Scheduling in Warehouse-scale Data Centers

Francisco Brasileiro
fubica@computacao.ufcg.edu.br

Agenda

- Day one: state-of-practice
  - Warehouse-scale data center
  - Typical workloads
  - The scheduling problem
  - Google's Borg priority-based scheduler
- Day two: QoS-driven scheduling
  - Typical Service Level Agreements (SLA)
  - Limitations of priority-based schedulers
  - QoS-driven scheduling
  - Availability-driven scheduler

Warehouse-scale data center typical structure

- Large cloud computing providers such as Google, Microsoft and Amazon operate this kind of data center

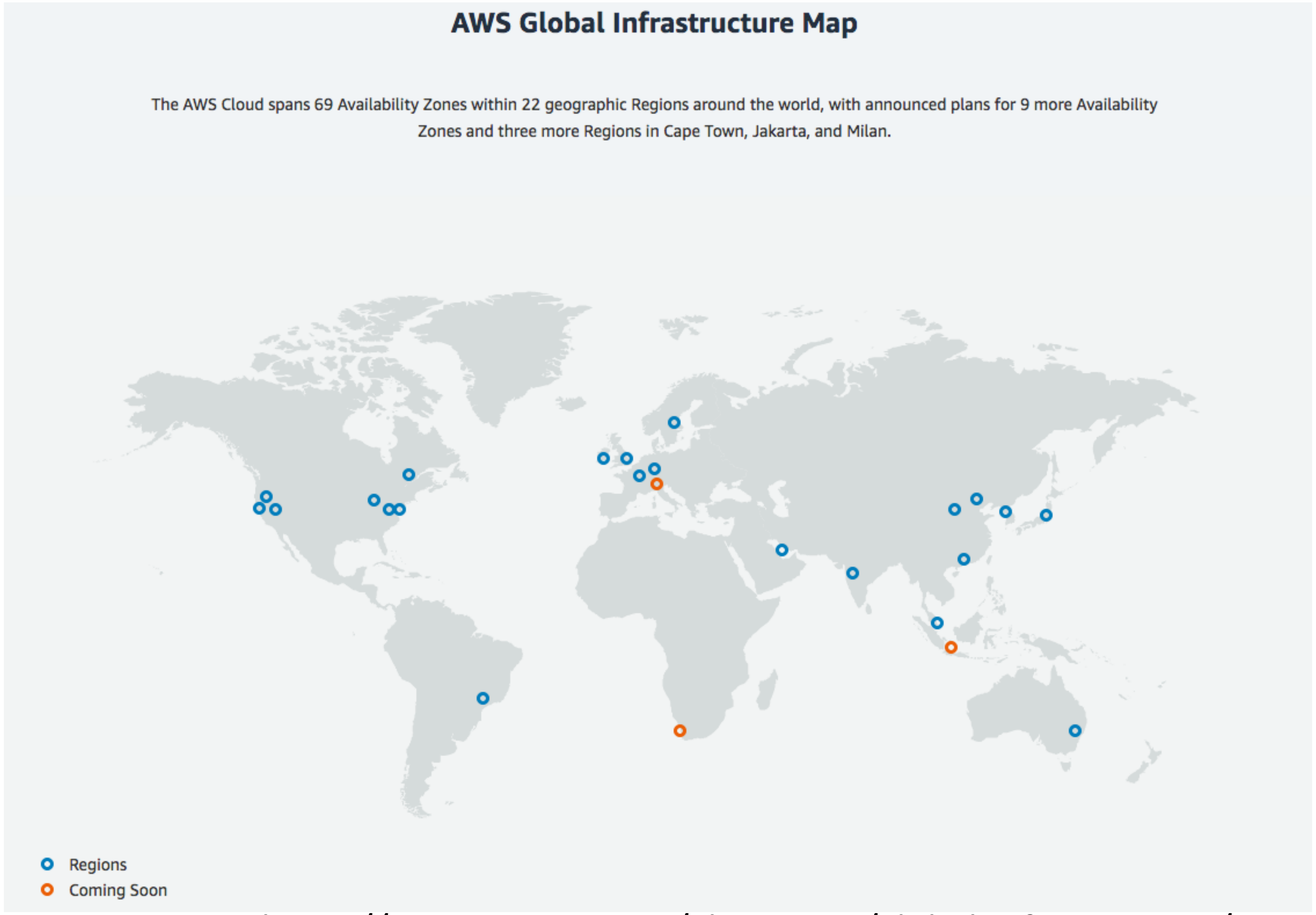
Microsoft data center in Cheyenne, Wyoming, USA

Source: https://www.datacenterknowledge.com/microsoft/microsoft-build-fifth-massive-western-us-azure-region

Google's data center campus in Eemshaven, Netherlands

Source: https://www.datacenterknowledge.com/google-alphabet/google-spend-11-billion-new-data-centers-netherlands

Warehouse-scale data center typical structure



AWS Global Infrastructure Map

The AWS Cloud spans 69 Availability Zones within 22 geographic Regions around the world, with announced plans for 9 more Availability Zones and three more Regions in Cape Town, Jakarta, and Milan.

