QAWARE SOFTWARE ENGINEERING

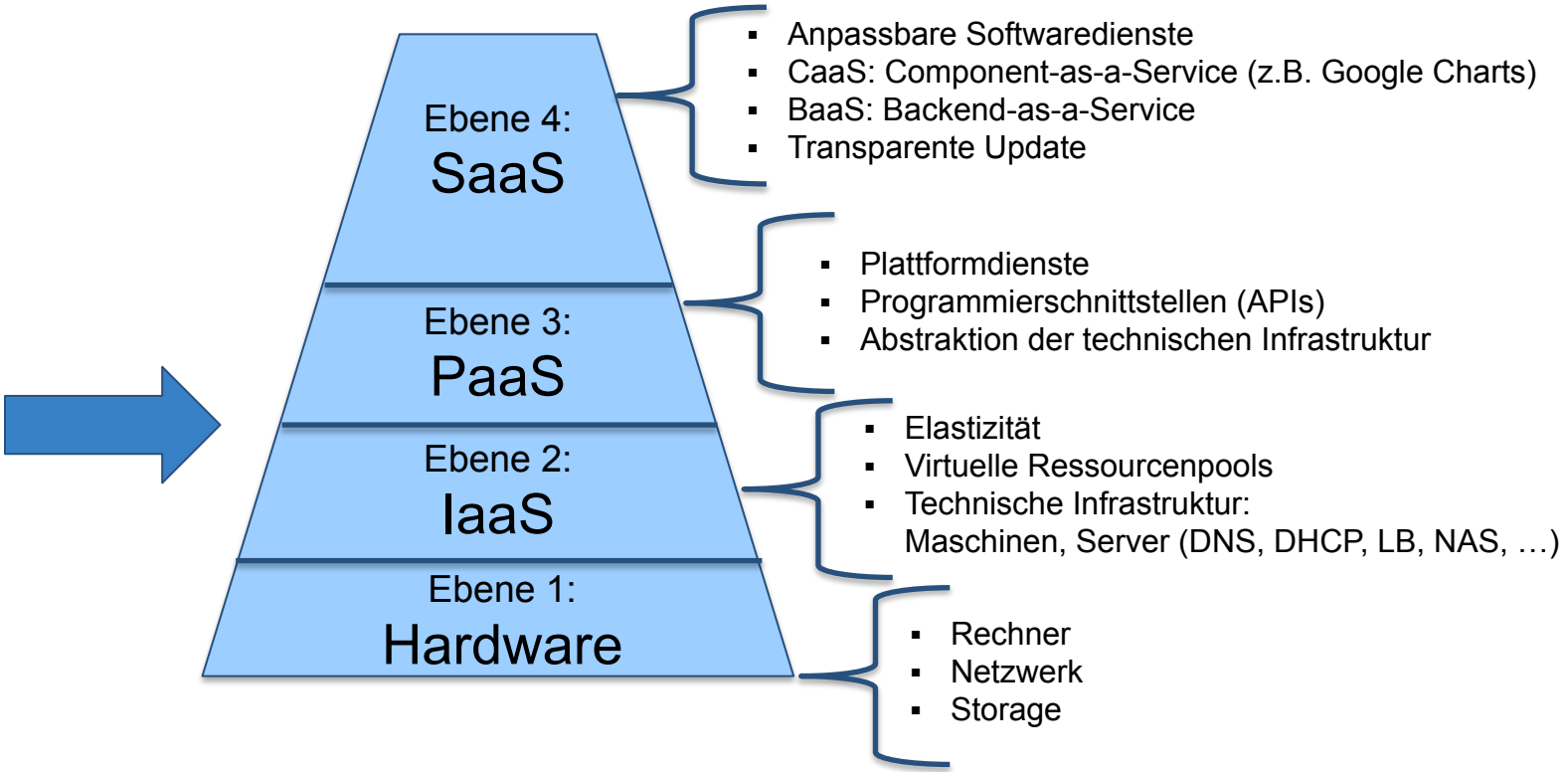
HM Hochschule München University of Applied Sciences

Cloud Computing
Cluster Scheduling

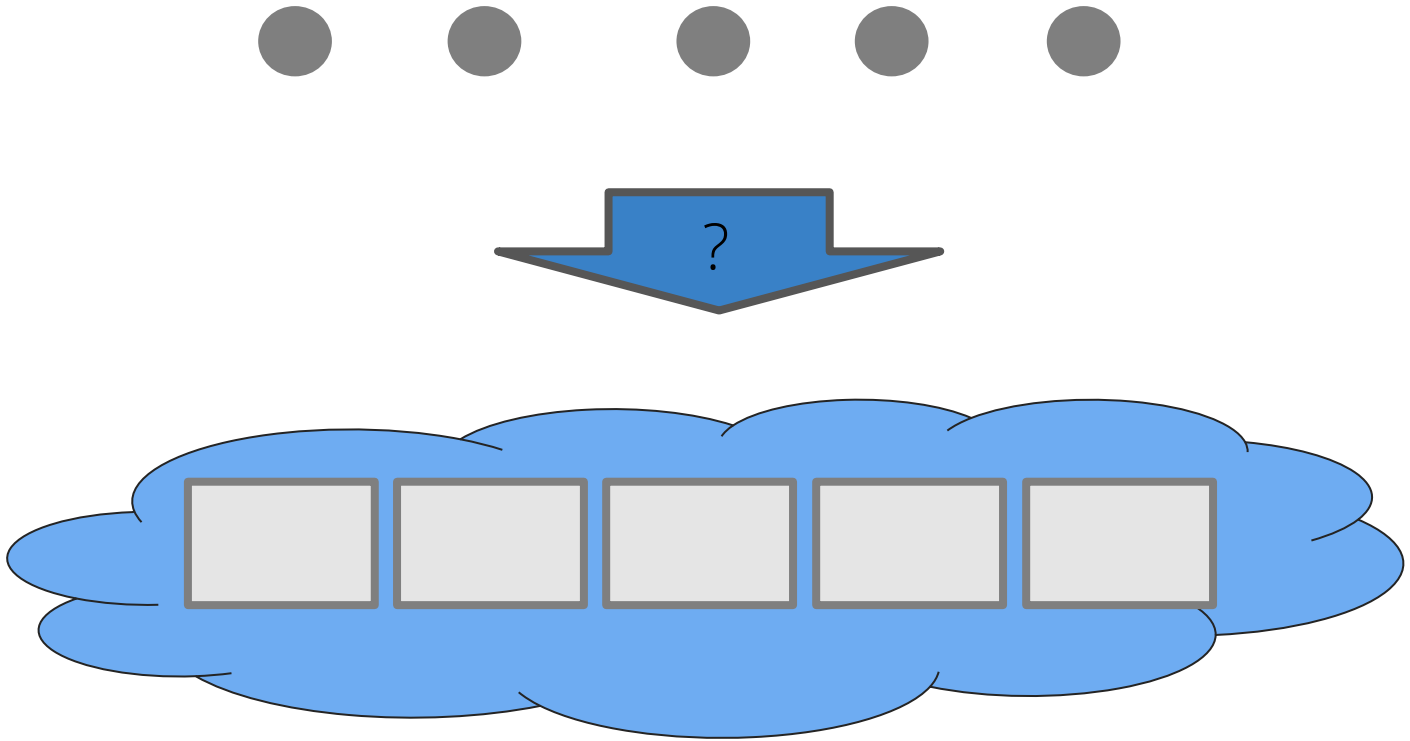
Johannes Ebke

2023-05-10

Reminder: Das Schichtenmodell des Cloud Computing



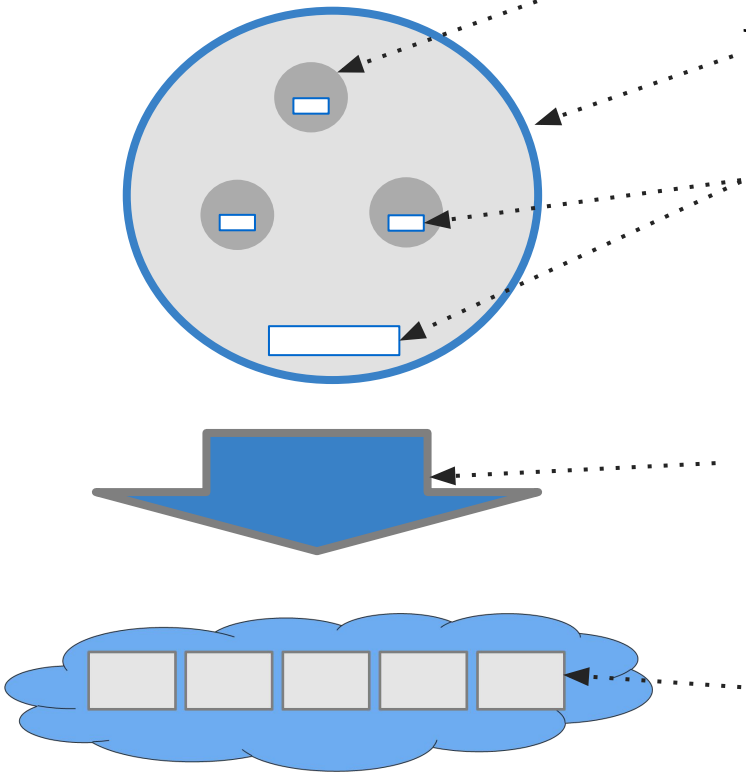
Das Problem



Rechenaufgaben

Rechen-
Ressourcen
(z.B. per IaaS)

Terminologie



Task: Atomare Rechenaufgabe inklusive Ausführungsvorschrift.

