Karlsruhe Institute of Technology

GridKa

Review of HTC scheduling strategies for HEP at GridKa Tier 1

Max Fischer, Matthias Schnepf, Andreas Petzold
HEPiX Autumn 2022 Workshop

High-Level Scheduling Goals and Background

- Pledges: fair share
  - Right now and over a long time
  - Adjust as capacity changes
- Fairness: “MC/SC” [1]
  - No bias for specific requests
  - Problem: Resource Fragmentation
- Allocation: efficiency
  - Use as many resources as possible
  - Mix jobs with different requirements

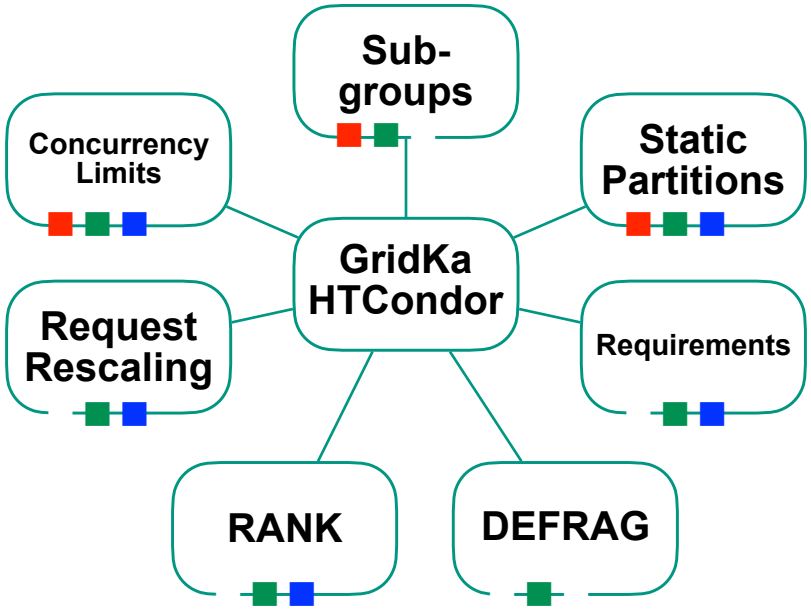
- Multi-VO Site: LHC VOs, Belle, others
  - Diverse usage requirements
  - Need generally usable strategies
- Significant size at ~800k HS06
  - Aim for performance and simplicity
  - Reject solutions for small scale

- We might be doing some things before they become cool…
- …or be amongst the first to realise they do not work.

