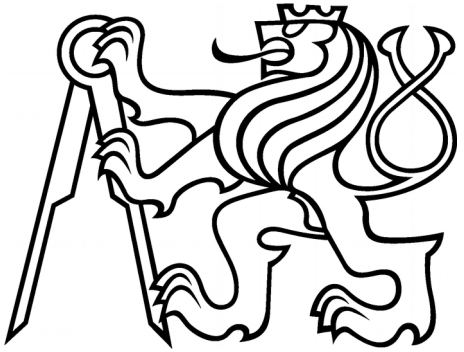


Optimizing resource usage with HTCondor

Alexandr Mikula, Petr Vokáč

NEC2019

1st October 2019

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

HTCondor

- Job management system
  - different roles
    - local batch system for scheduling compute-intensive jobs on cluster
    - grid gateway (HTCondor-CE) supporting many local batch system
- High-Throughput Computing (HTC)
  - run as much work as possible on all available resources
  - large amount of computational power over longer period of time
    - huge number of independent jobs or with DAG dependencies
    - vs. HPC with tremendous power of single massively parallel jobs
  - maximize machine utilization
    - don’t leave resources idle
    - optimize time-to-finish not time-to-run
- Operator/user requirements and expectation more complex
  - fair resource distribution, reasonable time to start job, job execution priorities, resource allocations without artificial constraints, …

HTCondor configuration

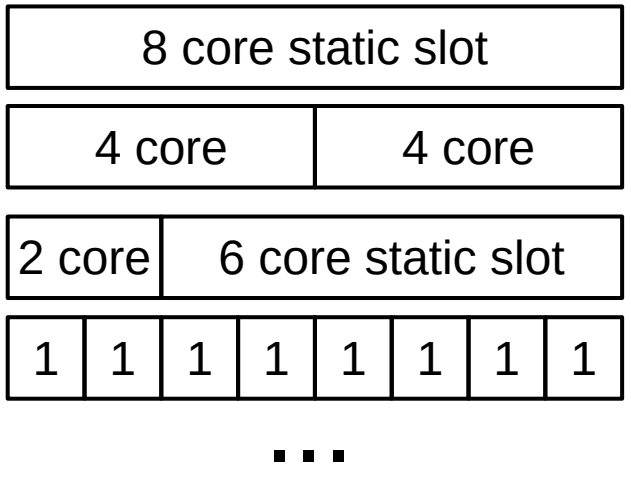
- HTCondor provides extremely flexible configuration
  - provides tools to enforce local job scheduling policies
  - default / simple configuration usually lack some features
    - job executed to maximize use of available resources
    - missing concept of “queues” known from other batch systems
  - flexible configuration has downsides
    - no single standard way to achieve requested behavior
      - custom attributes (also in submission files, e.g. MaxRuntime, JobFlavor)
      - expression language powerful, but can become complex – hard to read
- Clusters shared between groups with different requirements
  - jobs with non-standard requirements can wait long (infinite) time
    - number of CPUs / GPUs, memory size, disk space, connectivity, …
    - small jobs scheduled to fill idle resources not big enough for big one
      - cluster providing resources for grid jobs have plenty of small jobs
      - special jobs waits till number of small jobs finish at the same time

Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)

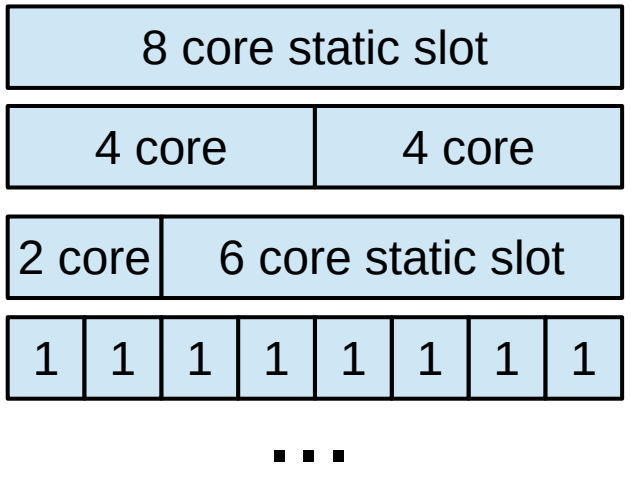
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn’t take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation



