P^2RUTOR: A Programming Tutor for Parallel Programming

Deepak B. Hegde, Preeti Malakar, Amey Karkare
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, India
{deepakbh, pmalakar, karkare}@cse.iitk.ac.in

***Abstract*—We present an enhanced parallel programming tutor (P^2RUTOR) for parallel programming assignment evaluation system. This is based on an existing programming tutor, PRUTOR, which is a comprehensive programming interface and evaluation system. Automatic evaluation of programming assignments for large enrollment courses is helpful for the instructors of introductory programming courses. Evaluation of parallel coding assignments (on several compute nodes) of large enrollment courses on introductory parallel programming is more challenging. Thus a comprehensive interface to develop, test and execute MPI C programs is useful for instructors who teach distributed memory programming related courses.**

***Index Terms*—Parallel programming assignments, MPI, PRUTOR, Web interface**

I. Introduction

In today's exascale era, it is imperative to impart education on parallel and distributed computing (PDC) to future generations from the beginning of undergraduate study. There are numerous efforts around the world to adopt parallel and distributed computing courses as core courses in the undergraduate curriculum [7], [8], [15]. One of the hurdles in imparting parallel computing education in a hands-on approach may be unfamiliarity with the hardware resources in the early years (freshmen/sophomore). Compute resources are the skeletons of learning large-scale parallel programming [12], [15]. However, learning to program message passing based large-scale parallel programming paradigms requires access to compute clusters and familiarity with basic Unix shell commands [16]. Students may be more familiar or comfortable in coding from graphical interfaces rather than operating via Unix-based terminals [15]. Programmers who want to learn parallel programming may find it easier to develop and submit their program on the cluster from a web or graphical interface [12], [15], [16].

Another challenge that instructors of PDC courses may often encounter is assignment evaluation on multiple compute nodes. This is required for programming assignments using message passing interface where the assignment often requires scaling the parallel algorithm on multiple nodes and cores. This necessitates in-house development of automated scripts to collect the submitted assignments through some portal (for e.g. GitHub) followed by verifying the correctness of the code via execution and manual code inspection, followed by grading. This is often a cumbersome task and hinders productivity in case of large enrollments.

In this work, we present a unified platform Parallel Programming Tutor (P^2RUTOR) that enables development of parallel codes (based on MPI), compilation and execution on multiple nodes as well as evaluation by the instructor/teaching assistant. P^2RUTOR is developed on top of PRUTOR [6], [9] which is an existing programming tutor and assignment management system. It currently supports `C`, `C++`, and `Python` as primary languages. However it lacks support for parallel programming, especially distributed memory programming models such as the Message Passing Interface (MPI).

One of the main concerns of instructors is availability of sufficient hardware resources during the course [5]. Supercomputers have high queue waiting times which renders them as a less preferred option while teaching basic parallel programming using MPI. Also, access to supercomputers for course instructors and students are subjected to availability and current load on the systems. Some of the publicly available resources are also limited and often charged; for example, Google Colab provides only up to 2 cores [11].

A viable resource for teaching PDC in computer science departments is the department computing infrastructure which in many cases include at least a few tens of compute nodes. However, these are shared resources which may be used by several students of the department for other courses. Thus an intelligent scheduler for managing resources in the presence of dynamic load is required. Our unified tool, P^2RUTOR, can automatically schedule and launch MPI jobs on the required number of nodes and required number of cores per node. This is especially useful in computer laboratories, shared or dedicated clusters where a heavyweight job scheduler such as SLURM is not deployed and not widely used by diverse users who may be running sequential jobs. The department computing infrastructure is shared among all students, research staff and faculty members. This is a common scenario in academic computing laboratories which may be shared across all years and disciplines.

P^2RUTOR is integrated with a lightweight node allocator [10] that has been extensively tested on shared compute laboratory. This schedules jobs on less loaded nodes of the laboratory/cluster and thus improves performance of the parallel MPI code. We describe our work to integrate Node Allocator with PRUTOR to support MPI programming on clusters including those that may have variable load. The resulting system, called P^2RUTOR is used to teach parallel programming to a class of 30 students. We present our development of P^2RUTOR, experiments and experience of using P^2RUTOR for the class.

II. Background and Related Work

A. PRUTOR Architecture

PRUTOR (PRogramming tUTOR) is an innovative web-based tutoring framework to facilitate teaching introductory programming courses [6]. It offers an integrated development environment (IDE) to write, test and execute programs. Additionally, PRUTOR provides a comprehensive grading and feedback system, allowing instructors to provide feedback and grade students' submissions. PRUTOR presents an useful interface that shows the progress of students' programs, coupled with visualizations that illuminate their problem-solving approaches. PRUTOR also incorporates features to help identify students who may require additional support in programming [9]. This is especially helpful for instructors of large enrollment courses. The platform's successful implementation at the Indian Institute of Technology Kanpur (IIT Kanpur) demonstrates its efficacy in teaching introductory `C` programming. It has also been used for programming in C++ and Python. Its adaptable design allows for future integration with other programming languages.

