

263-3855-00: Cloud Computing Architecture

Section 04: Cluster Management Policies

Swiss Federal Institute of Technology Zurich (ETH Zürich)

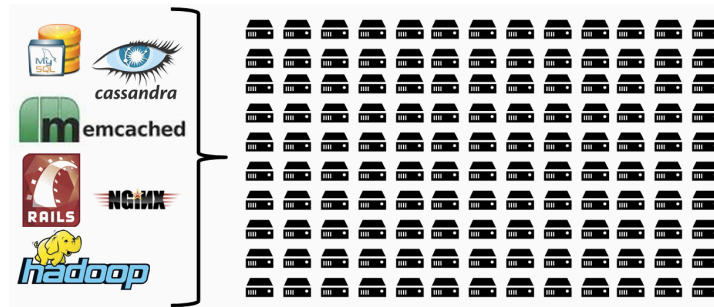
Last Edit Date: 05/31/2025

Disclaimer and Term of Use:

1. We do not guarantee the accuracy and completeness of the summary content. Some of the course material may not be included, and some of the content in the summary may not be correct. You should use this file properly and legally. We are not responsible for any results from using this file.
2. This personal note is adapted (derived) from the slides and codes from *Professor Ana Klimovic and people from ETH Zurich Systems Group*. Please [contact us](#) to delete this file if you think your rights have been violated.
3. This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

I. Resource assignment

How do we assign resources to applications? There are two major options: private v.s. shared assignment.



Private resource assignment

- Each app receives a private, static set of resources
- Also known as static partitioning
- Advantages
 - Simplicity
 - Performance isolation
 - Allows for hardware specialization
- Challenge
 - Low utilization
 - Failures and maintenance
- Still used today for some user-facing, scale-out workloads, e.g., web-serving, search, etc.



Shared resource assignment

- More common case: share machine resources between applications
- We need a scheduler to ensure
 - Fairness
 - Users should be granted resources proportional to how much they are paying
 - Flexibility to express priorities
 - Efficient resource usage
 - Resources should not idle while there are users whose demand is not fully satisfied (this is called work conservation)
 - Performance isolation
 - Guarantee that misbehaved controlled applications cannot affect me “too much”

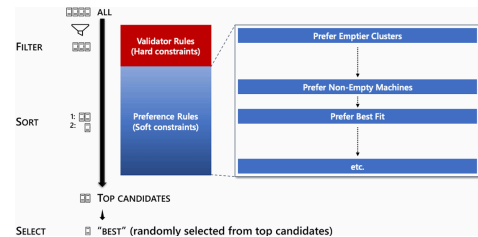
Scheduling: an old problem in operating systems

- CPU allocation
 - Multiple processors want to execute, OS selects one to run for some amount of time

- (Network) bandwidth allocation
 - Packets from multiple incoming queue want to be transmitted out some link, switch chooses one

How should we pick a specific physical node for a VM / container?

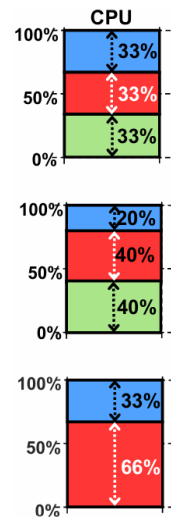
- Filter machines that satisfy hard constraints, e.g., VM may need a machine with a GPU.
- Rank candidate nodes to find machine that best satisfies soft constraints, e.g., best-fit to avoid resource fragmentation.



II. Fair resource allocation algorithms

1. Single resource scheduling - fair sharing

- n users want to share a resource (e.g., CPU)
 - Solution: give each $1/n$ of the shared resource
- Generalized by max-min fairness
 - Handles cases where a user wants less than its fair share
 - E.g., user 1 wants no more than 20%
 - Work conserving or work preserving
 - No unused capacity if there is demand
- Generalized by weighted max-min fairness
 - Give weights to users according to importance
 - User 1 gets weight 1, user 2 weight 2



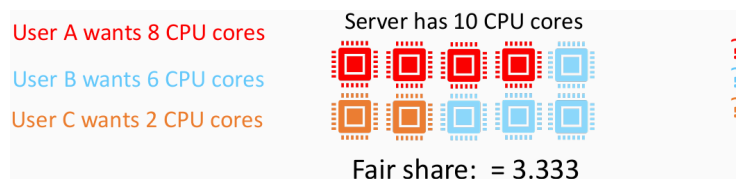
2. Max-min fair share algorithm

We achieve max-min fair share allocation as follows:

- Allocate resources in order to increasing demand
- No user gets a resource share larger than its demand
- Users with unsatisfied demands get an equal share of the resource

In essence, this allocates users with a “small” demands what they want, and evenly distributed unused resources to the “big” users

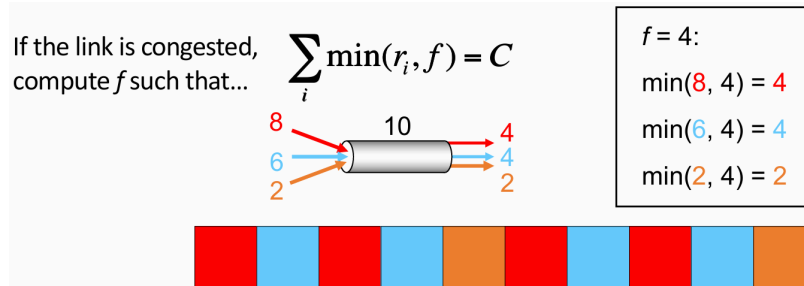
Example:



Fair Queuing

Fair queuing reduces bit-by-bit round robin among flows. Each flow has a separate queue. Each queue is served at rate $\min(r_i, f)$, where

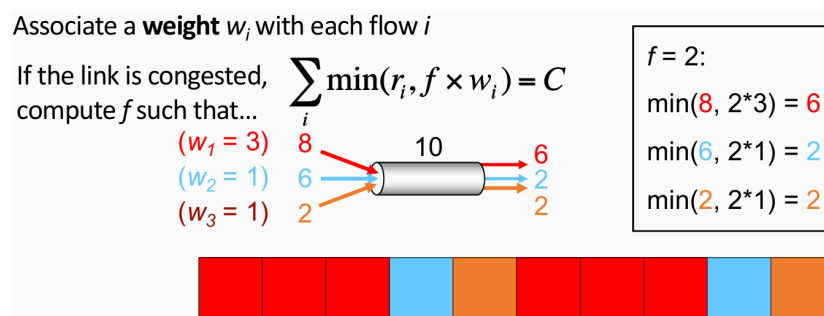
- r_i - flow arrival rate
- f - link fair rate



Weighted fair queuing

Fair queuing reduces bit-by-bit round robin among flows. Each flow has a separate queue. Each queue is served at rate $\min(r_i, f)$, where

- r_i - flow arrival rate
- f - link fair rate



Theoretical properties of max-min fairness

- Share guarantee
 - Each user gets at least $1/n$ of the resource
 - But will get less if their demand is less
- Strategy-proof
 - Users are not better off by asking for more than they need
 - Users have no reason to lie

Max-min fairness if powerful

- Weighted fair sharing / proportional shares
 - User u_1 gets weight 2, u_2 weight 1
- Properties: Give u_1 weight 1000, u_2 weight 1

- Reservations
 - Ensure u1 gets 10%: give u1 weight 10, sum weights ≤ 100
- Deadline-based scheduling
 - Given a job's demand and deadline, compute user's weight
- Performance isolation: Users cannot affect others beyond their share

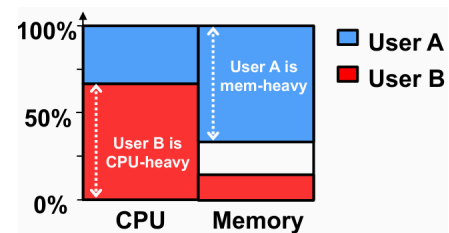
Why is max-min not enough?

- Job scheduling is not only about a single resource; tasks consume CPU, memory, network and disk I/O
- What about performance predictability?

3. Dominant resource fairness (DRF)

Suppose user 1 wants $\langle 1 \text{ CPU}, 4 \text{ GB} \rangle$ per task and user 2 wants $\langle 3 \text{ CPU}, 1 \text{ GB} \rangle$ per task.

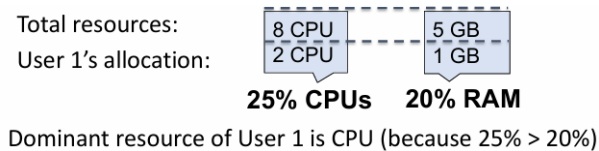
- Example:
 - If user A runs memory-heavy tasks and user B runs CPU-heavy tasks, DRF attempts to equalize user A's share of memory with user B's share of CPU
- Identify the dominant resource share of each user
- Maximize the minimum dominant share across all users.



Online DRF scheduler will maximize the smallest dominant share in the system, then the second-smallest etc.