Job: Menge an Tasks mit gemeinsamen Ausführungsziel. Die Menge an Tasks ist i.d.R. als DAG mit Tasks als Knoten und Ausführungsabhängigkeiten als Kanten repräsentiert.

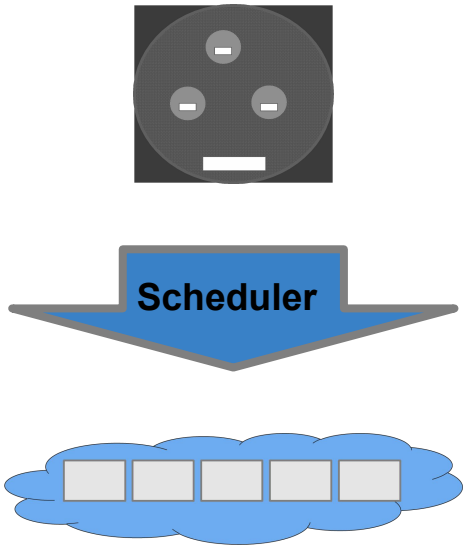
Properties: Ausführungsrelevante Eigenschaften der Tasks und Jobs, wie z.B.:

- Task: Ausführungsdauer, Priorität, Ressourcenverbrauch
- Job: Abhängigkeiten der Tasks, Ausführungszeitpunkt

Scheduler: Ausführung von Tasks auf den verfügbaren Resources unter Berücksichtigung der Properties und gegebener **Scheduling-Ziele** (z.B. Fairness, Durchsatz, Ressourcenauslastung). Ein Scheduler kann **präemptiv** sein, also die Ausführung von Tasks unterbrechen und neu aufsetzen können.

Resources: Cluster an Rechnern mit CPU-, RAM-, HDD-, Netzwerk-Ressourcen. Ein Rechner stellt seine Ressourcen temporär zur Ausführung eines oder mehrerer Tasks zur Verfügung (**Slot**). Die parallele Ausführung von Tasks ist isoliert zueinander.

Aufgaben eines Cluster-Schedulers



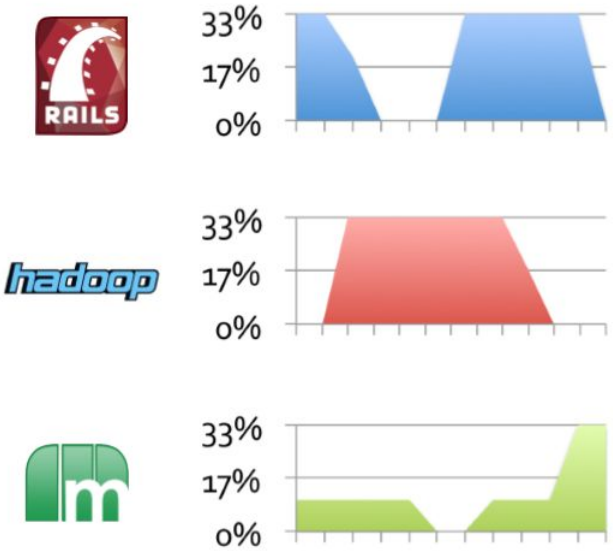
Cluster Awareness: Die aktuell verfügbaren Ressourcen im Cluster kennen (Knoten inkl. verfügbare CPUs, verfügbarer RAM und Festplattenspeicher sowie Netzwerkbandbreite). Dabei auch auf Elastizität reagieren.

Job Allocation: Zur Ausführung eines Services die passende Menge an Ressourcen für einen bestimmten Zeitraum bestimmen und allokieren.

Job Execution: Einen Service zuverlässig ausführen und dabei isolieren und überwachen.

Die einfachste Form des Scheduling: Statische Partitionierung

Auslastung pro Knoten



Vorteil:

- Einfach zu realisieren

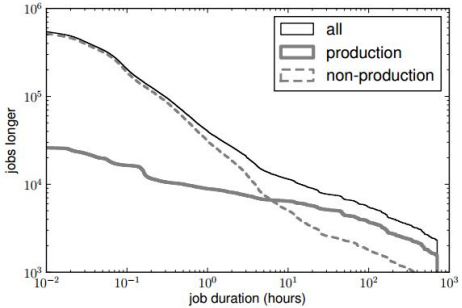
Nachteile:

- Nicht flexibel bei geänderten Bedürfnissen
- Geringere Auslastung
  □ hohe Opportunitätskosten

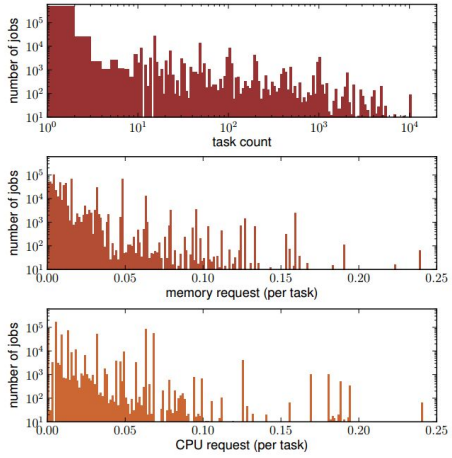
Bildquelle: Practical Considerations for Multi-Level Schedulers, Benjamin Hindman, 19th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) 2015

Heterogenität im Scheduling

- In typischen Clustern ist die Workload an Jobs sehr heterogen.
- Charakteristische Unterschiede sind:
  - Ausführungsdauer: min, h, d, INF.
  - Ausführungszeit: sofort, später, zu einem Zeitpunkt.
  - Ausführungszweck: Datenverarbeitung, Request-Handling.
  - Ressourcenverbrauch: CPU-, RAM-, HDD-, NW-dominant.
  - Zustand: zustandsbehaftet, zustandslos.
- Zu unterscheiden sind mindestens:
  - **Batch-Jobs**: Ausführungszeit im Minuten- bis Stundenbereich. Eher niedrige Priorität und gut unterbrechbar. Müssen i.d.R. bis zu einem bestimmten Zeitpunkt abgeschlossen sein. Zustandsbehaftet.
  - **Service-Jobs**: Sollen auf unbestimmte Zeit unterbrechungsfrei laufen. Haben hohe Priorität und sollten nicht unterbrochen werden. Teilweise zustandslos.