[1] “MC/SC”: Multicore vs Singlecore jobs

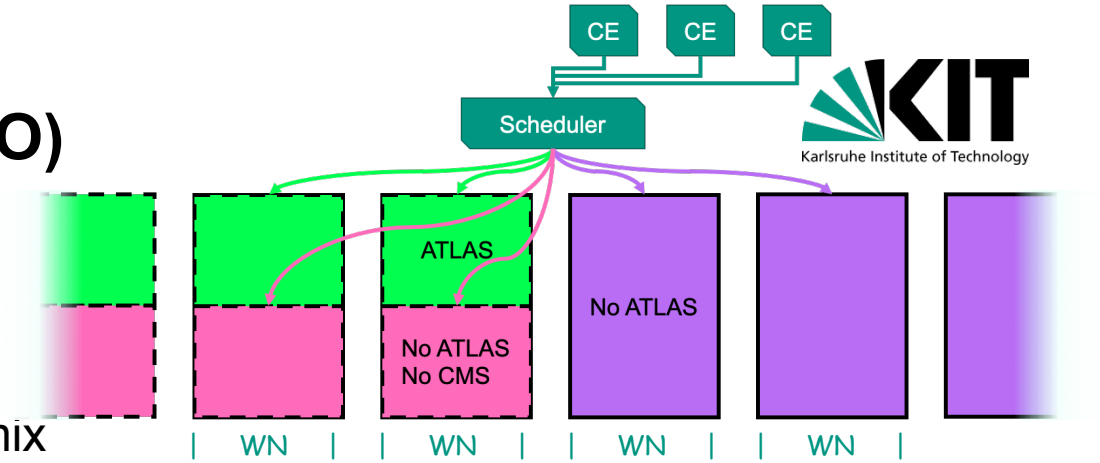
GridKa Scheduling Strategy 2021

- Pledges
- Fairness
- Allocation



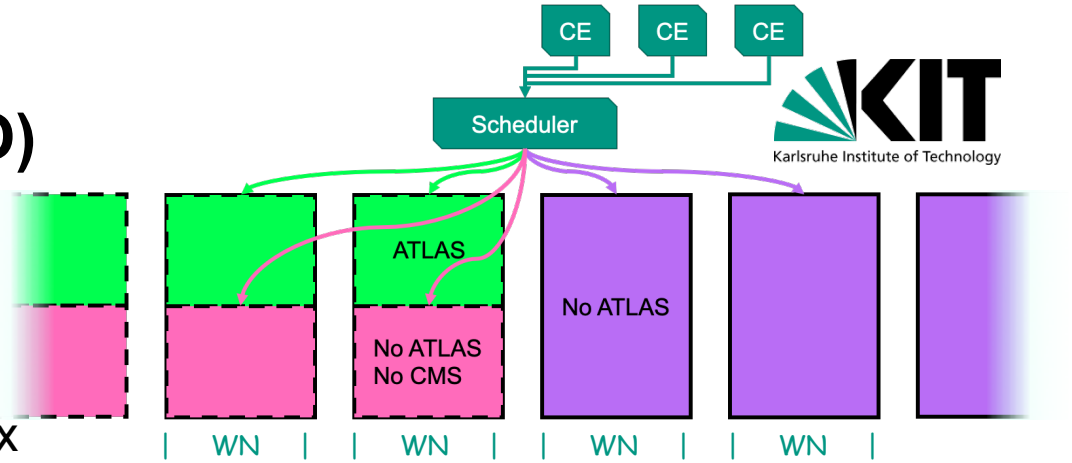
Static partitioning (by VO)

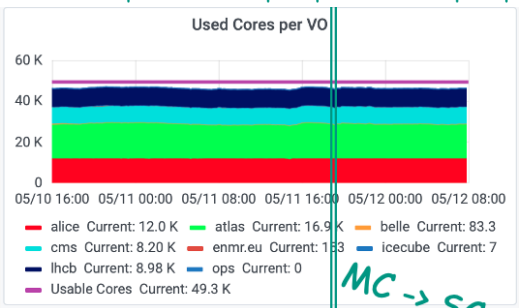
- WNs statically assigned VOs
  - Isolate dynamic demand from stable demand
  - Assign all VOs for good job mix

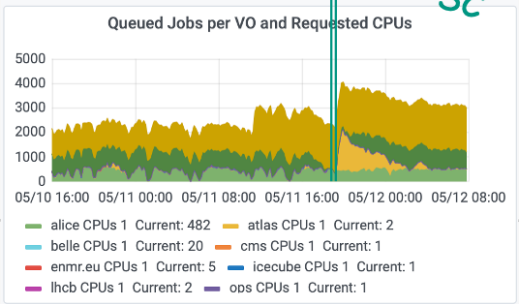


Static partitioning (by VO)

- WNs statically assigned VOs
  - Isolate dynamic demand from stable demand
  - Assign all VOs for good job mix
- The good: Works as designed
  - High fidelity scheduling of jobs from dynamic demand

