Description of Design Employed

Meshack Sandra, Thomas Iona

Contents

1. Introduction

In order to solve this problem, we decided to use discrete event simulation with a next-event time progression because this usually runs faster than fixed-increment time progression. The c++ patient simulator was used as a foundation. In this exercise, a doctor was receiving a queue of patients while in our case, we have nodes receiving jobs to run. There are five different categories of jobs(small,medium,large,huge,Gpu)(see requirement document for more precision) as opposed to the hospital that has a doctor that is capable of attending to his entire projects whereas not all nodes can run all type of jobs. Therefore, we replaced the queue from the patient simulator with a scheduler.
The scheduler keeps track of five different job queues,available nodes and matches the jobs with the nodes. Every item of the simulation shall be discussed in the following sessions.

2. Events

The event implementation for the c++ exercise:

According to the above diagram, an event is mainly a time stamp and an execute function. The execute function implements what happens at this time stamp. The event also has comparison relation in order to easily be sorted. we created different types of events to fit our HPC simulation(see figure 2) for the inheritance graph.

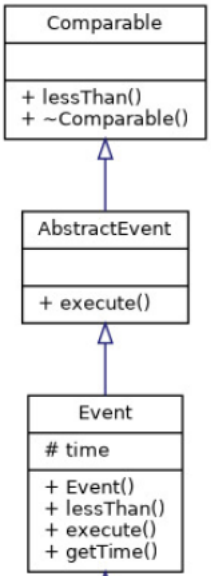

Figure 1: Event Inheritance

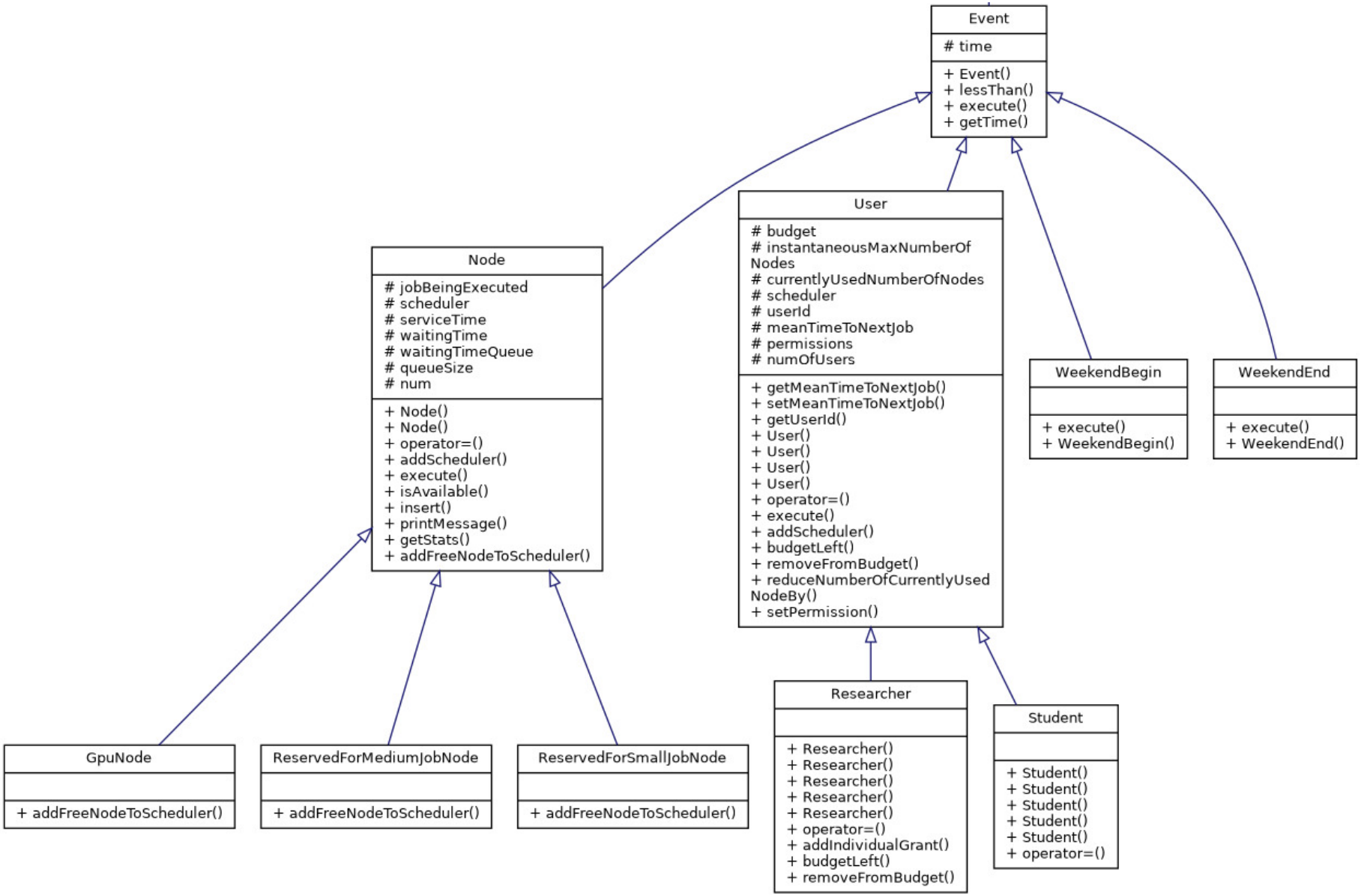

Figure 2: Event Subclasses

2.1. Weekends

As we need to implement a weekend cutoff, we use two events for delimiting the weekend. They are:

- WeekendBegin: When this happens,it tells the scheduler to run as many huge jobs as possible on all the available nodes. It reschedules itself for the next week if there are still available jobs to run or user that still has some budgets.
- WeekendEnd: This tells the scheduler to try to execute as many jobs as possible according to priority. It also reschedules itself if there is more than one event left for the next week.

2.2. User

