

Design and Implementation of an In-Browser Simulator for Teaching the Principles and Practices of Using Batch-scheduled Parallel Platforms

Herman Li

Abstract

Given the difficulties involved in teaching HPC and distributed computing topics using real-world platforms, simulation provides an attractive alternative that can provide students with a hands-on learning experience. Currently, the EduWRENCH project provides pedagogic modules covering a number of parallel and distributed computing topics, but it currently does not provide any module targeting the use of practical tools, and in particular tools requiring interactive use of the terminal, such as batch schedulers. This project designs and implements a simulation of batch-scheduled platforms with an interactive, in-browser terminal interface to teach the principles and the uses of batch schedulers.

1. Introduction

Every year, more students enroll in Computer Science as their major. Although Computer Science theory can be taught through traditional whiteboard and paper assignments, practical examples and hands-on assignments are more effective in deepening student comprehension of difficult topics. While many Computer Science classes can incorporate these elements, there are some topics for which it is difficult to implement. One of these topics is parallel and distributed computing, which a many concepts, techniques, and tools immensely benefit from hands-on learning. One of the key tools in this domain is batch scheduling, that is the use of software infrastructures that manage access to high performance computing platforms.

The impracticality of providing hands-on experience teaching parallel and distributed computing topics typically arises from the need to provide students with access to an actual parallel computing platform, such as a commodity cluster, for the duration of the class. Albeit such platforms available at many institutions, their use is typically devoted to research projects within universities. Thus, taking valuable processing time from these projects is unrealistic. An alternative is to simulate a cluster, which will remove the need to provide students with an actual compute platform [1].

Using simulation has further advantages in addition to removing the use of a real compute platform. A simulation can provide a controlled and consistent environment where students can observe repeatable results. In particular, a real platform will have many other jobs running and will result in a variable amount of time for student jobs to run, resulting in non-repeatable behavior. Similarly, a simulation does not necessarily have to limit itself to take one hour for a job which is expected to take one hour in a real system, but can accelerate the (simulated) execution or even allow for the jumping forward of time. This provides a way for students to experience what will happen during a long time span without actually sitting through the process. As a result, the use of simulation provides many options for hands-on learning that would be of questionable feasibility, even if a real compute platform were to be provided to students.

One development capitalizing on the use of simulation for parallel and distributed computing education is EduWRENCH, an online platform using simulation-driven pedagogic activities for teaching parallel and distributed computing concepts [2]. EduWrench hosts multiple pedagogic modules pertaining to hardware and use of single-core, multi-core, and distributed computing. These modules can be used to augment existing university courses with hands-on, simulation-driven activities for students to execute with the simulations built on the WRENCH and SimGrid frameworks.

Currently, EduWRENCH pedagogic modules only focus on the concepts behind parallel and distributed computing without any coverage of particular tools and methodologies for using real-world distributed computing platforms and systems. In particular, one fundamental component of distributed computing platforms, specifically high performance computing platforms, not covered in any EduWRENCH module is batch schedulers. These batch schedulers, as their name implies, are in charge of scheduling user jobs onto parallel computing platforms. Users of these platforms request access to platform resources via the batch scheduler, which will eventually grant access to the resources. Learning how to use batch schedulers correctly and efficiently is a key practical skill students (or professionals) need to acquire.

This project lays the foundation for including a batch scheduling pedagogic module to EduWRENCH. Consisting of a web server hosting the batch scheduler simulation and a front-end displaying a simulated shell in a terminal, the design provides a simulation of the typical workflow between a user and batch-scheduled compute platform (job submission, job cancellation, job status, picking appropriate job configuration parameters, etc.).

The remainder of this document is organized as follows: Section 2 provides the necessary background and explanation of terminology as relevant to this project; Section 3 describes the design and implementation of the project; and Section 4 gives concluding remarks and outlines future work and development directions.

