

# Utility Functions

First of all, there's an exciting bug in the currently released versions of cobalt wherein the queued time of a job is reported to utility functions in seconds whereas the wall time requested by a job is reported in minutes. So if you plan to write a function that does any computations with those values, you should make sure you have consistent units.

We have been testing out two utility functions which try to achieve two different goals. Both functions make use of the ratio of queued time to requested wall time. This is a value that increases as jobs wait, and also captures the fact that, for example, waiting an hour before running is more painful for a 20 minute job than for a 6 hour job. We usually refer to this ratio as the "unitless waiting time".

The first function is defined as:

```
def wfp():
    global wall_time
    wall_time = wall_time * 60
    val = (queued_time / wall_time)**3 * size
    return (val, 0.65 * val)
```

This function aims to avoid large job starvation. All jobs get increasingly angry the longer they wait, and bigger jobs moreso.

The second function is defined as:

```
def unicef():
    n = max(math.ceil(math.log(size)/math.log(2)), 6.0)
    z = n - 5
    # avoid dividing by zero
    val = (queued_time / (60*z*max(wall_time, 1.0)))
    return (val, min(0.75*val, 4.5))
```

This function is used on our T&D system and on the development portion of intrepid. It's goal is to provide fast turn around for small jobs. In this function, job size serves to penalize jobs by "stretching out" the wall time requested. This slows the rate at which large jobs accumulate their utility scores. The weird computation of n is to find the log base 2 of the size of the partition which will run the job. On our machines,  $2^{*}6 = 64$  is the smallest available partition. The value of z is thus 1 for the smallest partition, so small jobs don't get their wall time stretched, while any larger job will.