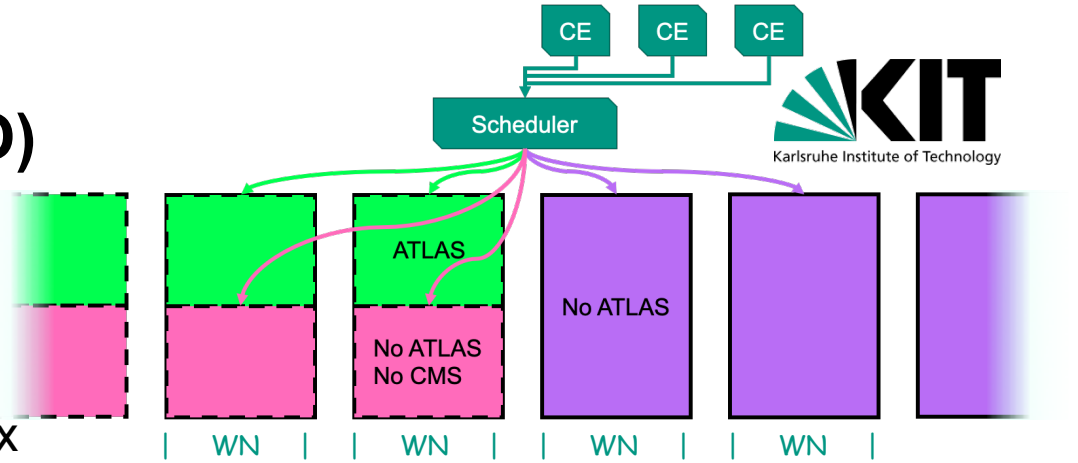


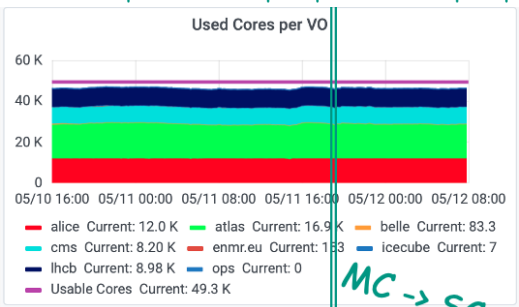


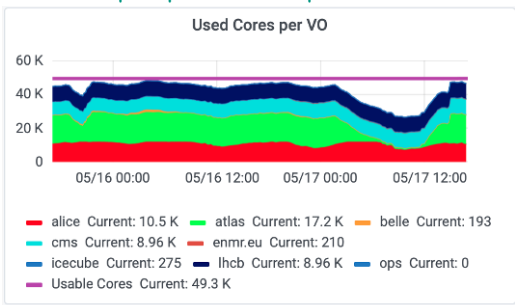


Static partitioning (by VO)

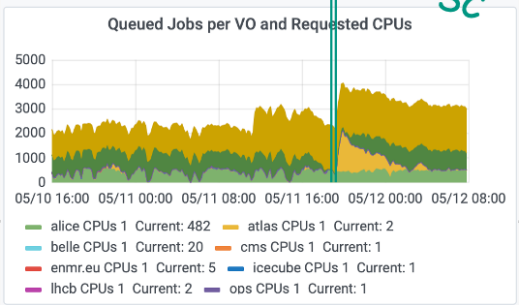
- WNs statically assigned VOs
  - Isolate dynamic demand from stable demand
  - Assign all VOs for good job mix
- The good: Works as designed
  - High fidelity scheduling of jobs from dynamic demand
- The bad: Works as designed
  - Idle resources if demand drops
  - Does not scale with cluster size

