Enhancing Grid Middleware Performance

Priority Management and Locking Time optimization in JAliEn

About

- Name: Jørn-Are Flaten
- Profession: Developer
- Master student - Software Engineering
  - Western Norway University of Applied Sciences
  - Supervisors: Bjarte Kileng & Håvard Helstrup
    - From CERN: Costin Grigoras

Context

- Grid middleware - Java Alice Environment (JAliEn)
- Job management to process data captured in the experiment
- Transitioning functionality from AliEn to JAliEn

The challenge

- Keeping resource information across all sites synchronized
- Bookkeeping service to track resource usage and update tables
  - Service exists in AliEn codebase
  - Locks database for a long duration
    - Causes long waiting time to access the database
    - Legacy algorithm must be investigated and improved

Submitting JDL

- Demo - Excalidraw
  - Simplified happy day scenario
  - https://excalidraw.com/#json=Z6SC8e9OKGK7TpEvqkoZ6,IHY0qCC1-SvVqupN88Qg3w

Categories of job states

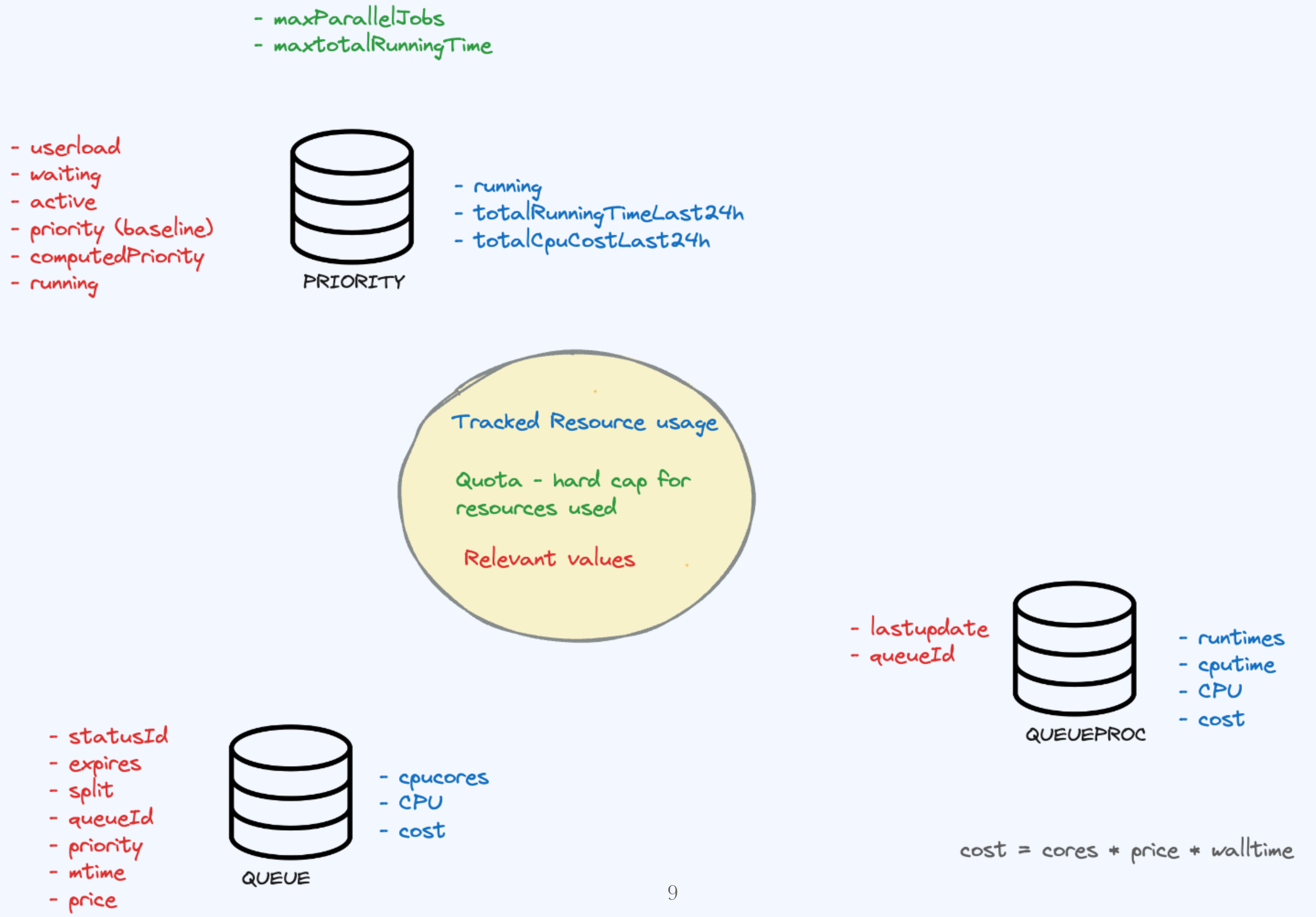
- Queued states
- Waiting states
- Running states
- Final states
- Error states
- Done states