Source: https://aws.amazon.com/about-aws/global-infrastructure/

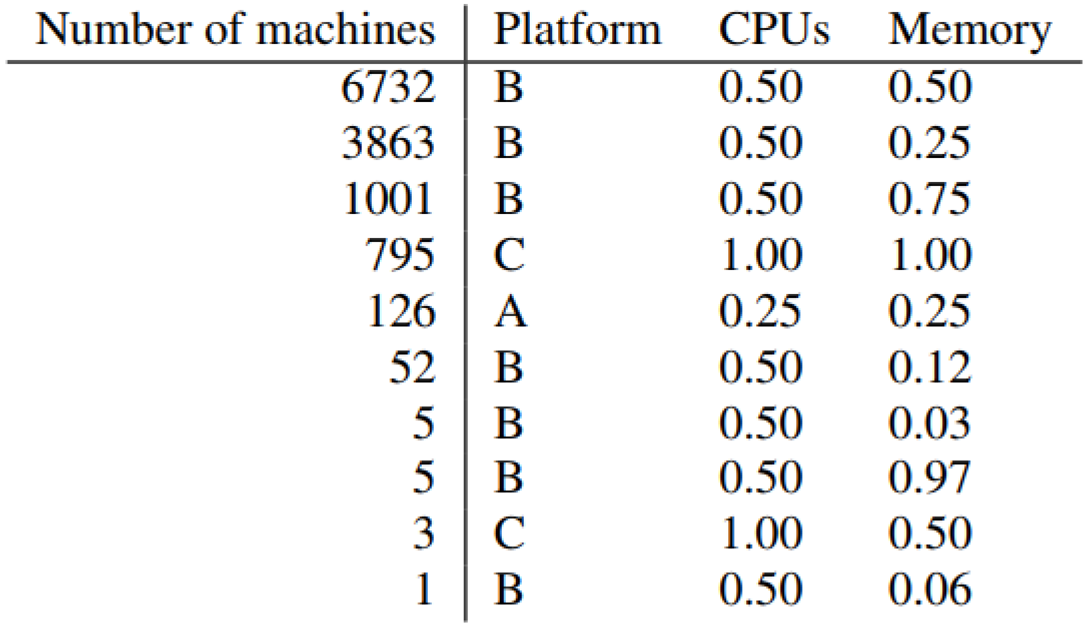
Warehouse-scale data center what is inside?

- Each data center holds a large number of clusters
- Typically, scheduling (and other management activities) are performed at the cluster level
- Google released some information in 2011 that provides some idea of how a typical cluster looks like

[John Wilkes. 2011. More Google cluster data. Google research blog.]

Warehouse-scale data center
a typical cluster at Google

- Configurations of the machines in the cluster
  - Resources are presented as a percentage of the server with the highest capacity for CPU and RAM



Number of machines	Platform	CPUs	Memory
6732	B	0.50	0.50
3863	B	0.50	0.25
1001	B	0.50	0.75
795	C	1.00	1.00
126	A	0.25	0.25
52	B	0.50	0.12
5	B	0.50	0.03
5	B	0.50	0.97
3	C	1.00	0.50
1	B	0.50	0.06

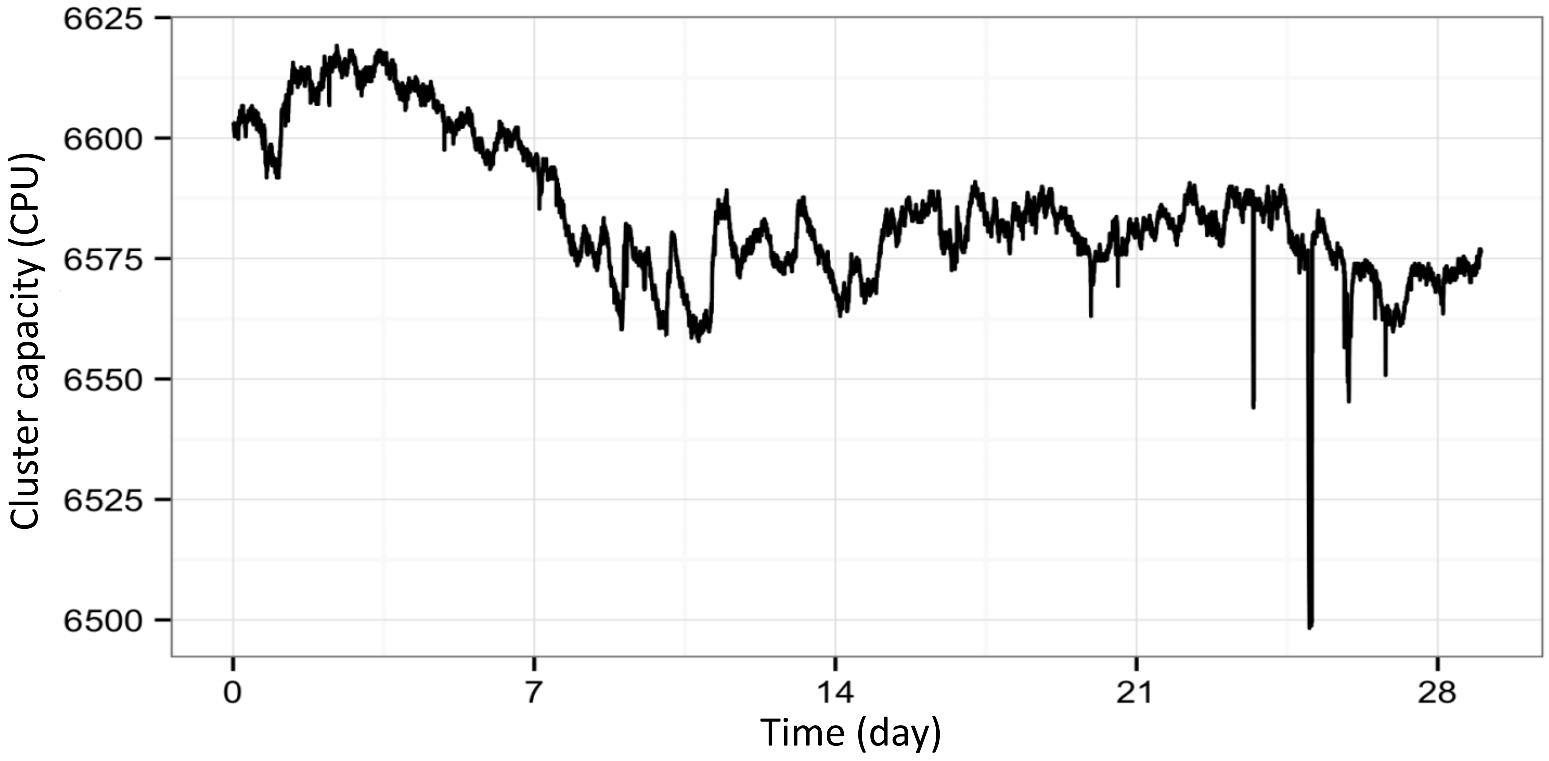
[Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. **Heterogeneity and dynamicity of clouds at scale: Google trace analysis**. In Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)]

Warehouse-scale data center
a typical cluster at Google

- Cluster heterogeneity also comes from “machine attributes”
  - <key,value> pairs
  - Total of 67 unique machine attribute keys
  - These attributes can be used to define placement constraints for tasks
    - E.g. chipset, SSD storage availability, number of disks, public IP, OS kernel version, etc.

