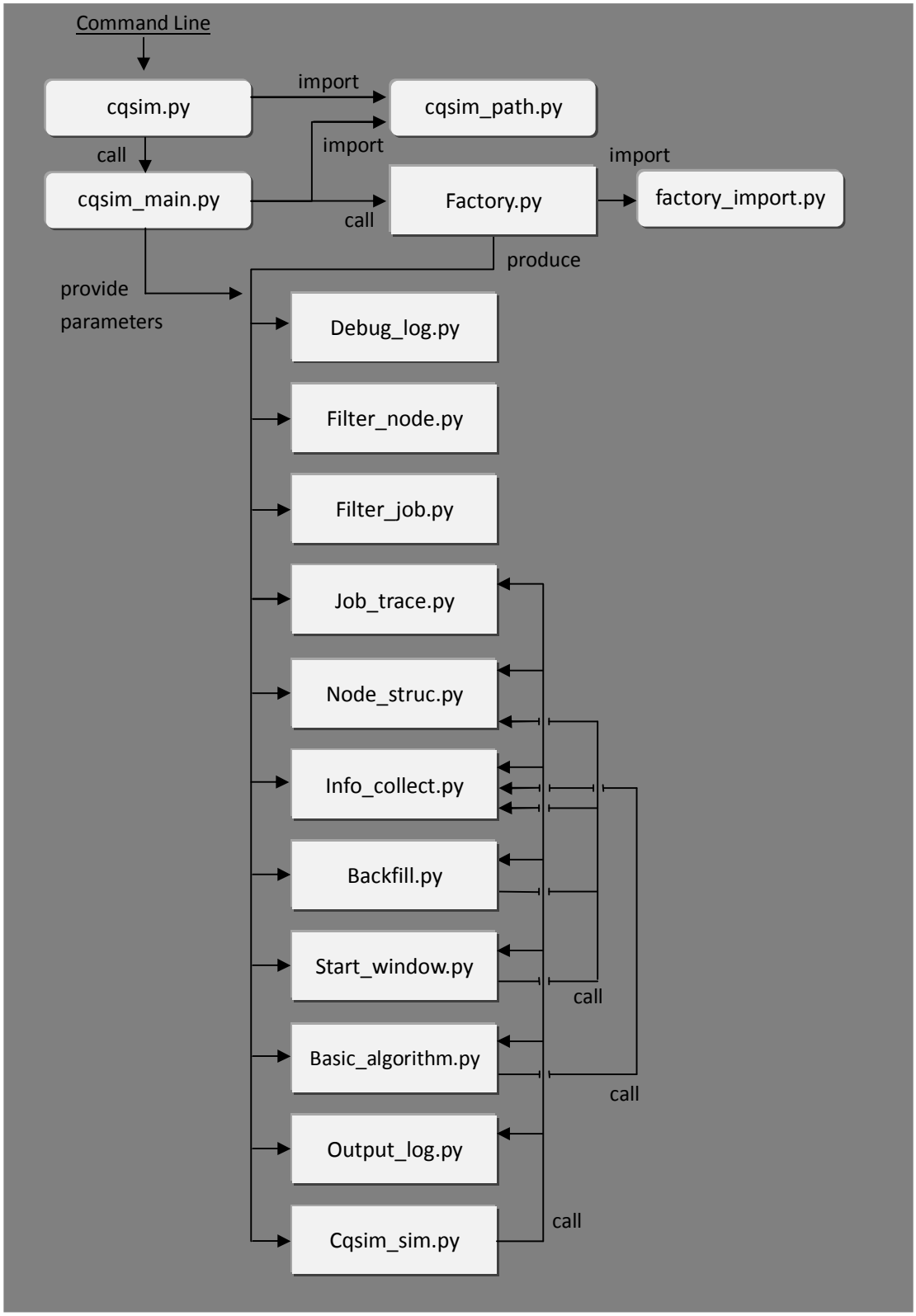
CQSIM -Report

Ren Dongxu

1. GOAL

- **An event driven job schedule**
  - Simulator scans the event sequence and do the operation related to every event in time order.
  - Event can be job submit/job finish, monitor event or other event added by the user.
  - An overall method invokes and initializes all the modules and the handles of the modules will be transported into the simulator.
  - The simulator should be able to support other modules and their subclasses.
- **A user command line interface**
  - User can pass all the parameters by command line
  - Advantage user interface can be used to call the command line entry automatically.
  - A system parameter config file can be used to initialize the command line parameter
  - A file name config file can be used to initialize all the temp, debug and output path, name and extension name.
  - The data read from the config file are on low level, so the parameter given in the command line will replace the same data read in config file.
- **Extendable module design**
  - These modules should have the standard interface.
  - All modules are supposed to know all the data formats. Hence, they can get correct data from the dictionary type of parameter. And any modification in data format should be specified clearly in the design document.
  - The modules can be extended in 2 ways: subclass and new method.
  - Also, new function can be added to the existed method. But this kind of modification should be static, which is used in all extension.
- **Input and output files**
  - Input raw files: Job trace and Node structure files
  - Formatted files: Job trace, Node structure, Job and Node config files
  - Output Result files: Job simulator result, Event log and Debug log.

