# HPC Workload Scheduling Simulator
# User Guide Version 0.1

Tyler A. Simon
tasimon@lps.umd.edu

February 22, 2017

## Contents

## 1 Overview

This document describes how to use the Workload Scheduling Simulator. The simulator enables a user to import, generate, or modify an existing workload with a specific number of jobs for a system of arbitrary size. Workloads consist of some number of jobs, jobs are considered to be collections of tasks, where the job size is the number of tasks that can be mapped and executed on resources in a one to one mapping, i.e. one task per processor, per unit time. The simulator assumes job properties associated with a typical high performance computing (HPC) cluster or cloud computing environment. Here system size means number of compute nodes, cores, or processors required for each job. All times are a single scheduling cycle which is consistent between systems and jobs and assumes 1 second. Thus a job length of 20, is 20 seconds and will take 20 schedule cycles to run. Waiting time is also represented in seconds. System size refers to the total number of processes that can be run per scheduling cycle, which is a tick of the scheduling clock or a simulation second. Additionally the user may choose to allow for three different failure modes for a give system mean time to failure (MTTF).

### 1.1 What does it do?

This simulator was developed to help better understand the performance trade offs between various scheduling policies typically employed in high performance computing (HPC) systems. By using the simulator you can evaluate various scheduling policies for a fixed or randomly generated workload for a variety of system sizes. One of the primary motivations for ”yet another simulator” was to make a very simple interface, to help get a general idea on determining a reasonable system size for a workload, and also to better determine system utilization and reduction of workload runtime under various scheduling policies. In addition to traditional scheduling policies I have included a *Dynamic* policy [1] which is determined to be the optimal, or best, policy for a given workload and system size.

**You should use the simulator if you have the following questions:**

1. Given a workload of $n$ jobs, how long will it take to run for a system of size $x$ under Policy $P$?
2. If I have lots of small jobs with an occasional large job, which policy will maximize system utilization?
3. If I double, triple, ..., etc. my system size how does that affect the total runtime of workload $n$?
4. If I modify the system MTTF, how does that affect workload runtime?
5. What is the best scheduling policy for workload $n$ for system size $x$?

6. How are the waiting times of jobs of a particular size or duration affected by the scheduling policy?

For more information on interpreting the scheduler results, see Section 4.

### 1.2 Software/System Dependencies

The simulator has been tested with the following software on an Ubuntu Linux 14.04 system.

- PyQt4
- Python 2.7.5
- GCC compiler v 5.2.0
- GNU Make version 3.82

## 2 Installation and Configuration

For any dependency issues see Section 1.2

```
$>tar -xzvf sched_sim.tgz
$>cd SchedSim/
$>make
cc -DIO=1 -DDEBUG=1 -o SchedSim SchedSim.c -lm
$>./SchedSim.py
```

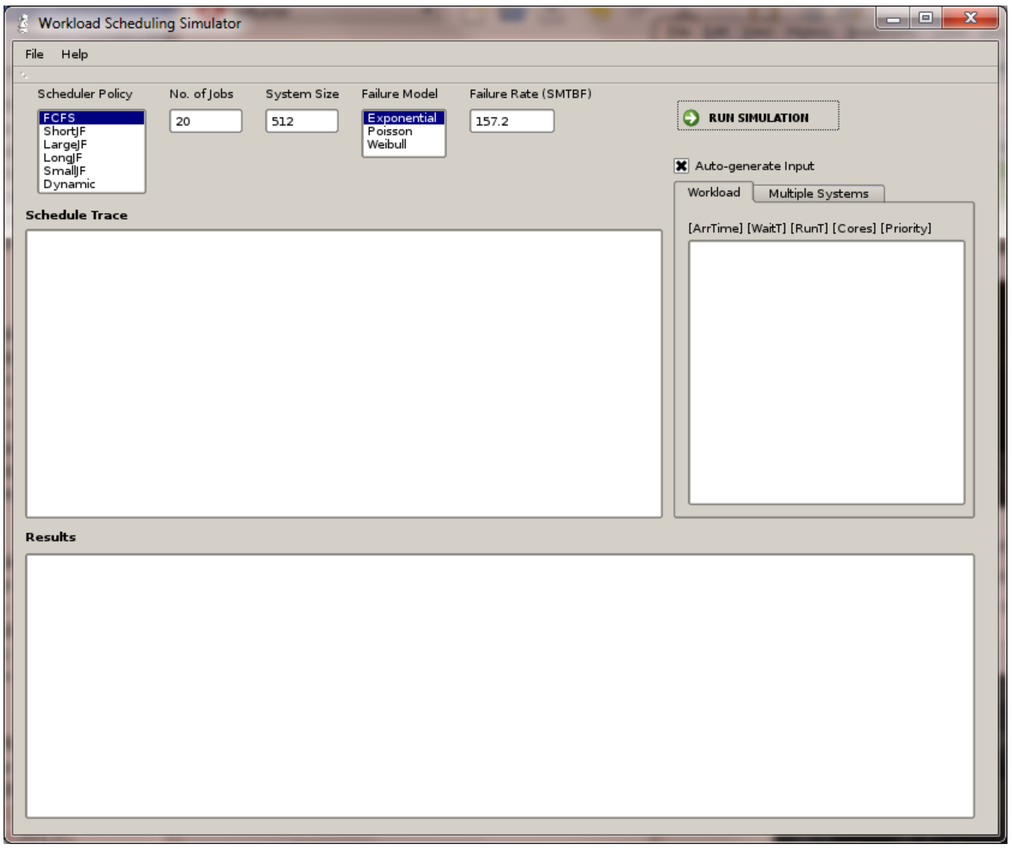
The IO and DEBUG compile flags assure a verbose output. For testing the C program you can disable these to improve the runtime, but you will not get any output in the results or trace windows.