Warehouse-scale data center
a typical cluster at Google

- Total cluster capacity over time



Typical workloads
Google's trace summary

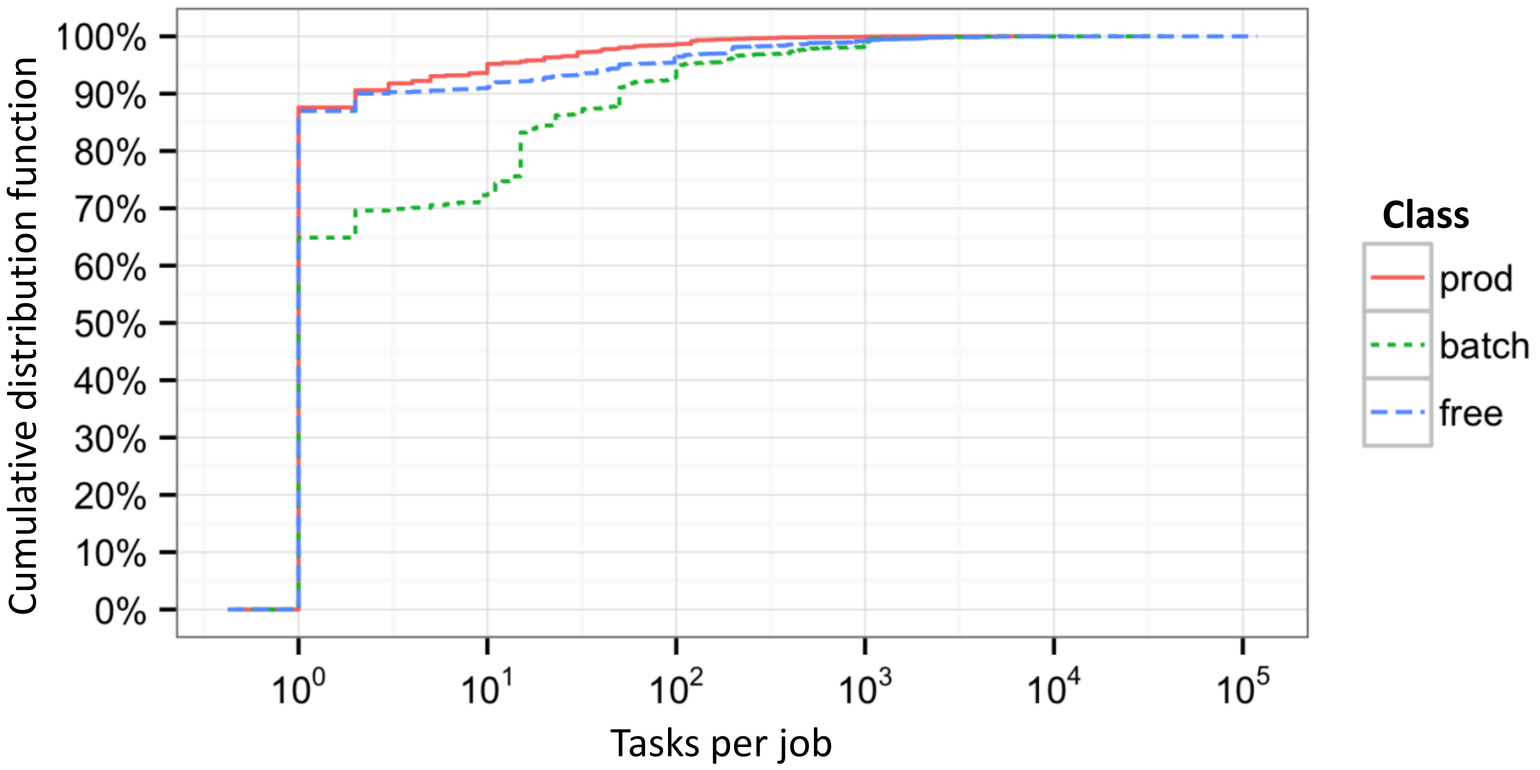
- Spans 29 days of May 2011
- Describes hundreds of requests submitted by 925 different users
- A request (or job) is composed of one or more tasks, which are programs to be executed on a machine
  - The trace contains more than 25 million tasks
- A task is defined by its service class, resource demand, duration, and constraints
  - There are 12 different service classes

Typical workloads
Google's trace analysis

- To simplify the analysis, the 12 service classes are grouped into 3 higher level service classes, that have common characteristics:
  - Prod (top 3 service classes)
    - These are critical long-running user-facing services or monitoring services that should be running all the time
  - Free (lower 2 service classes)
    - These are less important jobs, typically used in testing phase, that run only when there are free resources available
  - Batch (7 intermediate service classes)
    - These are batch jobs that need some minimal quality of service guarantees