2. STRUCTURE



- The flow is:
  - cqsim.py receive all input parameter from command line or config files.
  - cqsim_main.py instantiates all modules by factory mode, and pass the parameters to the

corresponding module.

- Modules are initialized in order. At last, the handles of 7 modules are passed into Cqsim_sim and start simulating.

3. FUNCTION

- **Simulating**
  Simulating the situation that jobs running on the node structure. And output the simulating result in text.
- **Filter**
  Filter different type of job and node structure into the same format. In this version, SWF job trace and be read and formatted.
- **Basic Algorithm**
  Run the simulating with the given algorithm. The algorithm is given by user when simulate starts.
- **Backfill**
  Implement EASY and conservative backfill.
- **Start Window**
  This is a function to optimize the local job arrangement.
- **Adapt**
  The simulator can modify the parameters in running time.

4. TEST

5 Job traces are used to test the simulator. For every job trace, 2 same algorithms are used to test.
For the same job trace, we got

$$\sum(\text{job running time}) = \sum(\text{utilization} * \text{interval time} * \text{total node number})$$

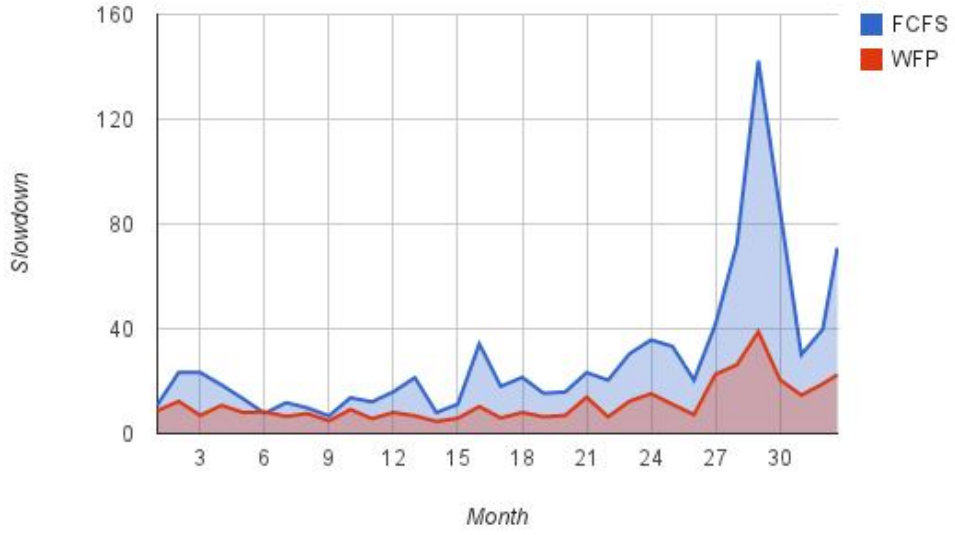
That proves the simulator runs well.

5. RESULT

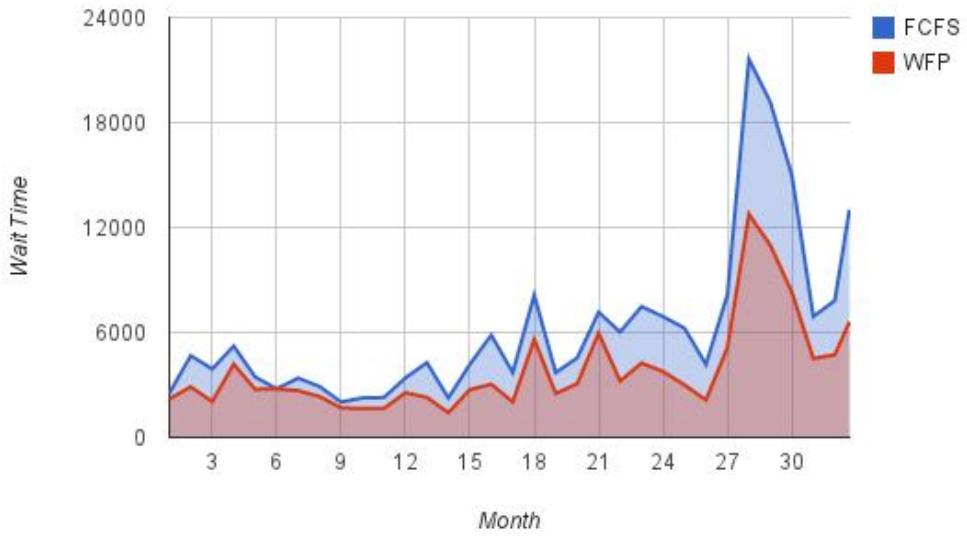
- **SDSC-BLUE-2000-4.1-cln.swf**
  - **Algorithm: FCFS / WFP**
  - **Backfill: EASY**