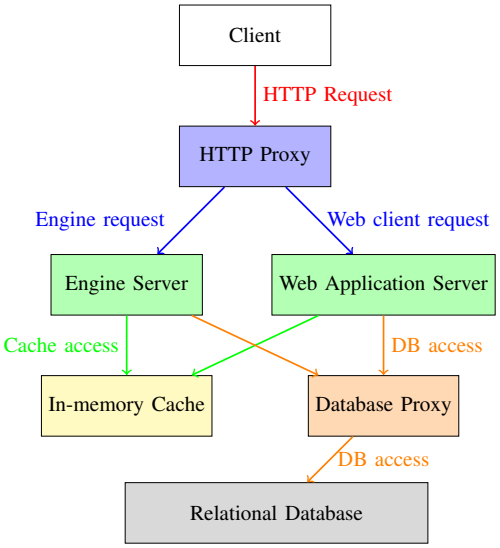

Fig. 1: PRUTOR Component Architecture

PRUTOR comprises diverse services, some are accessible to the client users and others reserved for internal use by other services. Docker containerization [14] enables smooth scalability of these services. The service discovery layer acts as a communication facilitator; it enables automatic and dynamic discovery of available services. The virtualization layer manages and allocates computing resources like CPU, memory, and storage to the various components and services within the tutoring system. The bottom layer is the hardware layer. Das et al. [6] describes the details of the PRUTOR architecture. The tutoring system consists of six primary components; these are briefly described below and depicted in Figure 1:

a. HTTP proxy: It redirects requests based on URL rules and employs the least connected load balancing algorithm to distribute requests among available instances.
b. Database proxy: This directs database access requests to available instances, using the least connected load balancing algorithm.
c. Relational database: This component serves as the central data storage for various types of information, such as user accounts, code history, programming problems, course events, grading details, logs, and system configurations.
d. In-memory cache: This component manages session storage and database caching. It utilizes in-memory storage for sessions to achieve optimal performance, as authentication is required for majority of web service requests.
e. Web application: This component delivers user interfaces and services for the web application. Multiple instances of this component are deployed to enhance performance, with each instance running on a dedicated thread.
f. Engine: This component handles compilation, execution, and evaluation requests, as well as tool execution requests within PRUTOR.

PRUTOR offers a range of user roles as listed below. The privilege of each role varies and may be set as desired.

a. Student
b. Teaching Assistant
c. Tutor
d. Instructor
e. Administrator

B. Node Allocator

Node allocator [10] is a simple job scheduler used for allocating nodes to MPI programs on shared/unmanaged clusters. This is based on the current network load and compute node load (processing and memory) which may vary significantly in a shared cluster. Node allocator is able to reduce the execution times by more than 30% in a shared cluster where the load (available from `psutil` [4]) can dynamically vary due to shared nodes and shared network. The allocator uses system parameters such as available memory, network bandwidth and latency, and CPU speed and load to determine the best resources (compute nodes) for an MPI job. The resource allocation employs a greedy algorithm to identify several candidate set of nodes (sub-graphs) given a required number of nodes by the user. Then we calculate the weighted sum of compute and network cost for each candidate and select the candidate set of nodes (sub-graph) that exhibits the minimum cost. This ensures efficient execution of MPI programs on a shared cluster with variable external load.

Figure 2 shows an overview of the tool. Lightweight daemons are used to periodically collect the real-time system parameters that vary dynamically, such as load, network bandwidth, latency, list of active (reachable) compute nodes. These are represented using circles around Central Monitor in the figure. The consolidated system state is stored in a file on a network-mounted filesystem which is accessible from all compute nodes. The node allocator accesses this information to generate a list of best compute nodes (Hostfile) which is used to execute the MPI program. Note that the user has to run the tool to obtain the list of best compute resources in the shared cluster.

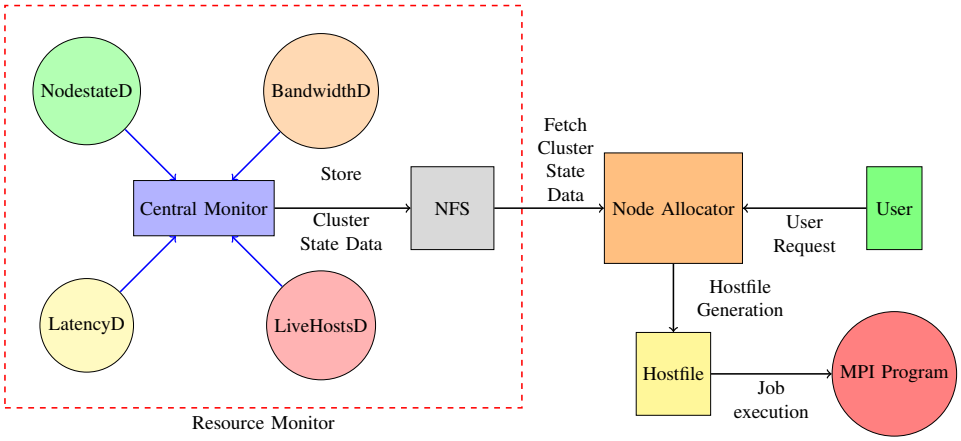

Fig. 2: Node allocator system workflow

C. Related Work

There exists prior work on developing web interfaces for various aspects of parallel programming. Lin et al. [12] presents a web interface for file manipulations. Sukhoroslov et al. [16], [17] present an easy-to-use interface to upload files, specify the number of processes and wall-clock time required. In contrast, our work P^2RUTOR also includes a web-based code editor for MPI and allows compilation and execution from the same interface. P^2RUTOR provides a unified code editor and has the capability to launch MPI programs from the browser

Ngo et al. [15] developed Jupyter notebooks for coding MPI programs in Python. The focus of this work was enabling JupyterHub for teaching PDC courses. In contrast, PRUTOR supports both C and Python, although we have extensively tested P^2RUTOR with C only. Our additional objective was simplifying and streamlining MPI assignment evaluation and verification using P^2RUTOR.