## 3 Running the Simulator

After running the executable you should see a window similar to Figure 1.

Once the GUI is launched, there will be several fields that you can modify, or you can run the simulation with the current values and modify them as needed. You will see five input boxes in the top left corner, a workload text box, and two output text boxes, labeled *Schedule Trace* and *results*. The Scheduler policies determine the characteristics of jobs that are prioritized first. The following common policies are currently available, with the ability to add other as needed.

- FCFS – First Come First Served, which is also Longest Waiting Job First



Scheduler Policy: FCFS, ShortJF, LargeJF, LongJF, SmallJF, Dynamic
No. of Jobs: 20
System Size: 512
Failure Model: Exponential, Poisson, Weibull
Failure Rate (SMTBF): 157.2
Auto-generate Input
[ArrTime] [WaitT] [RunT] [Cores] [Priority]

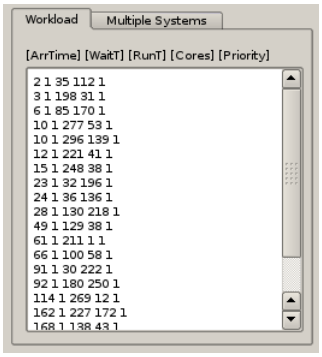
Figure 1: Main Window after launching ./SchedSim.py

- ShJF – Shortest Job First
- LgJF – Largest Job First
- LoJF – Longest Job First
- SmJF – Smallest Job First
- Dynamic – Chooses the *best* policy, **Note: This can take several minutes to run**

The Dynamic policy maximizes the total system utilization and minimizes the total workload runtime. Additional policies are also easy to include, such as Smallest and Shortest jobs first (SSJF), Long and large jobs first (LLJF), etc.

After running the example you will see the fields are now populated with a randomly generated workload, assuming the check box is highlighted and you will see two charts displayed as shown in figures 3, 4, and 5. In this case you have selected to simulate 20 jobs randomly selected, on a system that can run 512 processes at once using a First Come First Serve (FCFS) policy. The generated workload is displayed on the right hand side under the tab "workload". The workload is saved in a file *input.txt* which can be stored and used for later tests. Figure 2 shows either the uploaded or generated workload. The workload input is formatted with the following fields.