  - Avg. Utilization

  - Slowdown



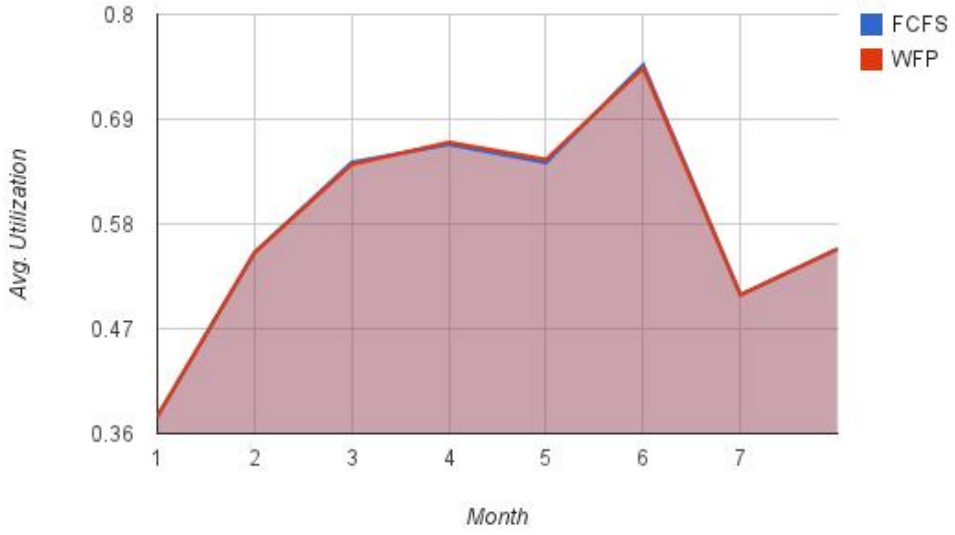
- Wait time



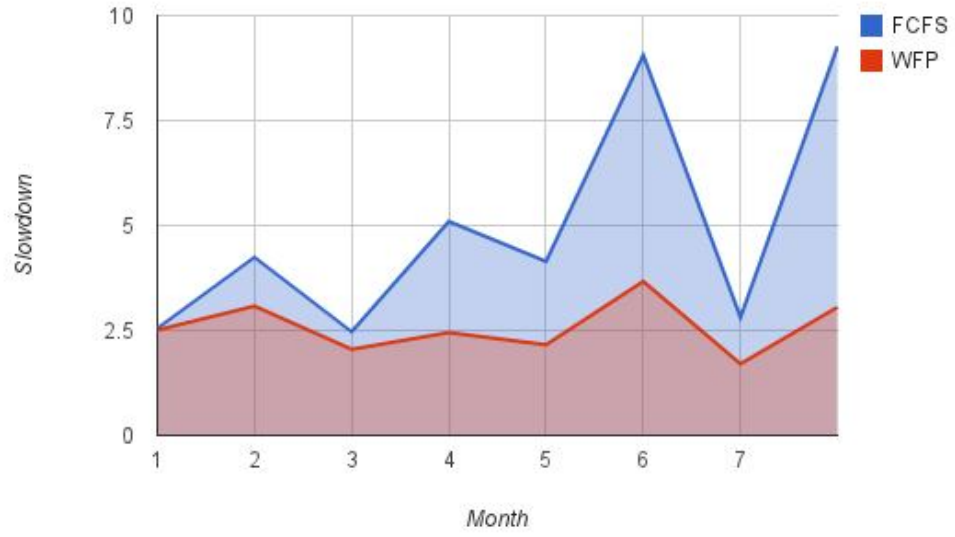
- **ANL-Intrepid-2009-1.swf**

  - **Algorithm: FCFS / WFP**
  - **Backfill: EASY**

  - Avg. Utilization



- Slowdown



- Wait time