SubMIT

Russell Pasetes, Jeremy Cowham, Christian Moroney

Introduction

Currently the MIT class *Computer Systems Engineering* (6.033) implements multiple systems for handling files, submissions, and grades that are problematic for students and staff alike. These problems include but are not limited to: split systems for handling grades, poor implementation for submission sites, and the lack of a system for shared work amongst groups. These problems have plagued the 6.033 community for too long, and a new system needs to be designed. Focused on solving these problems, we design a new system for 6.033, named SubMIT, that utilizes existing systems, but reengineers their implementations and interactions in a way which best meets the needs of students and staff.

A central server that communicates with all students and staff provides the foundation for our system. In designing SubMIT, we identified the core design principles of *simplicity, security,* and *utility*. Simplicity of the system means, for example, enabling students from different backgrounds to adapt quickly to the logistics of the class. We emphasize the intuitiveness of the storage and file management system, promoting safe and effective work by students and staff

alike. Furthermore, the security of SubMIT means avoiding technical failures, ensuring storage safety, and thwarting malicious activity. Finally, the utility of SubMIT means, in essence, addressing significant user needs. This principle means making design decisions which achieve optimal solutions and compromises among the other design principles and tradeoffs. SubMIT facilitates the communication between and among users and infrastructure services to meet the needs of students and staff. Our design focuses on the goals listed and priorities to provide the 6.033 class with a system design that scalably promotes group collaboration, while providing staff with necessary administrative tools.

System Overview

Our system utilizes the infrastructure services at our disposal: the MIT ID service (MIDS) based on the Kerberos system, the MIT File System (MFS), the MIT Sync Service (MSS), the MIT Locking Service (MLS), and Gradescope. Our system establishes a multipurpose platform that supports graded assignments for students in 6.033. Every communication request made by a user, either staff or student, will first be parsed through MIDS in order to coordinate with SubMIT and set proper access parameters. This enforces a secure framework for student and staff interactions.