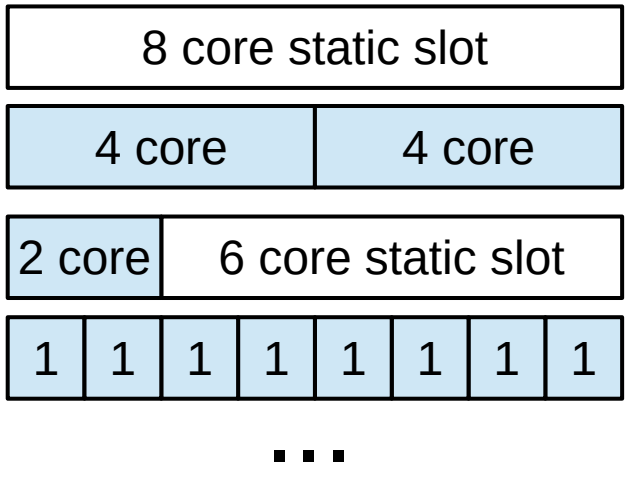
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn’t take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation



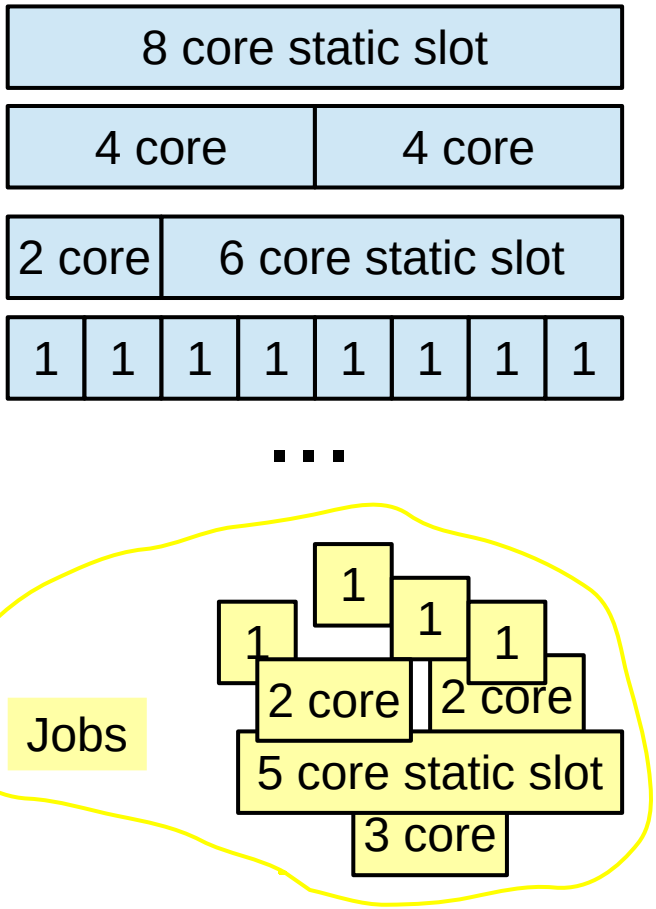
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn’t take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation



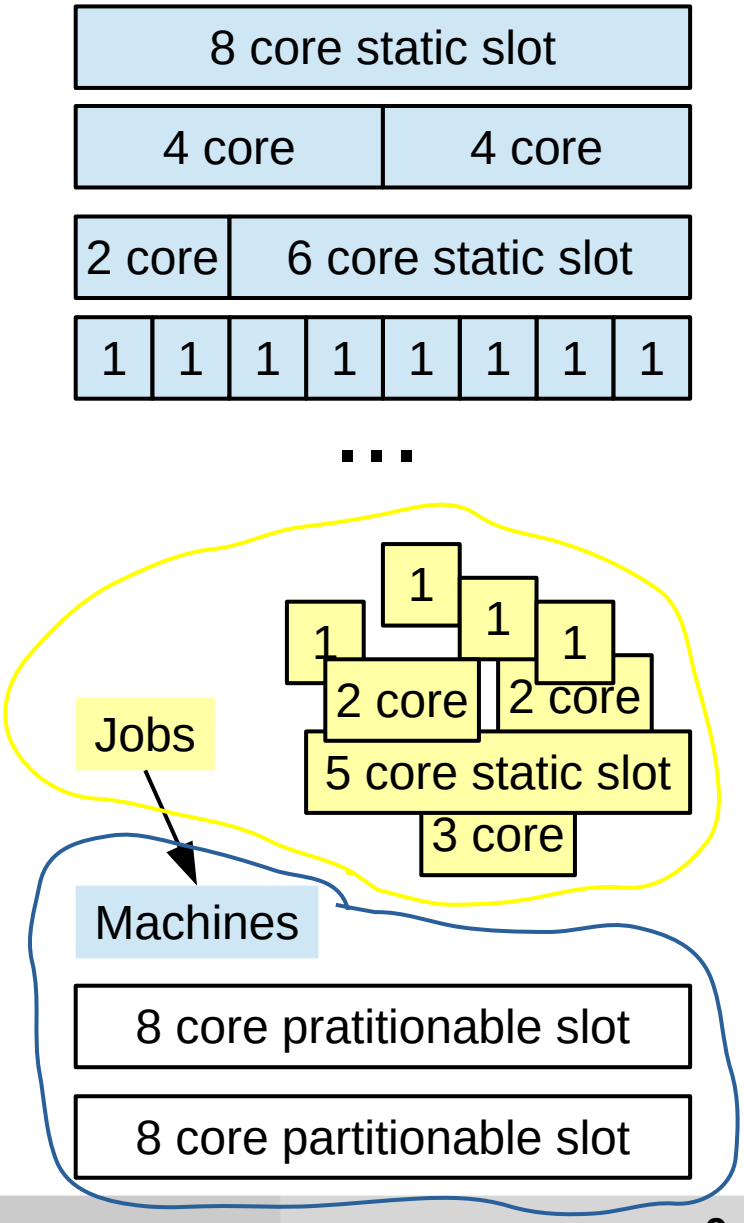
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



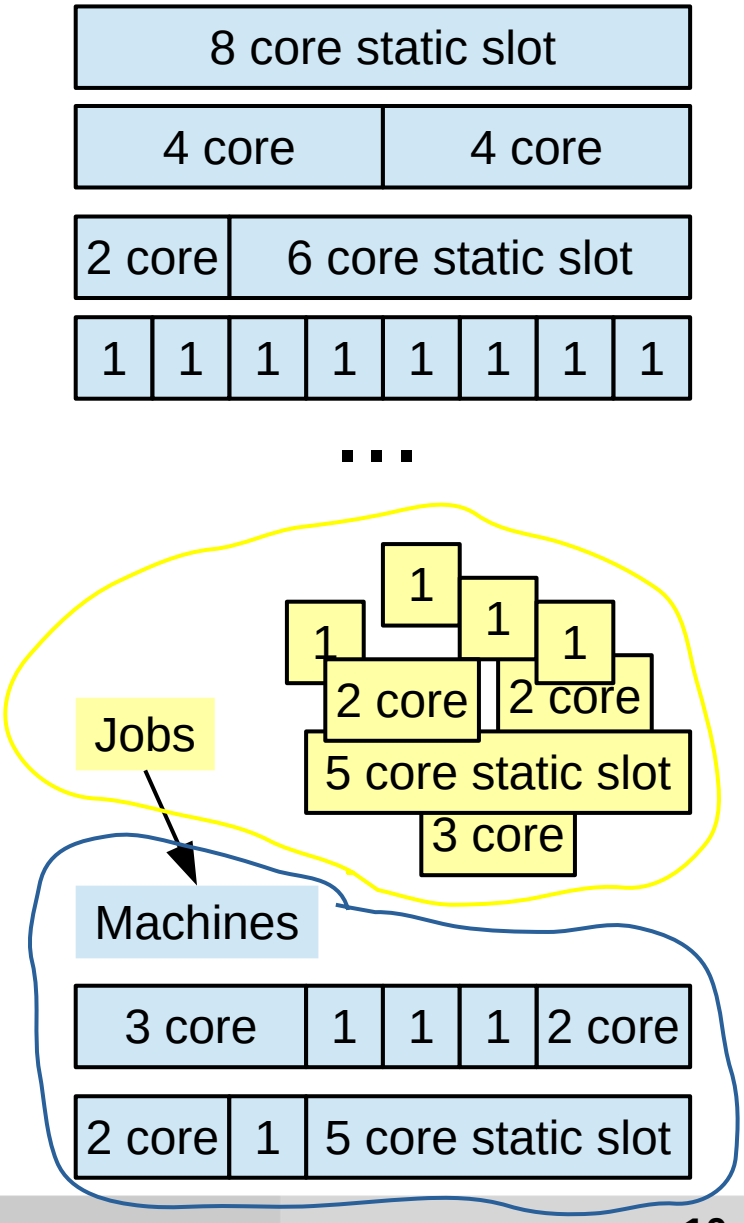
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