State diagram

- Demo - Excalidraw

Resources

- Several resources are being tracked across tables
- Used to synchronize information from sites
- Resource information is relevant
  - To calculating priority for each user
  - To check if user reached resource usage limits (quota)


- maxParallelJobs
- maxtotalRunningTime
- userload
- waiting
- active
- priority (baseline)
- computedPriority
- running
PRIORITY
- running
- totalRunningTimeLast24h
- totalCpuCostLast24h
Tracked Resource usage
Quota - hard cap for resources used
Relevant values
- lastupdate
- queueId
QUEUEPROC
- runtimes
- cputime
- CPU
- cost
- statusId
- expires
- split
- queueId
- priority
- mtime
- price
QUEUE
- cpucores
- CPU
- cost
cost = cores * price * walltime

The original algorithm

- Calculates computedPriority - basis for who gets to run a job
- Baseline priority of the user is weighted highly
- Originally part of a update query that joined three tables
- 1 job = 1 core
- Quota check for number of running jobs
  - Over quota (max CPU cores or CPU time) = computedPriority set to 1

New algorithm

- 1 job = can use multiple cores
- Quota check on cores used and CPU time spent
  - Over quota = computedPriority set to -1
- New parameter to account for CPU time spent in the last 24 hours
- Current cores in use and recent resource usage added to the calculation
  - Reduced the weighting of baseline priority -> improves fairness

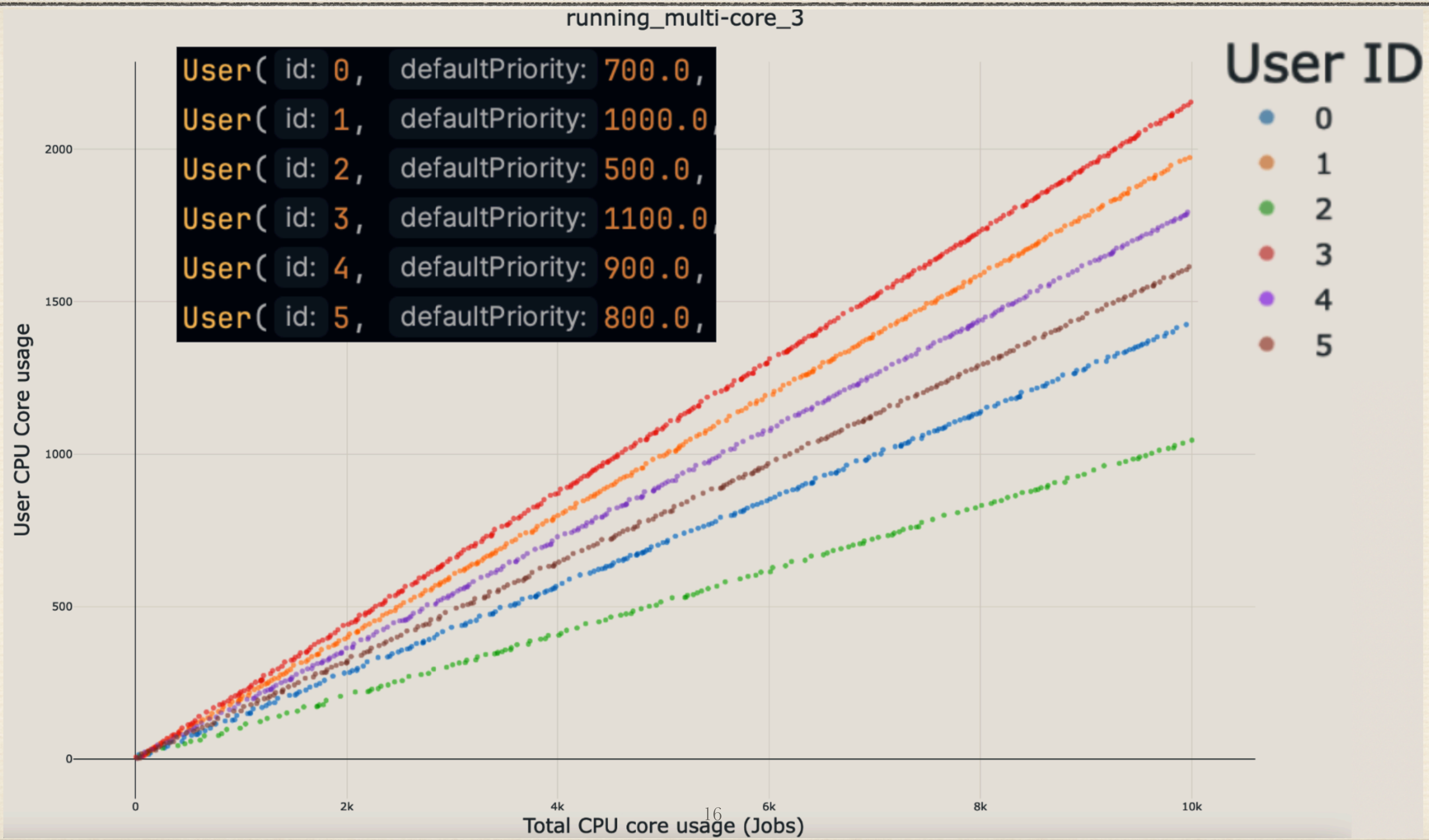
```java
public static double originalCalculateComputedPriority(double userPriority,
                                                        double running,
                                                        double maxParallelJobs) {
    double userload = running / maxParallelJobs;
    return (running < maxParallelJobs) ?
            ((2 - userload) * userPriority > 0 ?
                    50.0 * (2 - userload) * userPriority :
                    1) :
            1;
}
```

```java
private static void updateComputedPriority(PriorityDto dto) {
    if (isQuotaExceeded(dto)) {
        return;
    } else {
        int activeCpuCores = dto.getRunning();
        int maxCpuCores = dto.getMaxParallelJobs();
        long historicalUsage = dto.getTotalRunningTimeLast24h() / dto.getMaxTotalRunningTime();

        if (activeCpuCores < maxCpuCores) {
            double coreUsageCost = activeCpuCores == 0 ? 1 : (activeCpuCores * Math.exp(-historicalUsage));
            float userLoad = (float) activeCpuCores / maxCpuCores;
            dto.setUserload(userLoad);
            double adjustedPriorityFactor = (2.0 - userLoad) * (dto.getPriority() / coreUsageCost);

            if (adjustedPriorityFactor > 0) {
                dto.setComputedPriority((float) (50.0 * adjustedPriorityFactor));
            } else {
                dto.setComputedPriority(1);
            }
        } else {
            dto.setComputedPriority(1);
        }
    }
}
```