Several efforts exist in adopting and standardizing parallel and distributed computing curriculum [1]. This paper focuses on presenting our unified platform P^2RUTOR for simplifying coding for students and assignment evaluations for instructors.

III. P^2RUTOR: PRUTOR for MPI Programming

In this section, we present our work on extending PRUTOR to add support for MPI (Message Passing Interface) C and C++ programs. P^2RUTOR consists of a specialized wrapper that extends PRUTOR to handle MPI programs seamlessly along with the existing support for C, C++, and Python. We also integrated the node allocator tool (described in Section II-B) to optimize the allocation of compute nodes for execution of MPI programs on unmanaged clusters such as those available in computer laboratories of academic institutions. Our enhancements enable users (students) to develop MPI programs using P^2RUTOR IDE, test and compile their codes and execute from P^2RUTOR. P^2RUTOR internally execute the program on desired number of processes as specified by the users. We enhanced the PRUTOR engine server (see Figure 1), enabling it to handle MPI requests and efficiently allocate compute nodes for distributed-memory parallel programs. The modifications involve integrating a specialized module to support MPI requests seamlessly.

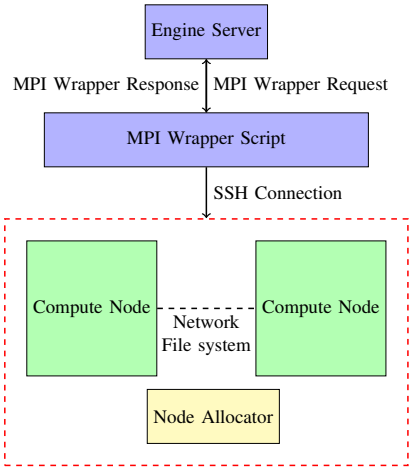

Fig. 3: Workflow of executing MPI programs via PRUTOR

A. P^2RUTOR Workflow, Design and Architecture

Figure 3 illustrates the extensions required to execute MPI programs via P^2RUTOR. We added a wrapper to manage all MPI-related requests. The wrapper assumes that the engine has access to a cluster of compute nodes that will be used for executing MPI programs. Users have the option of providing information about the number of processes `-np` and process per node `-ppn`. Users may also provide application-specific input parameters. We have the option to provide a default value of `-np` and `-ppn`. The parameters can also be explicitly specified by the user in their source code. We will demonstrate the P^2RUTOR GUI and coding and development in the next section. Users can develop their assignments on P^2RUTOR, and compile and execute from P^2RUTOR. In the backend, the wrapper and the engine ensures that the code is executed on the designated cluster. When a user compiles and runs her MPI code, the wrapper in the PRUTOR engine performs the following sequence of actions:

1) The wrapper package receives a request from the engine, and subsequently, it establishes an SSH connection with one of the compute nodes, designated as the login node. The purpose of this login node is to function as the initial access point for subsequent execution operations. The login node is chosen at random from a small subset of compute nodes. Subsequently, the necessary files for execution, such as the source code, is transferred to a predetermined location (a designated P^2RUTOR directory `execution_id`), where `id` is a unique identifier.
2) The wrapper proceeds with the compilation phase. The `mpicc` compiler is used to compile the user's MPI source code. This produces the corresponding MPI object code in the same P^2RUTOR directory `execution_id`. This ensures that all the compiled output is available in the proper location for subsequent MPI code execution. Note that here we used the `mpicc` compiler of MPICH [2]. Any of the available MPI implementations/installations may be used to compile and execute on the cluster.
3) The user may develop, debug, compile and execute her code from the P^2RUTOR GUI as explained in the next section. This triggers an MPI job execution request. Upon receiving this request, the wrapper module parses the necessary information regarding the total number of processes and the number of processes per node that are required for execution. This may be specified explicitly by the user or may also be set to a default value at the backend. This information is used by the Node Allocator to generate a file `hosts_<np>_<ppn>.txt` in the designated P^2RUTOR directory `execution_id`. This file contains the most efficient compute nodes for executing the MPI code as explained in Section II-B.
4) The wrapper executes the compiled code over the established SSH connection on the login node by running the below command:
   ```
   mpirun -np <number of processes>
   -ppn <process per node> -f
   hosts_<np>_<ppn>.txt <params>
   ```
   The output of the execution is stored in a separate log file in the designated P^2RUTOR directory `execution_id` and the response is sent back to the HTTP proxy server. The output is then displayed on P^2RUTOR's interface (user's output terminal view).
5) In the event of compilation or execution failure, the exact error response is sent to the client server and displayed on the interface. P^2RUTOR incorporates a pre-established time-out threshold to address scenarios wherein the code enters an infinite loop or experiences prolonged execution times. If the execution surpasses this threshold, the user will be presented with an error message depending on the case. Additionally, we also deploy a monitor (zombie process monitor) that continuously checks for any defunct MPI processes that were not properly terminated in case of a time-out situation.