2. Background

EduWRENCH pedagogic modules are developed to target a wide range of learning objectives in the area of parallel and distributed computing. Each module comes with a pedagogic narrative starting with introducing concepts and techniques, examples and simulation-driven activities, to practice homework questions with and without provided solutions. The primary intent for these modules is integration with university courses, while still allowing for individuals to study independently using these modules. From a pedagogic standpoint, the key innovative aspect of EduWRENCH is the use of simulation-driven activities. This project provides the approach and implementation to fill a current gap in the set of activities available in EduWRENCH, simulations driven by an interactive terminal session.

Each simulation-driven activity in EduWRENCH is built on WRENCH to simulate a particular parallel and distributed computing scenario. This project will similarly be built on the WRENCH framework. WRENCH is a framework to simulate the execution of parallel and distributed applications (in particular workflow applications) on a wide range of hardware and software infrastructures. It offers a high-level API allowing a straight-forward method to create an accurate representation of complex systems. In particular, WRENCH comes with in-simulation implementations of many core software services representative of production software infrastructures currently running on parallel and distributed computing platforms. These services, pertaining to computation, storage, and networking, can be assembled and

configured to quickly build a WRENCH simulator of an arbitrary system. The hardware platform is described by an XML file, which defines the capabilities of all hardware resources and how they are interconnected via a network. Using these simulated core services as the foundation, a WRENCH simulator typically implements a workflow management system (WMS), a software agent that interacts with the core services performing computations and data movements executing a particular application workload. WRENCH simulators can not only be used for educational purposes, like in pedagogic modules, but also for research purposes, typically to verify research hypotheses via experimental results obtained in simulation.

WRENCH itself builds and extends on the SimGrid simulation framework [3]. SimGrid provides low-level simulation abstractions and corresponding APIs to implement simulations of arbitrary distributed systems. The main focus of the SimGrid project has been the scalability and the accuracy of the simulation models it provides. As a result, SimGrid simulations have been shown to be highly scalable (i.e., simulations of complex and long-running scenarios running in a few seconds on a single computer with a low memory footprint) and accurate (as demonstrated by many validation and invalidation studies). Since SimGrid simulates low-level abstractions, a resulting drawback is the labor-intensiveness of developing a SimGrid-based simulator for complex systems giving rise to the motivation in developing the WRENCH framework.

With the new pedagogic module being based on batch scheduling, batch scheduling is a technique used to allow multiple users to share a single parallel computing platform. It is a form of job scheduling and implemented by a software agent called a “batch scheduler” controlling a set of compute resources (e.g., the compute nodes of a cluster). These compute resources can only be indirectly accessed by users through the batch scheduler when they submit jobs to the batch scheduler. Each job requests a certain amount of compute resources (i.e., a number of compute nodes) for a certain duration. The batch scheduler then places these job requests in a queue called the “batch queue” before using various algorithms to decide when each job should be granted access to the resources it had requested. Each job, thus, first experiences a “wait time” after which it is allocated the compute resources it needs. If a job exceeds its requested duration, it is terminated. The main goals of a batch scheduler are often to maximize job throughput while minimizing the amount of unused resources. Many batch scheduler implementations are available, with the most commonly used one being SLURM, the Simple Linux Utility for Resource Management [4]. In addition to submitting jobs to a batch scheduler, it is also possible to cancel jobs, monitor their progress, and inspect the state of the batch queue. WRENCH provides an in-simulation implementation of a batch scheduler which can be used by any simulator. Both schedulers, including SLURM, provide a command-line interface where users can invoke several commands in the terminal. In this work we provide a similar capability but all in simulation: the user issues commands in a simulated terminal and these commands interact with a simulated batch scheduler.

3. Design and Implementation

At the highest level, the design of the project is separated into two distinct parts: the *client* and the *server*. They use a REST API to communicate with each other. The client runs in the browser and contains a clock and a terminal for user input and is implemented using HTML and JavaScript. The server contains the batch scheduler simulation implementation, which is written in C++.