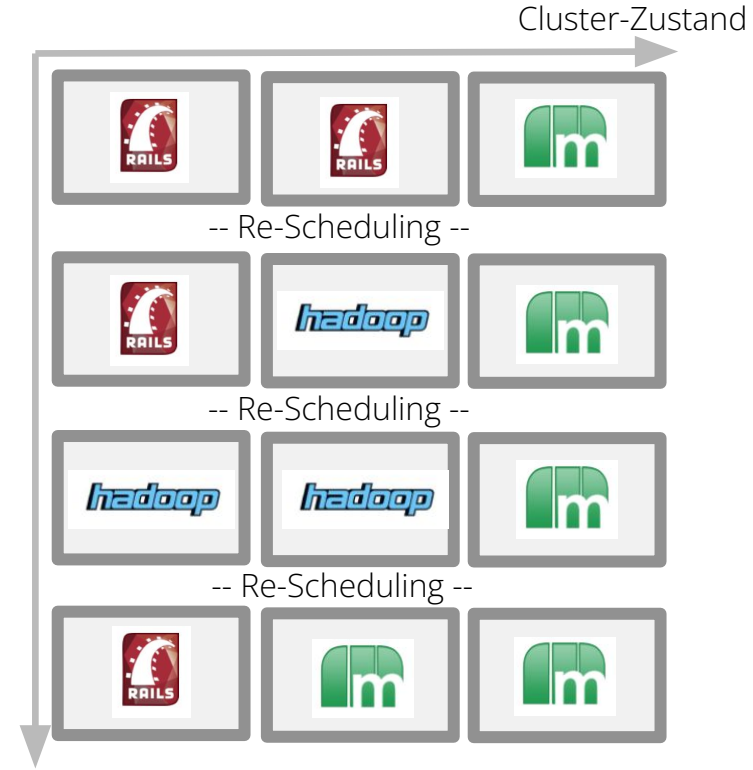
Ausführungsdauer



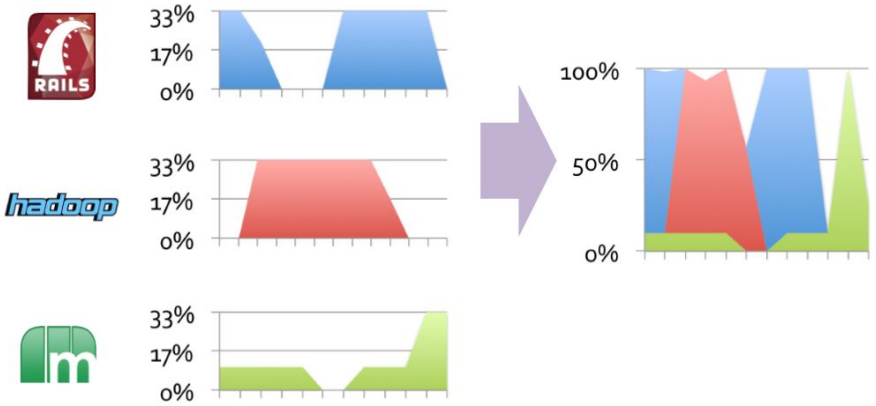
Ressourcenverbrauch

Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis, Charles Reiss et al., 2012

Bestehende Ressourcen einer Cloud können durch dynamische Partitionierung wesentlich effizienter genutzt werden



Statische Partitionierung Dynamische Partitionierung



Vorteile der dynamischen Partitionierung:

- Höhere Auslastung der Ressourcen □ weniger Ressourcen notwendig □ geringere Betriebskosten
- Potenziell schnellere Ausführung einzelner Tasks, da Ressource opportun genutzt werden können.

Cluster-Scheduler: Eingabe, Verarbeitung, Ausgabe

Eingabe eines Cluster-Schedulers ist Wissen über die Jobs und Tasks (Properties) und über die Ressourcen:

- **Resource Awareness**: Welche Ressourcen stehen zur Verfügung und wie ist der entsprechende Bedarf des Tasks?
- **Data Awareness**: Wo sind die Daten, die ein Task benötigt?
- **QoS Awareness**: Welche Ausführungszeiten müssen garantiert werden?
- **Economy Awareness**: Welche Betriebskosten dürfen nicht überschritten werden?
- **Priority Awareness**: Wie ist die Priorität der Task zueinander?
- **Failure Awareness**: Wie hoch ist die Wahrscheinlichkeit eines Ausfalls? (z.B. da ein Rack oder eine Stromvers.)
- **Experience Awareness**: Wie hat sich ein Tasks in der Vergangenheit verhalten?

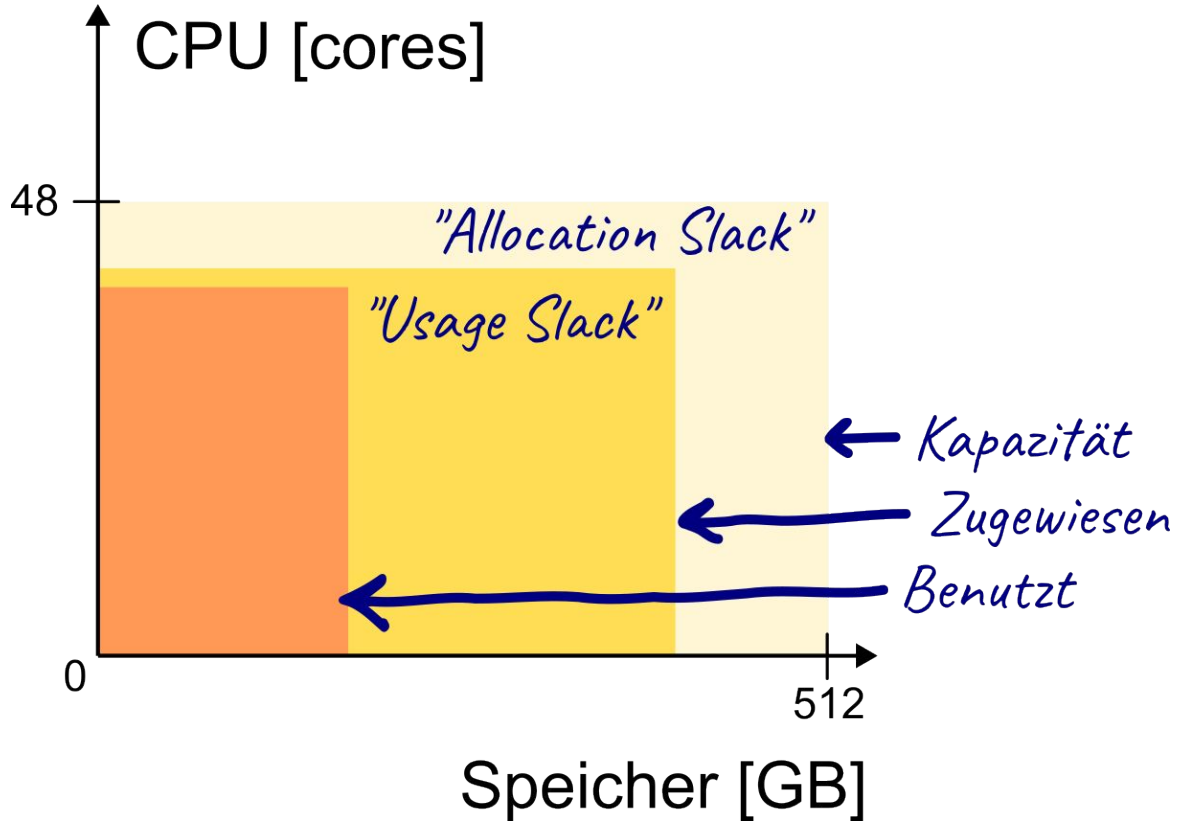
Ausgabe eines Cluster-Schedulers:
Placement Decision als

- **Slot-Reservierungen**
- **Slot-Stornierungen** (im Fehlerfall, Optimierungsfall, Constraint-Verletzung)

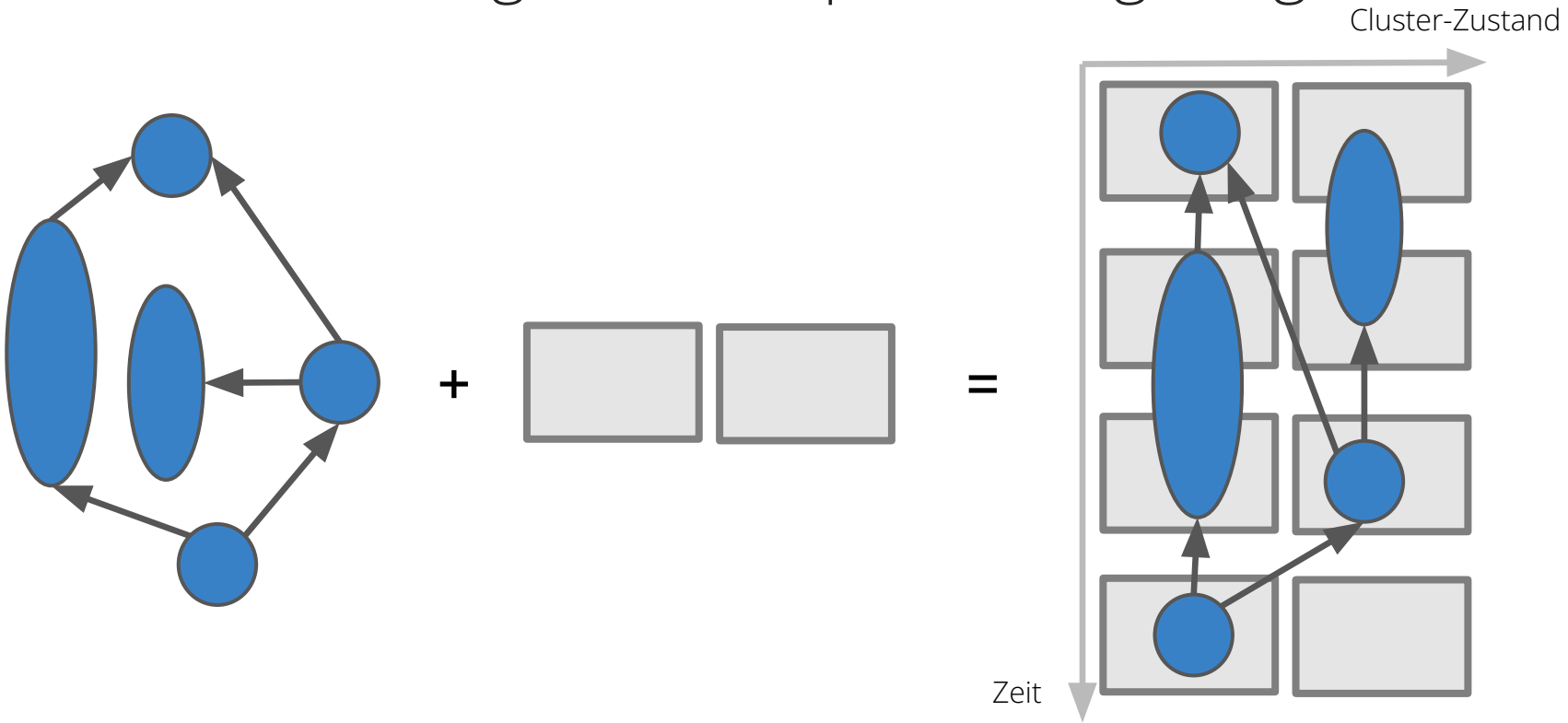
Verarbeitung im Cluster-Scheduler: **Scheduling-Algorithmen** entsprechend der jeweiligen **Scheduling-Ziele**, wie z.B.:

- **Fairness**: Kein Task sollte unverhältnismäßig lange warten müssen, während ein anderer bevorzugt wird.
- **Maximaler Durchsatz**: So viele Tasks pro Zeiteinheit wie möglich.
- **Minimale Wartezeit**: Möglichst geringe Zeit von der Übermittlung bis zur Ausführung eines Tasks.
- **Ressourcen-Auslastung**: Möglichst hohe Auslastung der verfügbaren Ressourcen.
- **Zuverlässigkeit**: Ein Task wird garantiert ausgeführt.
- **Geringe End-to-End Ausführungszeit** (z.B. durch Daten-Lokalität und geringe Kommunikationskosten, syn. Makespan)

Hauptziel ist es oft, die Ressourcen-Auslastung zu optimieren. Das spart Opportunitätskosten



Cluster-Scheduling ist eine Optimierungsaufgabe



Tasks mit Zeitbedarf (Höhe) und Ausführungsabhängigkeiten (Pfeile)

Verfügbare Ressourcen

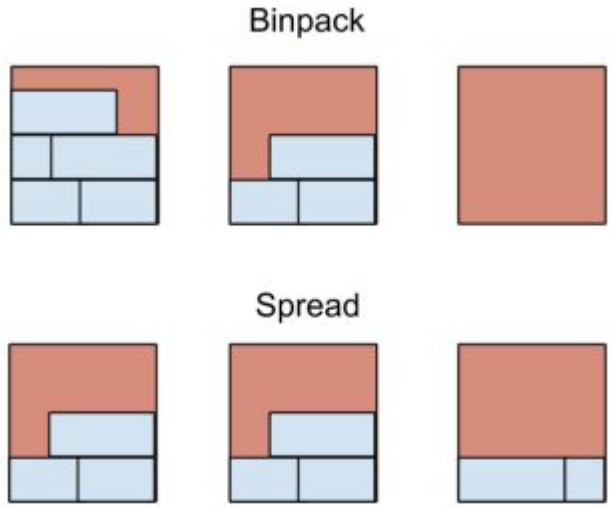
Slot-Reservierungen

Scheduling ist eine Optimierungsaufgabe…

- … und ist NP-vollständig.
  Die Optimierungsaufgabe lässt sich auf das Traveling Salesman Problem zurückführen.
- Das bedeutet:
  - Es ist kein Algorithmus bekannt, der eine optimale Lösung garantiert in Polynomialzeit erzeugt.
  - Algorithmus muss für tausende Jobs und tausende Ressourcen skalieren. Optimale Algorithmen, die den Lösungsraum komplett durchsuchen sind nicht praktikabel, da deren Entscheidungszeit zu lange ist für große Eingabemengen (|Jobs| x |Ressourcen|).
  - Praktikable Scheduling-Algorithmen sind somit Algorithmen zur näherungsweisen Lösung des Optimierungsproblems (Heuristiken, Meta-Heuristiken).
- Darüber hinaus kommen Job-Anfragen kontinuierlich an, so dass selbst bei optimalem Algorithmus der Re-Organisationsaufwand pro Job unverhältnismäßig hoch werden kann.

Einfache Scheduling-Algorithmen

- Optimieren das Scheduling von Tasks oft in genau einer Dimension (z.B. CPU-Auslastung) bzw. wenigen Dimensionen (CPU-Auslastung und RAM).
- Populäre Algorithmen:
  - Binpack (Fit First)
  - Spread (Round Robin)

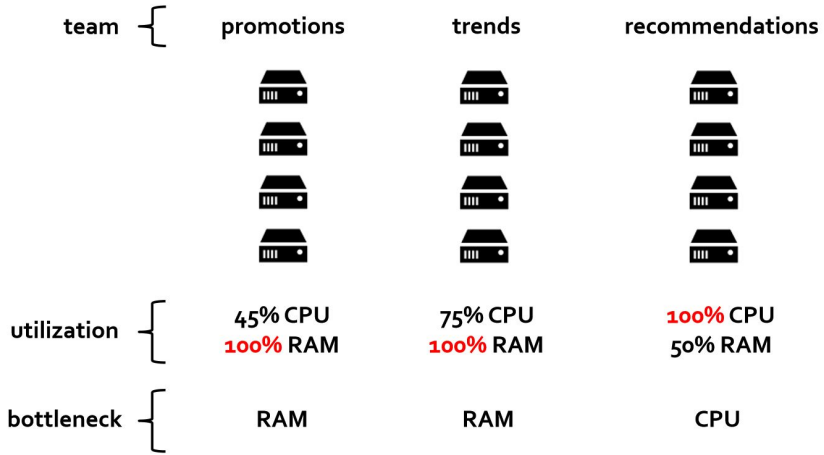


http://container-solutions.com/using-binpack-with-docker-swarm

Übung 1
Binpack und Spread

Multidimensionaler Scheduling-Algorithmus mit Fokus auf Fairness: Dominant Resource Fairness (DRF)

- Aufteilung der Ressourcen an verschiedene „Teams“ (Applikationen, Jobs).
- Ausgangslage: Würden die Ressourcen gleichteilig statisch an N Teams verteilt, so hat jedes Team eine dominante Ressource, die besonders intensiv genutzt wird. Diese dominante Ressource kann durch Beobachtung ermittelt werden und balanciert sich über alle Teams hinweg aus.
- Fairness-Auffassung: Jedes Team bekommt mindestens 1/N aller Ressourcen der dominanten Ressourcen. Der Scheduling-Algorithmus ist darauf ausgelegt, die minimal verfügbaren dominanten Ressourcen pro Team zu maximieren.
- Die Fairness kann justiert werden. Jedem Team kann ein gewichteter Anteil der Ressourcen in der statischen Ausgangslage zugesprochen werden. Die Fairness-Auffassung ist dann entsprechend gewichtet.



- Bildquelle: Practical Considerations for Multi-Level Schedulers, Benjamin Hindman, 19th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) 2015
- Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, Ghodsi et al., 2011
- DRF ist eine Generalisierung des Min-Max Algorithmus für mehrere Ressourcen: http://www.ece.rutgers.edu/~marsic/Teaching/CCN/minmax-fairsh.html
- https://stackoverflow.com/questions/39347670/explanation-of-yarns-drf

DRF - Beispiel

- Insgesamt verfügbare Ressourcen: 100 CPUs, 10000 GB Memory
- Es gibt 2 Applikationen:
  - Anforderungen Application A: 2 CPUs, 300 GB Memory je Container → A's dominante Ressource ist Memory (2% of CPUs vs. 3% of Memory)
  - Anforderungen Application B: 6 CPUs, 100 GB Memory je Container → B's dominante Ressource ist CPU (6% of CPUs vs. 1% of Memory)

Angenommen "A" bekommt x Container und "B" bekommt y Container

- Ressourcenbedarf A: 2x CPUs + 300x GB Memory (2 CPUs and 300 GB Memory pro Container)
- Ressourcenbedarf B: 6y CPUs + 100y GB Memory (6 CPUs and 100 GB Memory pro Container)

Bedingung: $2x + 6y <= 100$ CPUs & $300x + 100y <= 10000$ GB Memory

DRF - Beispiel

DRF versucht, die dominanten Bedarfe von A und B auszugleichen:

- A's dominanter Bedarf: 300x / 10000 GB (300x von den verfügbaren 10000 GB total memory)
- B's dominanter Bedarf: 6y / 100 CPUs (6y von den verfügbaren 100 CPUs)

DRF versucht damit einen Ausgleich von: (300x / 10000) = (6y / 100) → Auflösen ergibt x = 2y

Setzt man x = 2y in der Bedingung oben ein (4y + 6y <= 100 und 600y + 100y <= 10000) bekommt man x=20 und y=10.