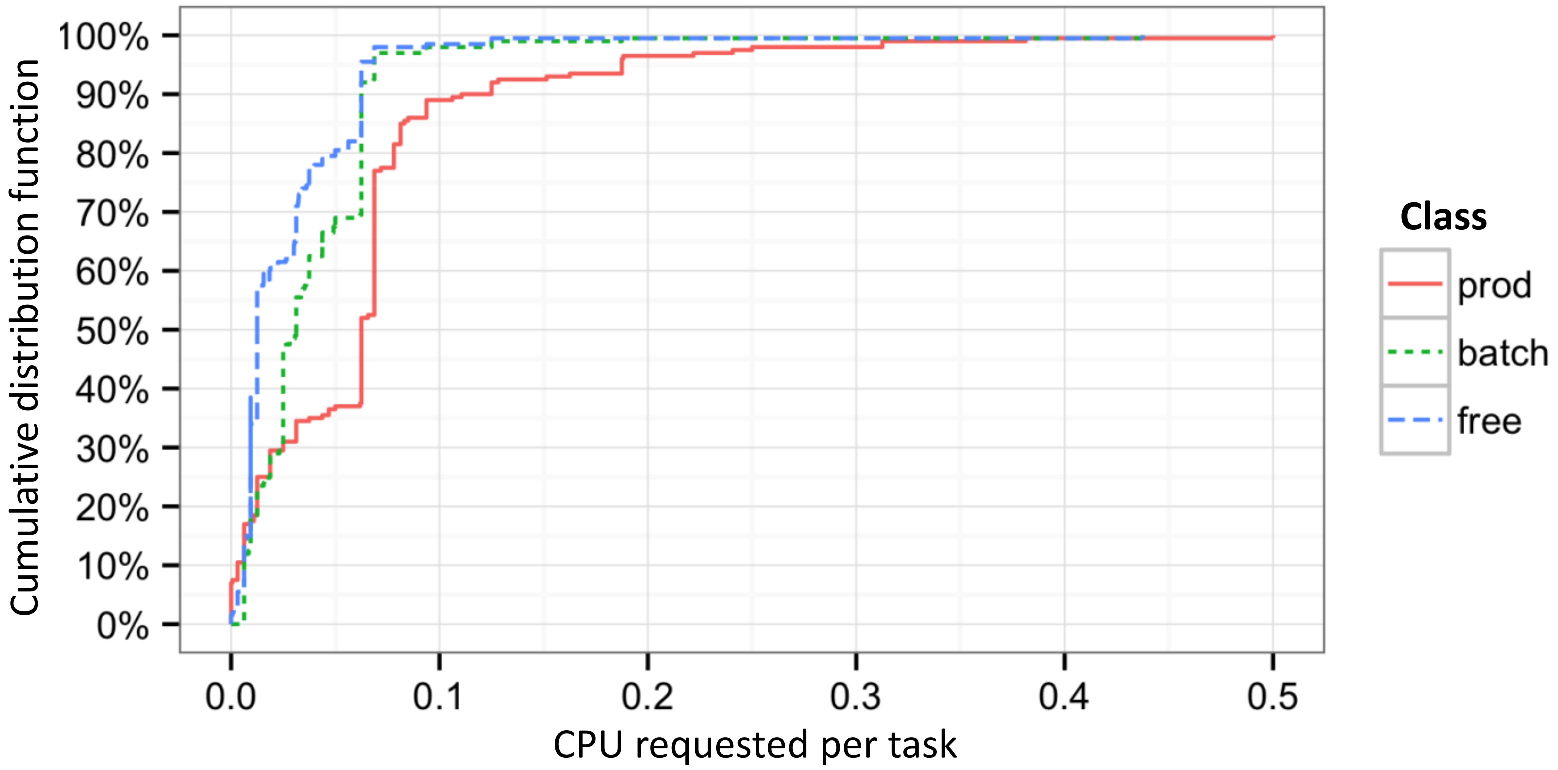
Typical workloads
workload heterogeneity

- Number of tasks per job



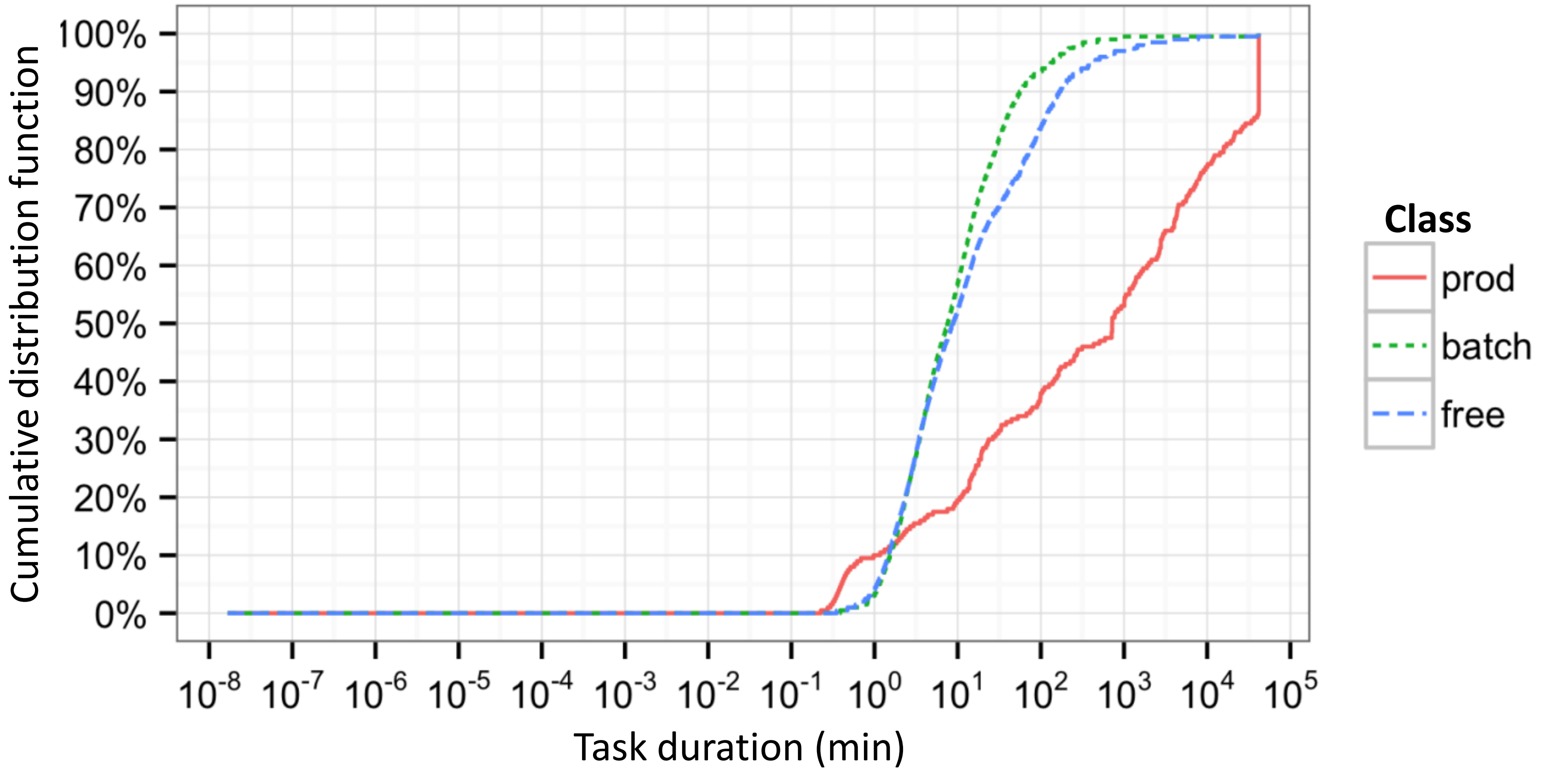
Typical workloads
workload heterogeneity

