CCU-LANL Batsim User Doc Manual

Craig Walker

October 28, 2024

Contents

i. Preface 4

ii. Style Of Document 4

1 Intro 6

1.1 How Batsim Works . . . 6

1.2 How Our FrameWork Works . . . 6

1.2.1 The Config File . . . 7

1.2.2 myBatchTasks.sh . . . 7

1.3 Our Folder Structure . . . 8

1.3.1 ’Project Folder’/ . . . 8

1.3.2 ’Experiment Folder’/ . . . 8

1.3.3 experiment_#/ . . . 8

1.3.4 id_#/ . . . 9

1.3.5 Run_#/ . . . 9

1.3.6 input/ and output/ . . . 9

1.3.7 Real Checkpointing: checkpoint_#/ and expe-out_#/ . . . 10

1.3.7.1 Real Batsim Checkpointing Options To Write Out Checkpoint . . . 10

1.3.7.2 How Checkpoints Are Written, checkpoint_#/ . . . 10

1.3.7.3 Starting From A Checkpoint, expe-out_#/ . . . 11

1.3.8 Putting ’Folder Structure’ all together . . . 12

1.4 How Our Analysis Works . . . 12

1.4.1 aggregate_makespan.py . . . 13

1.4.2 aggregate_aggregates.py . . . 13

1.4.3 .../tests/analysis.py . . . 13

1.4.4 .../tests/application_efficiency.py . . . 13

2 Installation/Deployment 14

2.1 Requirements . . . 14

2.2 Deploy Methods . . . 14

2.3 Deploy: How To... . . . 15

2.3.1 Bare-Metal . . . 15

2.3.2 Docker . . . 15

2.3.3 CharlieCloud with Internet . . . 16

2.3.4 CharlieCloud without Internet . . . 16

2.4 Make Sure Everything Works . . . 16

2.4.1 Bare-Metal works . . . 17

2.4.2 Docker works . . . 17

2.4.3 CharlieCloud works . . . 18

3 Batsim Environment **19**

4 Config File - In Depth **21**

4.1 Basic Outline . . . 21

4.2 Sweeps . . . 22

4.2.1 Explanation of Sweeps . . . 22

4.2.2 Types of Sweep Functions . . . 23

4.2.3 Current Sweeps Available - (Functions Allowed) . . . 24

4.3 Workloads . . . 26

4.3.1 grizzly-workload . . . 26

4.3.2 synthetic-workload . . . 28

4.4 Regular Options . . . 30

4.5 Algorithms . . . 37

4.5.1 fcfs_fast2 . . . 38

4.5.2 easy_bf2 . . . 38

4.5.3 easy_bf3 . . . 38

4.5.4 conservative_bf . . . 38

4.5.5 easy_bf_fast2 . . . 38

4.5.6 easy_bf_fast2_holdback . . . 38

4.6 Reservations: All Things About Reservations . . . 38

4.6.1 Reservations . . . 39

4.6.1.1 machines . . . 41

4.6.2 Reservation-Sweeps . . . 44

5 Command Reference **50**

5.1 Batsim Environment . . . 50

5.1.1 batEnv . . . 50

5.1.2 batVersion . . . 50
5.1.3 batExit . . . 50
5.1.4 batFile . . . 50
5.1.5 batFolder . . . 51
5.1.6 bind_all . . . 53
5.2 myBatchTasks.sh . . . 53
5.3 progress.sh . . . 56
5.4 aggregate_makespan.py . . . 60

i. Preface

This manual serves to be a definitive guide, for the user, to our CCU-LANL additions to the Batsim/Batsched simulator. If you need a guided tour through the CCU-LANL additions, make sure to read ***User_Doc_Walkthrough.pdf***.

As the user guide, this document will explain all options and cover different scenarios. This guide will *not* cover how to code for additions to the simulator, however. If one desires this (such as adding another scheduling algorithm, adding another option to Batsim, or adding sweeps) please look to our ***Developer_Doc_Manual.pdf***. It may still be advisable to read this document and/or ***User_Doc_Walkthrough.pdf*** to get a sense of Batsim before you start changing the code.

It is our hope that our additions to Batsim can serve the community well, and that this guide will explain things as clearly as we hope. Good luck with all things Batsim, and all things HPC.

ii. Style Of Document

There is a certain style to this guide that should be made apparent.

- **Inline style:**
  1. `Commands you would run ./from --the --terminal --look --like --this.`
  2. `Just an --argument to a command will look like this.`
  3. `A config 'property': will look like this`
  4. `A/folder/or/file/path/would/look/like/this.`
  5. `Code::would #look like() this.`

- **Block style:**
  1. **Terminal**

```
user > #this is a terminal block, and this is a comment in it.
user > ./and_this_would_be_a_command & | if [[ ]] ; for ;do echo
user > cd ~/our/path # and this is a known command
user > su -
Password:
root > ./this_would_run_as_root
```

  2. **Code**

```
//A c++ code block looks like this, and this is a c/c++ comment in it
and this::is::a::function()
{
  with an int definition;
  int a=10;
  string name="CCU-LANL";
  return 10;
}
```

```python
# and this is python code
import pandas as pd
with open("file.csv","r") as InFile:
  df = pd.read_csv(InFile,sep=",")
def hello:
  print("world")
  q = [ 5,10 ]
```

3. **Explanations**
   (a) **Additional Info**

   Explains Some Additional Info

   Additional info here

   (b) **Important Info**

   Explains Important Info

   This is very important

   (c) **Warning Info**

   Info That Warns You

   This will certainly break the internet

1 Intro

There are a couple things to define right now so it is easier to communicate.

These are things we wish to define:

1. How Batsim Works
2. How Our FrameWork Works
3. Our Folder Structure
4. How Our Analysis Works

1.1 How Batsim Works

Batsim uses 4 main tools:

1. **Simgrid**
   - This is what runs under the hood, simulating the passage of time and the running of jobs, as well as the cluster layout (taken from a platform xml file).
2. **Batsim**
   - This is what takes most of our options, reads in our workload, and tracks the progress of jobs. It stays in communication with Batsched through events that are sent over a messaging system (using a socket). That means if no events are happening and the last call to Batsched had finished, then Batsched is not being run.
3. **Batsched**
   - This is what is responsible for making scheduler decisions. It is alerted when a job is submitted, killed, and completed (along with some other notification alerts). But the main thing it decides is when a job starts and what machines to start it on. There are several algorithms to choose from for the scheduler.
4. **Robin**
   - Batsim is tied to the Simgrid library and runs as one process. Batsched runs as another process. Robin is a nice tool that bridges these two processes together as its own process and tracks the progress of Batsim and Batsched.

So from these 4 tools there are 3 processes that run during one simulation: batsim, batsched, and robin. Note that all 3 of the tools are lowercase executables.

1.2 How Our FrameWork Works

Our Framework is made to setup and run simulations, and then post-process after the simulation ends. Most of the work to this framework has gone into the setting up and running of the simulations. It relies on a single json config file that the user makes, as well as some additional options sent to the tool `myBatchTasks.sh`. This tool sets up your experiment's folder structure and options being sent to each simulation. It can be run on a single machine or a cluster.

1.2.1 The Config File

Without going into too much detail yet, the config file has 4 main parts to it:

1. **Sweeps**
   - These allow you to do a parameter range (or sweep) over certain values. Each sweep is a little different in its syntax. Each value that is generated as a parameter to an option represents a new experiment. If multiple sweeps are used the amount of experiments produced get multiplied. So 2 options generated by one sweep, then 3 options generated by another sweep would mean 6 experiments would be generated.
2. **Individual options**
   - These are just regular options added to each experiment.
3. **Workload options**
   - These are options to either a "grizzly-cluster-like" workload or a synthetic workload. When random numbers are used they can generate a subset of experiments we've called "id"s (as each workload generated has a unique ID) and their simulation's output are put into their own "id" folder under the experiment folder.
4. **Output options**
   - There are only a couple options as to what output you want. But the main option has to do with how many "runs" to do for each experiment. Given the random nature some options give to the simulation itself, it is advisable to run many "runs" per experiment when this is the case. Then you can use tools to average them. 400 runs should give you an idea of what is happening. 1000 or 1500 runs has seemed to be good for papers we have written.

1.2.2 myBatchTasks.sh

Our Framework relies on the command-line (for now), and so there are tools to make this easier. As mentioned, the main tool you will use is `myBatchTasks.sh`. What you will do is:

1. write your config file
2. point `myBatchTasks.sh` to this file
3. tell `myBatchTasks.sh` what kind of parallelization you want/need
4. run `myBatchTasks.sh`. This will:
   (a) create a workload and platform for each simulation (When needed. Workloads and platforms get reused if they can be reused)
   (b) set up the folder for each simulation
   (c) write the options each simulation gets inside its folders
   (d) depending on the parallelization will:
   - run each simulation in turn **(parallelization: none)**
   - run batches of simulations by backgrounding them **(parallelization: background)**
   - run sbatch on every simulation **(parallelization: sbatch)**
   - run sbatch on batches of simulations and run those simulations as tasks **(parallelization: tasks)**

   (e) In the case of parallelization:
   - **none | background**, `myBatchTasks.sh` will continue to run to completion
   - **sbatch | tasks**, `myBatchTasks.sh` will stop after handing over the jobs to SLURM

   (f) The simulations will start and after finishing, will run `post-processing.py` to post-process.

1.3 Our Folder Structure

Our folder structure has been a work in progress, so it may seem a little chaotic. Hopefully after explaining it, it will make sense to you.

There are 2 main folders in our FrameWork:

1. The **Simulator Folder**
   - This folder is basically your default location.
   - Holds a `configs` folder for your config files.
   - Holds an `experiments` folder for your projects.
   - Holds `basefiles` folder:
     - Holds all the scripts for deploying batsim, running simulations, processing simulations, and analysis.
     - Holds the `workloads` folder where workloads are kept.
2. The **Project Folder**
   - This folder is where a project resides.
   - This is normally in the `simulator/experiments` folder, but can be located anywhere.

1.3.1 'Project Folder'/

The Project Folder is where all your simulations will end up. This can be named anything and put anywhere. Consider the I/O requirements of your simulations and make sure that the drives you choose to store things on are up to the task. Each simulation writes out to I/O for each simulated job that completes, and the chosen options of your logging (it is highly recommended you turn logging off on 'real' simulations...ie ones that aren't just tests). So, drives that are somewhat responsive are important.

1.3.2 'Experiment Folder'/

In your json config file you can set options for a set of jobs known as an "Experiment". You must supply at least one Experiment, but you may choose to include more than 1 Experiment in your config file. Each Experiment can be named anything you want, but something descriptive is helpful. Also each Experiment name must be unique for each Project.

So to summarize, a place to store your project is chosen with the name of the Project Folder tacked on. Then the config file is read, and each first-level json key that includes an `"input":{ }` and `"output":{ }` section will be made into an 'Experiment Folder' in your `.../'Project Folder'`.

1.3.3 experiment_#/

When given the minimum required options, `myBatchTasks.sh` will start by running a setup script. This setup script will read in your config file and generate the project's experiment(s), one for each first level json key. Within each first level json key is an `'input'` key and an `'output'` key. The options, including the sweeps, will generate a job for each set of unique options. These jobs are labeled `experiment_#` where # is an integer, and reside under the `.../'Project Folder'/'Experiment Folder'/` folder.

In a future release we want to change these folder's names to `job_#` to make the naming clearer, or maybe even `options_#` to differentiate them when talking about SLURM 'jobs'. But for now, there are 'project's experiment(s)', and there are jobs within the 'project experiment' named `experiment_#`.

1.3.4 id_#/

Under each job folder is an `id_#` folder. This represents the workload `'index':<int>` that was used. An index on workloads helps make them unique. Sometimes you want to roll the dice on the randomness again when a workload is created. A unique index with the other options will force the creation of a workload because a workload is created when no other options match it, the index being one of them. So will `'force-creation':true` but an index can be referenced again later if you want to reuse that workload, as long as all the other options remain the same and the database hasn't been deleted. Also, `'workload-ids':<str>` can set multiple indices for a single 'project experiment', putting each index in its own `id_#` folder. This is how you would do some kind of statistical analysis on the workload options.

1.3.5 Run_#/

Under each `id_#` folder are `Run_#` folders. Instead of creating a bunch of new workloads for statistical analysis (like in id's), this represents a single workload for all Runs. However, certain options to Batsim will cause randomness (failures). To do statistical analysis for these situations we use multiple 'runs'. The amount of runs for a certain 'project experiment' can be set in the `'output':{ }` section. We have called the option `'avg-makespan':<int>` for historical reasons. Set that to how many runs you would like for each id and for every job. This adds up quickly. If a 'project experiment' has 3 jobs and we did 10 indices with `'workload-ids':<str>` option, and then we did 20 runs with the `'avg-makespan':<int>` option, that would equate to 3 * 10 * 20 = 600 simulations and 10 workloads being created (provided they were unique indices for those workload options).

1.3.6 input/ and output/

After `Run_#` there are two folders made: `input` and `output`. `input` gets the `config.ini` for the simulation options. Before running a simulation, this `config.ini` helps make the `experiment.yaml` file that `robin` creates and reads.

In a future release of Batsim we may move the workload creation into Batsim itself. Currently it remains part of the setup before launching the simulations.

After the setup script is completed it will enter the `run-experiments.py` phase. Each 'project experiment' is traversed and each `project/experiment/experiment_#/id_#/Run_#` will be submitted as a simulation. If done locally, `run-experiments.py` will continue until all of the simulations finish. If done using SLURM, `run-experiments.py` passes all of these simulations over to SLURM and the nodes will run their simulations.

Simulations are started in the `real_start.py` script. `robin` is invoked in this script and generates the appropriate `.../Run_#/input/experiment.yaml` file from the `.../Run_#/input/config.ini`. `robin` is invoked again to run the simulation from the yaml file. Once the simulation has ended correctly, the `.../Run_#/output/config.ini` is read and the appropriate options are sent over to the `post-processing.py` script. Finally, the `.../Run_#/output/progress.log` file is updated with `{ "completed":true }`

SLURM puts the job's log info in `.../Run_#/output/slurm-#.out`. Simulations, on the other hand, put their output in

`.../Run_#/output/expe-out`, mainly `.../expe-out/out_jobs.csv` and `.../expe-out/out_extra_info.csv`. Batsim log info goes to `.../expe-out/log/batsim.log` and Batsched log info goes to `.../expe-out/log/sched.err.log`

1.3.7 Real Checkpointing: checkpoint_#/ and expe-out_#/

1.3.7.1 Real Batsim Checkpointing Options To Write Out Checkpoint

Checkpointing Batsim is a recent feature added where Batsim is able to do in real life what it also simulates in the simulation, checkpointing its state so it can start from that point in the simulation later. There are three (3) options for this to happen:

- `'checkpoint-batsim-interval'`
  - Will set an interval to do real checkpoints
    - syntax: <string>
      - <string> syntax: `"(real|simulated)[:once]:days-HH:MM:SS[:keep]"`
        - ’real’ prepended will interpret the interval to be in real time
        - ’simulated’ prepended will interpret the interval to be in simulated time
        - ’optional :once will do one checkpoint and then stop doing any more checkpoints
        - days-HH:MM:SS is simply days followed by a ’-’ followed by hours:days:seconds.
          - days is simply multiplied by 3600*24, hours by 3600, and minutes by 60
          - days, hours, minutes, and seconds can actually be as many digits as you want greater than 0 digits.
          - This means the time can be completely in seconds, or minutes, or hours, or days
          - examples: ’0-0:0:537’ or ’3-00:00:00’ or ’1-12:30:00’
        - optional :keep will set the amount of checkpoints to keep. `'checkpoint-batsim-keep'` trumps this.