The entire process described above is automated through our wrapper script. This does not require any manual intervention. The users (students) do not require to manually log in to any of the compute nodes in order to run the jobs. The nodes are also automatically selected and the jobs are run, the output is displayed to the user through P^2RUTOR's web interface.

B. MPI Programming via P^2RUTOR

Figure 4 shows P^2RUTOR's web interface that is derived from PRUTOR interface. On the top left corner, there exists usual file menu options (under `File`) along with code compile and execute option (under `Run`). The left panel displays the user's directory structure. When an assignment is released, the left panel displays the assignment problem. The right panel displays the code written by the user. We have shown a simple MPI code in the figure. This code prints the rank, and the `hostname` along with two parameters that have been passed using #pragma directive (line 7 in the editor window). The total number of processes and the processes per node may also be specified using #pragma directive (line 6 in the editor window).

The output of the code is displayed in the bottom console as shown in Figure 4. The output simply prints the ranks 0, 1, 2, 3 and the host on which the ranks are running (`csews1` and `csews24`) in this case. Error message, if any, is also displayed similarly. Note that clicking the compile/execute button triggers the sequence of actions described in Section III-A. Primarily, this involves establishing a SSH connection with the cluster, compiling the code, invoking Node Allocator and executing the code on the assigned nodes.

When several users execute their code simultaneously, the wrapper establishes several simultaneous connections with the cluster login nodes. This is a common scenario during programming laboratories or near assignment submission deadline dates. We keep a detailed count of number of jobs launched and the number of nodes used for the current jobs. We also ensure that there are sufficient free nodes available for execution. If this is not the case, the wrapper waits till nodes are available. This is similar to job scheduling on supercomputers. Note that supercomputers are highly sought-after by researchers for production parallel code execution. It may not be always feasible to learn parallel programming on supercomputers. Learning requires a lot of testing and debugging for which million-dollar supercomputers may not be the right resource. Thus we developed P^2RUTOR for MPI to simplify coding, testing, learning and evaluating MPI programs.

One of the main goals of our work is to streamline assignment submission for large enrollment classes. P^2RUTOR allows creation and management of assignments. The instructor can create an assignment and release it for the desired duration. Students have an option to save draft code as well as submit the final code (from the File menu option). Each submitted assignment is assigned a submission identifier. This is tagged with the student's user identifier which helps the instructor to evaluate assignments. Every code state that is saved by the students is also stored in the P^2RUTOR database similar