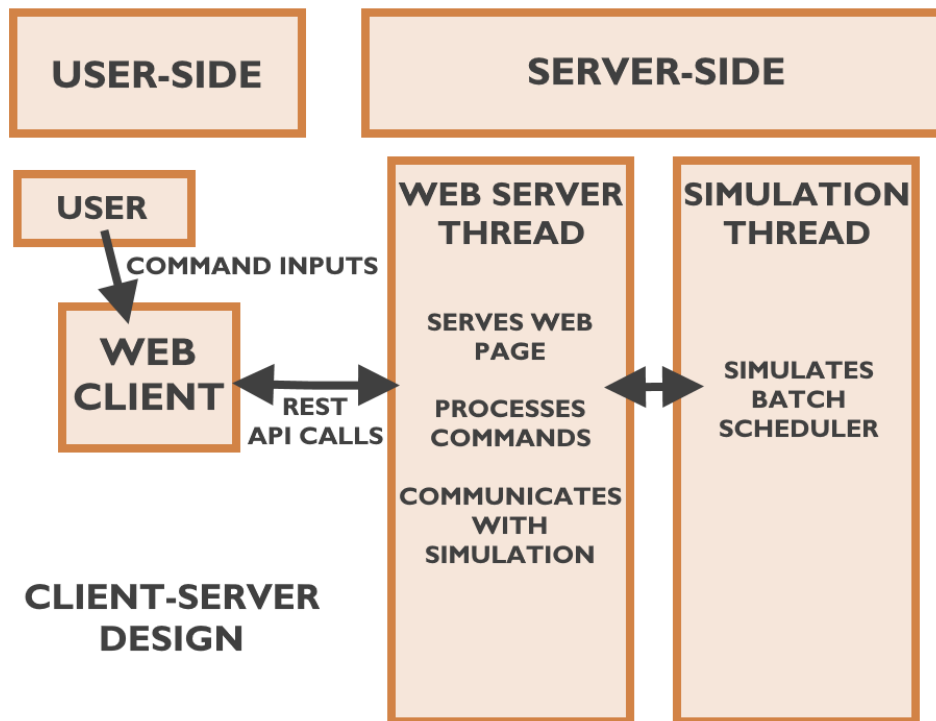


Figure 1. Client-Server design implemented in project

3.1. Client

The client consists of a digital clock to show the simulated time which matches actual time passing. Three buttons are provided to allow hops in time in increments of 1 minute, 10 minutes, and 1 hour. As a result, the user can fast-forward time in the simulation at will. The client also provides a terminal with an emulated bash shell. The user interacts with this terminal by typing shell commands. Finally, the client provides an editing window which will only appear when editing a text file. The client itself is written mostly in HTML and Javascript with only two libraries imported: Xterm.js for the terminal emulator to be created, and Bootstrap for structuring the HTML page.

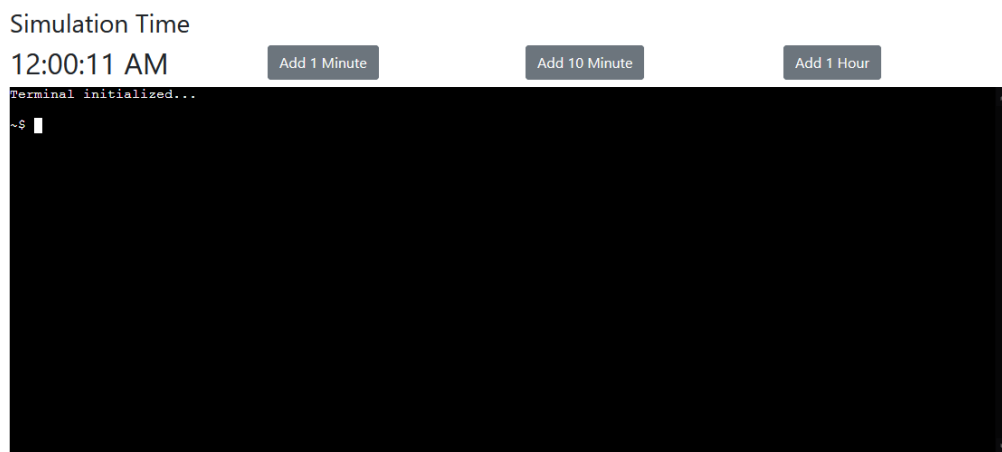
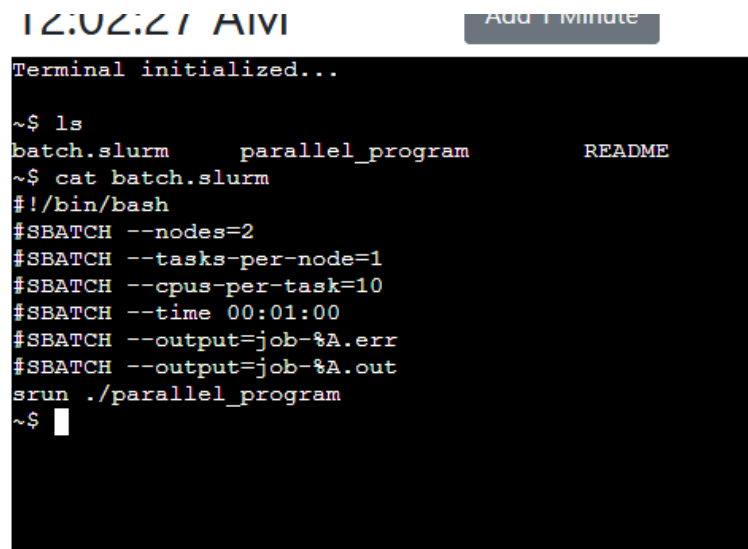