- `'checkpoint-batsim-keep'`
  - How many checkpoints to keep. default: 1
    - syntax: <int>

- `'checkpoint-batsim-signal'`
  - The signal number to use for signal driven checkpointing. SLURM can send a signal to checkpoint to the process before the job’s wallclock-limit is reached. see SBATCH_SIGNAL in SLURM docs.
    - syntax: <int>
      - <int> syntax:
        - You will want to either use SIGUSR1(10), SIGUSR2 (12), or preferably real-time signals from 35-64

1.3.7.2 How Checkpoints Are Written, checkpoint_#/

The reason we include Real Checkpointing in the ’Folder Structure’ section is to illustrate the folders that are made using this feature.

When a real checkpoint is written out, it will put the checkpoint in `.../Run_#/output/expe-out/checkpoint_1/`. There will also be a symbolic link created to this folder `.../Run_#/output/expe-out/checkpoint_latest`.

If another checkpoint is written out, what happens depends on whether you have chosen to 'keep' any checkpoints or not. This is set in either `'checkpoint-batsim-interval'` or `'checkpoint-batsim-keep'`. The default is 1 and should not be less than 1. If it is set to 1, then the next checkpoint simply overwrites the `.../expe-out/checkpoint_1/` folder.

If 2 is set to be kept then `.../expe-out/checkpoint_1/` is moved to `.../expe-out/checkpoint_2/`, and the current checkpoint is written to `.../expe-out/checkpoint_1/`. `.../expe-out/checkpoint_latest` always points to `.../expe-out/checkpoint_1/`. If another checkpoint is written then `.../expe-out/checkpoint_2/` is deleted and `.../expe-out/checkpoint_1/` moves down to `.../expe-out/checkpoint_2/`, and then the current checkpoint is written to `.../expe-out/checkpoint_1/`.

If 3 is set to be kept, we get a similar behavior as when 2 is kept except that on the 3rd checkpoint, `checkpoint_2/` moves to `checkpoint_3/`. And then on the 4th checkpoint, `checkpoint_3/` is deleted, everything else moves 'down', so `checkpoint_2/` becomes `checkpoint_3/`, `checkpoint_1/` becomes `checkpoint_2/`, and the current checkpoint is, again, written to `checkpoint_1/`.

1.3.7.3 Starting From A Checkpoint, expe-out_#/

Starting from a checkpoint starts with options to `myBatchTasks.sh`, mainly `-S, --start-from-checkpoint <int>`. Make sure that the number passed corresponds to a checkpoint that actually exists. Passing a number not corresponding to a checkpoint that exists will result in errors and stop execution of your simulations. There may be a case when some simulations didn't checkpoint for that iteration, but some did. In this case you can use the `-I,--ignore-does-not-exist` option. It will run those simulations that have that checkpoint and won't do anything to those that don't have the checkpoint.

The order that things happen is the following: First `generate_config.py` is called with the –start-from-checkpoint <int> option and that triggers `generate_config.py` not to read your config script. Instead it changes all the `.../input/config.ini` files to update what checkpoint they are starting from and other values mentioned below.

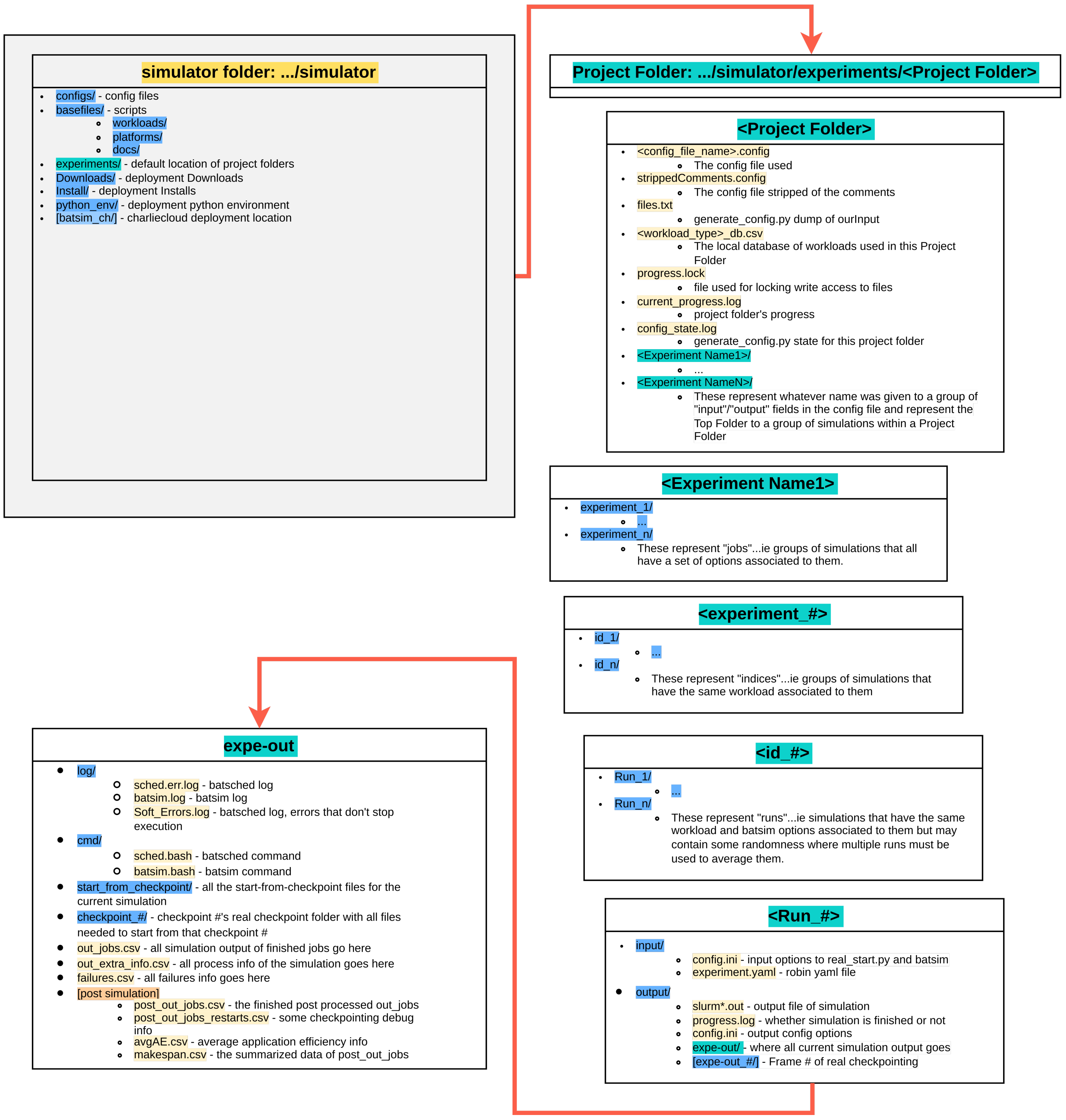
Second when a simulation starts from a checkpoint in the `run-experiments.py` phase and later in the `real_start.py` individual simulation script, it will first check that the checkpoint path exists and if it does, then it will move the current `.../output/expe-out` folder to `../output/expe-out_1`. This is what we call a 'Frame'. The new simulation always occurs in `.../output/expe-out`.

There is an option to `myBatchTasks.sh` `-K, --start-from-checkpoint-keep <int>`. This will keep <int> frames. Every time you start from a checkpoint a new frame is made if that checkpoint path existed. If you set the `-K <int>` option, that many Frames will stay around. If you set it to 2 and start from a checkpoint 3 times, then one frame will be deleted, much like checkpoints are moved 'down' and off.

To start from a certain frame use the `-F, --start-from-frame <int>` option. In conjunction with `-S, --start-from-checkpoint <int>` it will start the simulation at `.../output/expe-out_#A/checkpoint_#B` where #A is the frame and #B is the checkpoint. There is also an option `-D, --discard-last-frame`. This will make it like the last frame didn't happen, then will apply all the start options you gave it and run your simulations.

One last option before we move on to other things: `-C, --skip-completed-sims`. This option will check the `.../output/progress.log` for each simulation to see if the simulation had finished. If it had, then it will skip running it.

1.3.8 Putting 'Folder Structure' all together



simulator folder: .../simulator

- configs/ - config files
- basefiles/ - scripts
    - workloads/
    - platforms/
    - docs/
- experiments/ - default location of project folders
- Downloads/ - deployment Downloads
- Install/ - deployment Installs
- python_env/ - deployment python environment
- [batsim_ch/] - charliecloud deployment location

Project Folder: .../simulator/experiments/<Project Folder>

<Project Folder>

- <config_file_name>.config
    - The config file used
- strippedComments.config
    - The config file stripped of the comments
- files.txt
    - generate_config.py dump of ourInput
- <workload_type>_db.csv
    - The local database of workloads used in this Project Folder
- progress.lock
    - file used for locking write access to files
- current_progress.log
    - project folder's progress
- config_state.log
    - generate_config.py state for this project folder
- <Experiment Name1>/
    - ...
- <Experiment NameN>/
    - These represent whatever name was given to a group of "input"/"output" fields in the config file and represent the Top Folder to a group of simulations within a Project Folder

<Experiment Name1>

- experiment_1/
    - ...
- experiment_n/
    - These represent "jobs"...ie groups of simulations that all have a set of options associated to them.

<experiment_#>

- id_1/
    - ...
- id_n/
    - These represent "indices"...ie groups of simulations that have the same workload associated to them

<id_#>

- Run_1/
    - ...
- Run_n/
    - These represent "runs"...ie simulations that have the same workload and batsim options associated to them but may contain some randomness where multiple runs must be used to average them.

<Run_#>

- input/
    - config.ini - input options to real_start.py and batsim
    - experiment.yaml - robin yaml file
- output/
    - slurm*.out - output file of simulation
    - progress.log - whether simulation is finished or not
    - config.ini - output config options
    - expe-out/ - where all current simulation output goes
    - [expe-out_#/] - Frame # of real checkpointing

expe-out

- log/
    - sched.err.log - batsched log
    - batsim.log - batsim log
    - Soft_Errors.log - batsched log, errors that don't stop execution
- cmd/
    - sched.bash - batsched command
    - batsim.bash - batsim command
- start_from_checkpoint/ - all the start-from-checkpoint files for the current simulation
- checkpoint_#/ - checkpoint #'s real checkpoint folder with all files needed to start from that checkpoint #
- out_jobs.csv - all simulation output of finished jobs go here
- out_extra_info.csv - all process info of the simulation goes here
- failures.csv - all failures info goes here
- [post simulation]
    - post_out_jobs.csv - the finished post processed out_jobs
    - post_out_jobs_restarts.csv - some checkpointing debug info
    - avgAE.csv - average application efficiency info
    - makespan.csv - the summarized data of post_out_jobs

1.4 How Our Analysis Works

As mentioned, the analysis is mostly left up to the user. There are a couple exceptions.

1.4.1 aggregate_makespan.py

To average and aggregate a folder's `avg_makespan.csv` files you can use this script. It will output a single `total_makespan.csv` file in the project folder. It will also output an `errors_total_makespan.csv` file so that you can see what jobs did not complete. It lists which files it did not find and it has a summary at the bottom.

This file gives an alright example of how to traverse the project folder.

1.4.2 aggregate_aggregates.py

This is used for aggregating multiple `total_makesan.csv` files. This can be useful if you do a bunch of sims in batches.

1.4.3 .../tests/analysis.py

This file goes along with our config file `.../tests/configs/paper.config`. It has a lot of code in it, but hopefully you can find something that will pertain to your needs.

1.4.4 .../tests/application_efficiency.py

This file goes along with our config file `.../tests/application_efficiency.config`. It was one of the first things we used to verify that failures and simulated checkpointing were working. We hope you can learn something from this simple script, as well.

2 Installation/Deployment

Here you will learn how to deploy the simulator in an environment right for you.

2.1 Requirements

Requirements (bare-metal and charliecloud):

- linux os
- gcc >= 8.0 (bare-metal needs c++17, charliecloud method may allow for previous versions)
- cmake >= 3.15.4 (maybe previous versions. at least 3.11)
- python == 3.6
- python3-venv
- pip3
- typical build system
  - make
  - build
  - git
  - patch (bare-metal)
  - libtool (if not installed, deployment can attempt to build and install)
  - pkg-config (if not installed, deployment can attempt to build and install)
  - build-essential (ubuntu package. named other things on other distros)
- bash shell

Requirements (docker method):

- linux os
- git
- docker running and working

2.2 Deploy Methods

There are 4 methods of building and deploying our batsim applications.

- **bare-metal**
  - will compile and install everything you need into a directory
- **docker**
  - will compile and install everything you need into a docker container
  - currently there is no option of parallelism with this method

- **charliecloud with internet**
  - charliecloud is a container technology that works when docker is not an option (think clusters without docker)
  - will compile and install everything you need into a directory
- **charliecloud without internet**
  - charliecloud is a container technology that works when docker is not an option (think clusters without docker)
  - meant to be run where you have internet and then copy a folder (3.5GB) to the cluster without internet
    - will compile and install everything you need and will be packaged into a directory to be copied to your setup without internet, then you can attempt to unpackage it there.

2.3 Deploy: How To...

All of the methods rely on running `.../simulator/basefiles/deploy.sh`. One can run `deploy.sh --help` for complete usage info.