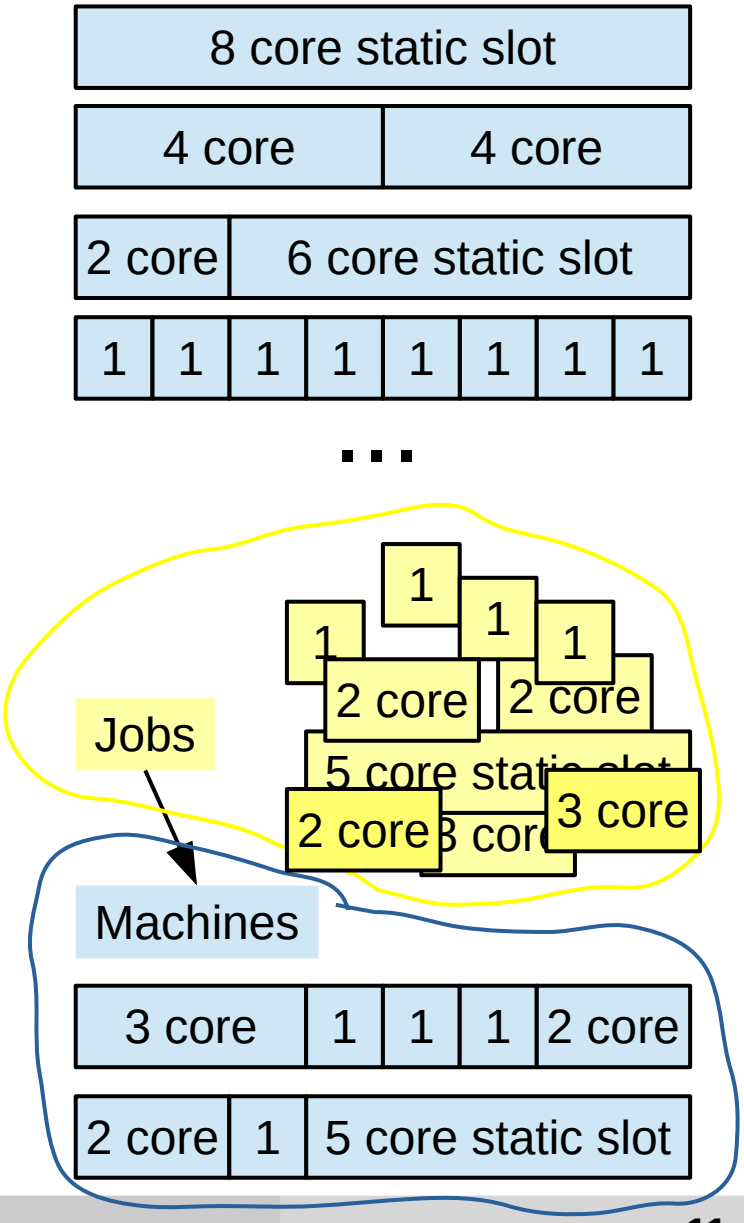
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