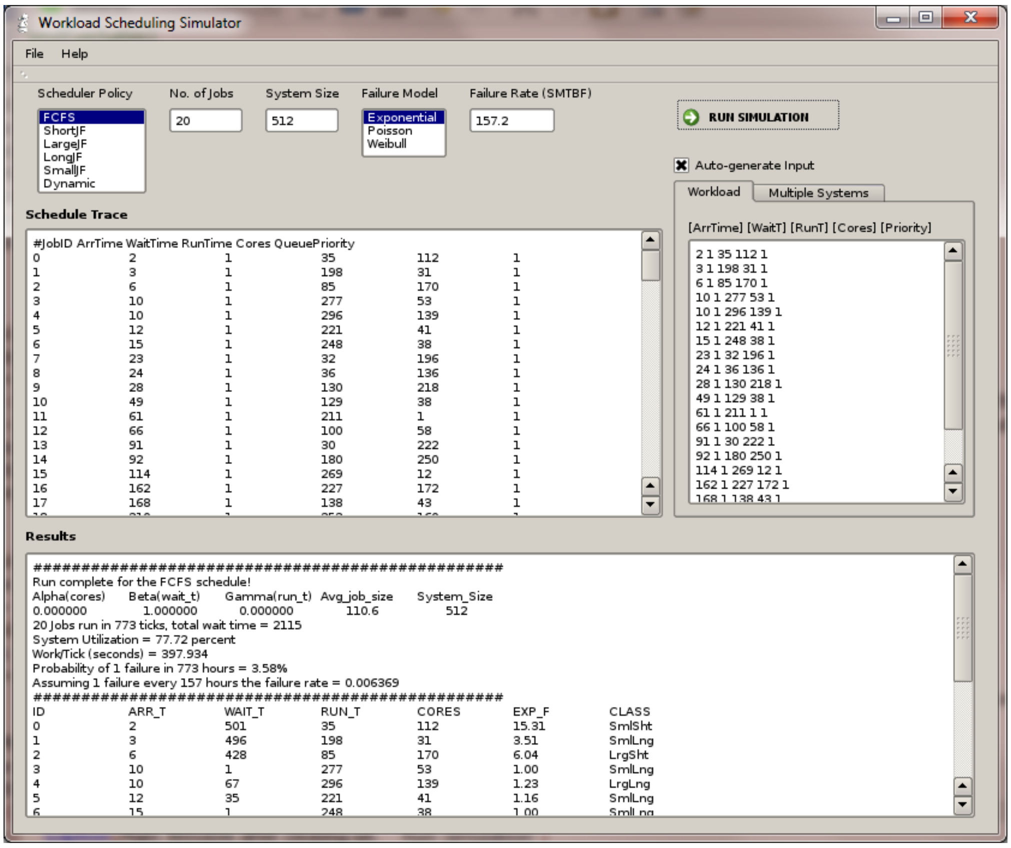
```
[Arrivaltime] [waittime] [run time] [cores/processors] [initial priority]
```



```
[ArrTime] [WaitT] [RunT] [Cores] [Priority]

2 1 35 112 1
3 1 198 31 1
6 1 85 170 1
10 1 277 53 1
10 1 296 139 1
12 1 221 41 1
15 1 248 38 1
23 1 32 196 1
24 1 36 136 1
28 1 130 218 1
49 1 129 38 1
61 1 211 1 1
66 1 100 58 1
91 1 30 222 1
92 1 180 250 1
114 1 269 12 1
162 1 227 172 1
168 1 138 43 1
```

Figure 2: Workload tab after "Run Simulation"



Scheduler Policy: [✓ FCFS] ShortJF LargeJF LongJF SmallJF Dynamic

No. of Jobs: 20

System Size: 512

Failure Model: [✓ Exponential] Poisson Weibull

Failure Rate (SMTBF): 157.2

[✓ Auto-generate Input]

**Schedule Trace**

```
#JobID ArrTime WaitTime RunTime Cores QueuePriority
0       2       1       35      112     1
1       3       1       198     31      1
2       6       1       85      170     1
3       10      1       277     53      1
4       10      1       296     139     1
5       12      1       221     41      1
6       15      1       248     38      1
7       23      1       32      196     1
8       24      1       36      136     1
9       28      1       130     218     1
10      49      1       129     38      1
11      61      1       211     1       1
12      66      1       100     58      1
13      91      1       30      222     1
14      92      1       180     250     1
15      114     1       269     12      1
16      162     1       227     172     1
17      168     1       138     43      1
```

```
[ArrTime] [WaitT] [RunT] [Cores] [Priority]

2 1 35 112 1
3 1 198 31 1
6 1 85 170 1
10 1 277 53 1
10 1 296 139 1
12 1 221 41 1
15 1 248 38 1
23 1 32 196 1
24 1 36 136 1
28 1 130 218 1
49 1 129 38 1
61 1 211 1 1
66 1 100 58 1
91 1 30 222 1
92 1 180 250 1
114 1 269 12 1
162 1 227 172 1
168 1 138 43 1
```

**Results**

```
###################################################
Run complete for the FCFS schedule!
Alpha(cores)   Beta(wait_t)   Gamma(run_t)  Avg_job_size   System_Size
0.000000         1.000000       0.000000       110.6           512
20 Jobs run in 773 ticks, total wait time = 2115
System Utilization = 77.72 percent
Work/Tick (seconds) = 397.934
Probability of 1 failure in 773 hours = 3.58%
Assuming 1 failure every 157 hours the failure rate = 0.006369
###################################################
ID      ARR_T   WAIT_T  RUN_T   CORES   EXP_F   CLASS
0       2       501     35      112     15.31   SmlSht
1       3       496     198     31      3.51    SmlLng
2       6       428     85      170     6.04    LrgSht
3       10      1       277     53      1.00    SmlLng
4       10      67      296     139     1.23    LrgLng
5       12      35      221     41      1.16    SmlLng
6       15      1       248     38      1.00    SmlLng
```

Figure 3: Main Window after clicking on "Run Simulation"

