Verarbeitung durch Reducer
- MapReduce in Hadoop
- Alternative Frameworks (Spark)



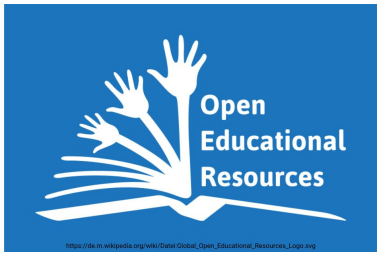
Weiterführende Informationen

- MapReduce-Tutorial: https://thirdeyedata.ai/hadoop-mapreduce/
- HDFS-Tutorial: https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/
- Hadoop vs. Spark: https://aws.amazon.com/compare/the-difference-between-hadoop-vs-spark/
- Apache Hadoop: https://hadoop.apache.org/
- Apache Spark: https://spark.apache.org/

Materialverfügbarkeit

PDF über GitHub verfügbar:

http://bit.ly/xxxxxx



Fragen, Feedback & mehr: b.pucker[a]tu-bs.de