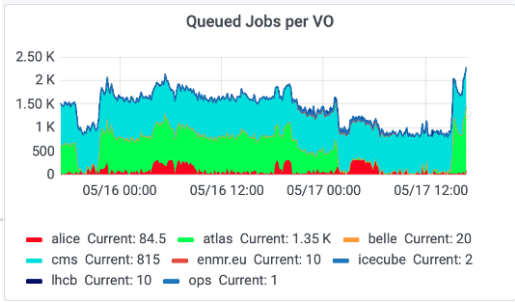






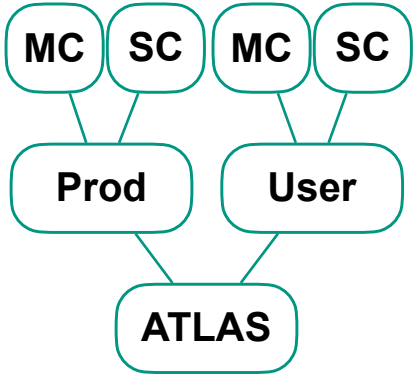
MC -> SC





Subgroups

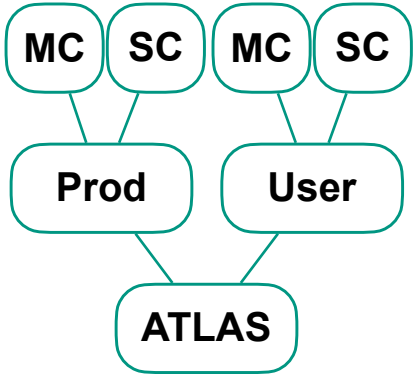
- Encode requests as HTC groups: VO.Kind.Size
  - Example: CMS.Prod.MC
  - Goal: Prefer MC via GroupSortExpr



[1] Futile battle: If it did, then preferring MC would not work…
[2] [HTCondor-users] Subgroups and GROUP_SORT_EXPR

Subgroups

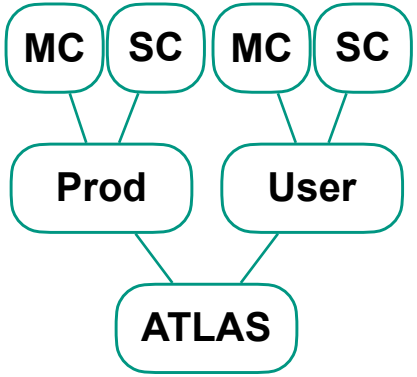
- Encode requests as HTC groups: VO.Kind.Size
  - Example: CMS.Prod.MC
  - Goal: Prefer MC via GroupSortExpr
- Fundamentally broken approach!
  - Sort is not subgroup aware [1,2]
  - Almost always prefers group with largest quota and substructure



[1] Futile battle: If it did, then preferring MC would not work…
[2] [HTCondor-users] Subgroups and GROUP_SORT_EXPR

Subgroups

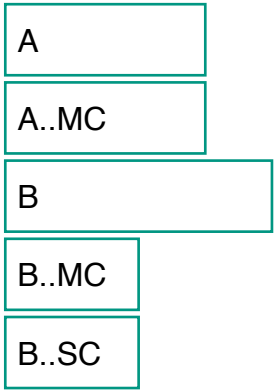
- Encode requests as HTC groups: VO.Kind.Size
  - Example: CMS.Prod.MC
  - Goal: Prefer MC via GroupSortExpr
- Fundamentally broken approach!
  - Sort is not subgroup aware [1,2]
  - Almost always prefers group with largest quota and substructure



7% Quota

14% Quota

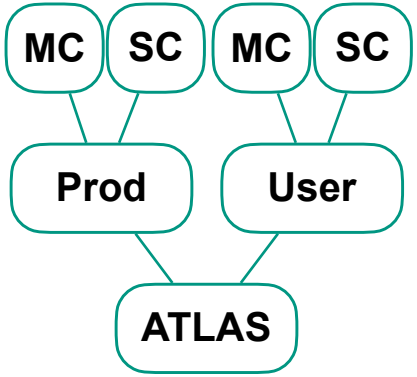
28% Quota



[1] Futile battle: If it did, then preferring MC would not work…
[2] [HTCondor-users] Subgroups and GROUP_SORT_EXPR

Subgroups

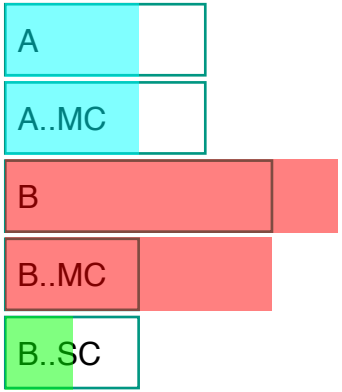
- Encode requests as HTC groups: VO.Kind.Size
  - Example: CMS.Prod.MC
  - Goal: Prefer MC via GroupSortExpr
- Fundamentally broken approach!
  - Sort is not subgroup aware [1,2]
  - Almost always prefers group with largest quota and substructure