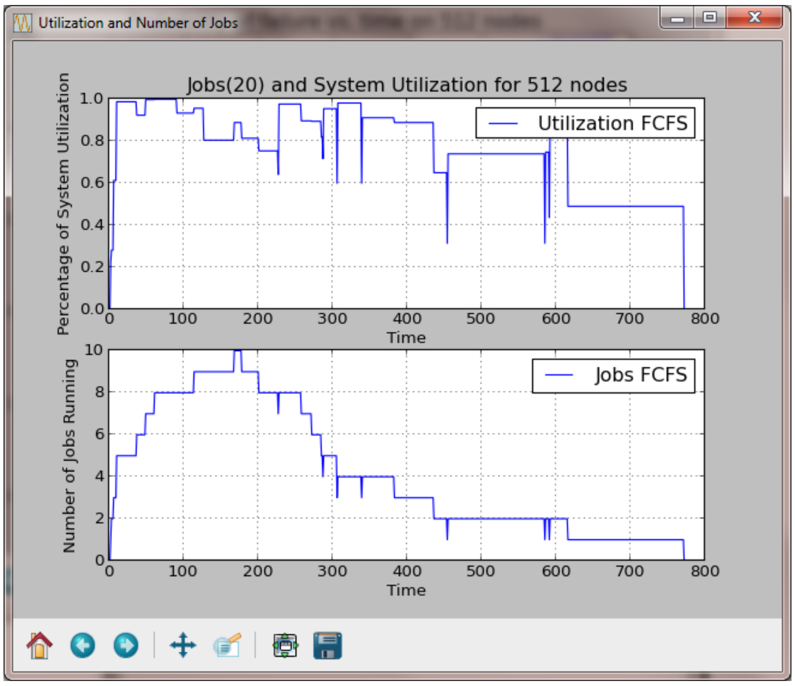


Figure 4: Utilization and Number of jobs for FCFS policy

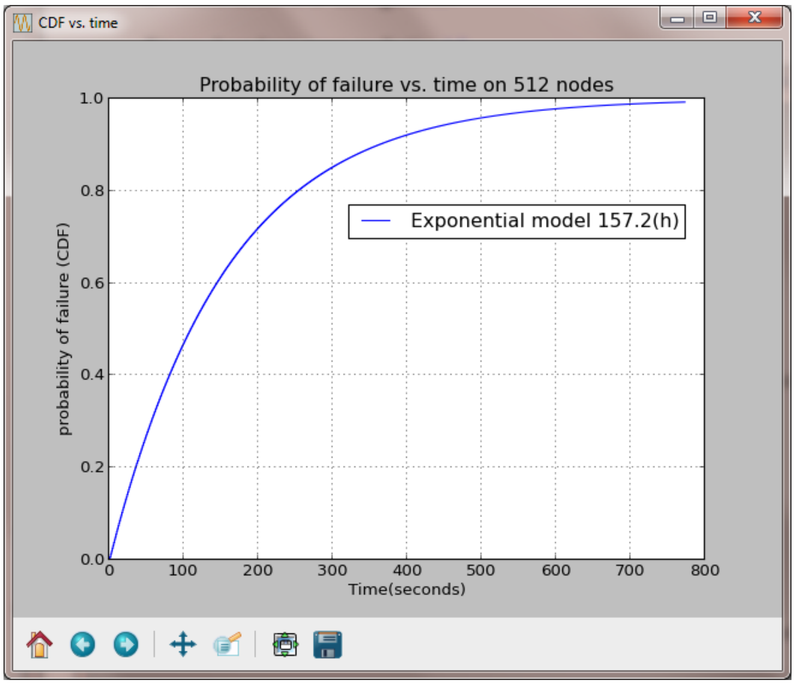


Figure 5: Cumulative distribution function of failure

By de-selecting the Auto-generate Input box, and selecting a different scheduling policy (LargeJF), then rerunning the simulation, we get an updated output text as well as an updated chart as shown in Figure 6. We can see by looking on the $x$ axis the LargeJF policy runs the same workload in around 720 seconds instead of around 775 for the FCFS policy. Also note that you can move the chart legends out of the way by clicking and dragging on them.