2.3.1 Bare-Metal

1. obtain the code
2. change directories
3. deploy

```
git clone https://cswalke1:ekhr1Q_mL356zvCt_p2B@gitlab.newmexicoconsortium.org/lanl-
↪   ccu/simulator.git
cd simulator/basefiles
./deploy.sh -f bare-metal --prefix $(dirname $(pwd))
```

2.3.2 Docker

1. obtain the code
2. change directories
3. deploy

```
git clone https://cswalke1:ekhr1Q_mL356zvCt_p2B@gitlab.newmexicoconsortium.org/lanl-
↪   ccu/simulator.git
cd simulator/basefiles
./deploy.sh -f docker
```

2.3.3 CharlieCloud with Internet

1. obtain the code
2. change directories
3. deploy

```
git clone https://cswalke1:ekhr1Q_mL356zvCt_p2B@gitlab.newmexicoconsortium.org/lanl-
↪  ccu/simulator.git
cd simulator/basefiles
./deploy.sh -f charliecloud
```

2.3.4 CharlieCloud without Internet

1. obtain the code
2. change directories
3. deploy package
4. change directories
5. scp folder
6. ssh to remote
7. change directories
8. unpackage

```
git clone https://cswalke1:ekhr1Q_mL356zvCt_p2B@gitlab.newmexicoconsortium.org/lanl-
↪  ccu/simulator.git
cd simulator/basefiles
./deploy.sh -f charliecloud --no-internet --package
cd ../../

#to be modified for your method of sending a folder to your remote location and logging
↪  in to your remote location
scp -r ./batsim_packaged user@remote.org:/home/USER/
ssh user@remote.org


cd /home/USER/batsim_packaged
./deploy.sh -f charliecloud --no-internet --un-package
```

2.4 Make Sure Everything Works

Here you will learn how to test that your deployment works. You can make sure your particular deployment works by using our tests: `.../simulator/basefiles/tests/test_simulator.py`.

Keep in mind that SLURM tests assume the following:

- You are on a cluster running SLURM
- You have access to at least two (2) nodes, otherwise it's not much of a parallel test

2.4.1 Bare-Metal works

Read the following list of instructions and then perform the commands below it.

1. change directories `/path/to/simulator/basefiles`
2. edit `batsim_environment.sh`
3. source `batsim\_environment.sh`
4. run `test_simulator.py`
5. make your selections:
   - choose local or slurm
   - choose bare-metal
   - choose either serial or parallel
     - serial will run 1 simulation per test, 1 at a time
     - parallel will give you options of how many simulations per test, and how many at a time per test
       - if local was chosen this will use background multiple processes
       - if slurm was chosen this will submit multiple jobs to SLURM
6. wait for results

```
cd /path/to/simulator/basefiles
# edit ./batsim_environment.sh
# make sure you point prefix to /path/to/simulator (don't include basefiles in the path)
source batsim_environment.sh
test_simulator.py
```

2.4.2 Docker works

Read the following list of instructions and then perform the commands below it.

1. create and run a container from your "simulator_compile" image
2. change directories (should already be in the correct directory)
3. edit `basefiles/batsim_environment.sh`
4. source `batsim_environment.sh`
5. run `test_simulator.py`
6. make your selections:
   - choose local or slurm

- choose docker
- serial is the only option here, so the simulations will start immediately

7. wait for results

```
docker run -it --name sim_test simulator_compile:latest
inside docker> cd /home/sim/simulator/basefiles
inside docker> # edit ./batsim_environment.sh  # prefix should be /home/sim/simulator
inside docker> source batsim_environment.sh
inside docker> test_simulator.py
```

2.4.3 CharlieCloud works

Read the following list of instructions and then perform the commands below it.

1. change directories `/path/to/simulator/basefiles`
2. edit `batsim_environment.sh`
3. source `batsim_environment.sh`
4. run `test_simulator.py`
5. make your selections:
   - choose local or slurm
   - choose charliecloud
   - choose either serial or parallel
     - serial will run 1 simulation per test, 1 at a time
     - parallel will give you options of how many simulations per test, and how many at a time per test
       - if local was chosen this will use background multiple processes
       - if slurm was chosen this will submit multiple jobs to SLURM
6. wait for results

```
cd /path/to/simulator/basefiles
# edit ./batsim_environment.sh
# make sure you point prefix to /path/to/simulator (don't include basefiles in the path)
source batsim_environment.sh
test_simulator.py
```

3 Batsim Environment

You always want to make sure you get into your batsim environment first.

To get into your batsim environment:

```
cd /path/to/simulator/basefiles
# make sure prefix is set in batsim_environment.sh
source batsim_environment.sh
```

Once you are in your batsim environment you have a few tools to use:

- **batEnv**
  - Tells you what batsim environment you are in (prefix). This is helpful if you have multiple deployments.
- **batVersion**
  - Tells you what version of simulator you are using. If one has problems it would be helpful to include:
    - batsim version
    - batsched version
    - batVersion
- **batExit**
  - gets you out of the batsim environment.
  - changes your:
    - PATH - removes some added paths
    - LD_LIBRARY_PATH - removes some added paths
    - deactivates your python environment
    - removes the (batsim_env) from your prompt
- **batFile**
  - makes it easier to select a file to pass to myBatchTasks.sh
    - displays your configs directory and allows you to choose the numbered config file to set as $file1 .
  - also helps you select a folder.
  - use `batFile --help` for full usage.
- **batFolder**
  - makes it easier to select a folder.
  - use `batFolder --help` for full usage.
- **batEdit**
  - makes selecting a file easier, as well as editing the file.
  - use `batEdit --help` for full usage.
- **bind_all**

  - will bind certain keys to certain functions while inside the command-line terminal.
    - very helpful bindings for the keyboard. For instance: type `cd` then continue to press 'alt+y' and you will see the history of just your `cd` command.
  - use `bind-all --help` to view all bindings that are made.

- **basefiles scripts**
  - ***myBatchTasks.sh***
    - the main script you will use.
    - use `myBatchTasks.sh --help` to view the full usage.
  - ***progress.sh***
    - a very helpful script to view the progress of running simulations.
    - use `progress.sh --help` to view the full usage.
  - ***aggregate_makespan.py***
    - will aggregate results after simulations are finished.
    - use `aggregate_makespan.py --help` to view the full usage.

4 Config File - In Depth

4.1 Basic Outline

Here you will see what a basic outline of a config file is. It will give you a good overview of what is included in one.

```
        The general format of a config file:

        {       <------------------------------------   Opening curly brace to be proper json

            "Name1":{       <------------------------   The name of an experiment comes first.
                                                        You can have multiple experiments
                                                        in one config file and each will end up
                                                        in it's own folder under the --output
                                                        folder.Notice the opening and closing
                                                        curly brace.  Make sure you put a comma
                                                        after the closing curly brace if you
                                                        plan on having another
                                                        experiment in the same config file


                                                        Json does not allow for comments
                                                        (unfortunately).  You may still want
                    #                          \        comments in your config,
                    # python/shell comment      \       however.  You can use all of these
                                                 \      types of comments and it will get
                    // c/c++ style comment        \     removed before parsing.Be aware that it
                    /* c/c++ block style comment  /     can get difficult to trace down a
                        Comments are fun.        /      simple mistake in your config when
                        This comment is too.    /       many comments are used due
                                               /        to the line numbers being off and
                                                        generally more clutter in your config.
                                                        But comments can make things a lot
                                                        clearer, too.The original and a
                                                        stripped version will be in your
                                                        --output folder.
                    */


                    "input":{    <-------------------   Always make sure you have an input and
                                                        an output in your experiment

                        "node-sweep":{  <------------   It is MOST advisable to always start
                                                        with a node-sweep.  All other sweeps
                                                        can come after this one


                        },
                        "synthetic-workload":{ <-----   Always include either a
                                                        synthetic-workload or a
                                                        grizzly-workload after your sweeps


                        },
                        "option":value,        <-----   Include any options that will affect
                                                        all of the jobs on the outside of any
                                                        sweep or workload


                    },    <--------------------------   Make sure you separate your input
                                                        options with commas, but also
                                                        remember to separate input
                                                        and output with a comma
                    "output":{   <-------------------   Again, always make sure you have an
```

```
                                                  input and output in your experiment

                     "option":value,   <----------  Output is a bit simpler than input.
                                                  Just make sure it is valid json
                     "option":value

                 }


        },     <--------------------------------  This closes the experiment and here
                                                  we have a comma because we included
                                                  another experiment "Name2"
        "Name2":{
            "input":{

                ...  <-------------------------   Make sure you replace this ellipsis
                                                  with at least:
                                                      * a node-sweep
                                                      * a workload
            },
            "output":{

                ...  <-------------------------   You should replace ellipsis with
                                                  at least:
                                                      * "AAE":true | "makespan":true

            }   <-----------------------------    Close output
        }  <----------------------------------    Close "Name2"
    }  <--------------------------------------    Close json
```

4.2 Sweeps

Learn what sweeps are and how to use them here.

4.2.1 Explanation of Sweeps

Here you will learn what Sweeps are.

Sweeps are what we call it when we make a parameterized option. When you start out you will have one job called 'experiment_1'. If you add a sweep that, say, sweeps over how many nodes your simulation will be using, then it will add to how many jobs you have.

Let's say you sweep from 1,000 nodes to 2,000 nodes with a step of 250. Then you will have:

- experiment_1: 1000 nodes
- experiment_2: 1250 nodes
- experiment_3: 1500 nodes
- experiment_4: 1750 nodes
- experiment_5: 2000 nodes

Now, the way sweeps work is that they loop over what is already there. So if we add a failure sweep like SMTBF (**S**ystem **M**ean **T**ime **B**etween **F**ailure) after the node sweep, then it will take the first parameter of the SMTBF sweep and set it to the experiments 1-5 above. But then it will copy those 5 experiments and set the failure parameter to the second parameter of the SMTBF sweep.

Let's say you sweep from a SMTBF of 20,000 seconds to 40,000 seconds with a step of 10,000. Then you will have:

- experiment_1: 1000 nodes SMTBF: 20,000 sec
- experiment_2: 1250 nodes SMTBF: 20,000 sec
- experiment_3: 1500 nodes SMTBF: 20,000 sec
- experiment_4: 1750 nodes SMTBF: 20,000 sec
- experiment_5: 2000 nodes SMTBF: 20,000 sec
- experiment_6: 1000 nodes SMTBF: 30,000 sec
- experiment_7: 1250 nodes SMTBF: 30,000 sec
- experiment_8: 1500 nodes SMTBF: 30,000 sec
- experiment_9: 1750 nodes SMTBF: 30,000 sec
- experiment_10: 2000 nodes SMTBF: 30,000 sec
- experiment_11: 1000 nodes SMTBF: 40,000 sec
- experiment_12: 1250 nodes SMTBF: 40,000 sec
- experiment_13: 1500 nodes SMTBF: 40,000 sec
- experiment_14: 1750 nodes SMTBF: 40,000 sec
- experiment_15: 2000 nodes SMTBF: 40,000 sec

So I hope you can see how the experiments add up quickly.

- We started with 5 node parameters
- We added 3 SMTBF parameters
- This totals 5 * 3 = 15 jobs

If we add another sweep after the SMTBF sweep with 4 parameters that would be 5 * 3 * 4 = 60 jobs

4.2.2 Types of Sweep Functions

Sweeps can parameterize in multiple ways. Here are the methods used:

- **(iMMS)** integer Min Max Step
  - start from the minimum to the maximum (inclusive) with a step (can be negative)

    ```
          "min":0,
          "max":10,
          "step":2
    ```

- **(fMMS)** float Min Max Step
  - same as iMMS except you can use floating point numbers
- **(iR)** integer Range
  - simply a list of integers

```
        "range":[10,20,30,80]
```

- **(fR)** float Range
  - same as iR except for floats
- **(iSR),(fSR)** integer Sticky Range and float Sticky Range
  - just like **iR** and **fR** except it requires the amount of values to equal the amount of jobs made from sweeps before it. Instead of adding any more jobs, it sets the values contained in it to the jobs already there.
  - example:

```
        "node-sweep":{"range":[1000,2000]},  //creates two jobs: experiment_1 and
                                             //experiment_2
        "SMTBF-sweep":{"sticky-range":[20000,30000]}
        // normally with "range" this would
            //set 20,000 to experiment_1(1000 nodes) and experiment_2(2000 nodes) and
            //set 30,000 to experiment_3(1000 nodes) and experiment_4(2000 nodes)
        //sticky-range, however, will
            //set 20,000 to experiment_1(1000 nodes)
            //set 30,000 to experiment_2(2000 nodes)
            //and that's all
        //No experiment_3 or 4. It 'sticks' to what was there before.
```

- **(F)** formula
  - used in conjunction with iR, fR, iSR, fSR, iMMS and fMMS. You can set a formula here with 'i' as your variable. Each number in your min/max/step or range will be passed in as 'i' to your formula and the result will be your number. Makes it easier to read.
  - Example:

```
        "range":[2,3,4],
        "formula":"i*3600"  // will make 2 hours, 3 hours, 4 hours.
                            // easier than 7200 sec,10800 sec,14400 sec
```

  - Example:

```
        "min":1,
        "max":5,
        "step":1,
        "formula":"(10**i)/i"  // 'i' can be used multiple times.
                               // Any python statement can be evaluated here.
```

4.2.3 Current Sweeps Available - (Functions Allowed)

Here are the current sweeps available and the parameterization allowed. All sweep names end in "-sweep"

- **checkpointError** ***(fMMS,fR)***

  - Used in our Application Efficiency tests. It adds/subtracts an error amount to optimal simulated checkpoint intervals

- **checkpoint** ***(iMMS,iR)***
  - The interval to use for simulated checkpoints. This value is an integer, but can also be set to ”optimal”.

- **coreCount** ***(iMMS,iR)***
  - How many cores per node. Currently only supported on fcfs_fast2, easy_bf_fast2, and easy_bf_fast2_holdback algorithms.

- **corePercent** ***(fMMS,fR)***
  - What percent of cores can be filled with 1 node jobs. Currently only supported on fcfs_fast2, easy_bf_fast2, and easy_bf_fast2_holdback algorithms.

- **jobs** ***(iMMS,iR)***
  - How many jobs out of the workload to use.

- **MTTR** ***(fMMS,fR,F)***
  - **M**ean **T**ime **T**o **R**epair. Used in conjunction with failures to set how long a repair lasts. It will come up with random repair times each time a machine goes down based on an exponential distribution.

- **node** ***(fMMS,fR,F)***
  - How many nodes the cluster will have.

- **performance** ***(fMMS,fR)***
  - Will increase/decrease the length of all jobs by this factor (floating point)

- **queueDepth** ***(iMMS,iR)***
  - In conservative_bf algorithm, will only schedule this amount of queued jobs before stopping. This will speed things up considerably.

- **repairTime** ***(iMMS,iR,F)***
  - Similar to MTTR, but, instead of a random MTTR, this will set a fixed repair time for the whole simulation.

