## Assumptions

· disk/tape separation at T1 sites, ie. the scheduling system does not have to worry about staging of input files from tape

· There is one global queue for jobs and the scheduler sees all resources globally. This document describes how the scheduling of these jobs for the global resources happens. We might provide “local” access to resources as well, maybe even within the same submission systems. But how that is handled is outside the scope of this document.

## CMS workflow types

- **Central production run by Operations teams**

  All central workflows consist primarily of many processing/production jobs with or without input data and a lesser number of various system jobs (merge, logCollect, cleanup). For instance (list isn’t necessarily complete, but covers the majority of workflows):

  · **ReReco**: re-reconstruction of data
    - input: RAW dataset
  · **ReDigi**:
    - input: GEN-SIM dataset
  · **MonteCarlo**:
    - input: none
  · **MonteCarloFromGEN**:
    - input: GEN dataset

- **Analysis run by individual users**

  All analysis workflows (at least at the moment) consist of only jobs of a single type, there are no system jobs. Even though what the jobs do can be very different, we just all call them analysis jobs.

  · **”Standard” Analysis**:
    - input: AOD or RECO dataset
  · **Other**:
    - private MonteCarlo production
    - arbitrary script execution

## CMS job types

· **Central job types:**

- Processing jobs: rereco of data or redigi of MC, needs access to input samples
- Skim jobs: I/O intensive jobs that are like processing jobs but with very little actual processing. Need to have site specific threshold not to overload the MSS.
- Production jobs: MC production, some need input samples ('/GEN' input), some don't (Pythia jobs)

  Note: For scheduling purposes the distinction between processing and production jobs is meaningless, production jobs are like processing jobs with or without input.

· **User jobs types:**
- Analysis jobs: analysis user jobs

· **System job types:**
- merge job: reads in small outputs of several processing/production jobs and stores the merged output in official namespace. Needs to have site specific threshold not to overload the MSS.
- logCollect job: tar of logArchives containing all the log files from the execution of a processing or production job and copies the tar ball to CERN via srm. Needs to have site specific threshold not to overload the MSS.
- cleanup job: deletes temporarily files. Needs to have site specific threshold not to overload the MSS.

**General scheduling policies**

· No jobs are preempted by other jobs with higher priority. Nothing needs to ever be done urgently enough to have to deal with the additional complexity of preemption.

· The system does not know how job slots are related to each other. Therefore we cannot make any scheduling decision based on the physical connection between job slots (same node, same rack, etc).

· Quota and share calculations should correctly handle a mix of single-core and multi-core jobs, ie. at the very least account for the higher number of cores used by a multi-core job slot. A strongly desired feature would be to not just use core counts for internal accounting used by quota and share calculations, but some measurable performance metric (like HS06). This could then also address the use case of systems with cores of different performance levels (Intel Phi for instance).

· We currently only foresee adapting the fraction of central to analysis jobs at a site, all other access to resources will either be handled via absolute prioritization or a strict fair share algorithm that treats all users the same. That means users with the same priority will get the same access to resources and the share they get might be temporarily different but balances out within the time integration limits of the fair share algorithm. If a user needs access to a larger share of the resources, the only way to accomplish that is to give the user higher priority, which means that users jobs will always run before any other jobs from lower priority users.

· Jobs should be matched to job slots primarily based on the central/analysis quotas and job/user priorities. If this isn't sufficient to determine what job to run next, one could look at how close the resource requirements of the job match the resource capabilities of the job slot, for this we consider wall time and memory to be the deciding factors in this order.

· Jobs that require input samples will be submitted with a site whitelist that have a replica of the dataset available locally
    - Exception to this rule:
        - § Overflow reschedules jobs to sites without local replica of input sample. Number of jobs that can be rescheduled to a site that does not have the data is a property of the source site, i.e. the site that has the data. (The limitation that is being scheduled is the capability of the storage at the site that serves the data).
        - § For now this will just be matching number of jobs reading remotely from a site against the overflow slot limit of the site. In the future we could envision providing some kind of weight factor related to both IO load of the job and the network connectivity between the job and data sites. Using such a weight factor could allow a better optimization of scheduling overflow jobs on a global scale, especially in an environment where a majority or at least a substantial minority of jobs are overflown. How CMS would calculate such a weight factor isn't clear yet.
        - § If multiple sites host some data, the total overflow limit is the sum of the individual sites overflow limits. This assumes that the xrootd redirection system will balance remote access to the various sites in a sensible way so that no individual sites gets overloaded.
        - § For central workflows we need a mechanism to limit the number of sites the workflow is overflown to. This is needed because of merge efficiency issues if a workflow is spread across too many sites. Possible implementations could be either attaching a list of sites the workflow can be overflown to or providing a maximum threshold for the number of sites it can be overflown to.