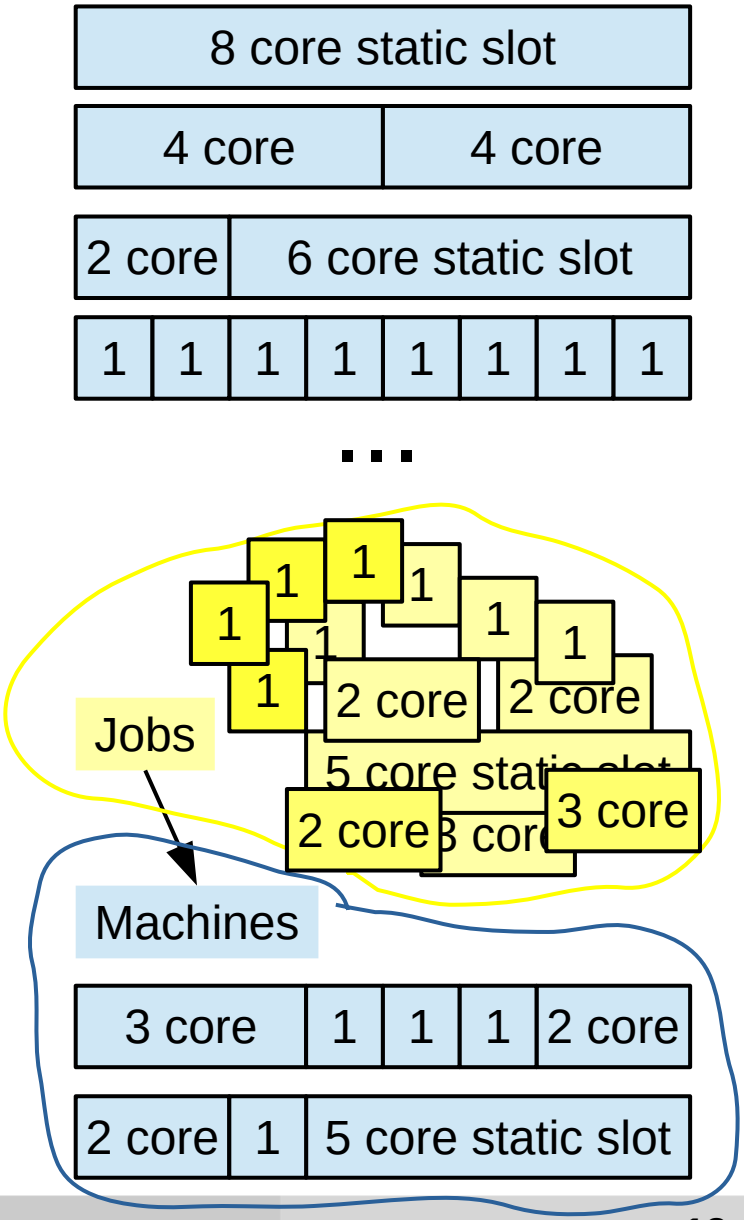
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



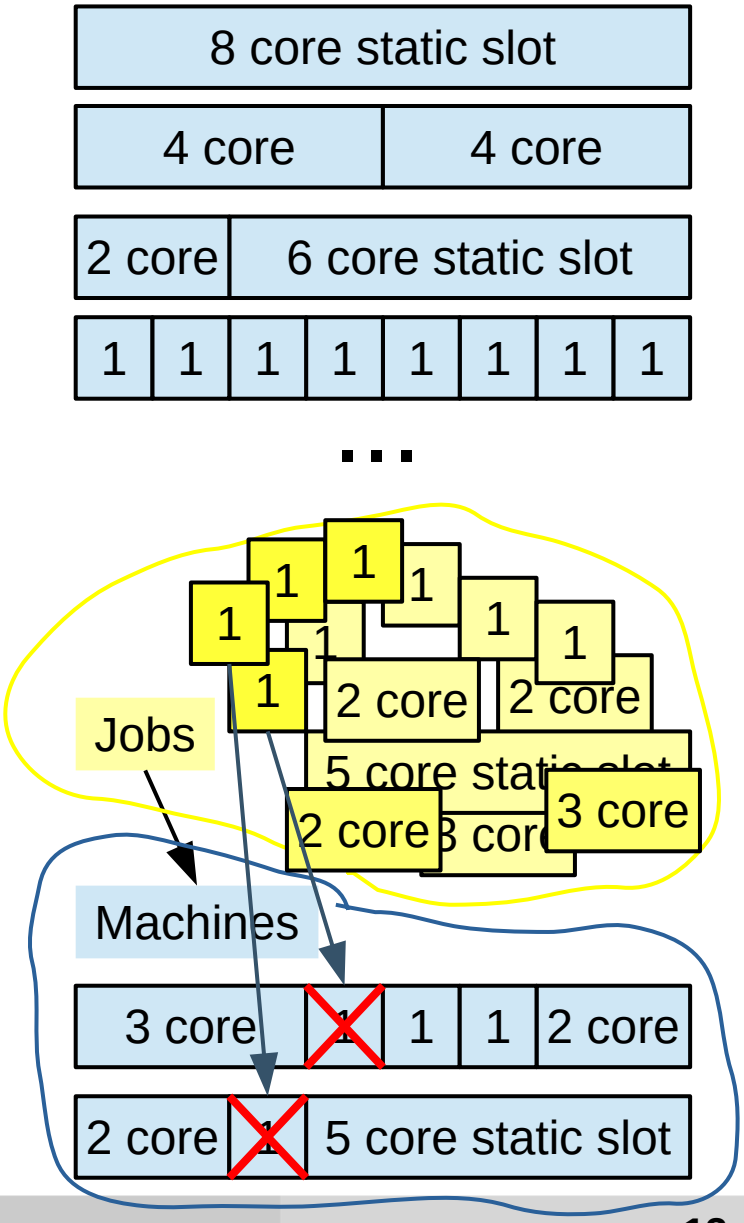
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