Damit bekommt Application A 20 Container: (40 CPUs, 6000 GB of Memory), Application B bekommt 10 Container: (60 CPUs, 1000 GB of memory)

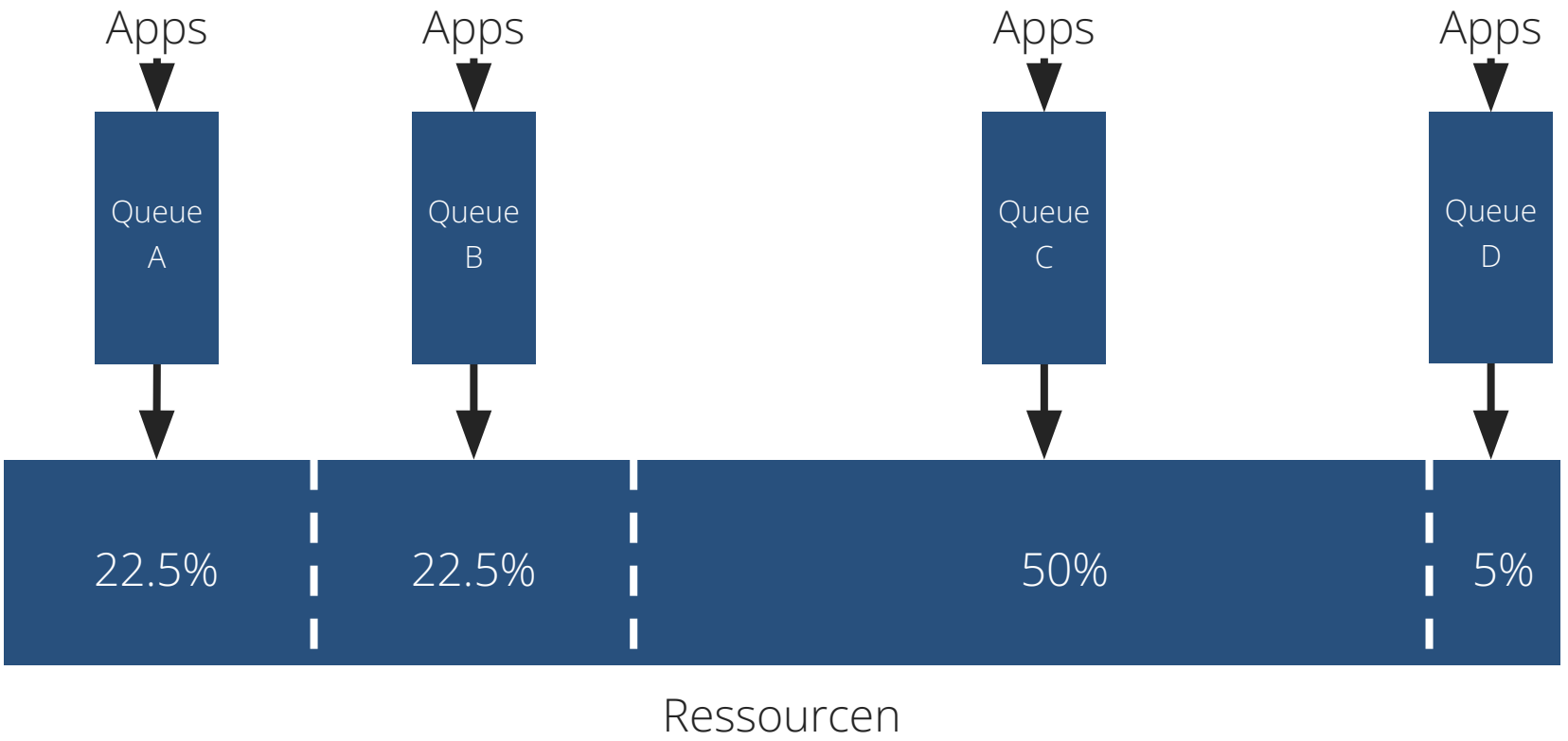
→ Total allocated CPU: 40 + 60 <= 100 CPUs available

→ Total allocated Memory: 6000 + 1000 <= 10000 GB of Memory available

Scheduling-Algorithmus mit Fokus auf Fairness: Capacity Scheduler (CS)

- Es werden Job Queues definiert und zu jeder Queue eine Kapazitätszusage in Ressourcenanteil vom Cluster definiert.
- Fairness-Auffassung: Diese Kapazitätszusage wird stets eingehalten. Der Scheduling-Algorithmus stellt sicher, dass diese Fairness stets sichergestellt wird.
- Damit das Cluster dafür nicht statisch partitioniert werden muss, ist ein sog. Over-Commitment von Ressourcen erlaubt.
- Wird durch ein Over-Commitment aber eine Kapazitätszusage gefährdet, werden die over-committeten Ressourcen entzogen. Hierfür ist also ein präemptiver Scheduler notwendig.

Capacity Scheduler (CS) - Beispiel:



Scheduling-Algorithmus mit Balance aus Fairness und Ausführungszeit: Tetris

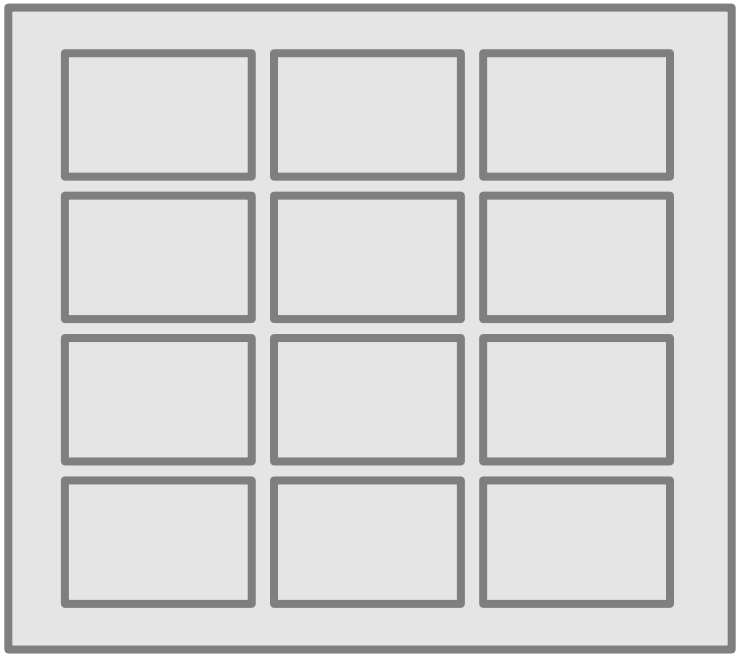
- Schritt 1: Effizientes multidimensionales Bin Packing hohe Auslastung
  - Gegeben:
    - Vektor des Ressourcenbedarfs entlang der Dimensionen des Ressourcenangebots wie z.B. CPU, RAM, Disk IO, NW IO.
    - Raum des Ressourcenabgebots entsprechend der Dimensionen plus einer diskretisierten Zeit-Dimension.
  - Gesucht: Wie packt man die Ressourcenbedarfe so in den Raum des Ressourcenangebots, dass möglichst viel freier Raum übrig bleibt (hohe Packungsdichte, geringe Fragmentierung).
  - Unscharfe Lösung über eine Heuristik.
- Schritt 2: SRT (Smallest Remaining Time) geringe Makespan (Zeit zur Verarbeitung eines Jobs)
  - Tasks werden entsprechend ihrer aufsteigend sortierten Rest-Laufzeit höher priorisiert bei der Ausführung.
- Fairness Knob Fairness vs. Performance
  - Es kann ein Wert zwischen 0 und 1 angegeben werden, der Fairness und Performance (bzgl. Makespan) balanciert. Je mehr Fairness gefordert wird desto höher werden Jobs priorisiert, die weiter Weg von einem fairen Zustand (im Sinne von DRF) sind.
- Deutlicher Performance-Vorteil bzgl. Makespan gegenüber DRF, gerade bei hoher Last im Cluster.

Multi-Resource Packing for Cluster Schedulers Paper, Grandl, 2014

Übung 2
Dominant Resource Fairness

Architektur eines Cluster Schedulers

The Datacenter as a Computer



Idee: Ein Cluster sieht von Außen aus wie ein großer Computer.

Konsequenz: Es müssen als Fundament viele Konzepte klassischer Betriebssysteme übertragen werden (ein Cluster-Betriebssystem). Das gilt insbesondere auch für das Scheduling.