- **reservation** ***(None - explained in its own section)***
  - This is used in conjunction with conservative_bf algorithm to simulate reservations. There is a whole syntax to this, so one should look at the documentation for info on it. More info at Section 4.6: Reservations.

- **sharePackingHoldback** ***(iMMS,iR)***
  - When using cores, this will holdback x amount of nodes for sharing jobs. All other nodes will not share jobs. Only used with easy_bf_fast2_holdback algorithm.

- **SMTBF** ***(fMMS,fR,fSR,F)***
  - **S**ystem **M**ean **T**ime **B**etween **F**ailures. Used as the primary source of failures. It will come up with random failure times with an exponential distribution, and will come up with a random machine to have the failure with a normal distribution.
  - has a compute-SMTBF-from-NMTBF option
    - Will treat the values generated from this sweep as NMTBF’s (**N**ode **M**ean **T**ime **B**etween **F**ailure) and will compute the SMTBF from the amount of nodes for that experiment

- **submissionCompression** ***(iMMS,iR,F)***
  - will compress/expand the time between submissions by a factor.

4.3 Workloads

Here you will learn about the mandatory workload keys in a config file.

The following keys will be explained:

- grizzly-workload
- synthetic-workload

4.3.1 grizzly-workload

A grizzly-workload is named based on a certain 'grizzly' cluster at Los Alamos National Lab. It is a 1490 node cluster and a 2018 real workload was acquired from the months of January to November. As long as the file the workload comes from conforms to the same requirements the 2018 workload conforms to, then the grizzly-workload is simply a 'real' workload that has options specific for it. Requirements for your own 'grizzly-workload' are laid out in `.../simulator/basefiles/docs/User/User_Doc_Manual.pdf`

- **Required Options**
  - ***type***
    - the type of profile to use: 'parallel_homogeneous' or 'delay'. With 'parallel_homogenous' run-time of a job is actually in terms of computational work done: flops/second. It just so happens that when `machine-speed` is set to 1 then it translates into time. `delay` deals only with time. Though 'parallel_homogeneous' may seem more complicated, it is recommended since other options and algorithms may use this flops/second functionality, such as using `cores`.
  - ***machine-speed***
    - used with 'parallel_homogeneous'. It is the amount of flops of computation done in 1 second. We highly recommend you set this to 1.
  - ***input***
    - the 'grizzly' or 'grizzly-like' workload file you will use. `sanitized_jobs.csv` is the 2018 workload file we use. Can be an absolute path or the name of a file in `.../simulator/basefiles`
  - ***time***
    - the time interval you would like to use in the workload.
      - example: '03-01-2018:04-01-2018' would do March 1 till April 1.
      - example: ':' would do all of the file. From Jan to November in the 2018 file.
      - example: '06-01-2018:' or ':05-01-2018' From June on or From start to May respectively.
- **Additional Options**
  - ***number-of-jobs***
    - once a time period is chosen with `time`, you may choose how many of those jobs you want with this. Starts from the front of `time` with a positive number. Starts from the back with a negative number. Takes precedence over regular option `number-of-jobs` and the `jobs-sweep`, so it should not be set if using the `jobs-sweep`.
  - ***random-selection***
    - used with `number-of-jobs`, will randomize which jobs are chosen.
      - example: 20 will seed the randomness with 20, making it deterministic
      - example: -1 will seed with time, making it random
  - ***submission-time***

    - The time between submissions and randomness used. If omitted, will use the actual submission time in the `input` file. If set to '0:fixed', all jobs will submit at time zero.
        - syntax: <float>:<exp|fixed>. will use <float> seconds as the mean time ('exp'onential) or the actual time (fixed)
        - syntax: <float1>:<float2>:unif. will use a uniform distribution between <float1> and <float2>

- ***wallclock-limit***
    - the amount of time that a job is able to use. If omitted, will use the actual wallclock-limit from the `input` file.
        - syntax: <float>|'<int>%' a percent will be based off of the runtime of the job.
        - syntax: <string> either 'min:max[:seed]' or 'min%:max%[:seed]' where min:max are floats and min% and max% are '<int>%'. These are random numbers from min to max and an optional seed
        - example: '98%:102%:10' from 98% of runtime to 102% of runtime with a seed of 10

- ***read-time***
    - The amount of time to read in from a simulated checkpoint if checkpointing is turned on. Follows the same syntax as `wallclock-limit`. Mandatory if using checkpointing, but can be set to 0.

- ***dump-time***
    - The amount of time to write out a simulated checkpoint if checkpointing is turned on. Follows the same syntax as `wallclock-limit`. Mandatory if using checkpointing, but can be set to 0.

- ***checkpoint-interval***
    - The amount of time between successive writes of a simulated checkpoint on a per job basis. Follows the same syntax as `wallclock-limit`. Not mandatory since a system-wide simulated checkpoint interval can be set.

- ***resv***
    - Sets what reservation definition to use. Only used if you are simulating reservations of time, and only used with conservative_bf algorithm.

- ***force-creation***
    - Workloads go in a database and will be re-used if they have the right characteristics. If you want to roll the dice again you should force the creation of a new workload.

- ***seed***
    - A seed that can be used on all randomness of the workload creation. Otherwise it will use time, making it random unless individual seed options are used.

- ***index***
    - Will set the index for a workload. Suppose you made a random workload and it was added to the database. You then wanted to run the experiment again but wanted a different roll of the dice for randomness, you could choose `force-creation` or just give it another index. The benefit of using an index is that you could come back to using the same workload as long as the other workload options remained the same.

Regular Options That Effect Workloads

- ***submission-compression***
    - will compress/expand the time between submission of jobs
        - syntax: '<int>%' . below 100% compresses, above 100% expands

- ***reservations-* and *reservation-sweep***
    - will define a reservation. If `resv` is set for the workload then this changes the workload. See the section on reservations. More info at Section 4.6: Reservations.

- ***workload-ids***
    - This option piggy-backs off the `index` idea.
        - Jobs and Runs
            - If you want different options you make multiple jobs in the form of `experiment_#` folders.

            - If those options make batsim use randomness and you want to do some statistics by running batsim multiple times you use 'runs' by setting `avg-makespan` in the `'output':{ }` section of your config.
        - Ids
            - In contrast, if you want to make multiple random **workloads** for statistics, then you use 'ids' which can also be used in tandem with 'runs', though this gets complicated and may not aggregate properly at the time of this writing.TODO
            - syntax: '[<comma seperated range of ids>]', example: '[1,5,8,20]' , yes this is a string
            - syntax: 'min;max;step', example: '5;20;1'
    - ***number-of-jobs***
        - Although `number-of-jobs` in the workload section takes precedence, `number-of-jobs` can be set in the regular options as well. This is purely for the benefit of the `jobs-sweep`.

4.3.2 synthetic-workload

A synthetic-workload is simply a completely, or almost completely, made up workload. There are many random options to give you the workload you require.

The reason we say 'almost completely' made up, is that there are 6 files that characterize different types of workloads based on the grizzly cluster. In fact 'wl2.csv' was made from the same distribution of jobs that is in a particular real grizzly workload.

Whether you use these files or not is completely up to you. We give you the tools to make a workload that suits you in the following list of options.

- **Required Options**
    - ***type***
        - the type of profile to use: `'parallel_homogeneous'` or `'delay'`. With `'parallel_homogenous'` run-time of a job is actually in terms of computational work done: flops/second. It just so happens that when `machine-speed` is set to 1 then it translates into time. `'delay'` deals only with time. Though `'parallel_homogeneous'` may seem more complicated, it is recommended since other options and algorithms may use this flops/second functionality, such as using cores.
    - ***machine-speed***
        - used with `'parallel_homogeneous'`. It is the amount of flops of computation done in 1 second. We highly recommend you set this to 1.
    - ***number-of-jobs***
        - The total number of jobs to make
    - ***number-of-resources***
        - The number of resources that each job will use
            - syntax: '<int>:fixed'
                - all jobs will have a fixed <int> amount of resources
            - syntax: '<int1>:<int2>:unif'
                - jobs will have from <int1> to <int2> uniformally random number of resources
            - syntax: '<float1>:<float2>:norm'
                - jobs will have from <float1> to <float2> normally random number of resources
            - syntax: '<str>:<int>:csv'
                - Will come from csv file at <str>. <str> can be an absolute path or a file in `.../simulator/basefiles`. <int> is the position in each row that holds the number of resources in the file. 0 is the first column.

            - csv files included are from `wl1.csv` to `wl6.csv`. `wl1.csv` starts at all 1 node jobs, `wl2.csv` is mostly 1 node jobs but resembles grizzly workloads in the past. `wl3.csv` is medium sized all the way up to `wl6.csv` which is the entire 1490 cluster on every job.

    - ***duration-time***
        - The length of time each job will use to complete
            - syntax: '<float>:<exp|fixed>'
                - all jobs will have a fixed <float> amount of runtime or an exponentially random runtime with a <float> mean time.
            - syntax: '<float1>:<float2>:unif'
                - jobs will have from <float1> to <float2> uniformally random number of runtime
            - syntax: '<float1>:<float2>:norm'
                - jobs will have from <float1> to <float2> normally random number of runtime
            - syntax: '<str1>:<int>:<str2>:csv'
                - Will come from csv file at <str1>. <str1> can be an absolute path or a file in `.../simulator/basefiles`. <int> is the position in each row that holds the number of resources in the file. 0 is the first column.<str2> is the unit of time that this represents in the file, so 'h|m|s' for hours or minutes or seconds. 'h' should be used with the included files.
                - csv files included are from `wl1.csv` to `wl6.csv`. `wl1.csv` is all 24 hour jobs as their width is only 1 resource. `wl2.csv` is varied but resembles grizzly workloads in the past. `wl3.csv` is medium length all the way up to `wl6.csv` which is entirely 24 hour jobs.

    - ***submission-time***
        - The time between submissions and randomness used. If set to 0:fixed, all jobs will submit at time zero.
            - syntax: '<float>:<exp|fixed>'
                - all jobs will have a fixed <float> amount of time between submissions or an exponentially random time with a <float> mean time between submissions.
            - syntax: '<float1>:<float2>:unif'
                - jobs will have from <float1> to <float2> uniformally random number of time between submissions
            - syntax: '<float1>:<float2>:norm'
                - jobs will have from <float1> to <float2> normally random number of time between submissions

- **Additional Options**
    - ***wallclock-limit***
        - the amount of time that a job is able to use. If omitted, will use -1, which means that it will not be used in Batsim.
            - syntax: <float>|'<int>%' a percent will be based off of the runtime of the job.
            - syntax: <string> either 'min:max[:seed]' or 'min%:max%[:seed]' where min:max are floats and min% and max% are '<int>%'. These are random numbers from min to max and an optional seed
                - example: '98%:102%:10' from 98% of runtime to 102% of runtime with a seed of 10

    - ***read-time***
        - The amount of time (in seconds) to read in from a simulated checkpoint if checkpointing is turned on. Follows the same syntax as wallclock-limit. Mandatory if using checkpointing, but can be set to 0.

    - ***dump-time***
        - The amount of time (in seconds) to write out a simulated checkpoint if checkpointing is turned on. Follows the same syntax as wallclock-limit. Mandatory if using checkpointing, but can be set to 0.

    - ***checkpoint-interval***
        - The amount of time (in seconds) between successive writes of a simulated checkpoint on a per job basis. Follows the same syntax as wallclock-limit. Not mandatory since a system-wide simulated checkpoint interval can be set.

- ***resv***
  - Sets what reservation definition to use. Only used if you are simulating reservations of time, and only used with `conservative_bf` algorithm. More info at Section 4.6: Reservations.
- ***force-creation***
  - Workloads go in a database and will be re-used if they have the right characteristics. If you want to roll the dice again you should force the creation of a new workload.
- ***seed***
  - A seed that can be used on all randomness of the workload creation. Otherwise it will use time, making it random unless individual seed options are used.
- ***index***
  - Will set the index for a workload. Suppose you made a random workload and it was added to the database. You then wanted to run the experiment again but wanted a different roll of the dice for randomness, you could choose `force-creation` or just give it another index. The benefit of using an index is that you could come back to using the same workload as long as the other workload options remained the same.

- **Regular Options That Effect Workloads**
  - ***submission-compression***
    - will compress/expand the time between submission of jobs
      - syntax: '<int>%' . below 100% compresses, above 100% expands
  - ***reservations*** and ***reservation-sweep***
    - will define a reservation. If `resv` is set for the workload then this changes the workload. More info at Section 4.6: Reservations.
  - ***workload-ids***
    - This option piggy-backs off the `index` idea.
      - Jobs and Runs
        - If you want different options you make multiple jobs in the form of `experiment_#` folders.
        - If those options make batsim use randomness and you want to do some statistics by running batsim multiple times you use 'runs' by setting `avg-makespan` in the `'output':{}` section of your config.
      - Ids
        - In contrast, if you want to make multiple random **workloads** for statistics, then you use 'ids' which can also be used in tandem with 'runs', though this gets complicated and may not aggregate properly at the time of this writing.TODO.
        - syntax: '[<comma seperated range of ids>]', example: '[1,5,8,20]' , yes this is a string
        - syntax: 'min;max;step', example: '5;20;1'
  - ***number-of-jobs***
    - Although `number-of-jobs` in the workload section takes precedence, `number-of-jobs` can be set in the regular options as well. This is purely for the benefit of the **jobs-sweep**.

4.4 Regular Options

All other available options are described here.

- **Required Options**
  - ***batsched-policy***
    - Sets which scheduling algorithm to use. Is mandatory.

      - options: `fcfs_fast2 | easy_bf_fast2 | easy_bf_fast2_holdback | easy_bf2 | easy_bf3 | conservative_bf`
      - algorithms discussed in Section 4.5: Algorithms

- **Options That Can Effect The Workload**
  - ***submission-compression***
    - will compress/expand the time between submission of jobs
      - syntax: '<int>%' . below 100% compresses, above 100% expands
  - ***reservations-*** and ***reservation-sweep***
    - will define a reservation. If `resv` is set for the workload then this changes the workload. More info at Section 4.6: Reservations.
  - ***workload-ids***
    - This option piggy-backs off the `index` idea.
      - Jobs and Runs
        - If you want different options you make multiple jobs in the form of `experiment_#` folders.
        - If those options make batsim use randomness and you want to do some statistics by running batsim multiple times you use 'runs' by setting `avg-makespan` in the `'output':{}` section of your config.
      - Ids
        - In contrast, if you want to make multiple random **workloads** for statistics, then you use 'ids' which can also be used in tandem with 'runs', though this gets complicated and may not aggregate properly at the time of this writing.TODO.
        - syntax: '[<comma seperated range of ids>]', example: '[1,5,8,20]' , yes this is a string
        - syntax: 'min;max;step', example: '5;20;1'
  - ***number-of-jobs***
    - Although `number-of-jobs` in the workload section takes precedence, `number-of-jobs` can be set in the regular options as well. This is purely for the benefit of the **jobs-sweep**.