Figure 2: Empty terminal with command prompt and buttons on startup

The terminal consists of a few parts in order for it to simulate a bash shell. It uses Xterm.js to create the terminal emulator itself, functions to handle commands commonly used in the shell, functions to handle Slurm commands for batch scheduling, and a simulated filesystem where (simulated) user files are located.

Xterm.js is a front-end component used to add a terminal emulator to an HTML document [5]. It is used in popular projects such as Microsoft Visual Studio Code and JupyterLab. The terminal emulator simply displays the terminal itself but has no ability to process commands or print characters, and we had to implement these capabilities ourselves. Although it is very limited in terms of what it does out-of-the-box, it provides an API to allow the terminal to either be connected to a remote computer offering terminal access or to implement the shell from scratch. In this case, the shell was implemented from scratch to simulate a bash shell.

To give the sense the user is actually using a bash shell, functions were implemented to provide simplified versions of the most commonly used commands including “ls”, “cat”, “touch”, “cd”, and “clear”, as well as some custom commands such as “date” to print file creation date and “edit” to allow editing of text files in the browser. Slurm commands are handled in this fashion as well (see their description hereafter). Generally, these functions perform basic parsing and error handling, before placing a call to internal functions to handle the operation similar to how UNIX systems work.

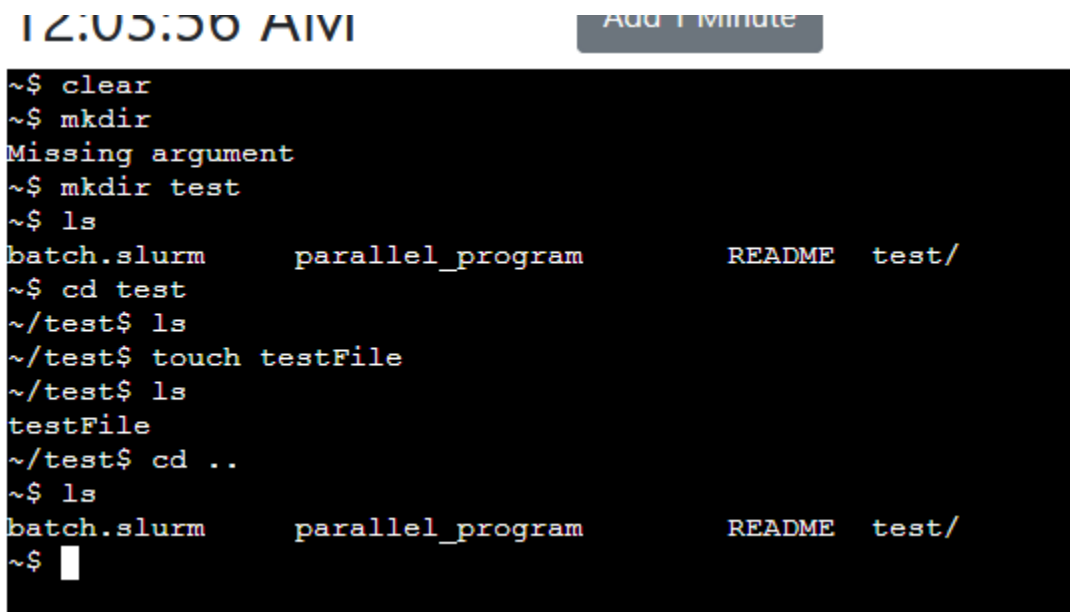


```
12:02:27 AM Add 1 minute
Terminal initialized...
~$ ls
batch.slurm      parallel_program  README
~$ cat batch.slurm
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=10
#SBATCH --time 00:01:00
#SBATCH --output=job-%A.err
#SBATCH --output=job-%A.out
srun ./parallel_program
~$
```