Figure 1: Layout of System

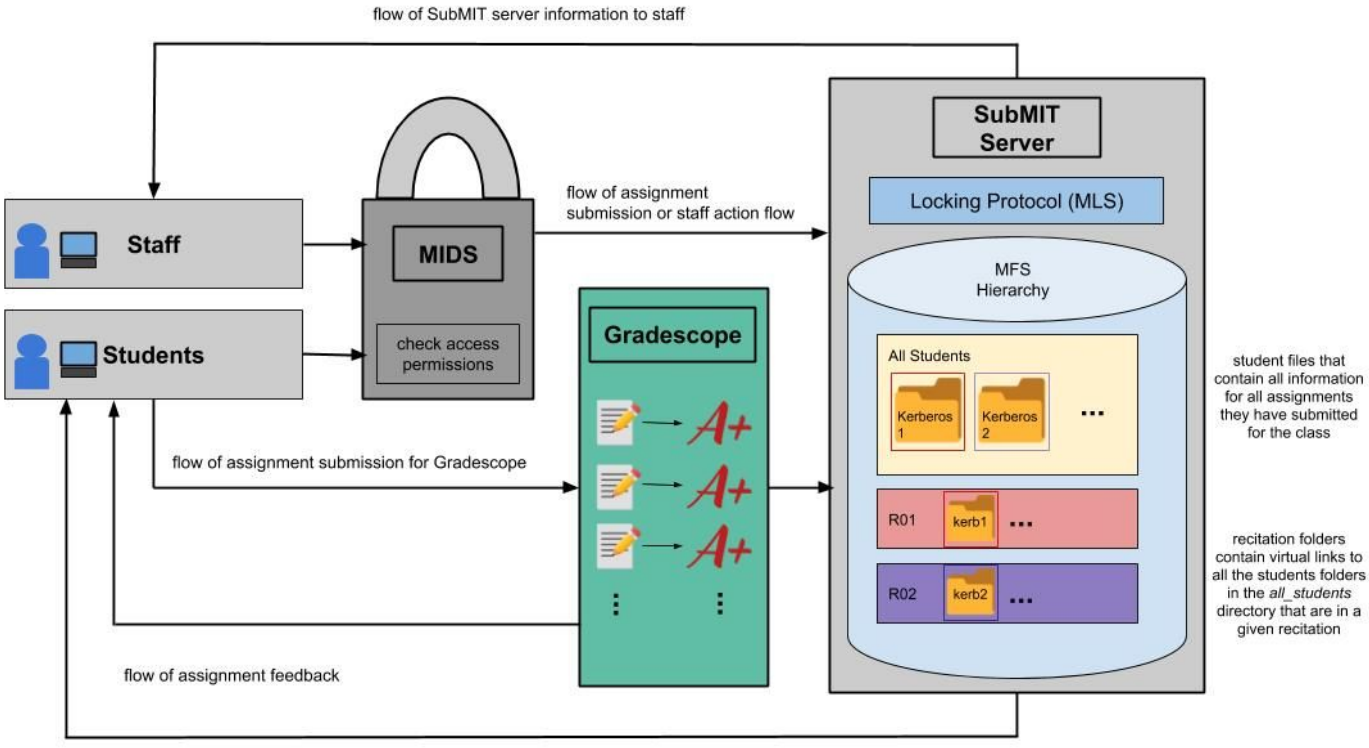

Figure 1: *The main interactions of our system with users. Students and staff access our system via remote computers. Then the MIT ID system (MIDS) validates the access for the requests they make to SubMIT. The MFS file hierarchy can be seen and the MIT Locking Protocol (MLS) dictates who can read or write to any of these files. Each recitation directory can be accessed by the WRAP instructors, Recitation Instructors, and Teaching Assistants that are assigned to a given recitation. Gradescope can periodically report grades to the SubMIT server. Students push assignments to Gradescope and SubMIT while still receiving feedback data from staff via the modules. SubMIT also provides staff with the feedback and information necessary to input feedback back into the system.*

Student Interactions

Assignment Organization & Access

SubMIT provides a hierarchical organization of student work through our implementation of the MIT File system (MFS). At the main directory, an *all_students* subdirectory contains all of the currently registered students' directories within the class. Only the Course Lecturer, Head WRAP Instructor, and Administrative TA have read and write access permissions for the all_students subdirectory, as the staff needs to add or remove students depending on the class' enrollment status. This subdirectory is hidden from all students to ensure that a student cannot access another fellow classmate's directory.

Also within the main directory is a collection of subdirectories for each recitation section. Each recitation subdirectory gives read and write access permissions only to the corresponding Recitation Instructor, Recitation TA, and WRAP Instructors. Within each recitation subdirectory are student directories for all of the students within that recitation section. Assume that recitations assignments are done manually. Each student has read access to their registered recitation section and will have read and write access to their own student directory. The student directories within each recitation subdirectory are actually symbolic links that accesses the student directories within the *all_students* subdirectory. Within the student directory, students can upload files that can be submitted to a given assignment.

Once teams are formed, a team directory is created using *create_dir(kerb1,kerb2,kerb3)* for all Design Project (DP) teams within the recitation section. These team directories can only be created if and only if all team members' individual directories exist within the same recitation section. Furthermore, these team directories exist in the recitation section subdirectory that the students are assigned to. All team members within a DP team are given read and write access to their corresponding team directory. We also place a lower limit on the minimum number of students in a given team. This ensures that there will be multiple teams of three students, with a few edge cases of four students on a DP for the overall class.

Assignment Storage

Inside each student directory, subdirectories are created that hold each individual grading element for the class. For the purposes of 6.033, there exists three subdirectories: reading questions, critiques, and peer reviews. These subdirectories allow for the upload of multiple files if a student decides to submit an assignment more than once. Inside each team directory, subdirectories are created that hold each team grading element for the class: the Design Project Preliminary Report (DPPR) and the Design Project Report (DPR). Similarly, these subdirectories allow for multiple file uploads if a team decides to submit an assignment more than once.

Submission Function

In order to support the submission of assignments, we provide students and DP teams with the function *submit_assignment(filename, directory name),* which gives students the ability to set a file as a submission for an assignment to the given directory if they hold the proper Kerberos ID permissions. These Kerberos ID permission are checked prior to the entrance to SubMIT via the MIDS. This ID check ensures the robustness and security of students work through redundancy, as a technical failure in SubMIT will be already double checked through MIDS. A copy of the file is created within the specified directory and is tagged as the most recent submission with a timestamp of when this function was called. Any previously uploaded file will retain their timestamp tags but only the tag with the most recent timestamp will be recognized as the submission considered for grading. To check whether the submission is considered on-time or late, we store a virtual link to the *most_recent_assignment*, for staff to access, that still falls under the accepted deadline time. This ensures that students are unable to send multiple files after a deadline. In the interest of time, we chose not to prioritize video submissions as we wanted to simplify our file types to strictly text documents, but we plan on adding support for videos in the future.