- **Logging Options**
  - KEEP IN MIND THESE CAN TAKE UP A LOT OF HARD DRIVE SPACE, not meant to be used on a large set of simulations
  - ***batsched-log***
    - sets the logging for batsched
      - options: `silent|debug|quiet|info|CCU_INFO|CCU_DEBUG_FIN|CCU_DEBUG|CCU_DEBUG_ALL`
        - **silent** - No logging
        - **debug** - Very verbose logging
        - **quiet** - Very minimal logging
        - **info** - A bit of logging
        - **CCU_INFO** - Minimal logging, tells you important things that are happening. Could use with actual experiments, though not recommended
        - **CCU_DEBUG_FIN** - Debug mode, but more important things. Not recommended to be used with real experiments.
        - **CCU_DEBUG** - Debug mode, very verbose. Not recommended to be used with real experiments.
        - **CCU_DEBUG_ALL** - Debug mode, extremely verbose. Not recommended to be used with real experiments.
  - ***batsim-log***
    - sets the logging for batsim
      - options: `network-only|debug|quiet|info|CCU_INFO|CCU_DEBUG_FIN|CCU_DEBUG|CCU_DEBUG_ALL`
        - **network-only** - only network logging
        - **debug** - Very verbose logging

    - **quiet** - Very minimal logging
    - **CCU_INFO** - Minimal logging, tells you important things that are happening. Could use with actual experiments, though not recommended
    - **CCU_DEBUG_FIN** - Debug mode, but more important things. Not recommended to be used with real experiments.
    - **CCU_DEBUG** - Debug mode, very verbose. Not recommended to be used with real experiments.
    - **CCU_DEBUG_ALL** - Debug mode, extremely verbose. Not recommended to be used with real experiments.

- ***log-b-log***
  - if set to true will log B_LOG files (these would need to be added to the scheduler code)
  - more info on these are in the Development Docs `.../simulator/basefiles/docs/Developer/Developer_Doc_Manual.pdf`

- ***output-svg***
  - Whether to output the schedule with algorithms that use the schedule
    - options: `none | all | short`
      - **'all'** - will output a svg every time an output_svg is encountered (basically every time the schedule changes). ONLY USE FOR SMALL WORKLOADS due to slow-down and Hard Drive space.
      - **'short'** - will output a svg every time there is a short output_svg (happens much less than 'all'). STILL ONLY USE FOR SMALLER WORKLOADS due to slow-down and Hard Drive space
      - **'none'** - will not output the schedule at all. The default.

- ***output-svg-method***
  - What method to output the schedule
    - options: `svg | text | both`
      - **'svg'** - will output the svg files
      - **'text'** - will only output the schedule in text form in the batsched-log which requires it to be on info or CCU_INFO
      - **'both'** - will output svg files and also text in the log

- ***svg-output-start***
  - What output number to start at. If you know 'when' you want to concentrate on, in terms of how many svg's have been output, then set this number and possibly also the `svg-output-end` .

- ***svg-output-end***
  - What output number to end at. If you know 'when' you want to concentrate on, in terms of how many svg's have been output, then set this number and possibly also the `svg-output-start` .

- ***svg-frame-start***
  - What frame number to start at. A frame number is incremented each time make_decisions() is entered.
- ***svg-frame-end***
  - What frame number to end at. A frame number is incremented each time make_decisions() is entered.
- ***svg-time-start***
  - What simulated time to start outputting the schedule.
    - syntax: <float>, in seconds

- ***svg-time-end***
  - What simulated time to end outputting the schedule. -1.0 to go to the end of the simulation, the default.
    - syntax: <float>, in seconds

- ***turn-off-extra-info***
  - Extra info is output to a file called `out_extra_info.csv` . It outputs a new line each time a job is completed. It consists of 'jobs completed','percent done','utilization', schedule metrics, 'utilization', and memory usage.

      - set to true to turn this off. Turning off will render progress.sh useless but may speed things up and will reduce Hard Drive space.

- **Failure Options**
  - ***MTTR***
    - **M**ean **T**ime **T**o **R**epair. Used in conjunction with failures to set how long a repair lasts. It will come up with random repair times each time a machine goes down based on an exponential distribution.
  - ***SMTBF***
    - **S**ystem **M**ean **T**ime **B**etween **F**ailures. Used as the primary source of failures. It will come up with random failure times with an exponential distribution, and will come up with a random machine to have the failure on with a normal distribution.
  - ***calculate-checkpointing***
    - If set to true, computes the optimal simulated checkpointing interval for each job based on read time and dump time and the failure rate
  - ***checkpoint-interval***
    - Sets the system-wide simulated checkpoint interval
      - syntax: <float>
  - ***checkpointError***
    - Used in conjunction with `calculate-checkpointing`. Will increase or decrease the computed optimal checkpoint by the factor given by `checkpointError`.
      - syntax: <float>, above 1.0 is an increase, below 1.0 is a decrease.
  - ***checkpointing-on***
    - if set to true, will turn simulated checkpointing on. Mandatory to do simulated checkpointing.
  - ***fixed-failures***
    - sets failures to be every simulated <float> seconds. Is very useful in debugging.
  - ***queue-policy***
    - What the policy for the queue is when dealing with a re-submitted job. The options are: `FCFS | ORIGINAL-FCFS`
    - Usually the queue is FCFS based on the submit time. ORIGINAL-FCFS would put resubmitted jobs at the front of the queue based on their original submit time.
  - ***reject-jobs-after-nb-repairs***
    - When failures result in machines going down because of a repair time on them, some jobs may not be able to run at all until machines become available. If there are only jobs in the queue that fall into this situation then a mode can be flipped to count how many times a repair is done before any job has executed. Once a job is able to execute, the count is reset. This setting waits <int> number of repairs being done before it gives up and rejects the jobs that are left. '-1' means the jobs will never be rejected in this situation, the default.
      - syntax: <int>
  - ***repair-time***
    - Sets a system-wide repair time in seconds.
      - syntax: <float>
  - ***seed-failures***
    - Will seed any random generators for failures, otherwise time is used.
      - syntax: <int>
  - ***seed-failure-machine***
    - Will seed any random generators for determining which machine should get the failure, otherwise time is used.
      - syntax: <int>

- ***seed-repair-times***
  - Will seed any random generators for repair time, otherwise time is used
    - syntax: <int>

- **Real Checkpointing Options**
  - ***checkpoint-batsim-interval***
    - Will set an interval to do real checkpoints
      - syntax: <string>
        - ”(real|simulated)[:once]:days-HH:MM:SS[:keep]”
          - **’real’** - prepended will interpret the interval to be in real time
          - **’simulated’** - prepended will interpret the interval to be in simulated time
          - **optional :once** - will do one checkpoint and then stop doing any more checkpoints
          - **optional :keep** - will set the amount of checkpoints to keep. `checkpoint-batsim-keep` trumps this
  - ***checkpoint-batsim-keep***
    - How many checkpoints to keep
      - syntax: <int>
  - ***checkpoint-batsim-signal***
    - The signal number to use for signal driven checkpointing.
      - syntax: <int>, You will want to either use SIGUSR1(10), SIGUSR2 (12), or preferably real-time signals from 35-64

- **Speed/Core Options**
  - ***core-percent***
    - Sets the limit on how many cores from a node can be used
      - syntax: <float>
  - ***core-count***
    - Sets the amount of cores each node will have in the platform file and turns on `'--enable-compute-sharing'`
      - syntax: <int>
  - ***share-packing***
    - If set to true, will pack single resource jobs onto one node until that node reaches `core-percent` * available cores
  - ***share-packing-holdback***
    - If set to true, will holdback a certain number of nodes for exclusive share-packing
  - ***speeds***
    - Will set the speed of the cluster in the platform file
      - syntax: <string>, flops per second
        - Where <string> is ’<int>f’.
        - One can use size prefixes in front of ’f’: K(10^3),M(10^6),G(10^9),T(10^{12}),P(10^{15}),E(10^{18})
      - syntax: <string1>,<string2>,...
        - The difference here is that a list of strings is given, one for each pstate you use. pstates will be explained in both User and Developer Docs.

- **ALL OTHER OPTIONS**
  - ***copy***
    - The amount of copies the ending workload will have, along with submission time optional options. This can be used to double up a workload when you double up the amount of nodes the cluster has. This operates at the Batsim level, and not during workload creation.

```
format:
<#copies>[:(+|-):#:(fixed|#:unif:(single|each-copy|all)[:<seed#>] ])'
    or
<#copies>[:=:#(fixed|((exp|:#:unif)[:<seed#>]) ]'
So you can just do number of copies, or
'=':
   you can copy and set the submission time of the copy as an
   exponential,uniform,or fixed amount with '=', or
'+|-':
   you can add a submission time to add some jitter. This submission time
   is either added or subtracted with (+|-)
   This time can be a fixed number followed by :fixed or uniform random
   number between 2 numbers
   If random:
     you need to specify the second number with :#:unif:
     you need to specify:  'single','each-copy',or 'all'
      'single' random number, single random number for 'each-copy', or
      random number for 'all'
2 copies here means if there are 10 jobs to start with, there will be
20 total after the operation.

 Examples:
                       '2'    - 2 copies no alteration in submission times
             '2:=:100:exp'    - 2 copies with 1 having original submission
                                times, 1 having exponential random with a
                                mean rate of 100 seconds.
             '2:=:0:fixed'    - 2 copies with 1 having original submission
                                times, 1 having fixed time of 0
       '2:=:20:40:unif:30'    - 2 copies with 1 having original submission
                                times, 1 having uniform random between 20
                                and 40 seconds. Use 30 as seed.
            '2:+:10:fixed'    - 2 copies, add 10 seconds fixed jitter to
                                submission times
            '2:-:10:fixed'    - 2 copies, subtract 10 seconds fixed jitter
                                from submission times
    '2:+:5:10:unif:single'    - 2 copies, get one random number between 5
                                and 10 and add it to all copied submission
                                times
    '3:+:5:10:unif:all:20'    - 3 copies, get random numbers between 5 and
                                10 for all jobs of all copies, add it to
                                submission times and seed the random
                                generator with 20
 '3:+:5:10:unif:each-copy'    - 3 copies, get one random number between 5
                                and 10 add it to all submission times
                                of first copy then get another random number
                                between 5 and 10 and add it to all sub
                                times of second copy

```

- ***submission-time-after***
  - This dictates the time between submissions and what kind of randomness. It happens AFTER the copy operation and after sorting the jobs based on submission time. This operates at the Batsim level, and not during workload creation.

```
format:
'<#:(fixed[:#])|(exp|#:unif)[:(#|s[:#]])'
   or
'shuffle[:#]'

It is applied after sorting the current workload by submit time and after
applying the copy option
If zero is used for a float,combined with ":fixed" then all jobs will
start at time zero.
If omitted, the original submission times will be used, be that grizzly
produced or synthetically produced

exp:    This will be exponentially distributed, random values with mean
        time between submissions to be FLOAT.
fixed:  All jobs will have this time between them unless zero is used
        for a FLOAT.
unif:   This will be uniform, random values from min:max
s:      Used after the random types (exp|fixed|unif) to specify you want
        the job's submit times shuffled after.
shuffle: Will simply shuffle around the submit times amongst the jobs.
#:      a seed can be put on the end of the string to use for deterministic
        behavior
ex:
       '--submission-time-after "200.0:exp:s"'
       '--submission-time-after "100.0:fixed"'
       '--submission-time-after "0.0:fixed"'
       '--submission-time-after "0:200.0:unif"'
       '--submission-time-after "200.0:exp:10"'  <-- 10 is the seed
       '--submission-time-after "0:200.0:unif:20"' <-- 20 is the seed
       '--submission-time-after "shuffle:20" <-- 20 is the seed

```

- ***submission-time-before***
  - This is the same as `submission-time-after` except it happens BEFORE the copy operation and before sorting the jobs based on submission time. Both `submission-time-before` and `submission-time-after` can be used or either can be used on their own.
- ***performance-factor***
  - Will increase/decrease the length of all jobs by this factor (floating point). This operates at the Batsim level and not during workload creation.
    - syntax: <float>, above 1.0 will increase, below 1.0 will decrease
- ***queue-depth***
  - The amount of items in the queue that will be scheduled
    - only used in conservative_bf

        - A lower amount will improve performance of the scheduler and thus the simulation but changes scheduling decisions and, so, gives different results
        - (-1) sets it to all items being scheduled, the default

- ***reservations-start***
    - Meant for monte-carlo with reservations, staggering their start time.
        - syntax: <string>
        - <string> is string in following format:

```
'<order#>:<-|+><#seconds>'

where order# is the order (starting at 0) in the reservation array
as described in your config file
where you (must) choose -(negative,behind) or +(positive,ahead)
where you specify the amount of seconds forward or backward

example_1: --reservations-start '0:+5'
              start the reservations with order# 0, 5 seconds ahead
example_2: --reservations-start '1:-2000'
              start the reservations with order# 1, 2000 seconds behind
example_3: --reservations-start '0:+5 , 1:-2000'
              only one invocation of this flag is allowed but values for
              different order #s can be acheived with a comma. spaces
              are allowed for easier viewing.

```

- ***test-suite***
    - If set to true, will assume the folder structure has an umbrella folder to it, where multiple configs were being used and so multiple base folders are used under the umbrella folder.
        - This affects where the `current_progress.log` file is kept.
        - This file keeps track of which simulations are finished and which successfully output a `post_out_jobs.csv` file. This helps the test-suite determine what simulations have finished and whether to go on to the next step or not.
        - If an umbrella folder is used then `current_progress.log` is located one folder up from its base folder. Otherwise it is located in its base folder.

4.5 Algorithms

While failures, simulated checkpointing, and real checkpointing should work with all algorithms, at the current moment only fcfs_fast2, easy_bf2, and easy_bf3 have had any rigorous testing.

It should be noted that real checkpointing does not produce the exact results as if it wasn't started from a checkpoint. The way the message passing works prevented this. It has been shown to be statistically the same with metrics like avg waiting time and makespan, however.

4.5.1 fcfs_fast2

fcfs_fast2 is a First Come First Serve basic algorithm. It may be basic, but it is also a good starting point. Jobs come in at the 'subtime', start when they are first in the queue and resources are available, and then run for the cpu/delay time. This algorithm is also equipped with some logic for 'cores' including the holding back of nodes for 1 core jobs. If you find the logic of cores useful you may be interested in easy_bf_fast2 and easy_bf_fast2_holdback.