Figure 3: Simple commands run within simulated shell

Given many of the above commands relate to file manipulation, there needs to be a simulated file system to handle the corresponding operations. The file system is simulated by using a JSON format with each folder and file represented by a JSON object. Each JSON object, which represents a file system object, contains three parts: A string detailing the object type, which can be either a directory, a binary file, or a text file; A string containing the time of creation based on the simulated system time; And for directories, a JSON array contains other file system objects, while for files, a string that contains the text data the file holds. This two-level structure allows for any extra details to be added in the future, such as the last edited date, without disrupting the file structure. The file system manages current directories by holding a pointer to the current directory, with another pointer holding the full filesystem, and an array holding the path to the current directory. This means going into a directory changes the pointer to the new directory and adds the directory name to the array. Going up

a directory requires starting from the root directory and tracing the path down to the parent directory. Since the path to the directory is saved in an array, it provides the ability to return the current working directory.



```
12:05:50 AM | Add 1 Minute
~$ clear
~$ mkdir
Missing argument
~$ mkdir test
~$ ls
batch.slurm      parallel_program  README  test/
~$ cd test
~/test$ ls
~/test$ touch testFile
~/test$ ls
testFile
~/test$ cd ..
~$ ls
batch.slurm      parallel_program  README  test/
~$
```

Figure 4: Simulated shell running filesystem commands

There are three Slurm commands implemented in the client: “sbatch”, “squeue”, and “scancel”. The sbatch command is used to submit simulated batch jobs to the batch scheduler. Using a prefilled and editable configuration file and a fake executable which the jobs “run”, sbatch will parse the configuration file to retrieve the number of nodes requested and the amount of compute time requested for the batch job before placing an API call to the server to add a batch job to the queue in the simulated batch scheduler. If successful, it will print the job name onto the terminal. The squeue command requests from the server the list of jobs in the queue and currently running handled by the batch schedule and displays them in the terminal. Details included in the information printed include the job name, the name of the user who submitted the job, the number of nodes requested, when the job started running, and the state of the job (running or pending). Lastly, the scancel command cancels a job based on the job’s name. Only the jobs submitted by the user can be cancelled, otherwise an error message will be displayed.

```

12.05.05 AM
Add Minute

Terminal initialized...

~$ ls
batch.slurm  parallel_program  README
~$ sbatch batch.slurm
job_0
~$ squeue
JOBNAME      USER      NODES  TIME          STATUS
job_0        slurm_user 2      00:00:20 UTC  RUNNING
~$ sbatch batch.slurm
job_1
~$ squeue
JOBNAME      USER      NODES  TIME          STATUS
job_0        slurm_user 2      00:00:20 UTC  RUNNING
job_1        slurm_user 2      0          READY
~$ scancel job_1
Successfully cancelled job_1
~$ squeue
JOBNAME      USER      NODES  TIME          STATUS
job_0        slurm_user 2      00:00:20 UTC  RUNNING
~$ ls
batch.slurm  parallel_program  README  job_0.out
~$ █

```

Figure 5: Batch scheduler commands in simulated terminal showing the user submitting two jobs, viewing the job queue, and canceling the second job.