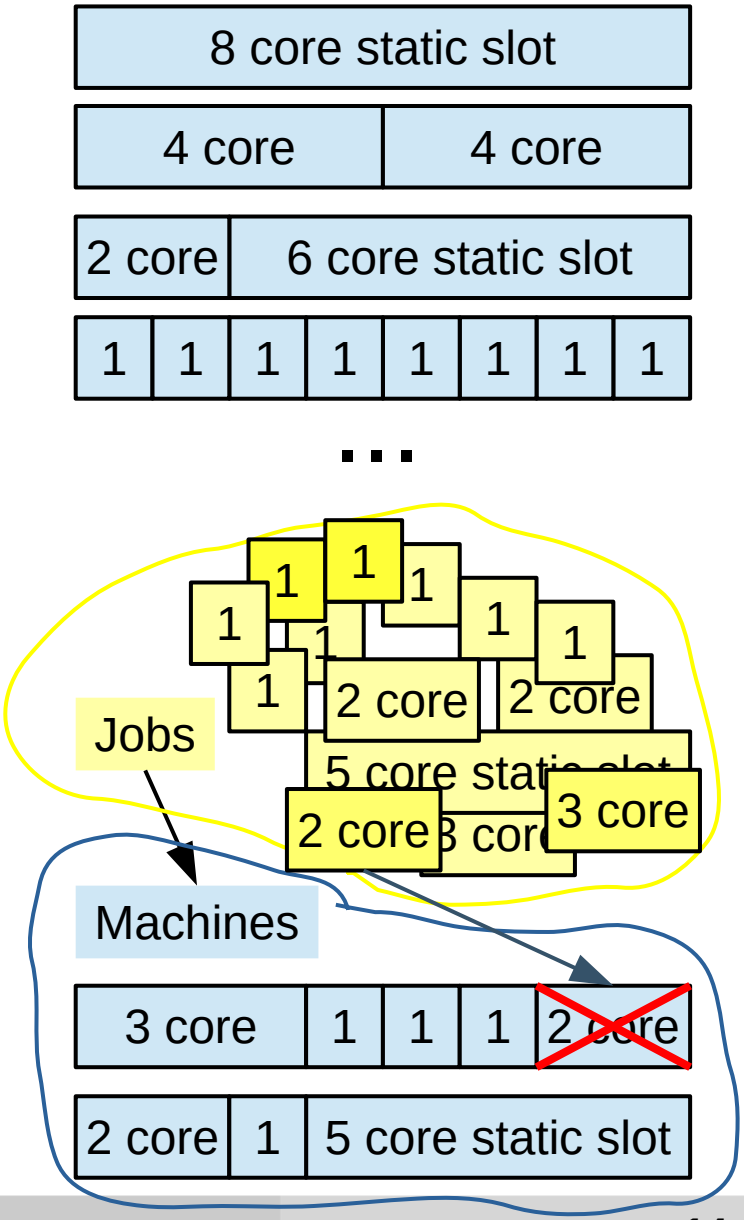
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