Team File Sharing

In order to support access permissions for teams, we construct a secure and sensible locking mechanism through via the lock service. The following summarizes the SubMIT locking protocol:

- Locks will be designated as either exclusive or reader/writer.
- An exclusive lock cannot be obtained as a reader nor as a writer, and a reader/writer lock cannot be obtained as an exclusive lock (either user action will alert the user that this is illegal).
- Obtaining an exclusive lock will allow that user, and that user only, both reading and writing abilities.
- Obtaining a write lock will allow that user, and that user only, to write the file associated with that lock. Other users may still obtain read locks for that file.

- Obtaining a read lock will allow that user to read the contents of the associated file.

To enable and encourage collaboration on team projects, we develop check-out system for students working on a design project concurrently. This check-out system aims to strike a compromise between allowing users to edit files concurrently and not allowing users to silently overwrite each other's changes. We utilize the lock system within our SubMIT server to handle this information. Students acquire locks for the given byte sections of the shared file. The lock information is stored in a separate directory in our SubMIT file system. They can write to this and then upload changes to the file, at which point, the lock acquired on the section of bytes will be released. While a given student is editing a section of a file, other students can read the most previously uploaded portion of the file. This feature enables students to work on assignments with other students and completely avoid the difficulties of merge conflicts. This also avoids the complexity of handling real-time edits to a shared file. Furthermore, it is a *simple* solution for students, and allows for effective team work on these shared assignments. This is a key feature which provides simplicity for students interactions with SubMIT.