- Quantity of CPU requested by a task



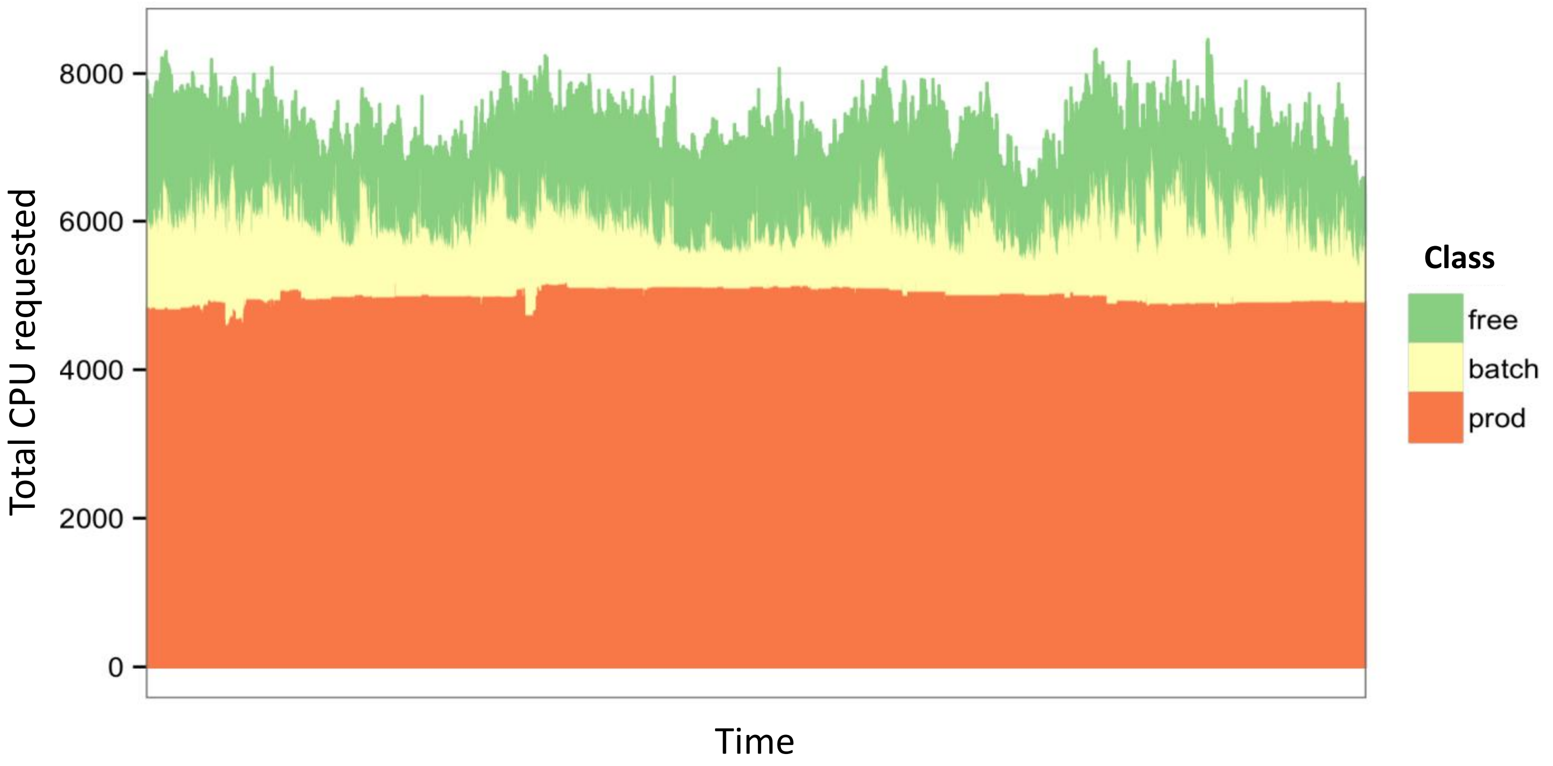
Typical workloads
workload heterogeneity

- Task duration



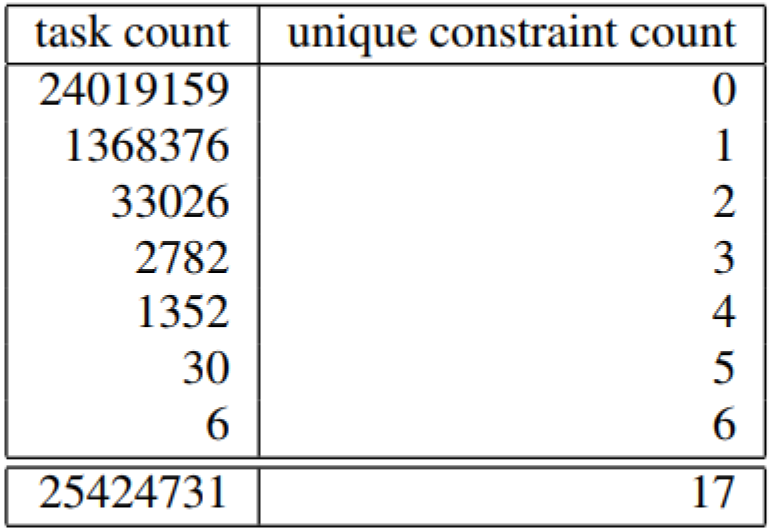
Typical workloads
workload heterogeneity

- Quantity of CPU requested over time



Typical workloads
workload heterogeneity

- ~6% of all tasks submitted have constraints
- 17 attributes are used in the constraints
  - Most of constrained tasks do so over a single attribute



task count	unique constraint count
24019159	0
1368376	1
33026	2
2782	3
1352	4
30	5
6	6
25424731	17

[Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. **Heterogeneity and dynamicity of clouds at scale: Google trace analysis**. In Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)]

The scheduling problem
some basic terminology

- Servers
  - A server s can be modelled as a tuple:
    - s = <id, rc, attrib>, where:
      - id is the identification of the server
      - rc = (c_1, c_2, ..., c_n) is the server capacity for different resource types (eg. CPU, RAM, disk, etc.), where c_i is the server capacity for resource type i
      - attrib are the attributes associated to the server

The scheduling problem
some basic terminology

- Requests
  - A request r can be modelled as a tuple:
    - r = <id, sc, T, pc>, where:
      - id is the identification of the request
      - sc is the service class to which the request has been submitted
      - T is the set of tasks that comprises the request
      - pc are the placement constraints that might be imposed to the set of tasks T

The scheduling problem
some basic terminology

- Tasks
  - A task t can be modelled as a tuple:
    - t = <id, sw, hw>, where:
      - id is the identification of the task
      - sw are the software requirements to execute the task
      - hw are the hardware requirements to execute the task
        - » Hardware requirements are specified as a sequence of demands for the different resource types
          - hw = (d_1, d_2, ..., d_n), where d_i is the demand for the resource type i
    - Typically, tasks that belong to the same requests have the same hardware and software requirements

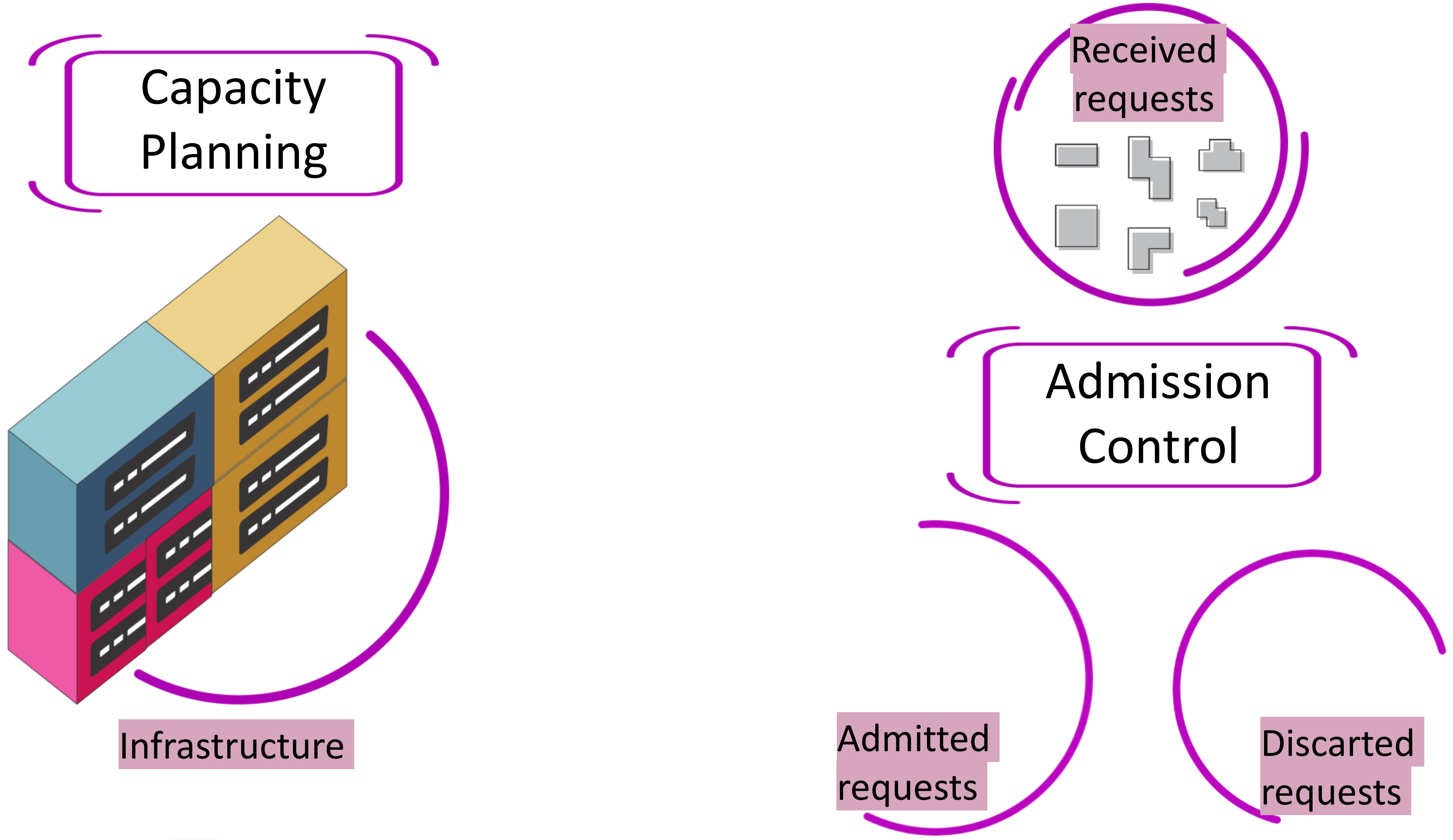
COMPUTAÇÃO UFCG

Universidade Federal de Campina Grande

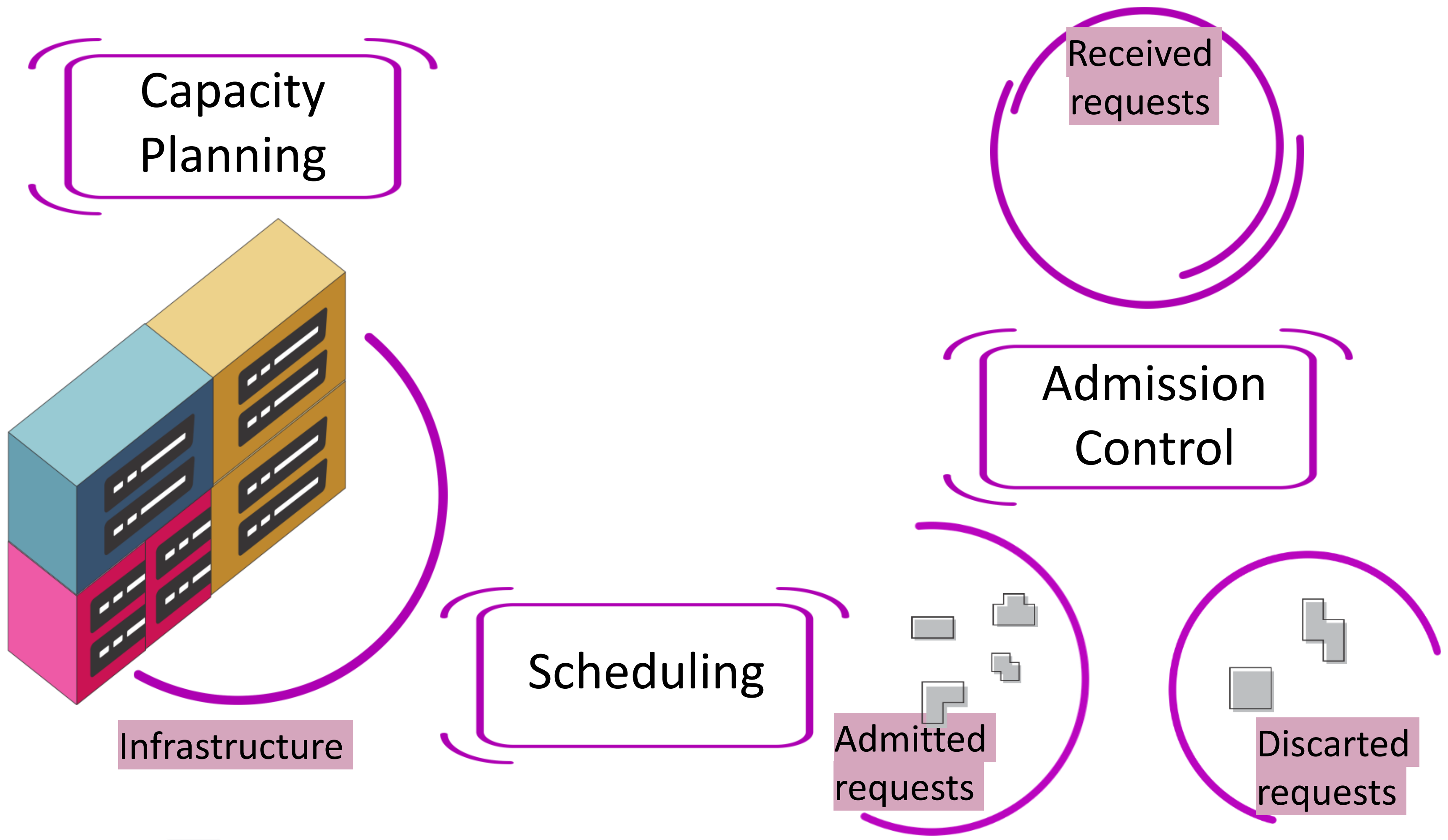
The scheduling problem
task isolation

- Tasks must execute inside private environments
  - This provides security and performance isolation
  - Virtualization technologies are typically used to provide such isolation
    - e.g. virtual machines or containers
  - The bottom line, however, is that a task is executed as an isolated process running at some physical server

The scheduling problem
before scheduling



The scheduling problem
scheduling



The scheduling problem
scheduling at different time frames

- Multiple schedulers co-exist