A user's dominant resource is the resource that the user has the biggest share of

- Example:



A user's dominant share is the fraction of dominant resource allocated

- User 1's dominant share is 25%

Equalize the dominant share of the users

Example:

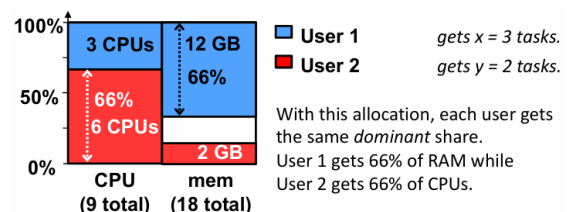
Total resources: $\langle 9 \text{ CPU}, 18 \text{ GB} \rangle$

User 1 demand: $\langle 1 \text{ CPU}, 4 \text{ GB} \rangle$, dominant resource is memory ($1/9 < 4/18$)

User 2 demand: $\langle 3 \text{ CPU}, 1 \text{ GB} \rangle$, dominant resource is CPU ($3/9 > 1/18$)

How to calculate DRF allocation?

- User 1 gets $\langle x \text{ CPU}, 4x \text{ GB} \rangle$



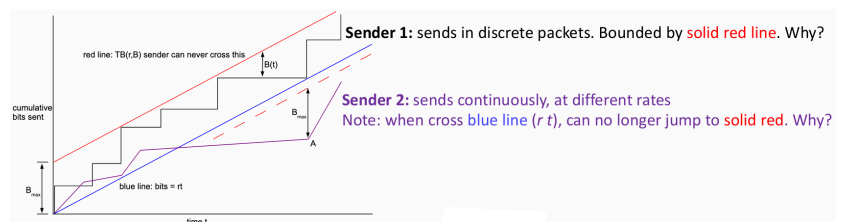
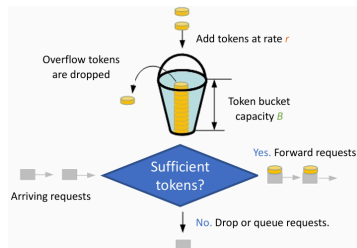
- User 2 gets $\langle 3y \text{ CPU}, y \text{ GB} \rangle$
- Total resources = $(x + 3y) \text{ CPU}$ and $(4x + 7) \text{ GB}$
- Dominant share for user 1: $4x/18 = 2x/9$
- Dominant share for user 2: $3y/9 = y/3$
- Maximize x and y subject to
 - $x + 3y \leq 9$ (CPU constraint)
 - $4x + y \leq 18$ (memory constraint)
 - $2x/9 = y/3$ (equalize dominant shares)

4. Token bucket

- Commonly used for controlling network and storage traffic
 - Goal: guarantee a baseline bandwidth, but also allow bounded bursts
- Motivating example:
 - Network link capacity is 4 requests / ms
 - We want to guarantee user A gets 1 request / ms bandwidth, regardless of collocated workloads
 - Strawman approach:
 - Wait 1ms before transmit A's next request
 - But what is A provided at times 1ms, 2ms, 3ms, 4.5ms, 5ms? Do we wait until 5.5ms to transmit the last request ?
 - What if we want to support an average rate of 1 request / ms and allow bursts if the user was idle for some time?

Idea of token bucket

Control traffic by scheduling requests when they accumulate sufficient tokens. We denote r as the fill rate and B as the bucket capacity.



III. Quasar cluster manager

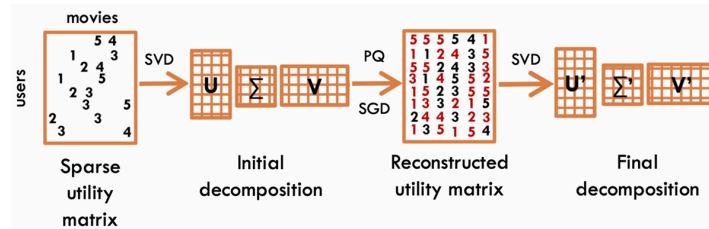
Resource demands are hard for users to specify. To address this problem, Quasar proposes letting users specify performance goals, the cluster manager profiles applications and dynamically adjusts resource allocations.