4.5.2 easy_bf2

This is a basic backfilling algorithm, however we have found it to be slightly different than a 'normal' easy backfilling algorithm. In addition, it uses the 'schedule' class to determine what nodes the priority job runs on. The schedule class becomes quite slow with a large queue so that should be considered. A good alternative to this algorithm is the easy_bf3 algorithm which we designed to be in step with how most people think about easy backfilling algorithms, as well as skipping the use of the 'schedule' class. This speeds the algorithm up quite a bit, but it does not determine what nodes the priority job is going to run on in advance.

4.5.3 easy_bf3

We designed this algorithm to be in step with how most people think about easy backfilling algorithms, as well as skipping the use of the 'schedule' class. This speeds the algorithm up quite a bit, but it does not determine what nodes the priority job is going to run on in advance.

4.5.4 conservative_bf

This algorithm is the only algorithm that you can use reservations with at this point in time. This is because this algorithm uses the 'schedule' class and schedules out the whole queue. This makes it ideal for adding the reservation features. It also makes the algorithm our slowest one, by far. Something that can speed it up is using the `'queue-depth'` option. This option will limit how much of the queue gets scheduled before making decisions.

4.5.5 easy_bf_fast2

This algorithm is a bit different than easy_bf2. It does not follow your typical easy backfill algorithm. But it has the 'cores' functionality that 'fcfs_fast2' has with some version of backfilling to boot.

4.5.6 easy_bf_fast2_holdback

This algorithm is the same as easy_bf_fast2 just with the ability to holdback nodes for 'cores'.

4.6 Reservations: All Things About Reservations

Reservations are only compatible with the `"batsched-policy":"conservative_bf"`.

- They are blocks of time that appear like jobs, and actually run like a job with purpose: 'reservation' as opposed to 'job'.
- But their priority is higher than any other job.
- If there are other jobs running on the machines that the reservation takes during the time the reservation takes, then they are killed.
- Another distinction is that the reservation can specify which machines it runs on, whereas a job does not.

4.6.1 Reservations

Reservations start with a `'reservations-<name>':{}` section in the config file. You are defining a set of reservations with this key, named '<name>'.

There are two places you will potentially use this <name>:

- You can include it in a `'grizzly-workload':{}` or `'synthetic-workload':{}` item.
  - You do this using the `'resv'` property within either of those workload items.
- The other place to use the <name> is in a `'reservation-sweep':{}` to specify which set of reservations the sweep applies to.
  - You do this using the `'name'` property within the `'reservation-sweep':{}`.

The first thing to start a `'reservations-<name>':{}` section is the `'reservations-array'`. This is mandatory and includes an array of reservation types.

Here is an example array inside of a definition named 'test' of just 1 reservation type. Everything will be explained below:

```
"reservations-test":
{
    "reservations-array":
    [
      {
        "type":"parallel_homogeneous",
        "machines":
        {
            "prefix":"a",
            "machine-speed":1,
            "total-resources":"0-1489",
            "interval":"0-1489"
        },
        "subdivisions":128,
        "subdivisions-unit":"1month 0days 00:00:00",
        "repeat-every":"1months 0days 00:00:00",
        "time":"09:00:00",
        "start":"1months 0days 12:00:00",
        "submit-before-start":"0months 7days 00:00:00",
        "submit":-1,
        "count":30
      }
    ]
}
```

So let's define each part of this reservation:

- **Time syntax:**
  - <string> in format: `"[<int>month[s]][<int>day[s]] HH:MM:SS"`
    - **#month** - optionally the amount of months (simply the amount of seconds in 30 days * #months). Include an 's' at the end of month if you wish.
    - **#day** - optionally the amount of days (simply the amount of seconds in #days). Include an 's' at the end of day if you wish.

    - **HH** - mandatory, the amount of Hours. Must be greater than or equal to two digits. Can set to 00 for zero.
    - **MM** - mandatory, the amount of Minutes. Must be greater than or equal to two digits. Can set to 00 for zero.
    - **SS** - mandatory, the amount of Seconds. Must be greater than or equal to two digits. Can set to 00 for zero.

- **Required Fields:**
  - ***type*** (line 6)
    - <str>. The profile type of this job/reservation `'parallel_homogeneous' | 'delay'`. This is the same option for workloads. Keep in mind, if you set it to parallel_homogenous and the machine speed goes above/below what is set in this `'machines':{}` section, the reservation will go faster/slower than the `'time'` set.
  - ***machines*** (line 7)
    - <json>. The machines this reservation will use. This gets its own section below, in order to properly explain it.
  - ***start*** (line 18)
    - <str>. The time to start the reservation. Look at the Time Syntax at the top.
  - ***time*** (line 17)
    - <str>. The duration of the reservation. Look at the Time Syntax at the top.
  - ***count*** (line 21)
    - <int>. The amount of these reservations. Should be 1 in a lot of cases, but pertains to certain `'machines':{}` configurations, and also with `'repeat-every'` field.

- **Optional Fields:**
  - ***submit-before-start*** (line 19)
    - <str>. required if `'submit'` is omitted. The amount of time to submit the reservation before the start of the reservation. Look at the Time Syntax at the top. This is a convenience field in place of using `'submit'` and trumps this field, as well. It also allows for keeping up with different start times with `'repeat-every'`.
  - ***submit*** (line 20)
    - <str>. required if `'submit-before-start'` is omitted. The absolute time at which to submit the reservation. Look at the Time Syntax at the top. You can use -1 to signify submit time of zero.
  - ***repeat-every*** (line 16)
    - <str>. The amount of time before repeating the reservation. Look at the Time Syntax at the top. Used in conjunction with `'count'`.
  - ***subdivisions*** (line 14)
    - <int>. The amount of subdivisions to divide the machines into. Imagine a 100 node reservation. If we have 2 subdivisions then there will be 2 reservations of 50 nodes each. One will start at `'start'` and the other one will start at `'subdivisions-unit'`/2.
  - ***subdivisions-unit*** (line 15)
    - <str>. required if `'subdivisions'` used. The amount of time to complete all subdivisions in, spaced evenly. Look at the Time Syntax at the top. If the unit is 1 hour and we have 2 subdivisions, one will start at time `'start'` and the other at 30 minutes after that. With 4 subdivisions that would be `'start'`, 15 min, 30 min, 45 min.

4.6.1.1 machines

The `'machines':{}` section determines what machines the reservations will occur on.

- Without the random aspects, the machines are static and will be applied to all reservations.
- With the random aspects, the machines will be different, or at least randomly chosen, for the reservations.
- The random aspects work with the `'count'` field of the reservation type.

- **IntervalSet syntax**
  - <str> in format: ”#-# # #-#”. Intervals are produced with ”<min>-<max>” syntax. Numbers and intervals are seperated with a space.
  - examples:
    - ”1” - a single number. Gives you [1]
    - ”1-5” - a single interval. Gives you [1,2,3,4,5]
    - ”1-5 8-10 14 21-25 100” - multiple intervals and numbers. Gives you [1,2,3,4,5,8,9,10,14,21,22,23,24,25,100]

- ***prefix***
  - <str> required. The prefix of the machine name. This actually isn’t quite used yet, but is included as we use forward-thinking designs. Use ”a” for now. TODO

- ***machine-speed***
  - <int> required. The machine-speed of the machine(s). This will translate the times given in the reservation definition to the amount of flops of work when using `'type':'parallel_homogeneous'`.

- ***total-resources***
  - <str> required. This is what resources we are starting with when talking about random selections of machines. The string is an intervalset, with its syntax above.

- ***interval***
  - <str> required if `'resources'` omitted. These are the actual machines that the reservation will use. Shows up as `'alloc'` for the machines, and as `'res'` for the amount of machines in the workload json file. The string is an intervalset, with its syntax above.

    ```
    or
    ```

  - ”random:unif [options]” - Will randomize the interval based on what resources are available given in `'total-resources'` and the following options:
    - **res-number** - The amount of resources to choose for each random interval. Can be a ”min,max” separated by a comma, in which case a random number between min-max will be chosen. Can be an intervalset inside brackets, in which case a random number in the intervalset will be chosen.
    - **different-res-numbers** - The amount of different resource numbers. Can be a ”min,max” separated by a comma. Can be an intervalset inside brackets. When set to -1 will do all of the choices in **”res-number”**.
    - **different-intervals** - For each resource number, how many different intervals. Can be a ”min,max” separated by a comma. Can be an intervalset inside brackets. When set to -1 will make each one different.
    - **Examples:**
      - **With each example assume** `'total-resources':"0-99"` and `'count':5`

        - `"random:unif res-number=5 different-intervals=1"`
            - will make one interval with 5 randomly chosen machines out of 0-99. For instance [10,20,55,72,85] for all 5 reservations
        - `"random:unif res-number=5 different-intervals=2"`
            - will make 2 intervals with 5 randomly chosen machines out of 0-99. For instance [10,20,55,72,85] and [30,45,50,60,99]. It will assign the first interval to reservations 1-2 and the second to reservations 3-5.
        - `"random:unif res-number=5 different-intervals=-1"`
            - will make 5 intervals with 5 randomly chosen machines out of 0-99. For instance [10,20,55,72,85] ,[30,45,50,60,99],[1,5,10,48,97],[8,9,22,24,63], and [33,76,82,93,99]. It will assign intervals 1-5 to reservations 1-5 respectively.
        - `"random:unif res-number=5,8 different-intervals=1`
            - will make 1 interval of 5-8 randomly chosen machines out of 0-99. For instance, if 6 is randomly chosen for the resource number, we can have [10,20,30,40,50,86] and this will be applied to all 5 reservations.
        - `"random:unif res-number=[5-8 10] different-intervals=1`
            - will make 1 interval of 5,6,7,8 or 10 randomly chosen machines out of 0-99. For instance, if 6 is randomly chosen for the resource number, we can have [10,20,30,40,50,86] and this will be applied to all 5 reservations.
        - `"random:unif res-number=5,8 different-intervals=2`
            - will make 2 intervals of 5-8 randomly chosen machines out of 0-99. For instance, if 6 is randomly chosen for the resource number, we can have [10,20,30,40,50,86] and [11,22,33,44,55,66]. Interval 1 will be applied to reservations 1-2 and interval 2 will be applied to reservations 3-5.
        - `"random:unif res-number=5,8 different-res-numbers=2 different-intervals=2`
            - will choose 2 resource numbers between 5-8. For each resource number ,'res', 2 intervals will be made from 'res' number of machines out of 0-99.
            - Say 5 and 7 are chosen as resource numbers.
            - For resources = 5, 2 different intervals of 5 machines each are chosen at random from 0-99.
            - For resources = 7, 2 different intervals of 7 machines each are chosen at random from 0-99.
            - For instance, [10,20,30,40,50],[15,25,35,45,55],[10,20,30,40,50,60,70],[15,25,35,45,55,65,75],[15,25,35,45,55,65,75] are chosen for reservations 1-5. 2 size 5, 3 size 7, with only 1 repeat.

- ***resources***
    - <int> required if `'interval'` omitted. These are the amount of machines used, but not specific machines. The reservation will try not to kill any jobs, but still has the priority. Shows up as `'res'` in the workload json file.

      `or`

    - <str> "random:unif [options]" - Will randomize the resources based on what resources are available given in `'total-resources'` and the following options:
        - **res-number** - The amount of resources. Can be a "min,max" separated by a comma. Can be an intervalset inside brackets.
        - **different-res-numbers** - The amount of different resource numbers. Can be a "min,max" separated by a comma.Can be an intervalset inside brackets. When set to -1 will do all of the choices in **"res-number"**.
        - **Examples:**
            - **With each example assume** `'total-resources':"0-99"` and `'count':5`
            - `"random:unif res-number=5"`
                - will make one in resource number 5 for all 5 reservations.
            - `"random:unif res-number=5,8 different-res-numbers=2`
                - will make 2 resource numbers, randomly chosen from 5-8. It will assign the first resource number to reservations 1-2, and the second to reservations 3-5.
            - `"random:unif res-number=[5-8 10] different-res-numbers=2"`

    - will make 2 resource numbers, randomly chosen from 5,6,7,8,10. It will assign the first resource number to reservations 1-2, and the second to reservations 3-5.

- `"random:unif res-number=[1-4] different-res-numbers=-1`
    - will make 4 resource numbers because the choices length is 4: 1,2,3,4. It will assign resource numbers 1-4 to reservations 1-4, respectively, and reservation 5 will get 4 resources as well.

- `"random:unif res-number=[1-4] different-res-numbers=[1-3]`
    - will make a random choice on how many different resource numbers to choose, from 1-3, let's call that 'random1'. It will choose 'random1' numbers from 1,2,3,4 at random, let's call these 'random2'. It will distribute those to reservations 1-5 with any repeats happening at the end.

4.6.2 Reservation-Sweeps

Reservation sweeps apply to an already defined reservation type, or types.

Let’s look at an example to let you dive right in:

```
      "reservation-sweep":{
          "name":"test",

          "reservations-array":[
            { "{-3} start":"[1;4;1]days [12]:[30,30,00,00]:[00]",
              "{-2} time":"[1;3;1]:[30]:[00]",
              "{-1} time":"[00]:[30,60,90]:[00]",
              //the ones above this are called 'before-the-base'

              "subdivisions":"[2,4,8]", //this is the base sweep
              "subdivisions-unit":"[1]month [00]:[00]:[00]", //this is also part
                                                              //of the base sweep

              "{1} subdivisions-unit":"[7,14,21]days [00]:[00]:[00]", //this is a multiplier


              //the ones below are called 'after-the-base'
              "{+1} subdivisions":[10],
              "{+1} subdivisions-unit":"[2]months [00]:[00]:[00]",
              "{+2} subdivisions-unit":"[3]months [00]:[00]:[00]"

            }

          ]
      }

```

This sweep starts out with the `'name'` of the definition this sweep applies to. It then starts the `'reservations-array'`.

Each json object in this array maps onto the reservation type in the `'reservations-array'` of the reservation definition. If you define two types of reservations in your definition, then you should have two reservation sweep json objects in your `'reservations-array'` in your `'reservation-sweep':{}` section.