7% Quota

14% Quota

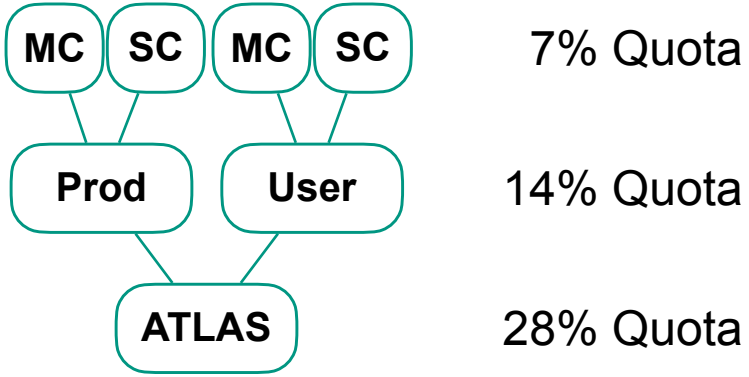
28% Quota

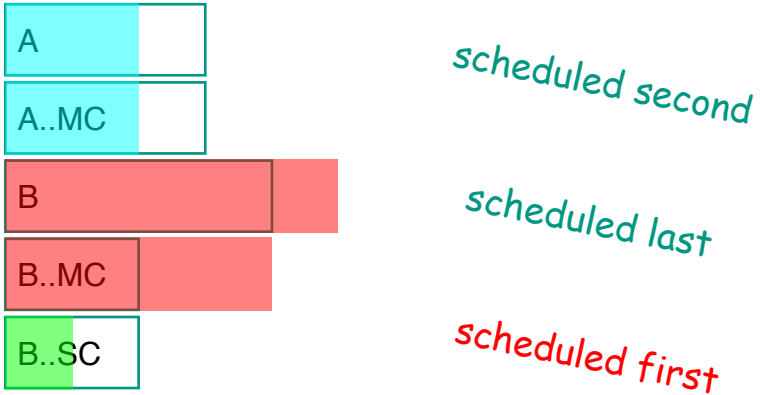


[1] Futile battle: If it did, then preferring MC would not work…
[2] [HTCondor-users] Subgroups and GROUP_SORT_EXPR

Subgroups

- Encode requests as HTC groups: VO.Kind.Size
  - Example: CMS.Prod.MC
  - Goal: Prefer MC via GroupSortExpr
- Fundamentally broken approach!
  - Sort is not subgroup aware [1,2]
  - Almost always prefers group with largest quota and substructure

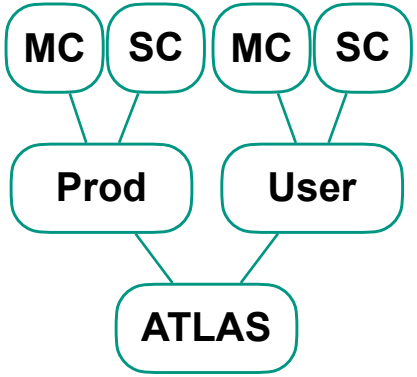




[1] Futile battle: If it did, then preferring MC would not work…
[2] [HTCondor-users] Subgroups and GROUP_SORT_EXPR

Subgroups

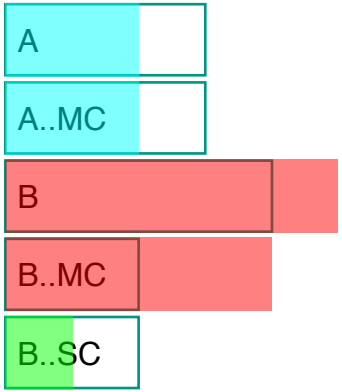
- Encode requests as HTC groups: VO.Kind.Size
  - Example: CMS.Prod.MC
  - Goal: Prefer MC via GroupSortExpr
- Fundamentally broken approach!
  - Sort is not subgroup aware [1,2]
  - Almost always prefers group with largest quota and substructure
- Could not provide fairness nor pledge for groups with uniform job requests



7% Quota

14% Quota

28% Quota



scheduled second

scheduled last

scheduled first

[1] Futile battle: If it did, then preferring MC would not work…
[2] [HTCondor-users] Subgroups and GROUP_SORT_EXPR