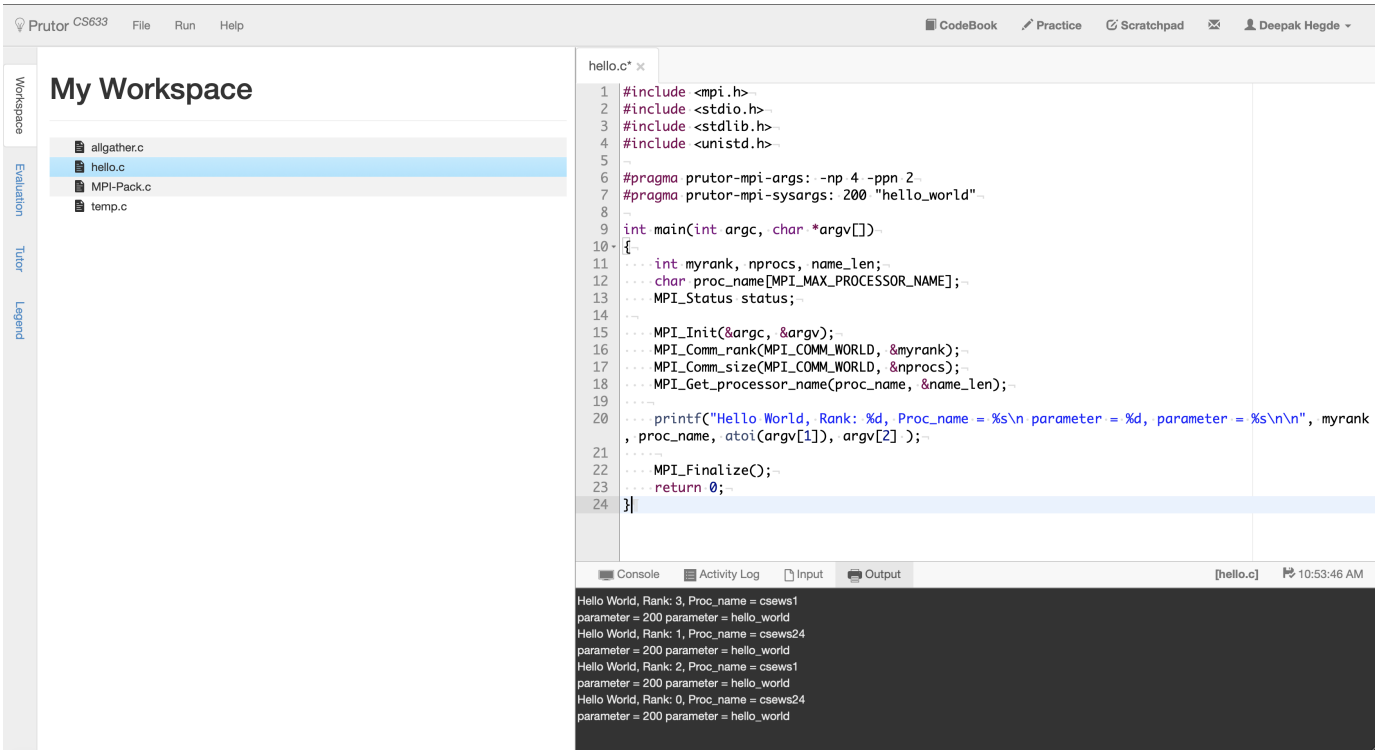

Fig. 4: P^2RUTOR Web Interface for MPI Programming.

to a time-stamped event. This also helps the instructor in understanding how the student develops code logic and finally completes the code. An example is shown in Figure 7. We will explain this in detail in Section IV-C.

C. Setup and Installation

Figure 5 summarizes the setup and installation required to use P^2RUTOR to execute MPI programming assignments in a compute cluster of any academic institution. P^2RUTOR can be easily set up on any local server of the department. Some of the essential libraries are shown in the figure. Any MPI implementation may be installed on the cluster where the administrator/instructor desires to execute the student submissions. We used the compute facilities of the Department of Computer Science and Engineering (CSE), IIT Kanpur. The detailed configurations of the systems are specified in Section IV-A. The Node Allocator component is optional in case the cluster is shared among external users. In our case, the external users were students of the CSE department or other departments who required access to compute nodes for simple tasks such as web browsing, document editing, basic C programming by freshmen/first year students. The Node Allocator is not required if there is no variable load in the cluster. The Appendix lists the steps to install Node Allocator on a cluster (small/medium/big).

IV. Evaluation

In this section, we describe the conduct of assignment submissions during our course, the development timeline, the assignment submission interface and the job submission statistics.

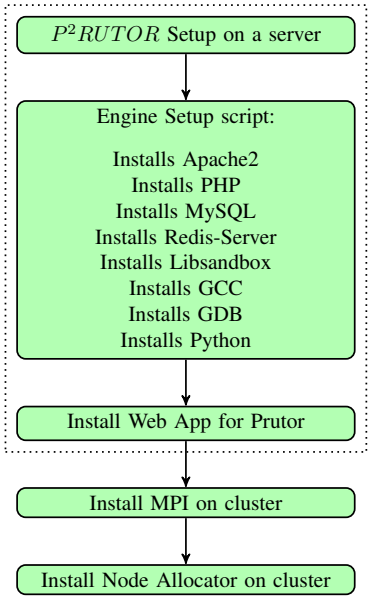

Fig. 5: Workflow of using PRUTOR for MPI Program Development and Execution on a Cluster.

A. Experimental Setup

We used P^2RUTOR for the course `CS633` on "Parallel Computing" at IIT Kanpur during the Spring Semester of 2023. This course covers parallel programming based on message passing interface as well as parallel communication algorithms. A detailed syllabus can be found in [13]. There were 30 enrollments with an equal mix of third-year and fourth-year students undergraduate students and two post-

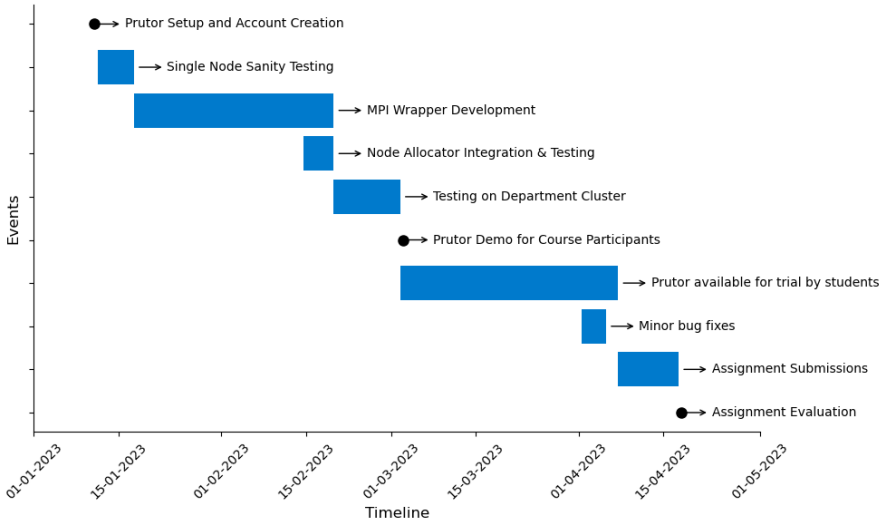

Fig. 6: Development Timeline of P^2RUTOR for MPI between January – April 2023.

graduate students. We leveraged 50 compute nodes in the compute laboratory of the Department of Computer Science and Engineering (CSE), Indian Institute of Technology (IIT Kanpur) as our cluster. Each Intel® Core™ i7-8700 node has 12 logical cores and 16 GB memory. The nodes are connected in a tree-like topology via 1 Gbps Ethernet. We installed MPICH [2] and the Hydra process manager in the cluster. All compute nodes share a common Network File System.