Simulations - legacy algorithm


running_jobs3
User(id: 0, defaultPriority: 700.0,
User(id: 1, defaultPriority: 1000.0,
User(id: 2, defaultPriority: 500.0,
User(id: 3, defaultPriority: 1100.0,
User(id: 4, defaultPriority: 900.0,
User(id: 5, defaultPriority: 800.0,
User ID
1
3
4
User CPU Core usage
0
1000
2000
3000
4000
5000
0
2k
4k
6k
8k
10k
Total CPU core usage (Jobs)

Simulations - new algorithm


running_multi-core_3
User(id: 0, defaultPriority: 700.0,
User(id: 1, defaultPriority: 1000.0,
User(id: 2, defaultPriority: 500.0,
User(id: 3, defaultPriority: 1100.0,
User(id: 4, defaultPriority: 900.0,
User(id: 5, defaultPriority: 800.0,
User ID
0
1
2
3
4
5
User CPU Core usage
2000
1500
1000
500
0
0
2k
4k
6k
8k
10k
Total CPU core usage (Jobs)

The optimizers

- ActiveUserReconciler
- InactiveJobHandler
- JobAgentUpdater
- PriorityRapidUpdater
- PriorityReconciliationService
- SitequeueReconciler
- CheckJobStatus
- OldJobRemover
- OverwaitingJobHandler

Recent activity

- Optimizer: ActiveUserReconciler
- Ensure active users are marked as active in the PRIORITY table
  - Or set to inactive when no activity last 24h
- Active = CPU time used within 24h

Resource usage as input

- Optimizers: PriorityRapidUpdater and PriorityReconciliationService
- Collect resource usage from sites
    - Used as inputs to calculate computedPriority
- Synchronize resource usage recorded in QUEUEPROC to PRIORITY
    - Trigger recalculation of computedPriority for all users

Synchronize priority

- Optimizer: JobAgentUpdater
- Updates the JobAgent table column:
  - JOBAGENT.priority = PRIORITY.computedPriority

Removing waiting or old jobs

- Optimizers: OldJobRemover and OverwaitingJobHandler
- Jobs in a WAITING state for longer than 7 days will be moved to ERROR_EW state
    - Possible to set shorter duration through flag in JDL
- Jobs in a final state for 5 days will be removed

Alive or dead?

- Optimizer: InactiveJobHandler
- Heartbeat sent from site to central services to show liveness of job
- If no heartbeat in 1h, move to ZOMBIE state
- After another hour without heartbeat, move to EXPIRED state
  - Unless it finishes within the hour

Verify master and subjob states

- Optimizer: CheckJobStatus
- Masterjob in final state:
    - Ensure all subjobs final
        - Else set master to split
- Masterjob in running state
    - If all subjobs are in final state
        - Set master to done

Cost and jobs per site

- Optimizer: SitequeueReconciler
- Count number of jobs in each state per siteId
- Aggregate total cost for jobs per siteId
-

siteId	cost	SAVING	DONE	KILLED	ERROR_V	EXPIRED	RUNNING	WAITING	ERROR_E
51	6266370000	3	13455	2327	0	0	100	0	0
132	5370019800	0	25005	6304	0	0	322	6	0
203	694810970000	581	186623	52348	0	0	3145	0	0

Thank you - Questions or comments?

- Email: jorn-are.klubben.flaten@cern.ch
- Mattermost: @jflaten