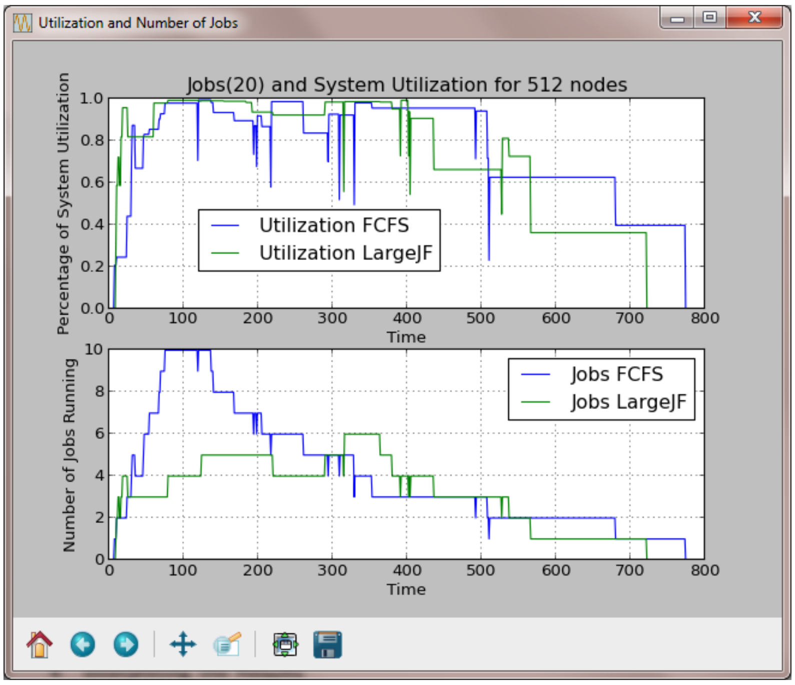


Figure 6: Same workload but with ShortJF policy

## 3.1 saving the results to a file

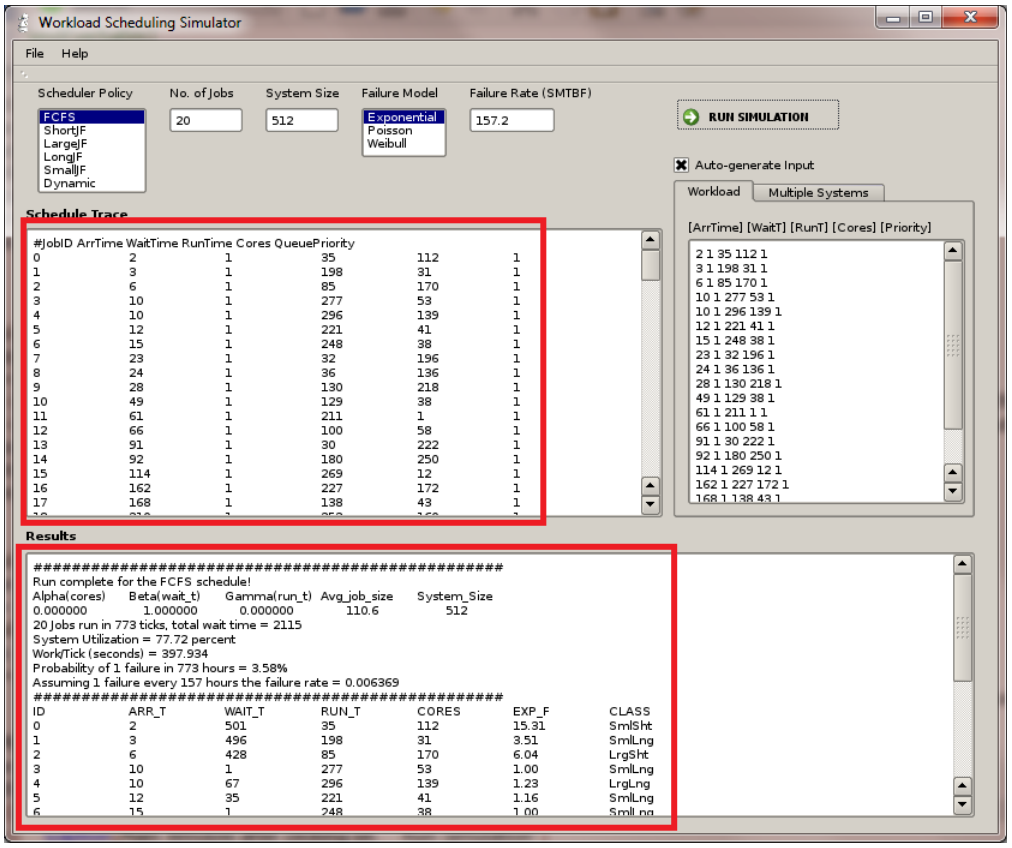
To save the output from the *Results* text box to a file for later review, go to the $File->$ *Save* menu item and type in a filename, it will be saved as a *.txt* file.

## 3.2 opening a workload file

To open a workload file from a previous run, just go to the $File->$ *Open* Menu and select a file on your local system. You can also edit the file in the Workload tab prior to running the simulation. **Note: If the auto-generate input box is checked it will overwrite the input.txt file in the local directory.**

## 4 Interpreting the Results

There are two text boxes where you can review and scroll through the results of the simulation as well as two graphs. Figure 7 shows the *Schedule Trace* and *Results* text boxes highlighted.