Concurrency Limits

- Pool-wide limit on resource usage
  - Relies on job tags applied by CEs
  - Limit for MC jobs, SC jobs, ALICE jobs, ATLAS jobs, CMS Jobs …
  - Intended as an emergency switch

```
[root@htcondor-ce-1-kit ~]# condor_q -af ConcurrencyLimits | sort -u
alice.default:1, alice.default_sc:1, alice:1, slots.sc:1, jobs.sc:1
alice.default:8, alice.default_mc:8, alice:8, slots.mc:8, jobs.mc:1
atlas.pilot:1, atlas.pilot_sc:1, atlas:1, slots.sc:1, jobs.sc:1
atlas.prod:8, atlas.prod_mc:8, atlas:8, slots.mc:8, jobs.mc:1
auger.default:1, auger.default_sc:1, auger:1, slots.sc:1, jobs.sc:1
belle.default:1, belle.default_sc:1, belle:1, slots.sc:1, jobs.sc:1
cms.pilot:8, cms.pilot_mc:8, cms:8, slots.mc:8, jobs.mc:1
enmr_eu.default:1, enmr_eu.default_sc:1, enmr_eu:1, slots.sc:1, jobs.sc:
lhcb.default:1, lhcb.default_sc:1, lhcb:1, slots.sc:1, jobs.sc:1
```

[1] See HTCondor Week 2018

Concurrency Limits

- Pool-wide limit on resource usage
  - Relies on job tags applied by CEs
  - Limit for MC jobs, SC jobs, ALICE jobs, ATLAS jobs, CMS Jobs …
  - Intended as an emergency switch
- Works well as safety mechanism and emergency switch
  - But hid the subgroup problem… :/

```
[root@htcondor-ce-1-kit ~]# condor_q -af ConcurrencyLimits | sort -u
alice.default:1, alice.default_sc:1, alice:1, slots.sc:1, jobs.sc:1
alice.default:8, alice.default_mc:8, alice:8, slots.mc:8, jobs.mc:1
atlas.pilot:1, atlas.pilot_sc:1, atlas:1, slots.sc:1, jobs.sc:1
atlas.prod:8, atlas.prod_mc:8, atlas:8, slots.mc:8, jobs.mc:1
auger.default:1, auger.default_sc:1, auger:1, slots.sc:1, jobs.sc:1
belle.default:1, belle.default_sc:1, belle:1, slots.sc:1, jobs.sc:1
cms.pilot:8, cms.pilot_mc:8, cms:8, slots.mc:8, jobs.mc:1
enmr_eu.default:1, enmr_eu.default_sc:1, enmr_eu:1, slots.sc:1, jobs.sc:
lhcb.default:1, lhcb.default_sc:1, lhcb:1, slots.sc:1, jobs.sc:1
```