The sweep above goes with the following reservation definition:

```
    "reservations-test":
    {
        "reservations-array":
        [
          {
            "type":"parallel_homogeneous",
            "machines":
            {
                "prefix":"a",
                "machine-speed":1,
                "total-resources":"0-1489",
                "interval":"0-1489"
            },
            "subdivisions":128,
            "subdivisions-unit":"1month 0days 00:00:00",
            "repeat-every":"1months 0days 00:00:00",
            "time":"09:00:00",
            "start":"1months 0days 12:00:00",
            "submit-before-start":"0months 7days 00:00:00",
            "submit":-1,
            "count":30
          }
        ]
    }

```

So the main thing to learn in our sweeps are:

- What jobs are being modified
- What jobs are being added
- In what order are jobs being modified and added
- The number of changes
- The syntax for sweeping

We will continue to show the sweep so that you won't have to look back too far. So here it is again:

```
    "reservation-sweep":{
        "name":"test",

        "reservations-array":[
          { "{-3} start":"[1;4;1]days [12]:[30,30,00,00]:[00]",
            "{-2} time":"[1;3;1]:[30]:[00]",
            "{-1} time":"[00]:[30,60,90]:[00]",
            //the ones above this are called 'before-the-base'

            "subdivisions":"[2,4,8]", //this is the base sweep
            "subdivisions-unit":"[1]month [00]:[00]:[00]", //this is also part
                                                           //of the base sweep

            "{1} subdivisions-unit":"[7,14,21]days [00]:[00]:[00]", //this is a multiplier


            //the ones below are called 'after-the-base'
            "{+1} subdivisions":[10],
            "{+1} subdivisions-unit":"[2]months [00]:[00]:[00]",
            "{+2} subdivisions-unit":"[3]months [00]:[00]:[00]"

          }

        ]
    }

```

The first thing to note is that all the options for the sweep are just the options that are used to define a reservation type in the first place.

But you'll notice that there are *curly braces* with numbers inside for some of the **keys** and *brackets* in some of the **values**.

- The curly braces have to do with the order of the jobs as well as the grouping of multipliers
- The brackets have to do with lists of values and min;max;step

I think it's easiest to start with the base of the sweep on line 10. you see `'subdivisions':"[2,4,8]`.

- No curly braces in the key says that this is the base.
- brackets in the value are mandatory, and a list is provided, as evidenced by the commas.
- If only the base were provided then this would:
  - take all the jobs already made from previous sweeps
  - apply the reservation definition 'test' to all of the jobs, but substitute `'subdivisions':2` for all those original jobs (the original 128 is not used at all in this case)
  - copy all of those original jobs, but now substitute `'subdivisions':4`
  - copy all of those original jobs, but now substitute `'subdivisions':8`
  - So if we had 10 jobs originally, we would have 30 now

You might be observant and question the other base property `'subdivisions-unit'` and ask why that doesn't do anything. Well it does. I just wanted to complete at least 1 **key**:**value** train of thought.

For the 30 jobs that are made from the `'subdivisions'` property, add on to it these details:

- In the base sweep we see we have 3 values so far: [2,4,8]
- All other **key**:**values** in the base should have 3 values to them as well, unless you use just 1 value. If you use 1 value, it will just copy that value (3 times in this case).
- So what will happen with `'subdivisions-unit':"[1]month [00]:[00]:[00]"` is the following:
  - It sees [1]month but the largest expansion in the base has 3 values, so it is really [1,1,1]month
  - It sees [00] for hours, but the largest expansion in the base has 3 values, so it is really [00,00,00]:
  - It sees [00] for minutes, but the largest expansion in the base has 3 values, so it is really[00,00,00]:
  - It sees [00] for seconds, but the largest expansion in the base has 3 values, so it is really[00,00,00]:
  - For all 10 jobs that has `'subdivisions':2` those will have `'subdivisions-unit':"1month 00:00:00"`
  - For all 10 jobs that has `'subdivisions':4` those will have `'subdivisions-unit':"1month 00:00:00"`
  - For all 10 jobs that has `'subdivisions':8` those will have `'subdivisions-unit':"1month 00:00:00"`
- suppose we had `'subdivisions-unit':"[1,2]month [00]:[00]:[00]"`
  - this would error as there are two values for month being mapped onto the largest expansion in the base of 3 values.
- suppose we had `'subdivisions-unit':"[1,2,3]month [6,12,18]:[00]:[00]`
  - this would be fine:
    - `'subdivisions':2` would have `'subdivisions-unit':"1month 06:00:00"`
    - `'subdivisions':4` would have `'subdivisions-unit':"2month 12:00:00"`
    - `'subdivisions':8` would have `'subdivisions-unit':"3month 18:00:00"`
- suppose we had `'subdivisions-unit':"[1,2,3,4]month [00]:[00]:[00]"`
  - this would error, but not because of `'subdivisions-unit'`, but because of `'subdivisions'`.
  - `'subdivisions'` would have 3 values being mapped onto the largest expansion in the base of 4 values.

- if `'subdivisions'` did have the correct amount of values, say `'subdivisions':[2,4,8,16]`, then instead of 30 jobs, we would have 40 jobs.

’expansion’

You might’ve been aware of our use of the word ’expansion’ in the base explanation. An expansion is just a list or a min;max;step which ’expands’ into a list.

For instance, this: `'subdivisions-unit':"[1,2,3]month [6,12,18]:[00]:[00]"`
is the same as this: `'subdivisions-unit':"[1;3;1]month [6;18;6]:[00]:[00]"`

’curly braces’

So What about the curly braces? Well if they have just a number inside they are multipliers and the numbers are used to group them.

So let’s look at our sweep again:

```
 1        "reservation-sweep":{
 2            "name":"test",
 3
 4            "reservations-array":[
 5              { "{-3} start":"[1;4;1]days [12]:[30,30,00,00]:[00]",
 6                "{-2} time":"[1;3;1]:[30]:[00]",
 7                "{-1} time":"[00]:[30,60,90]:[00]",
 8                //the ones above this are called 'before-the-base'
 9
10                "subdivisions":"[2,4,8]", //this is the base sweep
11                "subdivisions-unit":"[1]month [00]:[00]:[00]", //this is also part
12                                                               //of the base sweep
13
14                "{1} subdivisions-unit":"[7,14,21]days [00]:[00]:[00]", //this is a multiplier
15
16
17                //the ones below are called 'after-the-base'
18                "{+1} subdivisions":[10],
19                "{+1} subdivisions-unit":"[2]months [00]:[00]:[00]",
20                "{+2} subdivisions-unit":"[3]months [00]:[00]:[00]"
21
22              }
23
24            ]
25        }
26
```

Line 14 shows us a multiplier and it’s grouping is 1. What a multiplier does is it takes all the jobs that came before it and applies it’s changes as new jobs. So here, if 10 jobs resulted from previous sweeps, we would have 30 jobs from the base part of the sweep. From the multiplier with grouping ’1’ we would have the following:

- To start we have:
  - we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':1month 00:00:00`
  - we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':1month 00:00:00`
  - we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':1month 00:00:00`
- From line 14 we have:
  - an additional 30 jobs with:
    - we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':7days 00:00:00`

        - we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':7days 00:00:00`
        - we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':7days 00:00:00`
    - an additional 30 jobs with:
        - we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':14days 00:00:00`
        - we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':14days 00:00:00`
        - we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':14days 00:00:00`
    - an additional 30 jobs with:
        - we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':21days 00:00:00`
        - we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':21days 00:00:00`
        - we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':21days 00:00:00`

Now imagine we had `'{1} time':'[1,2,3]:[00]:[00]` on line 15. This is fine because the largest expansion of multiplier grouping '1' is 3. It would do the following:

- To start we have:
    - we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':1month 00:00:00` and `'time': 09:00:00`
    - we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':1month 00:00:00` and `'time': 09:00:00`
    - we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':1month 00:00:00` and `'time': 09:00:00`
- From line 14 we have:
    - an additional 30 jobs with:
        - we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':7days 00:00:00` and `'time': 01:00:00`
        - we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':7days 00:00:00` and `'time': 02:00:00`
        - we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':7days 00:00:00` and `'time': 03:00:00`
    - an additional 30 jobs with:
        - we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':14days 00:00:00` and `'time': 01:00:00`
        - we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':14days 00:00:00` and `'time': 02:00:00`
        - we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':14days 00:00:00` and `'time': 03:00:00`
    - an additional 30 jobs with:

- we have 10 jobs with `'subdivisions':2` and `'subdivisions-unit':21days 00:00:00` and `'time': 01:00:00`
- we have 10 jobs with `'subdivisions':4` and `'subdivisions-unit':21days 00:00:00` and `'time': 02:00:00`
- we have 10 jobs with `'subdivisions':8` and `'subdivisions-unit':21days 00:00:00` and `'time': 03:00:00`

So you'll notice having another property with grouping '1' will not multiply the amount of jobs more than what the multiplier already does. So at this point we have 120 jobs. If we add another multiplier grouping the property to multiplier '2' then it will multiply those properties to what is there, just like what multiplier '1' did after the base.

'non-multiplier groupings'

So now we will discuss 'before-the-base' and 'after-the-base'. Let's queue up the config file again.

```
      "reservation-sweep":{
          "name":"test",

          "reservations-array":[
            { "{-3} start":"[1;4;1]days [12]:[30,30,00,00]:[00]",
              "{-2} time":"[1;3;1]:[30]:[00]",
              "{-1} time":"[00]:[30,60,90]:[00]",
              //the ones above this are called 'before-the-base'

              "subdivisions":"[2,4,8]", //this is the base sweep
              "subdivisions-unit":"[1]month [00]:[00]:[00]", //this is also part
                                                             //of the base sweep

              "{1} subdivisions-unit":"[7,14,21]days [00]:[00]:[00]", //this is a multiplier


              //the ones below are called 'after-the-base'
              "{+1} subdivisions":[10],
              "{+1} subdivisions-unit":"[2]months [00]:[00]:[00]",
              "{+2} subdivisions-unit":"[3]months [00]:[00]:[00]"

            }

          ]
      }

```

Let's continue from the example we were using. If you had all the 120 jobs but you wanted to just add one job to the mix you could decide if you want that job being run **before** or **after** the other 120 jobs. Making a non-multiplier with a '-' would put the job **before** the 120 jobs. If you use a '-1' then all properties with -1 will be applied to the 1 job before the 120. If you want another job even before that, then group those properties with a '-2'.

The same logic is used for putting the job **after** the 120 jobs. If you make a job group with '+1', then it will come after the 120 jobs. If you want to have another job come after that one, then use '+2' to group the properties.

And that is the completion of the reservations explanation.

5 Command Reference

5.1 Batsim Environment

```
user >   cd ./simulator/basefiles
user >   source batsim_environment.sh
(batsim_env) user >
```

5.1.1 batEnv

```
(batsim_env) user >   batEnv
/path/to/simulator
```

5.1.2 batVersion

```
(batsim_env) user >   batVersion
1.0.6
```

5.1.3 batExit

```
(batsim_env) user >   batExit
user >
```

5.1.4 batFile

```
(batsim_env) user >  batFile --help

batFile                     batFile can be invoked after sourcing
                            $prefix/basefiles/batsim_environment.sh
It will help you choose a config file in your $prefix/configs and set it to file1
It will also help with setting folder1

Usage:
batFile [-NF] [ls options]

  -NF                       No Folder, will not ask you for a folder after asking
                            for the file

  -[ls options]             Normally 'ls' is invoked to read your $prefix/configs
                          folder without any options.
                          You may find it helpful to use options to 'ls' such as: sort by
                          time, reverse etc...
                          Just pass the ls options you would normally want to use to
                          batFile and you should be fine.
                          I suggest using batFile like so:
                          batFile -ltr
-----------------------------------------------------------------------------------
post-running                After running this script you will see the last myBatchTasks
                          command you entered
                          It checks your ~/.bash_history file for this.  If all that
 ↪   shows
                          up is:
                          "myBatchTasks -f ${file1} -o ${folder1} "
                          Then you may need to flush your history with "history -w"
                          I leave this to the user as you may not want this flushed
```

```
(bat_env) user >  batFile -ltr  #newest configs on the bottom
  1  total 100
  2  -rw-r--r-- 1 user users  1254 Jul 21  2023 my_long_named_config.config
  3  -rw-r--r-- 1 user users  3369 Jul 31  2023 my_other_long_named_config.config
  4  -rw-r--r-- 1 user users  2651 Aug  4  2023 MTTR_from_2h_to_4h_step_0.5.config
Enter a choice (0 to exit): 2
Use a dot to exit: folder1 = my_long_named_config
file1=my_long_named_config.config
(bat_env) user >  myBatchTasks.sh -f ${file1} -o ${folder1} -m bare-metal -p tasks -t 7
```

5.1.5 batFolder

```
(bat_env) user >  batFolder --help
batFolder                   batFolder can be invoked after sourcing
                            $prefix/basefiles/batsim_environment.sh
                            It will help you choose a folder in your
                            $prefix/experiments and set it to folder1
Usage:
batFolder [-#] [-i <path | -d <bookmark>] [ls options]

  -#                        Normally sets the chosen folder to folder1 but you can choose
                            folder1-folder9 with -1,-2,...,-9 flag

  -i <path>                 Normally $prefix/experiments is used, but if you have another
                            folder to use set it here. Relative paths are not recommended.

  -d <bookmark>          If using dirB  will use the bookmark specified as the folder
                            *tab-completion works with this option

  -[ls options]          Normally 'ls' is invoked to read your $prefix/experiments
                            folder without any options.  You may find it helpful to use
                            options to 'ls' such as: sort by time, reverse etc...
                            Just pass the ls options you would normally want to use to
                            batFolder and you should be fine. I suggest using batFolder
                            like so:
                            batFolder -ltr
```

```
(bat_env) user >  batFolder -3 -ltr    #newest folders on the bottom
                                        #value goes in $folder3 with -3 option
0
  1  drwxrwxr-x 1 craig users  210 Sep 14  2023 SMTBF_from_200k_to_300k_step_10k
  2  drwxrwxr-x 1 craig users  134 Oct 18  2023 October_Projects
  3  drwxrwxr-x 1 craig users  134 Oct 18  2023 test_synth_200
  4  drwxrwxr-x 1 craig users  134 Oct 23  2023 test_synth_5hr
Prepend selection with a '.' (period) to enter directory
Enter a choice (0 to exit):.2  # the folders that follow will not have -ltr
                               # applied to them
-i /path/to/simulator/experiments/October_Projects
  1 SMTBF_from_200k_to_500k_step_50k
  2 test_synth_800
  3 test_synth_10hr
Prepend selection with a '.' (period) to enter directory
Enter a choice (0 to exit): 1
(batsim_env) user >  echo $folder3
/path/to/simulator/experiments/October_Projects/SMTBF_from_200k_to_500k_step_50k
```