  - The long term scheduler decides which processes should be active at which servers
  - The short term scheduler decides which of the processes stored in main memory should be using the CPU of the server it controls
- Our focus in this course is on the long term scheduler

The (long term) scheduling problem formal definition

- Let:
  - ω_τ be the workload at time τ, i.e. the set of tasks associated to all requests that have been admitted at or before τ, and have not yet completed
  - σ_τ be the set of servers that comprise the infrastructure at time τ
- A schedule $\mu(\omega_\tau, \sigma_\tau)$ produces a mapping $<\alpha_\tau, \pi_\tau>$, where α_τ is the allocation map, and π_τ is the pending list at time τ
  - The allocation map α_τ defines which tasks from ω_τ should be allocated at each of the servers that comprise σ_τ at time τ
  - The pending list π_τ is comprised by all tasks in ω_τ that have not been allocated to a server at time τ

The (long term) scheduling problem formal definition

- The schedule $\mu(\omega_\tau, \sigma_\tau) = <\alpha_\tau, \pi_\tau>$ must satisfy all software and hardware requirements, as well as placement constraints of all tasks assigned to some server s
- In particular, the sum of the demand (d_i) of all tasks that have been assigned to a server s, for any resource type i, needs to be at most equal to s's capacity (c_i) for resource type I
- Other constraints that the scheduler need to comply are related to the objective function of the scheduler
  - E.g. maximize resource utilization

The (long term) scheduling problem main challenges

- While trying to achieve its goals, the scheduler needs to deal with two basic challenges
  - Avoid problems that lead to low resource utilization
  - Deal with the scale of the infrastructure and the workload received

The (long term) scheduling problem resource utilization issues

- Minimize the quantity of wasted resources due to fragmentation
  - Resource fragmentation is caused by very small resource leftovers in the hosts that are not big enough to serve typical requests
- Minimize resource stranding
  - Resource stranding occurs when there are relatively large quantities of resources that cannot be allocated because there is not enough resources of another kind that need to be bundled together to serve a request