Scheduler Policy: FCFS (ShortJF, LargeJF, LongJF, SmallJF, Dynamic)
No. of Jobs: 20
System Size: 512
Failure Model: Exponential (Poisson, Weibull)
Failure Rate (SMTBF): 157.2
[✓] Auto-generate Input

**Schedule Trace**

```
#JobID ArrTime WaitTime RunTime Cores QueuePriority
0       2       1       35      112     1
1       3       1       198     31      1
2       6       1       85      170     1
3       10      1       277     53      1
4       10      1       296     139     1
5       12      1       221     41      1
6       15      1       248     38      1
7       23      1       32      196     1
8       24      1       36      136     1
9       28      1       130     218     1
10      49      1       129     38      1
11      61      1       211     1       1
12      66      1       100     58      1
13      91      1       30      222     1
14      92      1       180     250     1
15      114     1       269     12      1
16      162     1       227     172     1
17      168     1       138     43      1
```

```
[ArrTime] [WaitT] [RunT] [Cores] [Priority]
2 1 35 112 1
3 1 198 31 1
6 1 85 170 1
10 1 277 53 1
10 1 296 139 1
12 1 221 41 1
15 1 248 38 1
23 1 32 196 1
24 1 36 136 1
28 1 130 218 1
49 1 129 38 1
61 1 211 1 1
66 1 100 58 1
91 1 30 222 1
92 1 180 250 1
114 1 269 12 1
162 1 227 172 1
168 1 138 43 1
```

**Results**

```
#######################################################
Run complete for the FCFS schedule!
Alpha(cores)    Beta(wait_t)    Gamma(run_t)  Avg_job_size    System_Size
0.000000          1.000000        0.000000       110.6           512
20 Jobs run in 773 ticks, total wait time = 2115
System Utilization = 77.72 percent
Work/Tick (seconds) = 397.934
Probability of 1 failure in 773 hours = 3.58%
Assuming 1 failure every 157 hours the failure rate = 0.006369
#######################################################
ID      ARR_T   WAIT_T  RUN_T   CORES   EXP_F   CLASS
0       2       501     35      112     15.31   SmlSht
1       3       496     198     31      3.51    SmlLng
2       6       428     85      170     6.04    LrgSht
3       10      1       277     53      1.00    SmlLng
4       10      67      296     139     1.23    LrgLng
5       12      35      221     41      1.16    SmlLng
6       15      1       248     38      1.00    SmlLng
```

Figure 7: Results after clicking on “Run Simulation”

The output format for the Trace displays a tick by tick, or second by second trace of the scheduler, so you can track decisions made. For example in the Schedule Trace window you can see when certain jobs arrive and watch the priority of waiting jobs increase at different rates. The trace view in figure 8 shows the state of all jobs that have arrived over the course of the workload runtime. This shows the ordered priorities or each job and whether or not it is running or has completed. Also shown is the number of available cores. Note that jobs with a 0 in the running and complete column are waiting and thus their priority will increase. Running and complete jobs have a zero priority.For each running job the runtime will decrease one second and the job wait time will increase if the job is not complete or running.

Figure 9 displays the Summary.txt file which includes per job statistics and the total

```
# Tick (312)
# Available cores =  8/512
##############################################
#ID     Tasks   Wait_t  Run_t   Prio.   Running Complete
6       203     263     88      1.664   0       0
12      186     231     161     1.525   0       0
13      185     212     194     1.516   0       0
14      109     198     125     0.893   0       0
18      17      10      65      0.139   0       0
7       118     100     0       0.000   0       1
1       139     1       0       0.000   0       1
15      1       1       52      0.000   1       0
4       71      1       0       0.000   0       1
3       119     1       0       0.000   0       1
11      7       1       0       0.000   0       1
5       221     35      0       0.000   0       1
10      44      214     225     0.000   1       0
8       125     235     124     0.000   1       0
17      115     29      79      0.000   1       0
9       219     46      3       0.000   1       0
16      137     56      0       0.000   0       1
2       69      1       0       0.000   0       1
0       161     1       0       0.000   0       1
(19)job 19 arrived at 312, tick = 312
19      136     2       11      0.000   0       0
```

Figure 8: example from a job trace at time 312, job 19 arrived

run time and wait time of the workload, with the line **20 Jobs run in 723 ticks, total wait time = 2100**.

Additionally we see the total system utilization was 75.77% over 723 seconds. The bottom section prints out per job statistics including expansion factor, where any values close to 1 are optimal. Each job is also placed into one of four classes based on its size and duration vs. the average for all jobs. The last line lists the average expansion factors for each job class. This is useful for doing multiple tests and determining which schedule helps reduce the expansion factor for each job type. The last two fields of the last line shows the total workload runtime and the system utilization.