· Test scheduling for every workflow:
    - Schedule the first X central or analysis jobs of a workflow immediately (make X configurable for central or analysis jobs or maybe even by workflow)
    - This does not mean the rest of the jobs have to wait for completion of the first X jobs, if they would otherwise be scheduled to run they can run already
    - Objective of this functionality:
        - § Give the user quick feedback on small subset of jobs.
        - § Given 100% failure rate in the first subsample allow automatic killing of the remaining jobs.
        - § Monitor job performance of job, update description of remaining jobs with

that information, feedback information to user if jobs cannot succeed (optionally kill them in that case).

· System job types should be scheduled first. They are generally short and focused on freeing up resources, e.g. disks. It is thus best to guarantee that they are run first whenever an appropriate slot frees up. System jobs can generally not be run at any site, but this is not a scheduling system requirement, we can always configure this ourselves with a site whitelist and not allowing the job to overflow.

· Scheduling at a site:
  - site pledges resources to CMS (converted in number of slots) which are divided into central slots and analysis slots by Ops (centrally decided)
  - site can provide access to additional resources on top of pledges, these are used automatically but the site can tell Ops how these extra resources should be divided into central slots and analysis slots (can be a different mix than for the pledged resources).
    - § site can give priority access for group of users that the site defines (regional users) to the analysis share of the additional resources
  - site can specify additional resources can *only* be accessed by special group of users
  - Ops always tries to use all central slots and all analysis slots at a site (or anything above that for that matter as long as the site accepts more CMS jobs).
    - § If there are operational reasons to not use a site's pledge or additional resources above the pledge, this would have to implemented at the site level, they have full control over how many of our jobs they let run at any given time.
  - **T1 sites**: number of central slots is set to all slots of the site
  - **T2 sites**: number of central slots is typically set to 50% of the pledged slots plus slots of additional resources dedicated to central jobs by site
  - **T3 sites**: number of central slots is set to slots dedicated to central jobs by site
  - **opportunistic sites**: number of central slots is set by Ops
  - **clouds**: number of central slots is set by Ops

· Scheduling order within job types:
  - **System jobs**:
    - § always have priority over any other types of jobs
    - § FIFO
  - **Central jobs**:
    - § single user => no fair share needed
    - § workflow has specified priority
    - § same priority: FIFO
    - § priority of a workflow can be adapted while being in the queue

§ production jobs should first fill T1 resources and then use T2/T3/opportunistic resources
§ Use case: queue is stacked deep, new workflow is submitted which should run immediately, workflow is given a higher priority and jobs of the workflow should be scheduled on the next available resources
§ Use case: workflow has been processed for a while and is almost done (fixed percentage, configurable per workflow ?). We want to boost its priority to get it done quickly finishing out processing tails and making the output available, priority is changed for the jobs and they should be scheduled on the next available resources. We assume here that the system knows the total number of jobs in the workflow and how many are still pending.

- **Analysis jobs**:
  § multiple users => fair share
  § users can be grouped and given a higher priority for all or individual resources
  § possible groups:
    - users of a physics group
    - priority user within a physics group
    - users of a region or specific site
  § Use case: physics group gets higher priority resulting in “expedient” access to 16k slots for their work (for example: discovery of supersymmetric particle)
  § Use case: priority user of higgs group is producing group ntuples and therefore needs larger priority to get more resources than the rest of the group
  § Use case: site x provides more resources than pledged and specifies that these extra resources are only available for analysis

**Summary and remove CMS specifics as much as possible**

- **Parameters of a workflow:**
  - Central or Analysis: this is a quantity of the workflow, the jobs inherit it.
  - ID / Name: we want to have scheduling policies that apply within jobs belonging to the same workflow, therefore the system needs to be able to group jobs by workflow (this might or might not be extractable from other information in the JDL). Also inherited by the jobs.
- **Parameters of a job:**
  - SiteWhiteList: job input is at these sites.
  - AllowOverflow: allow jobs to be scheduled ignoring the site whitelist (up to the limit of the sites overflow slots). Could be globally defined for a whole workflow, but we need the ability to override at job level too.
  - HighIO: job is heavy on the storage system. Does not just mean high bandwidth,

can also cause load on the storage system in other ways (like many file deletions). We need separate site limits for high IO jobs.
- SystemJob: job is a special system job and has priority (over all other jobs not just from this workflow) because it's completion will usually release resources at a site (keeping them pending for too long can cause infrastructure problems).