The clock and buttons to fast-forward time provide a realistic sense of passing time while the user submits or cancels batch jobs, checks the queue status, or waits for job completion. To prevent the requirement to wait the realistic time-frame for a batch job to complete, which can be multiple hours long, the buttons to fast-forward the time allow the user to get a feel for how long a job can take without actually requiring them to sit there waiting for the simulated job to run. The implementation of the clock uses the default JavaScript standard library for handling dates and to minimize the complexity of handling time while keeping consistency for each run, the time is actually set to UNIX time since epoch of 0. Since only the hours, minutes, and seconds are displayed, there should be no confusion of the date since it can be corresponding to any arbitrary date. To change the time second by second and to synchronize with the server, a scheduled function is run every second to add 1 second to the clock and request any events from the server and the current server time. It uses the server time to determine if the client's displayed time is accurate to within half a second since this function is run asynchronously meaning the time might not be ticking along accurately. The buttons to jump time forward operate in a similar fashion with a request to the specific API to tell the server to jump forward in time as well.

To handle events from the server, using the query to the server sent every second, which returns the current server time and any events which occurred during the second. If there are no events, the client just receives an empty array. When jumping forward in time using the buttons, the client receives all events which occurred during that time period within the array. When there is an event received, each job completion generates a new text file called <job-name>.out to represent how the results of the compute task are returned in a real system. Since the creation time of the output file needs to match when the task was "done" even if it completed during the skipped time period, the time is extracted from the event and the file creation time is set accordingly.

To enable editing of text files, including the Slurm configuration file, there is a text editing area appearing below the terminal when the “edit” command is used. The editing area will have the file contents displayed for users to change with a save and quit button above to close the editor and saving the contents back to the simulated filesystem. For the “batch.slurm” file, the configuration file for Slurm, instead of allowing a completely editable text area, there is a form with input boxes for allowable adjustable values which includes the duration of compute time and the number of nodes to be requested. This will modify the text values of the form accordingly and when closing the editor, save the values to the simulated file system.