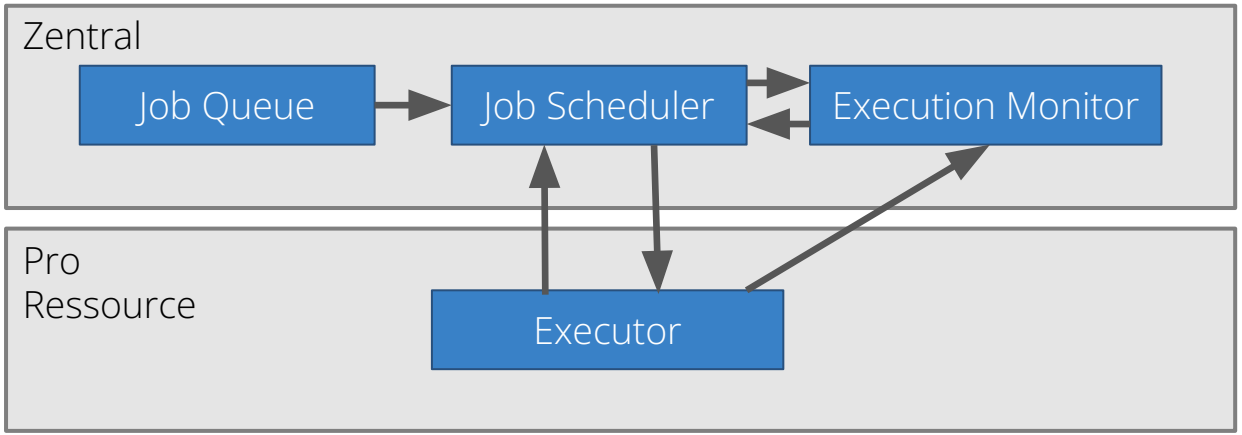
Buch: The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, 2009, Luiz André Barroso und Urs Hölzle

Eine konzeptionelle Architektur für Cluster-Scheduler

Job Queue:
- Eingehende Jobs zur Ausführung
- Events zu eingegangenen Jobs

Job Scheduler:
- Jobs einplanen
- Taskausführung steuern

Execution Monitor:
- Task-Ausführung überwachen
- Ressourcen überwachen



Anforderungen:
- Performance
  - Geringe Queing-Time
  - Geringe Decision-Time
  - Geringe Ausführungslatenz
- Hoch-Verfügbarkeit und Fehlertoleranz
- Skalierbarkeit bzgl. Anzahl an Jobs und verfügbaren Ressourcen.

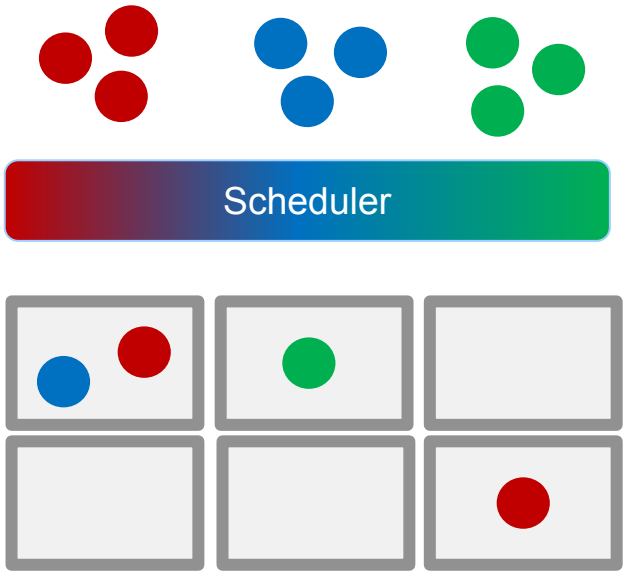
Executor:
- Task ausführen
- Informationen zur Ressource bereitstellen

Scheduler-Architektur Variante 1: kein Scheduler

Statische Partitionierung

Scheduler-Architektur Variante 2: Monolithischer Scheduler

- **Google Borg**
  Large-scale cluster management at Google with Borg, Verma et al., 2015
- **Hadoop YARN**
- **Kubernetes**
- **Docker Swarm**



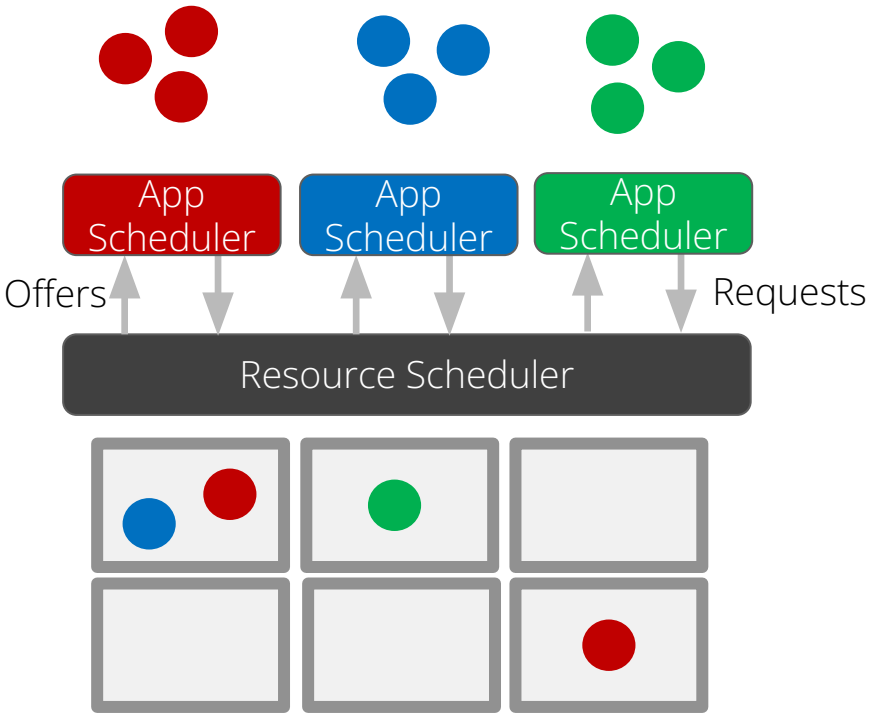
Vorteile:

- Globale Optimierungsstrategien einfach möglich.

Nachteile:

- Heterogenes Scheduling für heterogene Jobs schwierig
  - Komplexe und umfangreiche Implementierung notwendig
  - … oder homogenes Scheduling von geringerer Effizienz.
- Potenzielles Skalierbarkeits-Bottleneck.

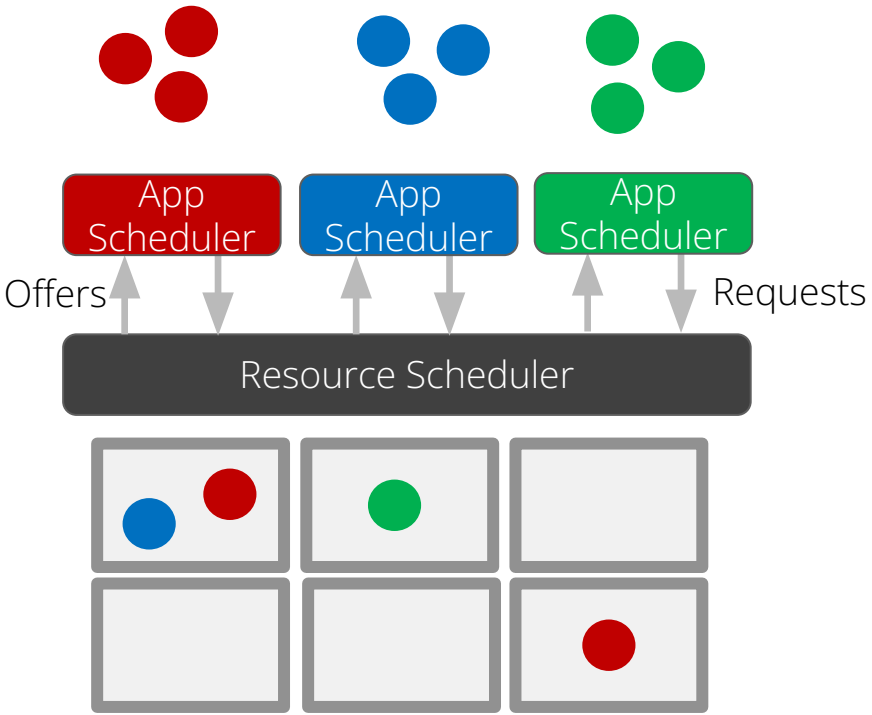
Scheduler-Architektur Variante 3: 2-Level-Scheduler



- Auftrennung der Scheduling-Logik in einen Resource Scheduler und einen App Scheduler.
  - Der **Resource Scheduler** kennt alle verfügbaren Ressourcen und darf diese allokieren. Er nimmt Ressourcen-Anfragen (Requests) entgegen und unterbreitet entsprechend einer Scheduling-Policy (definierte Scheduling-Ziele) Ressourcen-Angebote (Offers).
  - Der **App Scheduler** nimmt Jobs entgegen und „übersetzt“ diese in Ressourcen-Anfragen und wählt applikationsspezifisch die passenden Ressourcen-Angebote aus.
- Offers sind eine zeitlich beschränkte Allokation von Ressourcen, die explizit angenommen werden muss.
- Grundsätzlich **pessimistische Strategie**: Disjunkte Offers. I.d.R. sind aber auch optimistische Offers verfügbar, bei denen eine gewisse Überschneidung erlaubt ist.
- Im Sinne der Fairness kann ein prozentualer Anteil der Ressourcen für einen App Scheduler garantiert werden.