The (long term) scheduling problem
scheduler architectures: scale issue

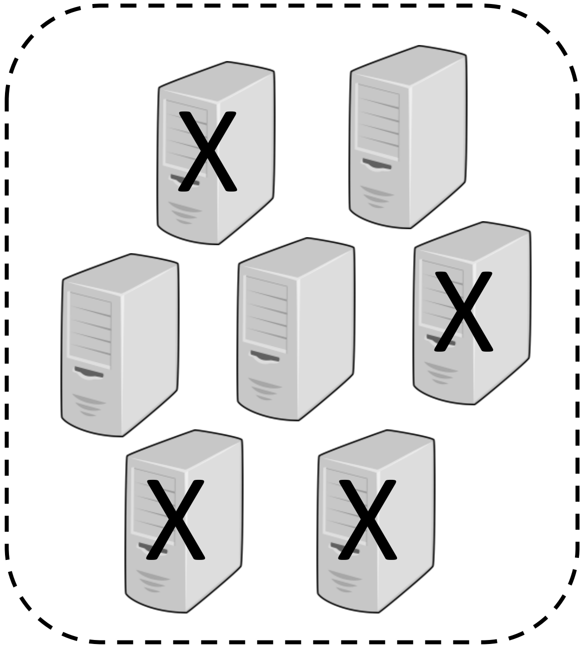
- Monolithic
  - Make decisions in a centralized way, considering the cluster as a whole (e.g. Firmament, Quincy)
- Two-level
  - Have a single resource manager with cluster state information and offer resources to multiple parallel “scheduler frameworks” (e.g. Mesos, YARN)
- Distributed
  - Share the cluster state among different scheduler instances (e.g. Apollo, Tarcil, Sparrow, Omega, Borg)
- Hybrid
  - Make centralized decisions for part of the workload and distributed ones for the remaining part (e.g. Hawk, Mercury)

Google's Borg priority-based scheduler basic algorithm

- Each task has a priority assigned to it, which depends on the service class it belongs
- The pending queue is ordered considering the priority of the tasks
  - User-based Round-Robin order for tasks of the same priority
- Periodically, the scheduler tries to allocate all tasks that are left in the pending queue
- For each task t, two operations are performed
  - Feasibility checking of servers for task t
  - Ranking of feasible servers for task t

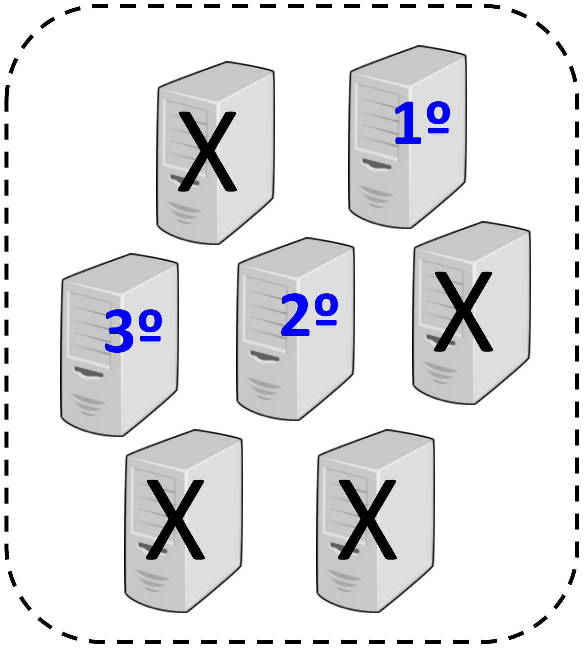
Google's Borg priority-based scheduler feasibility check

- Are hardware, software and placement restrictions for the task satisfied?
- Does the server have enough free resources to allocate the task?
- Can already allocated tasks be preempted to make room for the new task?

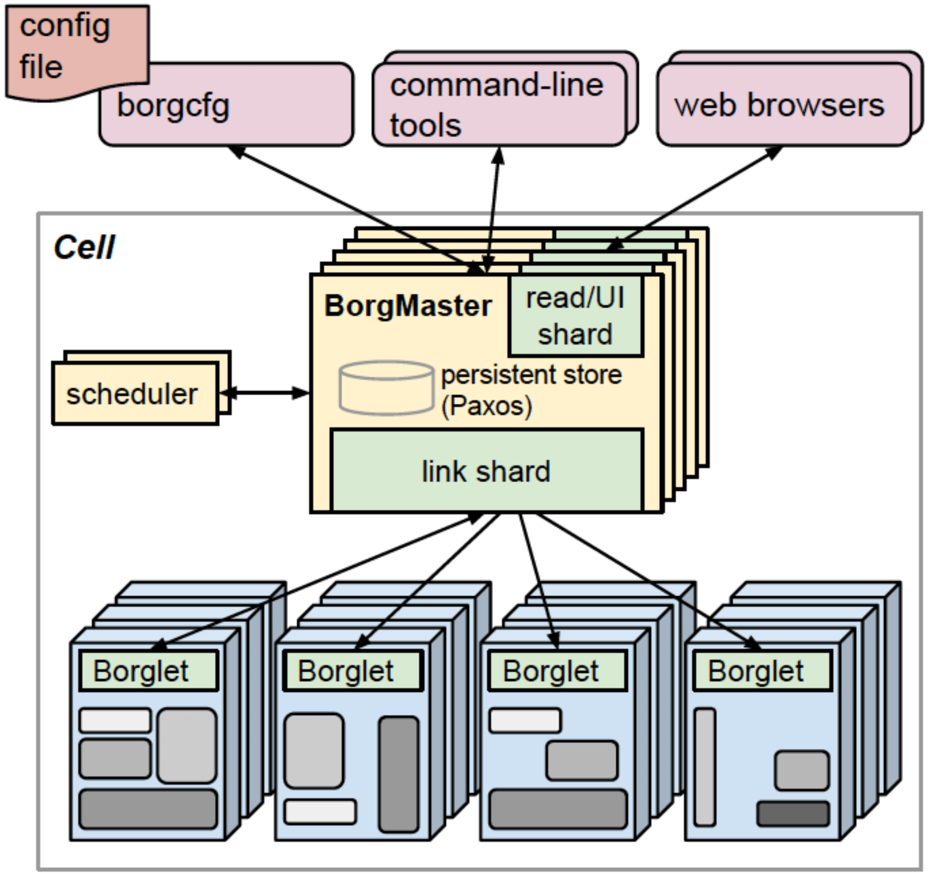


Google's Borg priority-based scheduler ranking

- Main criterea
  - Minimize the number and priority of preempted tasks
  - Mix low and high priority tasks
    - Better deal with over commitment
  - Sum of the fraction of CPU and RAM requested by the task



Google's Borg priority-based scheduler scalability



Google's Borg priority-based scheduler scalability

- Optimizations
  - Score caching
    - Only re-computes scores when changes occur in the server
  - Equivalent task scoring
    - Tasks of the same job have the same requirements, and thus, the server scores are the same for all tasks
  - Relaxed randomization
    - Checks the feasibility of servers only until a minimum number of feasible servers are found