CS633 is aimed at teaching basic MPI programming, where students learn and develop assignments based on point-to-point and collective communications. 50 compute nodes sufficed in our case. PRUTOR has been used in introductory `C` and `Python` programming in classes with 600+ enrollments. P^2RUTOR is an extension of PRUTOR, thus will be able to handle such large courses. However, the instructor should use the required number of nodes as per the class strength.

P^2RUTOR requires password-less SSH to the login node from where the programs are executed. MPI program execution also requires password-less SSH in the cluster. We created a user account `evaluator` on the cluster with the help of system administrators. Students accounts were created on the P^2RUTOR server using which the students logged into P^2RUTOR. When the students execute their programs from P^2RUTOR, the programs run on the cluster under the user account `evaluator`. Thus individual accounts in the cluster were not required. All jobs submitted from P^2RUTOR run with this user name. However, we use the user's submission identifier to differentiate between the jobs of different users.

The wrapper transfers the user's code to a launch node (we have a set of launch/login nodes) and the compilation and execution phases complete as explained in Section III-A. The home directory of `evaluator` is NFS-mounted on all compute nodes. We did not require users to directly access the cluster, the students were able to test their MPI programs on desired number of nodes and desired number of processes per node from P^2RUTOR. It is possible that several users run programs simultaneously from P^2RUTOR. We added a lightweight waiting mechanism in our script that automatically checks the current number of jobs in execution and determines whether there are enough free nodes to schedule the next job. We enforced a strict limit during the assignment submission durations when the students collected timing data from their runs. We relaxed this limitation during testing and development phases by students.

B. Development Timeline

This tool was developed during the course that was offered between January – April 2023. Figure 6 illustrates the development and testing timeline of P^2RUTOR for MPI. The x-axis shows the timeline. The y-axis shows various events. The events shown in a black circle spanned only 1 day. The blue bars show the extent of the event (in days). This primarily includes:

1) Development of wrapper for P^2RUTOR: The major development phase happened during the first four weeks.
2) Integration of node allocator: We integrated Node Allocator with the wrapper so that the user's code is executed on an optimal set of compute nodes.
3) Testing on 50 compute nodes: We performed testing on several nodes and node configurations using #pragma directive as well as using an execution script for assignment submissions.
4) Test and development by students: The students were familiarized with the platform and its usage.
5) Assignment submission by students: We evaluated the last assignment of the course using P^2RUTOR, we specify its details in the next section.

C. Assignment Evaluation on P^2RUTOR

In this section, we elaborate the details of assignment submission and evaluation during April 8 – 17, 2023. The assignment was to test concepts of 1D and 2D domain decompositions and to develop understanding of efficient