To determine the optimal policy when the dynamic scheduler is selected, the values for total workload runtime(ticks) and utilization are used.

```
#################################################
Run complete for the LargeJF schedule!
Alpha(cores)    Beta(wait_t)    Gamma(run_t)    Avg_job_size    System_Size
1.000000            0.000000        0.000000           119.1            512
20 Jobs run in 723 ticks, total wait time = 2100
System Utilization = 75.77 percent
Work/Tick (seconds) = 387.938
Probability of 1 failure in 723 hours = 4.61%
Assuming 1 failure every 157 hours the failure rate = 0.006369
#################################################
ID      ARR_T   WAIT_T  RUN_T   CORES   EXP_F   CLASS
0       9       427     183     161     3.33    LrgLng
1       10      1       50      139     1.02    LrgSht
2       12      288     2       69      145.00  SmlSht
3       16      214     138     119     2.55    SmlLng
4       18      29      7       71      5.14    SmlSht
5       27      13      61      221     1.21    LrgSht
6       51      266     88      203     4.02    LrgSht
7       56      1       30      118     1.03    SmlSht
8       57      1       146     125     1.01    LrgLng
9       77      100     194     219     1.52    LrgLng
10      78      1       247     44      1.00    SmlLng
11      79      1       141     7       1.01    SmlLng
12      83      1       161     186     1.01    LrgLng
13      102     323     194     185     2.66    LrgLng
14      116     56      125     109     1.45    SmlSht
15      124     1       240     1       1.00    SmlLng
16      138     1       98      137     1.01    LrgSht
17      157     1       207     115     1.00    SmlLng
18      304     235     65      17      4.62    SmlSht
19      312     1       11      136     1.09    LrgSht
Class_Averages(LL,LS,SL,SS)(1.0 0.0 0.0)  1.9  1.7  1.3  31 723 75.77
```

Figure 9: Summary window for 20 jobs on 512 processor system

## 4.1 Performance metrics

The simulator uses the following performance metrics:

1. Expansion factor: $\frac{JobRuntime+JobWaittime}{JobRuntime}$,which assumes job should wait for as long as the run, this should be as close to 1 as possible.
2. System Utilization: The area under the curve for processors iun use vs. time. The total number of processors used per time step until all of the jobs are complete.

3. Workload runtime: The total time it takes from the first job to arrive and start to the last job to finish.
4. Job Class Expansion factor: Each job can ble classified into four classes, as short or long and large or small, thus (ShLrg, ShSm, LonLrg, LonSm) We are interested in schedules that affect a jobs in a particular class.

### 4.2 Scheduling Assumptions

The scheduler uses a single queue and a greedy fractional knapsack, which is sorted on individual job priorities. The priorities are dynamic with respect to all other jobs in the wait queue, so in every scheduling cycle each job gets an updated priority based on its size, wait time and expected runtime. We make the following assumptions in the job scheduler:

- Single Queue: All jobs are placed into one queue.
- Backfill enabled, a lower priority job may be run if it does not delay the start time of the higher priority job
- No Preemption, jobs will not be stopped once they have started running.
- Non-Malleable jobs, jobs cannot be split into independent tasks by the scheduler. Thus all tasks must run for the job to be considered complete.
- Wait time, a job waiting time increases by one second if it has arrived and has not been run yet and has not completed.

## 5 Details of the Simulation Algorithm

The schedule simulator is written in C with a PyQt GUI, and consists of only a few hundred lines of code. The simulator allows the user to specify the system size, scheduling parameters, and a workload input file. The simulator can also read formatted input files. The simulator generates output in text fields and as charts. The input workload may be modified in the text box. Algorithm 1 illustrates the execution of dynamic policy for the scheduling simulator.

When running experiments we run the simulator with a workload which consists of with various job sizes (number of processors), and duration (length of job). The workloads are summarized in the *workload* tab of the GUI. Each job is prioritized in a single queue using the priority equation 1. We use variables $\alpha$, $\beta$ and $\gamma$ as exponents, which changes the rate of the increase or decrease of job priority, $P_j$ in the queue. Enumerating each parameter for the workload using the simulator runs on the order of tens of seconds. Job priorities are updates after each scheduling cycle. In future work, we will parallelize

```
Data: job file, system size
Result: Schedule performance
Read job input file;
for $\alpha, \beta, \gamma = -2 \rightarrow 2, + = 0.1$ do
    while jobs either running or queued do
        calculate job priorities;
        for every job do
            if job is running and has time remaining then
                update_running_job();
            else
                for all waiting jobs do
                    pull jobs from priority queue and start based on best fit;
                    if job cannot be started then
                        increment waittime
                    end
                end
            end
        end
        Print results;
    end
end
```