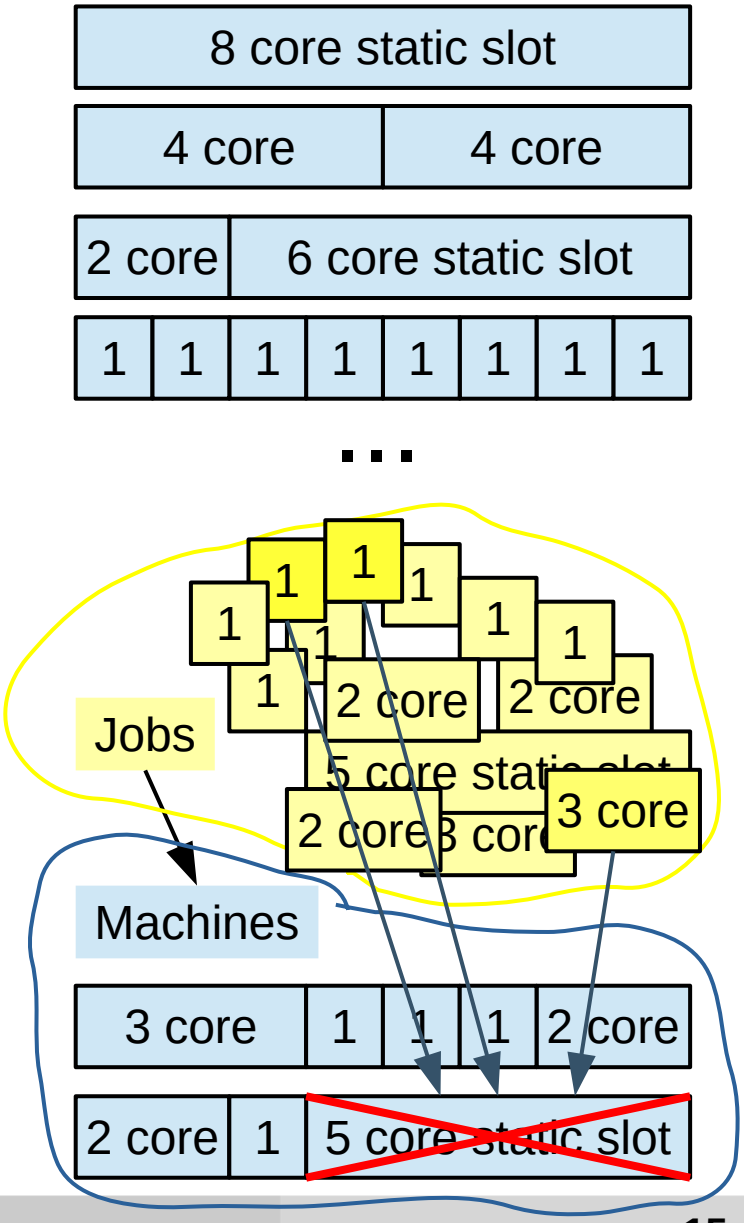
Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



Cluster resource fragmentation

- Most simple case – multicore jobs (request_cpus > 1)
- Static jobs slot configuration
  - dedicate portion of resources to multicore
  - fixed ncore configuration
  - doesn't take into account queued jobs
    - can lead to idle resources
    - fairshare by static allocation
- Partitionable batch slots
  - dynamically allocated resources
  - resource fragmentation
  - no / minimized idle resources
  - small jobs "preferred"
    - big jobs delayed (negotiation cycle)
    - configured fairshare not followed



Cluster defragmentation

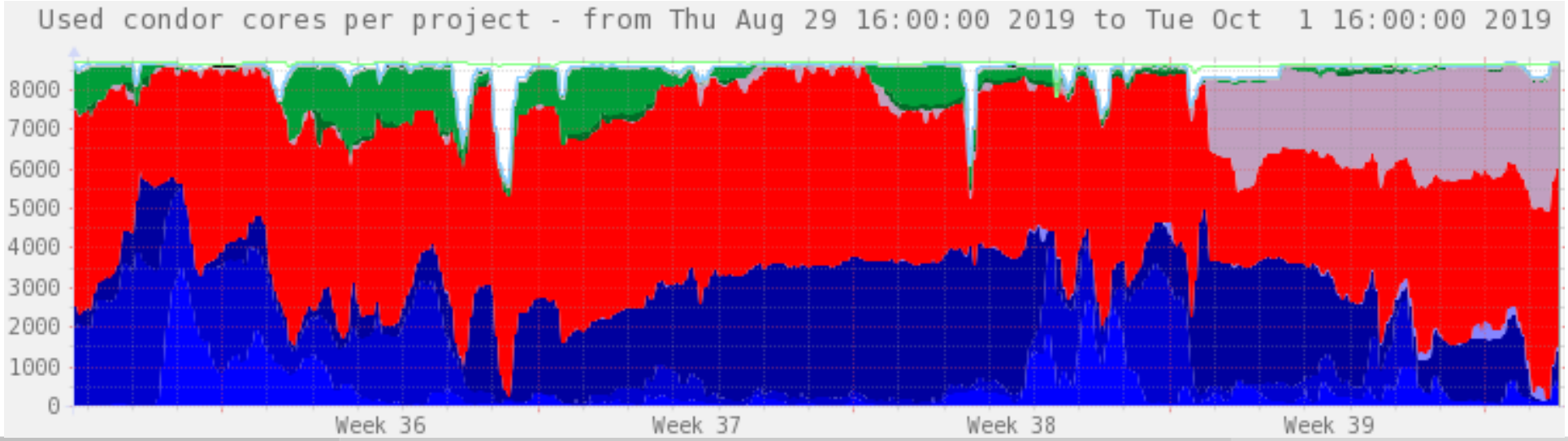
- Peemptible jobs
  - jobs with lower priority killed by batch system
    - wasted CPU unless job continuously store results (ES, multi-payload)
    - rescheduled to run later once jobs with higher priority completes
  - more jobs can be preempted from machine at same time
    - higher chance (not guaranteed) for bigger slot for higher priority jobs
  - usually not applicable for all jobs