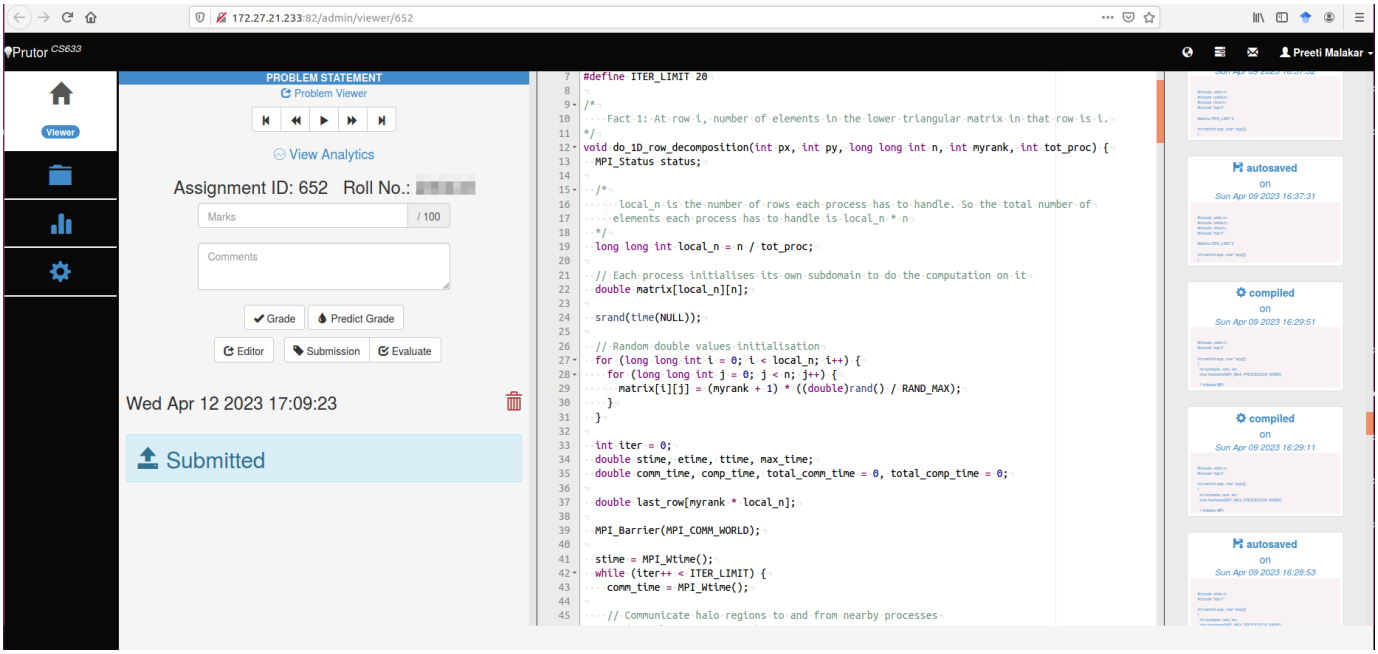

Fig. 7: MPI Assignment Submission on P^2RUTOR.

parallelization of a simple kernel (Gaussian elimination in this case). The submission requirements were scaling studies on up to 16 processes for different matrix sizes.

We created an assignment submission script that automatically executed the student's code on the desired configurations as required by the assignment. This is an alternate option to specifying the input arguments using #pragma as was explained in Figure 4. Once the student clicks the Run option, her code is compiled and executed on the cluster on the most optimal set of nodes in the designated cluster.

Figure 7 shows one of the submitted assignments which the instructor or teaching assistants may review later. This is a different view of the same interface shown in Figure 4. The leftmost panel shows various options available to the administrators. The second panel from the left shows some assignment statistics such as the assignment identifier, the roll number of the student (blurred to preserve identity of the student), the time of submission, and a simple interface to grade the evaluation. The assignment may be auto-graded if test cases and the actual output are specified while designing the question. The grades may be revised using this same interface in case of assignments that may include a manual evaluation component. The third panel from the left shows the code editor. The rightmost panel shows snapshot of the user's code. Snapshots are autosaved every 90 seconds (this is configurable). Snapshots also show the compilation times and submission times. This is helpful for the course evaluator.

Figure 8 shows the job runtimes of 663 job (assignment) submissions during April 8 – 17, 2023. The x-axis shows the timeline, the y-axis shows the job runtimes in seconds. The maximum runtime was 950 seconds. We observe a few runtimes below 10 seconds, these were erroneous jobs that may have failed at runtime. A student may submit multiple times. The last submission of the student is considered as the final submission to be evaluated. The due date of this assignment was April 15, which explains the maximum number of submission during April 11 – 13 because of early submission credit points. We also observed about 300 failed submissions, some of which were due to bad terminations. All students successfully submitted their assignments using P^2RUTOR.

V. Conclusions and Future Work

We demonstrated P^2RUTOR, which extends PRUTOR for streamlining MPI programming assignment submission and evaluation. This reduces the burden of manually evaluating the correct execution of code by teaching assistants and instructors. This also simplifies learning, developing, testing, debugging and assignment submission by students. We found this to be immensely helpful during the previous offering (Spring 2023 semester) of the course on Parallel Computing at IIT Kanpur. We plan to regularly use this interface in all future offerings of this course.

One of the current challenges in using P^2RUTOR in an unmanaged cluster environment is the dependency on availability of sufficient number of nodes during the peak assignment submission phase and multiple SSH connections from the server node of P^2RUTOR. This may affect the user experience of job submissions via P^2RUTOR in case of large enrollment courses because currently we have only one node as the P^2RUTOR server. We plan to address this in future.

VI. Acknowledgements

We thank Abhishek Reddy for his help with running Node Allocator and assignment evaluations. We also thank the laboratory staff of the Department of Computer Science and Engineering, IIT Kanpur for enabling access to the compute laboratory 24×7.

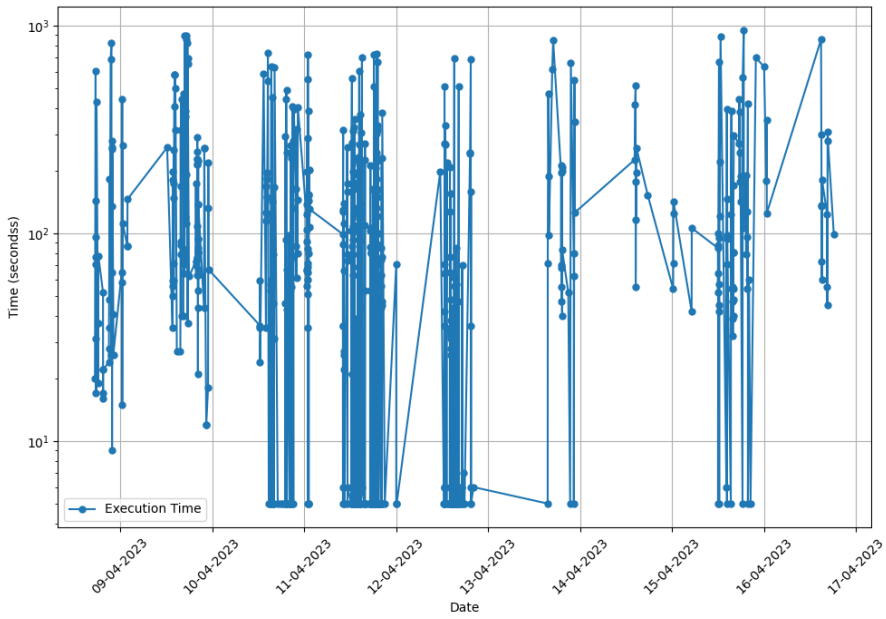

Fig. 8: Jobs Submitted during April 8 – 17, 2023 on P^2RUTOR.