**Algorithm 1:** Algorithm for the Dynamic Scheduler

the scheduler to bring the time down. In fact running a parameter sweep with no dependencies allows for testing a variety of scheduling policies.

$$P_j = \left(\frac{Tasks}{avg.Tasks}\right)^{\alpha} \left(waittime\right)^{\beta} \left(\frac{runtime}{avg.runtime}\right)^{\gamma} \tag{1}$$

## 6 Included Fault Models

The simulator allows the user to choose whether or not a system failure affects the workload, this option is enabled by the check box on the GUI. By modeling failures for the workload you can better determine that, in general, for long running jobs on a sufficiently faulty system you will see a lot of jobs needing to be rescheduled. This will affect overall workload runtime. No job or system restart or checkpoint times are included, yet.

The simulator currently models system failure with the three following distributions of the cumulative distribution function (CDF). The input for failures is $\lambda$ which is $\frac{1}{MTTF}$ where MTTF is **mean time to a single system failure** entered in the GUI.

- Exponential: $P(t) = e^{-\lambda t}$ where $t$ = workload runtime.
- Poisson: $P(X) = \frac{e^{-\lambda}\lambda^x}{x!}$. A failure occurs if the probability of failing at any time is constant and failures are independent, thus "at random". $x$ is the number of failures over the given time, in our case one (1).
- Weibull: $F(x; k, \lambda) = 1 - e^{-(x/\lambda)^k}$, where $k$ is the shape parameter, which is fixed at 1 for now.

**Note: The current implementation of failures for this version of the simulator selects only a single job for failure at some time, for the given rate, not all jobs running.**

## 7 Issues and Future Work

Below is a list of current issues that will invalidate the results, crash the simulator, or do nothing.

### 7.1 Known Issues

- Entering more than 300 jobs takes several seconds.
- Negative numbers do not work or will crash the simulator
- When uploading a workload file, it needs to be in sorted order by arrival time.

### 7.2 Future Tasks

Aside from improving the stability and robustness of the scheduling algorithms, here are some features that would be nice to have.

- Add a button to generate *.arff* formatted out files to be read by the WEKA Machine Learning toolkit.
- Add power usage (joules) for each job and prioritize each job by power consumption.
- Do more with system parameters for estimating failures, like include restart times, repair times, or flag jobs for check pointing after $t$ seconds, etc.
- Come up with good $\beta$ and $\alpha$ values for the Weibull distribution.
- Look at using probability distribution functions with or instead of cumulative distribution functions.
- Since everything is in C and python, it should be fairly easy to port to Windows.

## 8 Further Reading

There are hundreds of papers and books on job scheduling for HPC systems. A more recent book that provides a clearly written detailed overview, that also happens to be free, and motivated some of this work is *Workload Characterization and Modeling* by Dror Feitelson [2]. A more detailed description of the prioritization algorithm as well as its suitability for HPC workloads is provided in [1] and [3]. An example of its use in scheduling jobs for gridded satellite data for climate simulation studies can be seen in [4]. Additionally, we have shown how to incorporate dynamic priorities into Hadoop workloads [5]. For more information regarding the failures, the approach has been published in [6] and is based on work largely from Wolstenhome [7].

## References

[1] T. A. Simon, P. Nguyen, and M. Halem, "Multiple objective scheduling of hpc workloads through dynamic prioritization," in *Proceedings of the High Performance Computing Symposium*, ser. HPC '13. San Diego, CA, USA: Society for Computer Simulation International, 2013, pp. 13:1–13:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2499968.2499981

[2] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*, 1st ed. New York, NY, USA: Cambridge University Press, 2015.

[3] T. A. Simon and J. W. McGalliard, "Some workload scheduling alternatives in a high performance computing environment." in *Int. CMG Conference*, 2013.

[4] D. Chapman, T. A. Simon, P. Nguyen, and M. Halem, "A data intensive statistical aggregation engine: A case study for gridded climate records," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, May 2013, pp. 2157–2164.

[5] P. Nguyen, T. Simon, M. Halem, D. Chapman, and Q. Le, "A hybrid scheduling algorithm for data intensive workloads in a mapreduce environment," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 161–167. [Online]. Available: http://dx.doi.org/10.1109/UCC.2012.32

[6] T. A. Simon and J. Dorband, "Improving application resilience through probabilistic task replication," in *Workshop on Algorithmic and Application Error Resilience (AER)*, ser. ACM International Conference on Supercomputing (ICS'13), June 11 2013.

[7] L. C. Wolstenholme, *Reliability Modelling: A Statistical Approach.* Chapman and Hall/CRC., 1999.