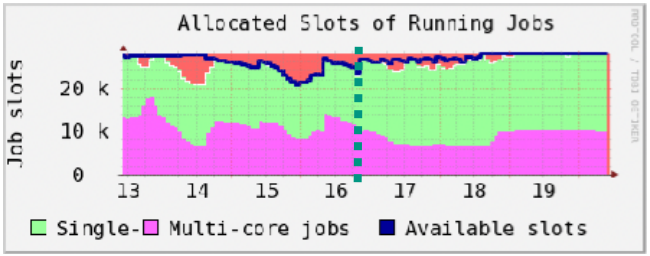
[1] See HTCondor Week 2018

Concurrency Limits

- Pool-wide limit on resource usage
  - Relies on job tags applied by CEs
  - Limit for MC jobs, SC jobs, ALICE jobs, ATLAS jobs, CMS Jobs …
  - Intended as an emergency switch
- Works well as safety mechanism and emergency switch
  - But hid the subgroup problem… :/
- Infeasible to actively balance MC/SC
  - We tried automations [1] but found no good cost function

```
[root@htcondor-ce-1-kit ~]# condor_q -af ConcurrencyLimits | sort -u
alice.default:1, alice.default_sc:1, alice:1, slots.sc:1, jobs.sc:1
alice.default:8, alice.default_mc:8, alice:8, slots.mc:8, jobs.mc:1
atlas.pilot:1, atlas.pilot_sc:1, atlas:1, slots.sc:1, jobs.sc:1
atlas.prod:8, atlas.prod_mc:8, atlas:8, slots.mc:8, jobs.mc:1
auger.default:1, auger.default_sc:1, auger:1, slots.sc:1, jobs.sc:1
belle.default:1, belle.default_sc:1, belle:1, slots.sc:1, jobs.sc:1
cms.pilot:8, cms.pilot_mc:8, cms:8, slots.mc:8, jobs.mc:1
enmr_eu.default:1, enmr_eu.default_sc:1, enmr_eu:1, slots.sc:1, jobs.sc:
lhcb.default:1, lhcb.default_sc:1, lhcb:1, slots.sc:1, jobs.sc:1
```

active limit balancing (2018)



[1] See HTCondor Week 2018

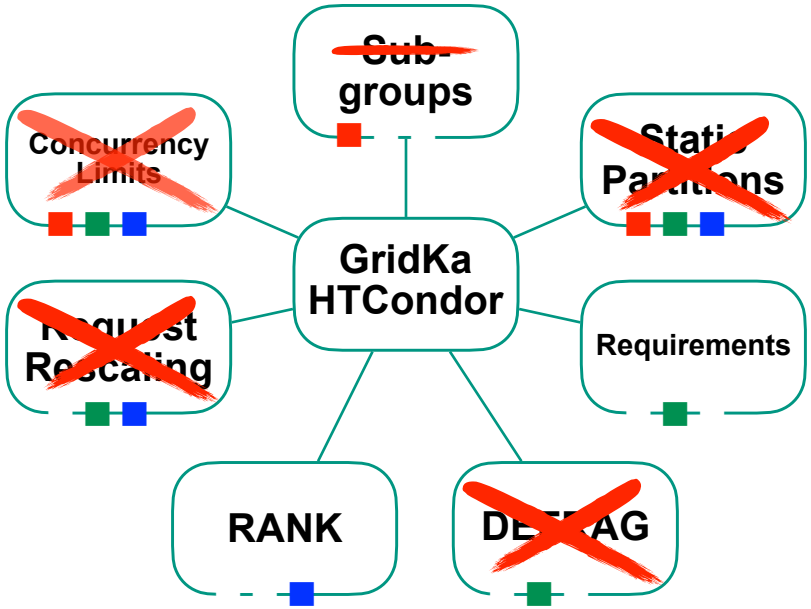
DEFRAG and Request Scaling

- Normalise jobs and thus dynamic slots
  - Idea: Leftovers after matching are regular (e.g. N x 512MB RAM)
  - Actually: No advantage at all 🤷
- DEFRAG to recover MC slots
  - Whole Node as Cpus >= 8
  - Works, but could not tune recovery speed vs draining loss well
  - Actually: Negligible benefit 🤷

Disclaimer:
This may due to
our scale

GridKa Scheduling Strategy 2022

- Pledges
- Fairness
- Allocation



Scheduling Strategy in a Nutshell

- One Group per pledge (aka per VO)
  - Relative pledge as QuotaDynamic
- Requirements of fairness
  - Matching remainder must support standard grid job
  - Fragmentation is not allowed
- Rank by possible allocation
  - Prefer more future matching possibilities

exactly what it says on the tin

might be negligible, still investigating

No-Fragmentation Requirement

- Idea: There must be an MC slot available after each job finishes
  - => Only START job if 8 Cores free