Understanding resource / performance tradeoffs

- Combine
 - “Small signal” from a short run of a new application
 - “Large signal” from previously run applications
- Get insights for resource management
 - Performance as a function of scale-up / out, hardware heterogeneity, interference, ...
- For each new application, we need to recommend a resource allocation and assignment

1. Recommender systems

- Lots of prior work, for example, recommending movies to users, like Netflix challenge.
- Common technique: collaborating filtering
 - Predict preferences of new users given preference of other users
 - Singular Value Decomposition (SVD) and PQ reconstruction (using SGD)



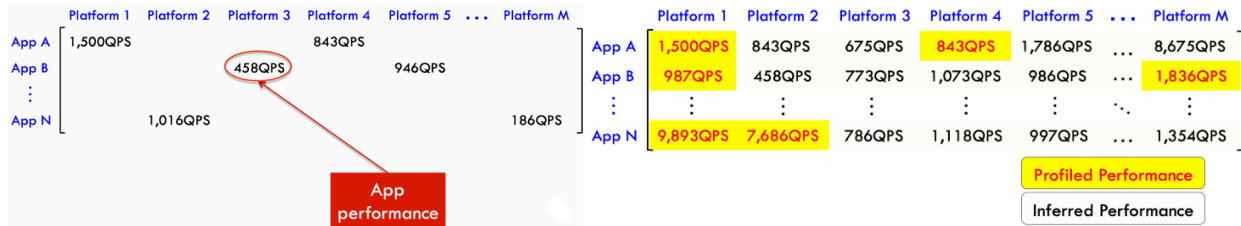
Recommend resource allocations to applications

- Need to consider how performance is affected by:
- Scale-out (e.g., using 4 nodes vs. 1 node)
- Scale-up (e.g., using 8-core VM vs. 2-core VM)
- Hardware heterogeneity (using different CPU / memory configurations)
- Interference (collocating different types of applications)

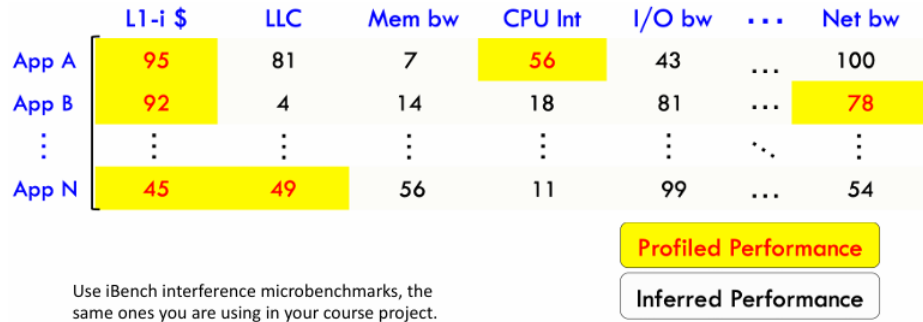
2. Applying collaborative filtering

We can recommend heterogeneous hardware platforms to applications.





We can predict interference sensitivity of applications



3. Steps of Quasar

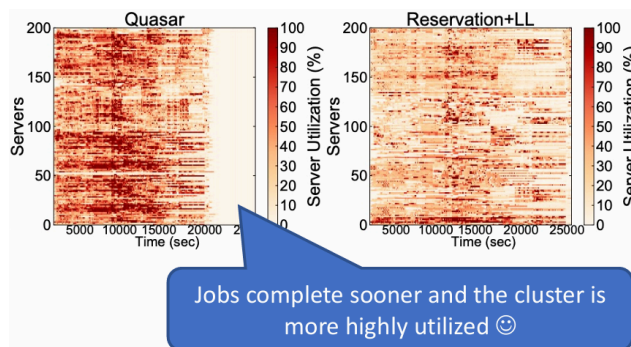
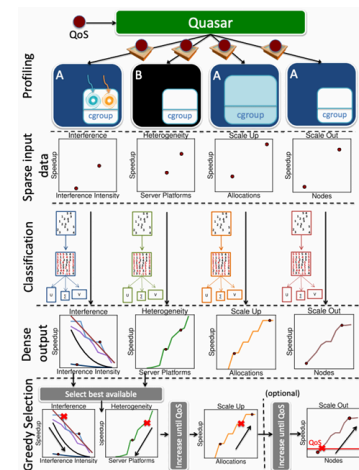
Step 1: Short profiling runs produce initial performance data (takes 10 - 60 s)

Step 2: Collaborative filtering techniques fill in missing data (takes around 20 ms)

Step 3: Greedy scheduler uses output to find the number and type of resources that maximize utilization and performance (takes around 50 ms - 2 s)

Evaluation

- **Setup:** 1200 jobs (analytics batch, latency-critical, single-server jobs) submitted to a 200-node cluster.
- **Baselines:** User reservation and least-loaded (LL) scheduler



IV. Related readings

1. (Optional) [Quasar: Resource-Efficient and QoS-Aware Cluster Management](#)
2. (Optional) [Borg: the Next Generation](#)