Scheduler-Architektur Variante 3: 2-Level-Scheduler

Apache Mesos
Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, Hindman et al., 2010



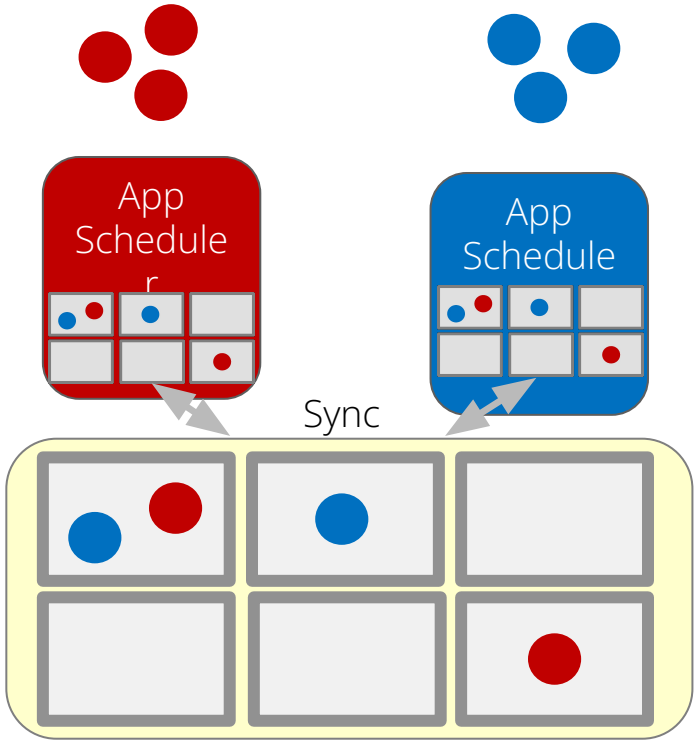
Vorteile:

- Nachgewiesene Skalierbarkeit auf tausende von Knoten (z.B. Twitter, Airbnb, Apple Siri).
- Flexible Architektur für heterogene Scheduling-Logiken.

Nachteile:

- App-Scheduler übergreifende Logiken nur schwer zu realisieren (z.B. globaler Ausführungsverzicht oder Gang-Scheduling)

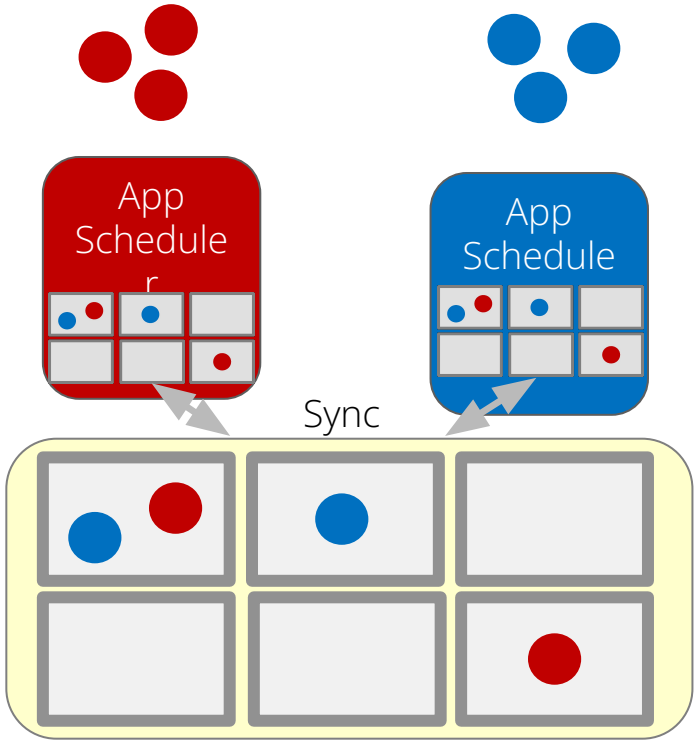
Scheduler-Architektur Variante 4: Shared-State-Scheduler



- Es gibt ausschließlich applikationsspezifische Scheduler.
- Die App Scheduler synchronisieren kontinuierlich den aktuellen Zustand des Clusters (Cluster-Zustand: Job-Allokationen und verfügbare Ressourcen).
- Jeder App Scheduler entscheidet die Platzierung von Tasks auf Basis seines aktuellen Cluster-Zustands.
- Optimistische Strategie: Ein zentraler Koordinationsdienst erkennt Konflikte im Scheduling und löst diese auf, indem er Zustands-Änderungen nur für einen der beteiligten App Scheduler erlaubt und für die anderen App Scheduler einen Fehler meldet.

Scheduler-Architektur Variante 4: Shared-State-Scheduler

Google Omega
Omega: flexible, scalable schedulers for large compute clusters, Schwarzkopf et al., 2013



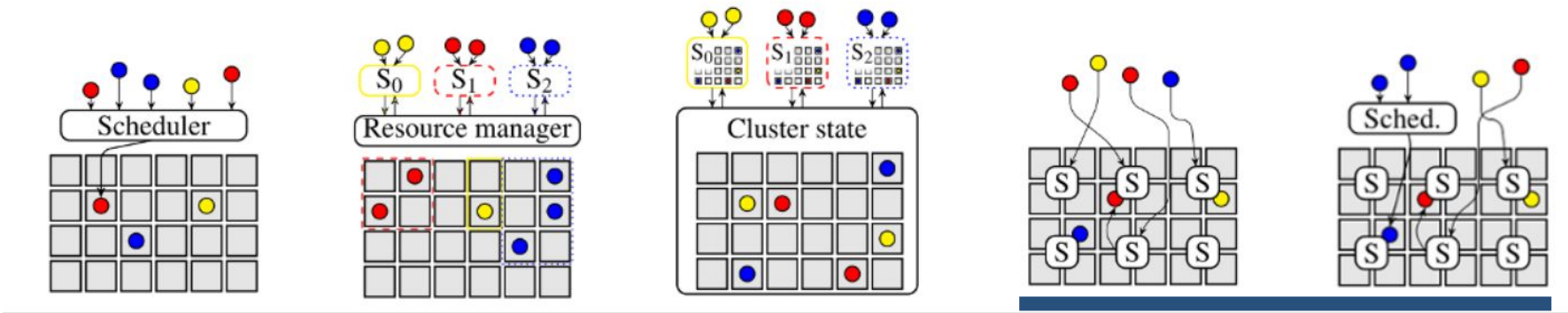
Vorteile:

- Tendenziell geringerer Kommunikations-Overhead.

Nachteile:

- Komplettes Scheduling muss pro App Scheduler entwickelt werden.
- Keine globalen Scheduling-Ziele (z.B. Fairness) möglich.
- Skalierbarkeit in großen Clustern unklar, da noch nicht in der Praxis erprobt und insbesondere Auswirkung bei hoher Anzahl an Konflikten ungeklärt.

Weitere Scheduler-Varianten



(a) Monolithic scheduler. (b) Two-level scheduling. (c) Shared-state scheduling. (d) Distributed scheduling. (e) Hybrid scheduling.