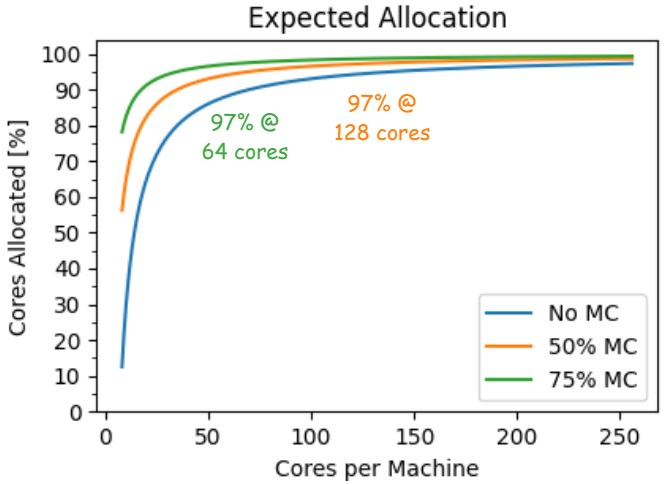
```
# HTCondor Startd Config
START = $(START) && (My.Cpus >= 8 || PartitionableSlot =!= True)
```

No-Fragmentation Requirement

- Idea: There must be an MC slot available after each job finishes
  - => Only START job if 8 Cores free

```
# HTCondor Startd Config
START = $(START) && (My.Cpus >= 8 || PartitionableSlot =!= True)
```

- Cost: Some resources will remain free when there are no MC jobs
  - This is what we already accepted for DEFRAG…
  - Worst case depends on machine!

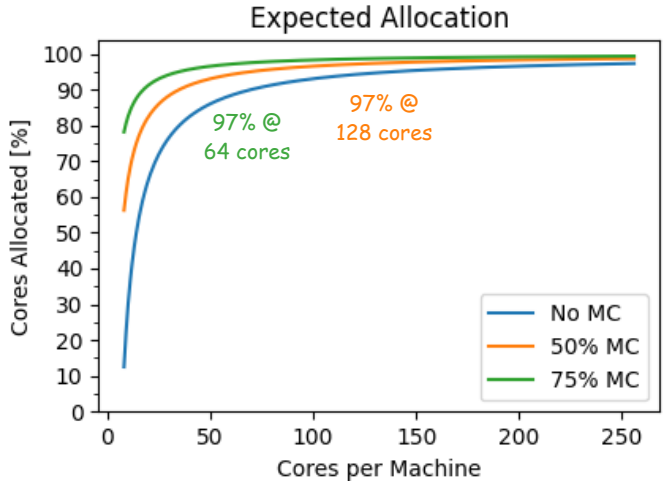


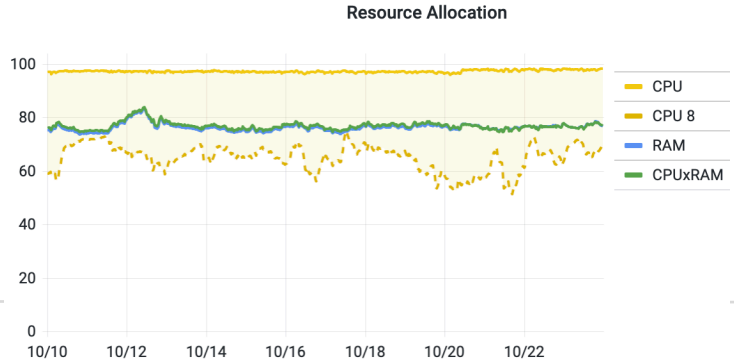
No-Fragmentation Requirement

- Idea: There must be an MC slot available after each job finishes
  - => Only START job if 8 Cores free

```
# HTCondor Startd Config
START = $(START) && (My.Cpus >= 8 || PartitionableSlot =!= True)
```

- Cost: Some resources will remain free when there are no MC jobs
  - This is what we already accepted for DEFRAG…
  - Worst case depends on machine!
- Replaced all our MC/SC settings
  - Fits our 192-256 Cores/WN well
  - Allows high MC/SC variability





Conclusions – HTCondor Strategies at GridKa

- Dismissed common MC/SC strategies
  - Subgroups: Unstable pledges
  - DEFRAG: Not responsive enough
- Deprecated custom approaches
  - Partitions: Not flexible enough
  - Limits: Separate from actual goals
- Satisfied with simple, flat approach
  - Groups for Pledge
  - Requirements/START for MC/SC
- Investigating benefit of RANK'ing
  - More impact due to less restrictions
  - Currently not sure if really needed