Cluster defragmentation

- Peemptible jobs
  - jobs with lower priority killed by batch system
    - wasted CPU unless job continuously store results (ES, multi-payload)
    - rescheduled to run later once jobs with higher priority completes
  - more jobs can be preempted from machine at same time
    - higher chance (not guaranteed) for bigger slot for higher priority jobs
  - usually not applicable for all jobs
- Defrag daemon
  - HTCondor built-in solution for cluster resource defragmentation
  - select machine passing DEFRAG_REQUIREMENTS (by rank expr.)
    - change machine state to drain (condor_status) – no new jobs
    - wait till DEFRAG_WHOLE_MACHINE_EXPR pass
    - limit number of machines in draining state, max draining per hour, …
  - idle resources waiting for defragmentation
    - no connection to queued job requirements

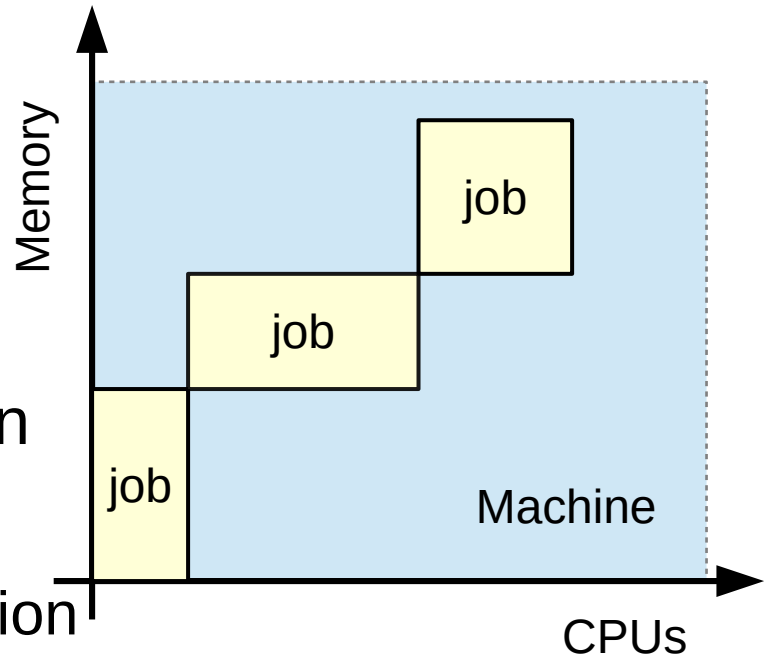
HTCondor defrag

- Doesn’t guarantee drained big slot matched with big job
  - additional configuration necessary not to match with small jobs
    - GROUP_SORT_EXPR – prefer matching multicore jobs first
      - HEP-puppet configuration
      - works quite reliable, but only for one group having multicore jobs
      - one starving muticore fairshare → prevents others to start mcore jobs
    - prevent small jobs to match big slot for few negotiation cycles
- Slow defragmenation vs. number of idle / draining machines
- Not optimal for arbitrary user resource requirements



Optimize resource utilization

- CPU not the only shared resource
  - GPU, memory
  - disk size, disk I/O, network
  - shared storage
- Draining – multidimensional optimization
  - resource requirements of queued jobs
  - monitor condor events for quick adaptation
  - node selection for fastest draining (MaxRuntime, preemption, …)
  - don’t limit users to semi-static resource partition (8 cores / 16GB RAM)
  - more intelligent job placement – group short jobs together
- Concurrency limits
  - could be used to protect (global) shared resource
    from overloading by running jobs (I/O, storage, network)
  - defaults values can be injected by JOB_TRANSFORM



Conclusion

- HTCondor provides means to define flexible job scheduling policies
- Default configuration doesn’t satisfy even simple requirements
- Optimal resource utilization is not an easy task
  - idle resources vs. time-to-start
    - important mainly for local users
  - prevent artificial resource limits
  - dynamic configuration updates necessary
    - HTCondor expression language not optimal for complex policies
- Accounting