5.1.6 bind_all

```
(batsim_env) user >  bind_all --help
      binds the following:
         alt+u: previous-history
         alt+m: next-history
         alt+y: history-search-backward
         alt+n: history-search-forward
         alt+j: backward-char
         alt+k: forward-char
         alt+h: unix-line-discard
         alt+l: beginning-of-line
         alt+;: end-of-line
         alt+p: yank
         alt+g: kill-line

      **Note: Will only last while the shell is open for your user
            If you like these binds you can make them permanent for your shell by
            editing ~/$USER/.bashrc
            and adding the following to that file:

            function bind_keys
            {
               bind '"\eu":previous-history'
               bind '"\em":next-history'
               bind '"\ey":history-search-backward'
               bind '"\en":history-search-forward'
               bind '"\ej":backward-char'
               bind '"\ek":forward-char'
               bind '"\eh":unix-line-discard'
               bind '"\el":beginning-of-line'
               bind '"\e;":end-of-line'
               bind '"\ep":yank'
               bind '"\eg":kill-line'
            }
            #this determines whether we are in an interactive shell,
            #the only time the commands are usable. It will bind if it is.
            #if unusable the bind command will print out error messages and gets
            #in the way. So it is best to check.
            STR="$-";SUB="i";INTERACTIVE=0
            if [[ "$STR" == *"$SUB"* ]]; then
               INTERACTIVE=1
               bind_keys
            fi
```

5.2 myBatchTasks.sh

```
myBatchTasks.sh -          automates the generate-config script and run-experiments
                           script. Meant to be run from the command line and probably
                           put into the background.
                           NOTE: make sure you are in a batsim_environment and the
                           correct one.

Usage:
myBatchTasks.sh -f <STR> -o <STR> (-p sbatch [-c <INT>] | -p tasks -t <INT> )[-m <STR>]
                [-s <INT>] [-w <STR>][--permissions <STR>][--start-from-checkpoint <INT>]
```

```

Required Options:

-f, --file <STR>          The config file.  Just the file name, the folder of where that
                          file is is worked out from $PREFIX being set and default
                          locations within

-o, --folder  <STR>       Where to output all the results of the simulations.  Just the
                          folder name of where to put stuff within the default locations
                          ( $PREFIX/experiments/<folder_name>)

-t, --tasks-per-node <INT>  How many tasks to put on each node
                            Only used with --parallel-method 'tasks' and 'background'
                            and mandatory with these parallel-methods

Optional Options:
-a, --add-to-sbatch <STR>   Commands to add to sbatch, in the form:
                            long:
                              -a "--option_s value --option_t value --option_u --option"
                            short/mixed:
                              --add-to-sbatch "-s value -t value -u --option"

-c, --cores-per-node <INT>  How many cores to use for each sbatch
                            Only used with --parallel-method 'sbatch'
                            Not mandatory

-m, --method <STR>          What method to run batsim:
                            'bare-metal' | 'docker' | 'charliecloud'
                            [default: 'bare-metal']

-p, --parallel-method <STR> What method to spawn multiple batsims:
                            'sbatch' | 'tasks' | 'none' | 'background'
                            sbatch: individual sbatch commands for each sim
                            tasks: --tasks-per-node sims per sbatch command, with enough
                              sbatch's to complete config file generated sims
                            none: no parallelism,only serial. Will run one sim after
                              another (may take a VERY long time)
                            background: will try to achieve parallelism by backgrounding
                              each sim, backgrounding (--tasks-per-node - 1) sims
                              before waiting
                            [default: 'tasks']

-s, --socket-start <INT>    What socket number to start at. You must do your own
                            housekeeping of sockets.  If you already have 100 sims
                            going and you started at 10,000, then you will want to do
                            your next set of sims at 10,100 for example.
                            You can use higher numbers.  I've used numbers up to 300,000
                            [default: 10000]

```

```
-w, --wallclock-limit <STR> How long your jobs will take as reported to SLURM
                            Will leave it up to slurm if not chosen.  Sometimes SLURM
                            will set it for UNLIMITED
                            STR is in format:
                            "minutes", "minutes:seconds", "hours:minutes:seconds",
                            "days-hours", "days-hours:minutes" and
                            "days-hours:minutes:seconds"
                            ex: '48' , '48:30', '2:48:30'
                                  = 48 minutes, 48.5 minutes, 2hours 48.5 minutes
                            ex: '3-0' , '3-12:0', '3-12:30:0'
                                  = 3days, 3days 12 hours, 3 days 12.5 hours

-P, --permissions <STR>     permissions to give files/folders after generate_config.py is
                            run but before run-experiments.py is run. It is still
                            ↪  suggested
                            to use SLURM_UMASK in batsim_environment.sh for files made
                            during the simulations.
                            STR = The octal numbers for the permissions
                            ex: '--permissions 777' = rwxrwxrwx
                                '--permissions 750' = rwxr-x---
                                '--permissions 755' = rwxr-xr-x

--test-suite                set this flag to create current_progress.log in one folder
                            up from FOLDER1

-C, --skip-completed-sims   Set this to skip sims that are in progress.log as completed
                            [default: false]
Checkpoint Batsim Options:

-S, --start-from-checkpoint <INT>   Set this if starting from a checkpoint.  The
                                    <INT> is the number of the checkpoint.
                                    Typically '1', the latest
                                    [default: false]

-D, --discard-last-frame            Used in conjunction with --start-from-checkpoint
                                    and can be used with --start-from-frame
                                    Does not make sense to use with
                                    --start-from-checkpoint-keep
                                    Flag to give the following behavior:
                                      Will not change any of the kept expe-out_#'s and
                                      Will not keep the current expe-out

-K, --start-from-checkpoint-keep <INT>  Used in conjunction with --start-from-checkpoint
                                        Will keep expe-out_1 through exp-out_<INT>.
                                        Only use with --start-from-checkpoint
                                        When starting from checkpoint, the current
                                        expe-out folder becomes expe-out_1.
                                        Previous expe-out_1 will become expe-out_2 if
                                        keep is set to 2
                                        If you have expe-out_1,expe-out_2,expe-out_3 and
                                        keep is 3:
                                          will move expe-out_1 to expe-out_2
                                          will move expe-out_2 to expe-out_3
                                          will delete old expe-out_3
                                        [default: 1]

-F, --start-from-frame <INT>          Only used with --start-from-checkpoint and in
                                      conjunction with --start-from-checkpoint-keep
                                      Will use the expe-out_<INT> folder to look for
                                      the checkpoint data. So if you were invoking
                                      --start-from-checkpoint-keep 2,
                                      you would have expe-out_1 and expe-out_2,
                                      once you started from a checkpoint twice
                                      If you use --start-from-frame 2 you will be using
                                      the checkpoint_[--start-from-checkpoint]
                                      folder located in the expe-out_2 folder
                                      ( the expe-out_[--start-from-frame] folder)
                                      Here, '0' is the original expe-out folder that
                                      becomes expe-out_1
                                      If --discard-last-frame is used, then default here
                                      is 1. 0 is not allowed and will become 1 if used.
                                      [default: 0]

-I, --ignore-does-not-exist           Normally if the path to one of the checkpoints
                                      does not exist it will ERROR out and stop.
                                      This option tells it that some paths may not exist,
                                      but run the others that do,
                                      and not to alter anything in the paths that do not
                                      exist.

-h, --help                            Display this usage page
```

5.3 progress.sh

```
progress.sh: used to show the progress of all simulations in a folder, or just certain
             ones, with options for what to show.

Usage:
  progress.sh [-# | -i <folder> | -d <bookmark> ] [-P <options>] [-e <folder>]
              [-j <array string>] [-D <array string>] [-r <array string>]
              [-b <int>] [-H] [-p] [-c] [-q] [-s]
              [-m <batsim|batsched|both|all|available>] [-M K|M|G|T|H]

Mostly Required Options:

```

```
-#               Will use the environment variable folder# for input.
                 It will act as if you used -i ${folder#}
                 This is best used in conjunction with batFolder
                 NOTE: using this method requires folder# to be exported: export folder#
                 NOTE: batFolder automatically exports the corresponding folder# variable

-d <bookmark>            Will use the dibB.sh bookmark

-i, --input <folder>     Where the experiments are.  This is supposed to be the outer
                         folder passed to ./myBatchTasks.sh
                         If it has a forward slash '/' in the name it will assume it
                         is an absolute path.
                         If it does not have a forward slash '/' in the name it will
                         assume it is located in "$prefix/experiments/<input>"
                         If --prompt is used, this folder is interpreted as where to look
                         for experiment directories.
                         If --prompt is used and this option is omitted, the default will
                         be used '${prefix}/experiments'

Optional Options:

Folder Options:
  -P, --prompt <options>     Another option instead of setting input directly
                             Will go into folder '--input <folder>' and list the
                             directories for you.  Select the folder you are after
                             with a number. If --input is omitted, will use
                             '${prefix}/experiments'
                             The options you want to send ls comes after prompt, but use
                             quotes around the whole list of options as to not confuse
                             this script.
                             Make sure you use empty quotes if not setting any options
                             Also look into using batFolder for repeated uses of
                             progress.sh
                             example:    progress.sh -P '-ltr'    or
                                         progress.sh -i "/some/other/path" -P ''

  --up                       Normally if you pass --input as a Run_#,output,or expe-out
                             folder you will get a single progress of that folder's
                             simulation. With this option, it will always set the
                             corresponding project folder as the input.
                             Must appear after -i option

Which-Simulation Options:

  -e, --experiment <folder>      If you want to focus on just one experiment (in the sense
                                 of config files where there was an input and output json
                                 for each experiment) then use this to enter the folder.
                                 The path retrieved will then be
                                 ".../<input>/<experiment>/*"
                                 <folder> can include wildcard (*) characters as well.
                                 For instance...
                                   progress.sh -i ${folder1} -e "*shuffled*"
                                 if you were going to shuffle some experiments and named
                                 those experiments with 'shuffled' in the name,
                                 then this would get you all the experiments with the word
                                 'shuffled' in it.

    -j, --job <array string>      If you want to focus on just one job (in the sense of
                                  folders named experiment_#) then use this to enter the
                                  folder number.
                                  If used with --experiment then it will retrieve paths
                                  ".../<input>/<experiment>/experiment_<job>/*
                                  If not used with --experiment, then it will retrieve
                                  paths ".../<input>/*/experiment_<job>/*
                                  If multiple jobs are wanted then put the job numbers
                                  into a string,
                                  for example: '-j "1 2-5 10"'
                                    will gather experiment_1,experiment_2,
                                    experiment_3,experiment_4,experiment_5, and
                                    experiment_10
                                  for example: '-j "3:1000:50 1 1050-1055"'
                                    will gather '3-1000 step of 50' '1' and
                                    '1050,1051,1052,1053,1054,1055'

    -D, --id <array string>       If you want to focus on just one id (in the sense of
                                  folders named id_#) then use this to enter the folder
                                  number. Basically the same as --job as far as how it
                                  works, except we focus on the id(s)
                                  Look at --job for string format

    -r, --run <array string>      If you want to focus on just one run (in the sense of
                                  folders named Run_#) then use this to enter the folder
                                  number. Basically the same as --job and --id as far as
                                  how it works, except we focus on the run(s)
                                  Look at --job for string format

    -F, --finished-sims           Only include sims that have:
                                  overall_completed_jobs == overall_jobs

    -U, --unfinished-sims         Only include sims that have:
                                  overall_completed_jobs != overall_jobs

    -T, --threshold <string>      Only include sims that are above or below a threshold
                                  for overall_percent_done
                                  format of string: "(+|-) <threshold float>"

  What-Info Options:

    -b, --before <int>              The amount of entries, starting from the last one, to
                                    include

    -H, --head                      Print the first entry in the out_extra_info.csv file
                                    first ( possible use is to check on how long the sim
                                    has been going )

    -a, --all                       Print out all data in the entry.  Default if nothing
                                    else is chosen to print.

    -E, --elapsed-time              Prints out the real elapsed time of the simulation.
                                    Must include the option to see it.
                                    Not included in the --all option, but keeps --all the
                                    default.
                                    Use -z "a" option to supress all, or choose what else
                                    to print out.

    -o, --overall-percent           Prints out the overall-percent when used with
                                    checkpoints.  Must include the option to see it.
                                    Not included in the --all option, but keeps --all the
                                    default.
                                    Use -z "a" option to supress all, or choose what else
                                    to print out.

    -O, --original-info             Prints out "overall-percent, original-jobs,
                                    number-resubmits,total-actually-completed"
                                    Not included in the --all option, but keeps --all the
                                    default.
                                    Use -z "a" option to supress all, or choose what else
                                    to print out.

    -p, --percent                   Include the percentage done that the sim has done

    -c, --completed                 Include the amount of ACTUALLY completed jobs and
                                    TOTAL jobs

    -t, --time                      Include the real-time and sim-time

    -S, --schedule-info             Include queue-size,schedule-size,nb-jobs-running

    -q, --queue-size                Include the queue-size in output

    -s, --schedule-size             Include the schedule-size in output

    -u, --utilization               Include utilization and utilization-without-reservations

    -m, --memory <string>           Include the memory usage.  Will include the USS,PSS,RSS
                                    for "batsim|batsched|both".
                                    Other options are "all|available"

    -M, --memory-size <string>     When displaying memory, what size to display:
                                   KB | MB | GB | TB | H.
                                   Single letter is fine too: K | M | G | T | H
                                   This will actually be in KiB, MiB, GiB, or TiB
                                   H signifies you want human-readable units.
                                   This is the largest unit that makes sense for the data.
                                   [default: H]
    -z, --no-info <string>         What categories NOT to show.<string> is a comma
     ↪  seperated
                                   string with all of the short options above
                                   but instead of displaying what the options would
                                    ↪  normally
                                   select, it displays everything except them.
                                   ex: "t m" no time, no memory, everything else
                                   ex: "a" nothing else except what is selected
                                       (in particular, use "a" with -O and -o options)

    -N, --normal-info              Prints a normal usage scenario: -O -E -c -p -Z
  Presentation Options
    -C, --condensed                When displaying data, will not follow the line breaks of
                                   the "format" at the bottom

    -Z, --super-condensed          When displaying format at the bottom, will use
                                   abbreviated names
                                   When displaying data, will not follow the line breaks of
                                   the "format" string
                                   When displaying path, will use a condensed version of it

    -h, --help                     Display this usage page
```

5.4 aggregate_makespan.py

```
Usage:
  aggregate-makespan.py -i FOLDER [--output FOLDER] [--try-frame1] [--start-run INT]
                        [--end-run INT] [--bins]

Required Options:
  -i FOLDER --input FOLDER    where the experiments are

Options:
  -o FOLDER --output FOLDER   where the output should go
                              [default: input]

  -t --try-frame1             if we don't find makespan.csv in normal expe-out,
                              then try expe-out_1
```

```

    --start-run INT             only include runs starting at start-run

    --end-run INT               only include runs ending at and including end-run

    --bins                      TODO If set, will calculate from binned data
                                This assumes you have run post-processing with bins
                                located in output/config.ini
                                and that it produced output/expe-out/bins/*.csv files
```