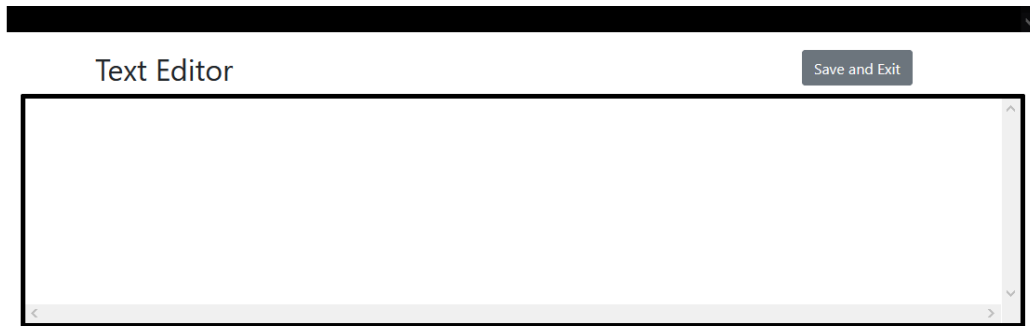


Figure 6: Empty text editor for any file

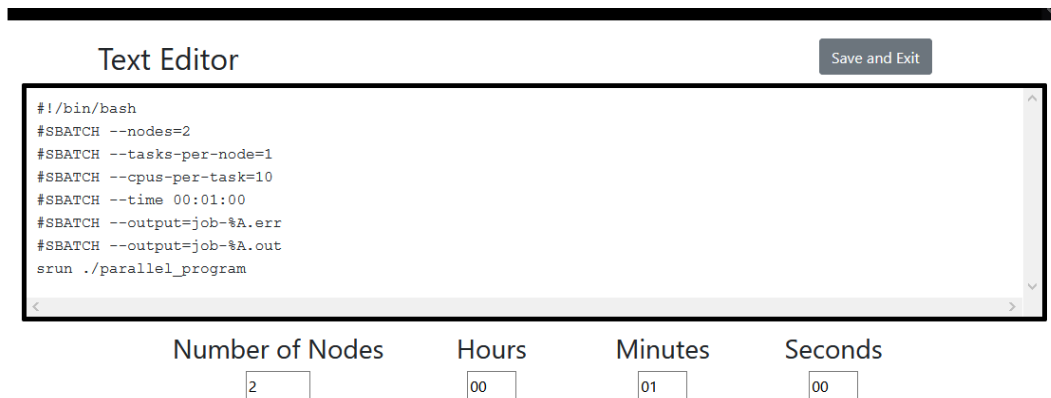


Figure 7: Text editor when editing Slurm batch file

3.2. Server

The server is written in C++ and makes use of multiple libraries to handle the web server side and the simulation side. On the web server side, `cpp-httplib` [6] is used to create the REST web server to communicate with the client. The `nlohmann-json` [7] library is used to parse the REST requests and generate the json body format for the responses. On the simulation side, the simulated compute platform and Slurm-like batch scheduler are built using the WRENCH developer API. Since the web server blocks when it is waiting for any network requests, the simulation and server parts execute on separate threads. Due to SimGrid’s own and particular usage of threads (recall that WRENCH builds on SimGrid) it is not feasible for the WRENCH simulation to be started from the web server thread. For this reason, the web server has to be run on a different thread than the simulation.

With the usage of separate threads for the simulation and web server, there needs to be a means for these two threads to communicate. Communication between them is primarily done using queue data structures, which store event-to-process objects. This allows any

modifications to the simulation to be passed forward in the order they were received as does any events generated from the simulation. The only exceptions to queue usage is the time synchronization and simulation shutdown between the web server and the simulation, where single variables are used instead. Because queues and variables are not thread-safe when used in this manner, mutex locks are required to prevent race conditions.

The web server operates as the intermediary between the client and simulation keeping track of time and handling all REST endpoints. To keep track of time, instead of incrementing a counter every second starting from zero, it uses a function to retrieve the current time in milliseconds since the UNIX epoch and stores it in a variable when the simulation starts. Using the same function again whenever the simulation or client needs to know the simulated time, it subtracts time it has stored, which should result in the time in milliseconds since the simulation start, in an accurate manner. The API implemented in the server has the following endpoints: “start”, “stop”, “add1”, “add10”, “add60”, “query”, “getQueue”, “addJob”, and “cancelJob”.

The endpoints “add1”, “add10”, “add60”, “start” and “stop” handle the timing in the simulation. The “add1”, “add10” and “add60” endpoints correspond to the buttons in the client that the user can click on to jump forward in (simulated) time by 1, 10 or 60 minutes. The “start” endpoint allows the client to be loaded first before starting the simulation. Otherwise, if the server were to start running first, there will be a slight desynchronization before the client corrects itself. The “stop” endpoint can be used to gracefully end the simulation.

The “query”, “getQueue”, “addJob”, and “cancelJob” endpoints relate to the processing of simulation events and the management of batch jobs. The “query” function serves two purposes: (i) jumping the simulation forward by a second, since it is called by the client every second; and (ii) returning any events completed in the queue from the simulation. The “getQueue” function requests from the simulation the lists of batch jobs running or pending, and returns these jobs and their corresponding information in an array. The “addJob” and “cancelJob” endpoints are, as their names imply, the methods to add a job and submit it to the simulated batch scheduler, or cancel a job that was previously submitted to the batch scheduler.

The simulation side of the server simulates the batch scheduler’s operation. Otherwise, the WRENCH simulation includes a simulated workflow management system (WMS) process in charge of managing batch jobs on behalf of the user. This process is implemented in a way which results in jumping forward in time until an event occurs or the timeout is exceeded. This means the simulation is required to jump forward in time in small increments (hundredth of a second) until it reaches the expected simulation time. Any event occurring during this time frame will be added to a queue to be read by the web server.