Jobs that do not declare a site whitelist are not restricted and can run anywhere. They might also have input, but as we can't know what that input is and where it is, any type of scheduling limits would be pure guesses. It's up to the submitter to declare a site whitelist if the jobs have site restrictions and can't just run anywhere.

Jobs with no site whitelist and the high IO flag are invalid in this scheme. We can either ignore the high IO flag and treat them as normal jobs or reject them outright as invalid.

**Parameters that need to be specified by Ops centrally and used for scheduling:**

- number of pledged slots per site
- number of overflow slots per site
- number of high IO slots per site

(it's an implementation detail, but we can define them in http://dashb-ssb-dev.cern.ch/dashboard/request.py/siteview#currentView=Pledges&fullscreen=true&highlight=true and retrieve them by API)

**Multi-core**

Allow scheduling both single-core and multi-core jobs on multi-core nodes.

As far as specifying how many cores a job should use, there are two models

- Job specifies exact core count (default 1)
- Job specifies minimum core count and uses up to N and/or the nodes max cores.

In the mode where we fill up all the available cores in the node we again have two options
- Job is only scheduled on node if it gets the complete node for itself
- Job can be scheduled on node even if jobs are already running

CMS has not really decided on a longterm policy here. We need to gain some experiences with the various models to find the approach that best guarantees a good resource utilization in a mixed single-core and multi-core job environment. We also have to wait and see which of these models works best together with the CMS framework.

In case we require node draining to make space for pending multi-core jobs, we want to use expected job runtime (which CMS provides) to decide which nodes to drain and how to drain them (whether to schedule shorter jobs while waiting for longer jobs to finish).

**How to manage special groups of users (for priority assignments)**

CMS wants to use VOMS here because we already use it to distinguish between central and analysis jobs (with the production role) and every CMS user needs to be registered there anyways to be able to use CMS resources. We don't want another parallel system. If there are problems with the way VOMS is used for group/user priority assignment we have to work that out.

**Open Questions / Potential Problems**

How do we protect the system job attribute against misuse ? At the moment all system jobs are central jobs, so under Ops control. That might not stay this way in the future though (if we add merge support for analysis).

If we make closeness of resource matching a deciding factor in matching of jobs to job slots, we have to look at how that will affect scheduling decisions for the production use case, ie. a large number of same priority workflows/jobs in the system. We have to avoid inadvertently blocking some workflows from getting enough resources. For example, in a scenario where the majority of slots supports 48 hours wall time jobs and we take wall time in consideration for scheduling, a workflow with 4 hour jobs would not get much resources as long as longer running workflows are present in the system. This is only relevant for central workflows, not analysis, because for the latter the fair share algorithm between users will make sure one users jobs cannot block other users jobs forever (it would only apply if the same users keeps different types of jobs with different wall time requirements pending all the time).

**Appendix (kept here to not have to deal with the same comments again)**

There were some comments on taking IO capabilities of sites and IO requirements of jobs into account for scheduling. For now we only have two job categories for IO (heavyIO and default). IO requirements can vary to a large degree and are somewhat unpredictable (especially once you get into analysis jobs), so quantifying what a job needs is difficult. Even if it could be done, there is then on the other side the problem of qualifying how much IO load a site supports and how to match sites IO capabilities to jobs IO needs. At the moment we have production jobs that we know will cause problems and the proposed simple solution protects us for this case. Anything more fine grained has the potential to cause more problems than it solves.

Some comments on retrying AsynchronousStageout failures. Either the ASO is run as a regular

job or it isn’t. In the former case we are dealing with a regular job retry, in the later case any retry would be outside the scope of this document (as it has to be handled in the CMS WM system).