Users event corresponds with a user attempting to submit a job according to his budget. The user generates a random job according to his permissions (the user has been granted privileges to run certain types of job) and his budget left. If the user does not have enough budget for this job, it does not reschedule itself which means that the user does not produce anymore jobs. Otherwise, it reschedules itself using exponential distribution for computing the next time.

There are two types of users:

- Researcher: The researcher shares a budget with his group members and may have additional individual grant.
- Students: The student is given a cap according to his curriculum both cumulatively and instantaneously.

Both classes inherit from user and implements two different version of the functions for computing the budget left and removing from budget.

In addition to that, the instantaneous cap for students has been implemented the following way:

- The student keeps a record of how many nodes he asked the scheduler for.
- When a job is complete, it removes the used nodes from the student counter.
- The student does not submit a job if the additional number of nodes required for a new job makes him exceed the students instantaneous cap.

Therefore, the instantaneous cap is applied on the submission and not on the execution of the jobs. Applying this cap to the execution will require a more advanced scheduler.

2.3. Group and Curriculum

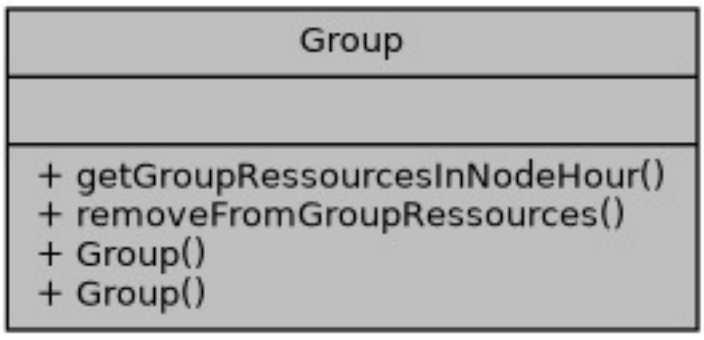

Figure 3: Group

- Group: This has a private variable that is initialised at the creation of the group with the available budget and it has a getter to ascertain the amount of budget left and a function to remove amount spent from the budget.
- Curriculum: This has two caps set up when creating new curriculum and two getters to get the cap values.

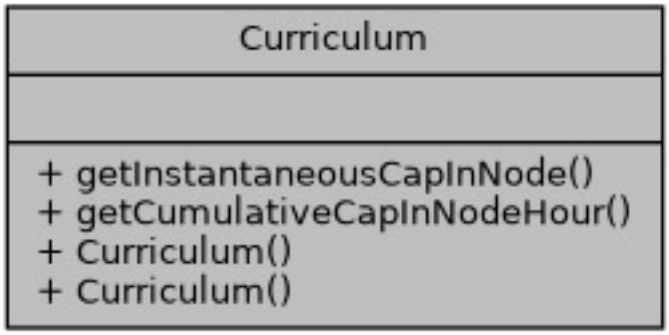

Figure 4: Curriculum

2.4. Nodes

A node event is when a node finishes its job and relays the message to the scheduler.
We have three sub-classes of nodes. They are:

- Reserved for small jobs
- Reserved for medium jobs Gpu accelerated jobs

The difference is apparent when they inform the scheduler that they are free and signifies the types of nodes they are which is done by the addFreeNodeToScheduler method.