For the simulation to manage jobs, any new jobs to be created will be added to the job queue from the web server thread and the simulation thread retrieves them from the queue to start. Jobs initiated by the user are stored within a hashmap hashed to the job name. The job name is the way in which the user can identify its jobs (i.e., as listed in the batch queue in response to the “squeue” Slurm command) or managing its jobs (e.g., passing job names to the “scancel” Slurm command). The simulation itself executes in its own event processing loop. As such, it is oblivious to when a job needs to be removed from the simulation, because it has either failed or successfully completed. For this purpose, these jobs are placed in a “to remove” queue after the web server processes the job completion.

3. Conclusion

This project designed and implemented an interactive, in-the-browser, simulation targeting the hands-on method to teach the principle and practice batch scheduler interaction. The intent for the project is for this simulation to support a new kind of pedagogic modules as part of the EduWRENCH project.

The project was implemented as a stand-alone program therefore, in its current state, the code remains to be integrated in the EduWRENCH codebase. Doing so will require slight modifications of the server implementation, so it can be started from an EduWRENCH Web page, but the client-server interaction currently implemented would be completely retained. Furthermore, more features will likely need to be added, so as to provide users with a more complete terminal (e.g., new bash commands, up arrow to recall the last command, man pages, better error handling).

To take this project to completion, I relied on my previous experience with C++ and several libraries. Certain libraries I had no previous experience using and were significantly more complex than libraries I have previously incorporated in other projects, providing a practical example for me to delve into the documentation to piece together what needs to be done. Particularly, the xterm.js library had minimal tutorials and lackluster documentation for certain functionality requiring me to look through the source code to comprehend what the functions were doing. Through the development process, I have also become more familiar with the CMake build environment, which should serve as a useful experience for my future career. Another part of the learning experience was the development of a simulation of a computer system, which I had never done before necessitating me to read summaries and documentation of tools similar to what I was doing and how they were implemented to serve as an example for me.

The design and implementation of this project opens the way for future EduWRENCH modules, specifically modules that will include simulation activities relying on the use of interactive terminals. Indeed, while the current EduWRENCH modules focus primarily on abstract principles, the batch-scheduling module to be developed based on this work will be the first EduWRENCH module targeting the use of a practical tool, such as Slurm. With many other such tools providing command-line interfaces and becoming standard for parallel and distributed computing, EduWRENCH plans to provide modules targeting these tools. Examples of these tools include Hadoop, Spark, HTCondor, etc.

A future direction for development for this and future simulation-driven EduWRENCH modules would be to use WebAssembly instead of using a web server and client-based system. WebAssembly, also known as Wasm, is a binary instruction format enabling compute-intensive code to be run in any browser [8]. Since it can be compiled from C++ and other LLVM-supported languages, it should be able to support compiling these types of projects. The idea would then be to run the simulation itself in the browser. However, currently Wasm might not be able to handle compiling WRENCH, likely because SimGrid requires the use of threads in a way that may not be supported by Wasm. If this issue is resolved in the future, rather than communicating with a web server, the simulation can be shifted completely to the client's browser. This would greatly simplify the design and implementation.

References

- [1] R. Tanaka, R. Ferreira da Silva and H. Casanova, "Teaching Parallel and Distributed Computing Concepts in Simulation with WRENCH," 2019 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC), 2019, pp. 1-9, doi: 10.1109/EduHPC49559.2019.00006.
- [2] eduWRENCH. [Online] Available: <https://wrench.isi.edu/> [Accessed April 16, 2021].
- [3] SimGrid. "The Modern Age of Computer Systems Simulation." [Online] Available: <https://simgrid.org/doc/latest/> [Accessed April 16, 2021].
- [4] Slurm Workload Manager. "Overview." [Online] Available: <https://slurm.schedmd.com/overview.html> [Accessed April 16, 2021].
- [5] Xterm.js. [Online] Available: <https://xtermjs.org/> [Accessed April 16, 2021].
- [6] cpp-http-lib. v0.7.5. [Online] Available: <https://github.com/yhirose/cpp-http-lib> [Accessed April 16, 2021].
- [7] JSON for Modern C++. [Online] Available: <https://github.com/nlohmann/json> [Accessed April 16, 2021].
- [8] WebAssembly. [Online] Available: <https://webassembly.org/> [Accessed April 16, 2021].