REFERENCES

[1] CsInParallel: Parallel Computing in the Computer Science Curriculum. https://csinparallel.org, 2023.

[2] MPICH Project Homepage. https://www.mpich.org/, 2023.

[3] Node Allocator. https://github.com/deepakbh21/Node-allocator, 2023.

[4] psutil Library. https://psutil.readthedocs.io, 2023.

[5] David W. Brown, Vitaly Ford, and Sheikh K. Ghafoor. A Framework for the Evaluation of Parallel and Distributed Computing Educational Resources. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 261–268, 2020.

[6] Rajdeep Das, Umair Z. Ahmed, Amey Karkare, and Sumit Gulwani. PRUTOR: A System for Tutoring CS1 and Collecting Student Programs for Analysis. https://arxiv.org/abs/1608.03828, 2016.

[7] Prasun Dewan, Andrew Worley, Samuel George, Felipe Yanaga, Andrew Wortas, James Juschuk, Mike Rogers, and Sheikh Ghafoor. Hands-On, Instructor-Light, Checked and Tracked Training of Trainers in Java Fork-Join Abstractions. In *2022 IEEE 29th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW)*, pages 28–35, 2022.

[8] Alec Goncharow, Anna Boekelheide, Matthew Mcquaigue, David Burlinson, Erik Saule, Kalpathi Subramanian, and Jamie Payton. Classifying pedagogical material to improve adoption of parallel and distributed computing topics. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 312–319. IEEE, 2019.

[9] Amey Karkare and Purushottam Kar. PRUTOR: An Intelligent Learning and Management System for Programming Courses. *Commun. ACM*, 65(11):62–64, Oct 2022.

[10] Ashish Kumar, Naman Jain, and Preeti Malakar. Network and Load-Aware Resource Manager for MPI Programs. In *Workshop Proceedings of the 49th International Conference on Parallel Processing*, ICPP Workshops '20, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Alina Lazar, Virginia Niculescu, and David P. Bunde. Peachy Parallel Assignments (EduPar 2023). In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 248–255, 2023.

[12] Hong Lin. Teaching Parallel and Distributed Computing Using a Cluster Computing Portal. In *2013 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum*, pages 1312–1317, 2013.

[13] Preeti Malakar. Experiences of Teaching Parallel Computing to Undergraduates and Post-Graduates. In *26th International Conference on High Performance Computing, Data and Analytics Workshop, HiPCW 2019, Hyderabad, India, December 17-20, 2019*, pages 40–47. IEEE, 2019.

[14] Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.*, 2014(239), mar 2014.

[15] Linh B. Ngo, Ashwin Trikuta Srinath, Jeffrey Denton, and Marcin Ziolkowski. Unifying computing resources and access interface to support parallel and distributed computing education. *Journal of Parallel and Distributed Computing*, 118:201–212, 2018.

[16] Oleg Sukhoroslov. Building web-based services for practical exercises in parallel and distributed computing. *Journal of Parallel and Distributed Computing*, 118:177–188, 2018.

[17] Oleg Sukhoroslov, Sergey Volkov, and Alexander Afanasiev. A Web-Based Platform for Publication and Distributed Execution of Computing Applications. In *2015 14th International Symposium on Parallel and Distributed Computing*, pages 175–184, 2015.

APPENDIX

The Node Allocator repository can be cloned from [3]. The steps to configure and run Node allocator is as follows:

1) Clone the repository to the root folder on the nodes and rename it to `UGP`:
```
git clone <repository_url> UGP
```

2) The code repository consists of two folders: `eagle` and `allocator`. The source code is present in the `allocator` folder, while the `eagle` folder contains a number of daemon programs that run in the background on the cluster.

3) To set the list of nodes to be monitored, create a file named `hosts.txt` under the `eagle` directory and add the host names to be monitored
```
cd UGP/eagle
nano hosts.txt
```

4) Launch the monitor using the following command:
```
python3 monitor/monitord.py start
```

5) To manually check whether the daemon is running and, if so, on which node, run the following command:
```
./monitor/globalDaemonStatus.sh
~/.eagle/hosts.txt
```

6) To terminate the monitor daemon and terminate all other associated daemons, execute the following command:
```
python3 monitor/monitord.py stop
```

7) The MPI Wrapper script is responsible for efficiently handling the distribution of nodes and streamlining the process of automatically allocating nodes. This allocation relies on the continuous monitoring of all host nodes by multiple monitor daemons.