3. Scheduler

The scheduler is the part of the simulation that is used for matching jobs and users request with nodes. This is the part of the simulation that the IT staff may want to tweak/replace to try different strategy. Therefore we created an AbstractScheduler class which is defining the minimal interface for being able to fit the scheduler in the simulation. This interface includes the method for inserting the five different kinds of jobs, methods for individual types of free nodes you want to add. All those jobs and free nodes are stored if they cannot be used right away.

We also have a tryToExecuteNextJobs method which is run at the end of the weekend for executing as many known huge job as possible.tryToExecuteNextHugeJobs method is run at the beginning of the weekend to start as many huge jobs as possible.

We also have a function that tries to execute the next job of a certain type

- small job
- medium job
- large job
- gpu job
- next non-gpu job that is short enough to run before the weekend shut off.

Finally, a method returning the number of none huge jobs waiting.

Scheduler class is a FIFO(first in first out) version of the scheduler.

For the scheduler, we used the one from the c++ exercise.

4. Simulator

The simulator's job is to keep track of the time and execute all the events registered in the order of the timestamp. We reused the abstractSimulator and Simulator from the c++ exercise and we added a new class(HPCSimulator) which is inheriting from the simulator. This new class allows us to initialise the simulation from a file, keep track of all the finished jobs and print statistics about the simulation.

The list of events is kept in a ListQueue which has been defined in the c++ exercise and reused as is.

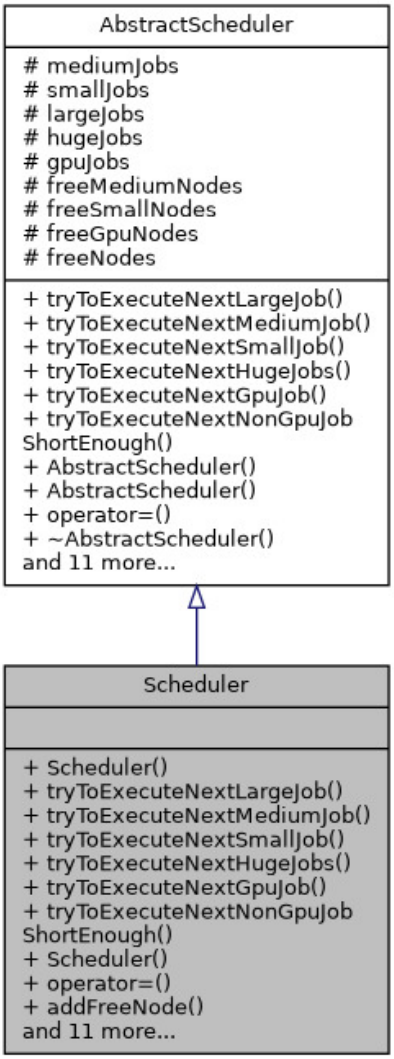

Figure 5: Scheduler

5. HPCParameters

This is a static class designed to hold the parameters of HPC machine. It includes maximum number of nodes and duration for several job categories, number of nodes, number of accelerated nodes and operation cost and prices.

6. Random

This is a static class for providing a convenient interface for random number generation according to different distributions(Normal, exponential, uniform and binomial).

7. Main Function

The main function requires a path to a file as a command line input. This is the file that it is going to use to initialise the HPC simulator.It then runs the simulation and prints the results.

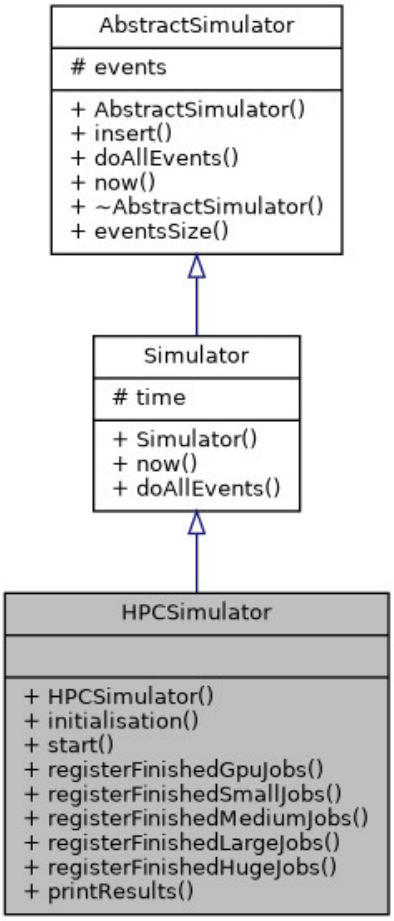

Figure 6: Simulator Inheritance