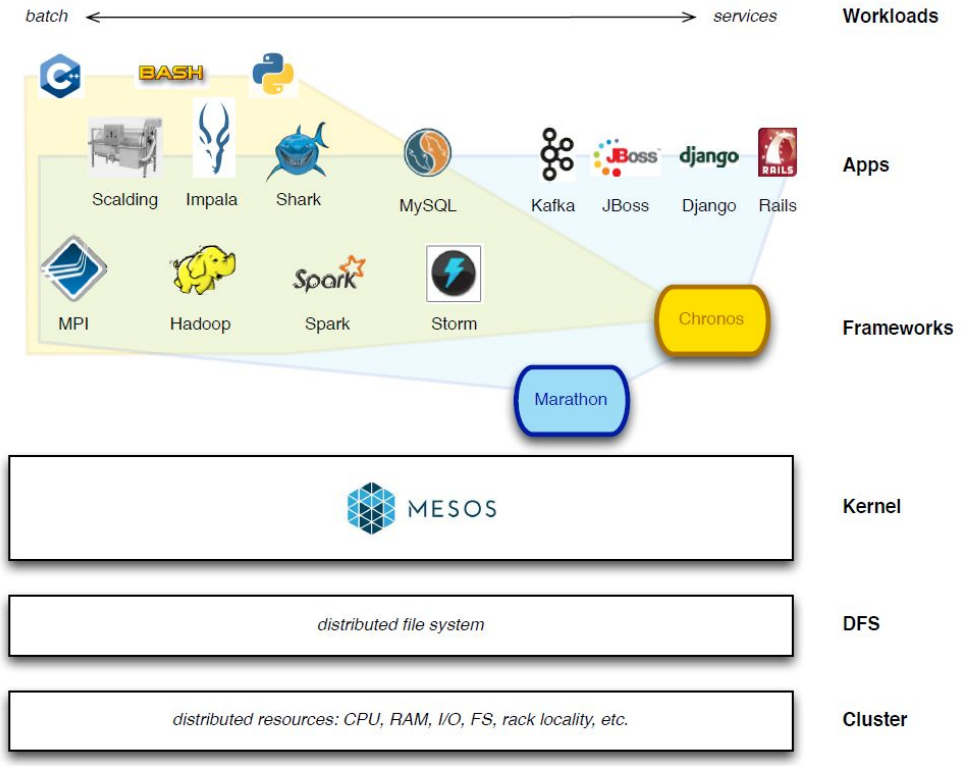
Figure 1: Different cluster scheduler architectures. Gray boxes represent cluster machines, circles correspond to tasks and S_i denotes scheduler i.

siehe: http://firmament.io/blog/scheduler-architectures.html

Beispiel: Apache Mesos

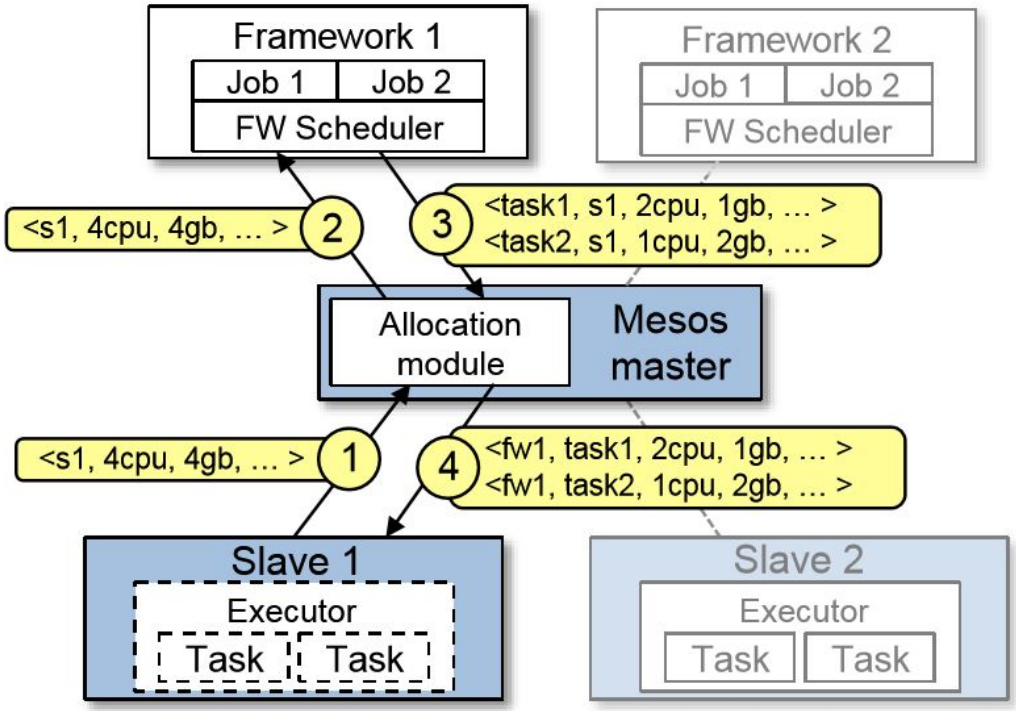
Apache Mesos

- Entstanden an der UC Berkeley im Rahmen der Arbeiten von Ben Hindeman. 1. Release 2009.
- Open Source Projekt unter der Apache Lizenz 2.0.
- Ist im Kern ein Cluster-Scheduler und fasst sich selbst als Cluster-Betriebssystem auf.
- Mesos ist als Cluster-Scheduler in DC/OS einem Open-Source Cluster-Betriebssystem enthalten.
- 2-Level-Scheduler mit DRF als Default-Scheduling-Algorithmus.
- Alle Bestandteile von Mesos können ausfallsicher ausgelegt werden.
- Wird im großen Stil eingesetzt bei Twitter, Apple, Microsoft, Verizon, CERN, Airbnb, ...
- Alle Teile sind per REST-API zugänglich. Task-Isolation per Docker oder eigenem Mechanismus.



Quelle: https://www.typesafe.com/blog/play-framework-grid-deployment-with-mesos

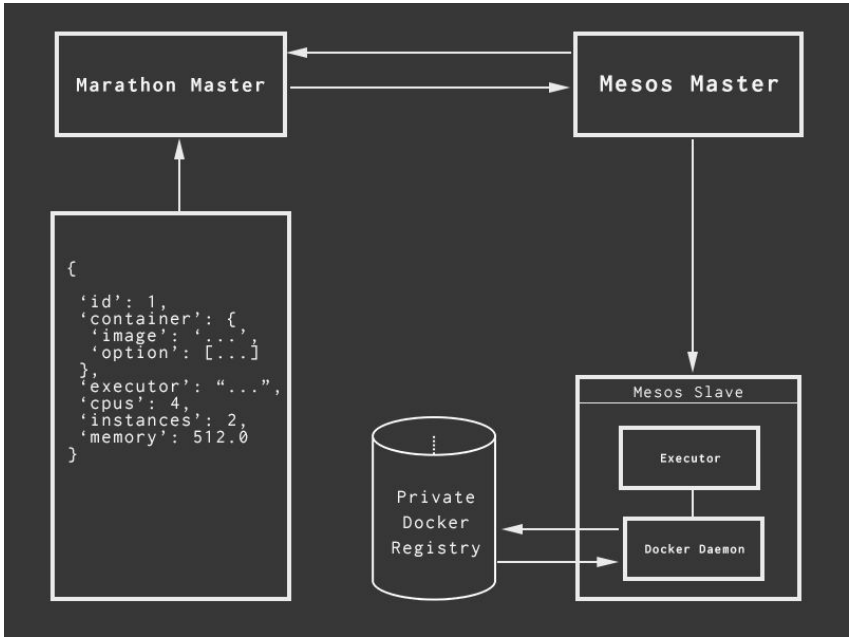
Apache Mesos: Zusammenspiel der Bausteine



Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, Hindman et al., 2010

Marathon ist der 2nd-Level-Scheduler in Mesos, der auf die Ausführung von zustandslosen Services ausgelegt ist

- Wurde initial von Tobi Knaup entwickelt als Mesos-Aufbau um langlaufende, zustandslose Services zuverlässig und komfortabel ausführen zu können.
- Besitzt eigenständige Web-UI und REST-API.
- Prozess werden kontinuierlich am Leben gehalten. Terminiert ein Prozess, so wird er automatisch wieder gestartet.
- Mechanismus für Health-Checking von Services.
- Eingebauter Mechanismus für Service Discovery und Load Balancing.



Übung 3:
Scheduling mit Nomad

Quellen

Videos zum Thema Cluster Scheduling

Scheduling in Cloud by Ms. Mala Kalra, https://youtu.be/hhwCc5yPylg

Omega: flexible, scalable schedulers for large compute clusters, https://youtu.be/XsXlm4wmB6o

Cluster Management and Containerization, Ben Hindman, https://youtu.be/F1-UEIG7u5g

Literatur

- Omega: flexible, scalable schedulers for large compute clusters, Schwarzkopf et al., 2013
- Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, Hindman et al., 2010
- Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis, Charles Reiss et al., 2012
- Scheduling Algorithms for Grid Computing: State of the Art and Open Problems, Dong et al., 2006
- Scheduling in Grid Computing Environment, Prajapati, 2014
- A Survey on Task Scheduling For Parallel Workloads in the Cloud Computing System, Abhijeet Tikar et al., 2014
- Multi-Resource Packing for Cluster Schedulers Paper, Grandl, 2014
- Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, Ghodsi et al., 2011
- Mesos, Omega, Borg – A Survey: http://www.umbrant.com/blog/2015/mesos_omega_borg_survey.html
- Übersicht zum Scheduling in Mesos: http://cloudarchitectmusings.com/2015/04/08/playing-traffic-cop-resource-allocation-in-apache-mesos