Figure 2: Shared File Schematic

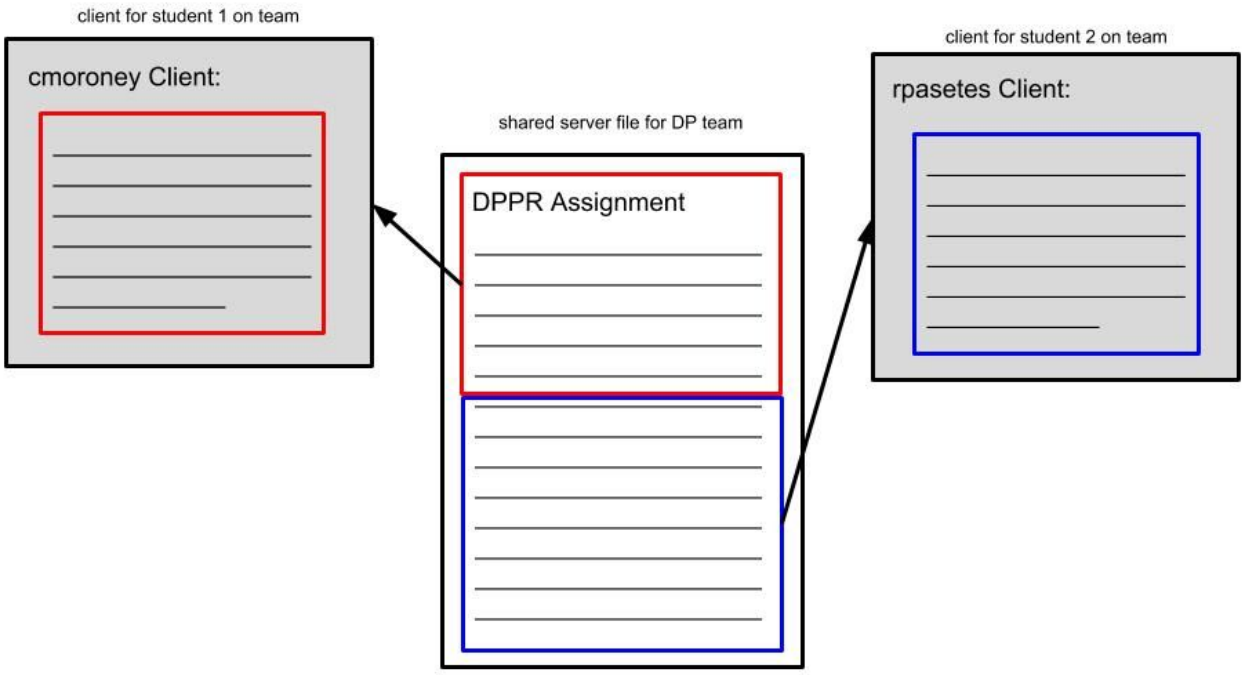

Figure 2: *The shared file in the diagram will exist in a DP's subdirectory within the recitation of the students. Student 1 with kerberos, cmoroney, can select portions of the document to 'check-out' and work on. This prevents any other student, such as student 2 with kerberos rpasetes, with write access to the portions of this document already checked out. The respective*

red and blue sections of the assignment illustrate the sections currently 'checked-out' and the work capable of being done on by separate clients for the same file on separate devices.

Staff Interactions

Grading

With the student assignments stored on our SubMIT server and Course Staff able to access students files with read or write privileges, we provide staff with the function *Assign_grade([student1,student2,...],[grade1,grade2,...],[assignment1,assignment2,...])* which gives the staff the option to upload multiple grades for multiple students for a given assignment. Furthermore, this process can be done in one function call for multiple assignments. We assume that this will never be called for more than 3 assignments at once, as the staff will be doing this multiple times throughout the semester, and will never be uploading a large amount of assignments at once. With staff access to the SubMIT server allowing more broad access to files, we limit their access to only read for student submitted assignments. This ensures privacy and security of student files, and limits staff to grading and commenting of files. Assignments that are uploaded into Gradescope are handled separately, but when a given student's Gradescope ecosystem sees a change or new assignment spawned, we will communicate with the SubMIT server, and in times of low activity in the SubMIT server and send a CSV screenshot of the atmosphere for the student, and update every assignment that was either spawned or changed in our SubMIT system with the Gradescope value. This is a tradeoff we implement as we assume students do not need a live and up-to-date report on all grades for the class.

Commenting

Furthermore, we provide staff with a function *Add_Commment([student1,student2,...], [comment1,comment2,...],[assignment1,assignment2,...])* which enables staff to add multiple comments to multiple students assignments in our SubMIT ecosystem. In our file system we allocate space in each assignment's final submission file that provides an area for comments for *each* type of staff (Recitation Instructor, WRAP Instructor, or TA). This design choice provides a simple solution to the complex problem of concurrent commenting from staff. This function does not attempt to add comments to Gradescope assignments, but rather focuses only on the SubMIT exclusive assignments. The comments and grade updating and changing will be handled strictly within the Gradescope ecosystem. Any comment that is added to a Gradescope file will not be updating the SubMIT grade assignment, as comments and written feedback for Gradescope assignment will not be stored on our SubMIT server, just the grades for these assignment and comments from staff.

Collection of Grades

For staff work we provide a function, *collect_all_grades(assignment)*, which pulls all data from the SubMIT server. This call initially takes in the assignment, and then communicate with the server that is requesting the information on this assignment. Our SubMIT server is always up to date with the latest information regarding any assignment submitted on SubMIT. With this in mind we then iterate across all students in the class and perform a lookup of their grade for the given assignment. Our SubMIT server is able to communicate with the Gradescope system to pull assignments that are contained within the Gradescope ecosystem. This is done by using the *Pull_gradescope_grades()* and then iterating across these values until the particular assignment is found. However, this updating of grades from Gradescope is not performed at every point in time that grades are submitted to Gradescope, as we incur a 10 second cost of processing time at every call. Instead, we update SubMIT periodically in times of low processing activity, such as every other morning at 4am, with Gradescope grades to avoid this penalty. Prior to the call of this function, there is a check to ensure that the identity of the caller is a staff member kerberos. This utilizes the capabilities of the identity system to ensure students do not have access to their peer's grades.

SubMIT Processing

Overview

Major design components of SubMIT we provide are: sensible access management and resource allocation, whether in terms of computation or storage. SubMIT adopts a balanced approach in this arena, generally providing freedom to users without that freedom infringing on standards of privacy or fairness.

Computing Resources

SubMIT addresses the challenge of sharing of computational resources. On one extreme, we could restrict each user or group to be be able to use *1/(#users)th* of the total computing resources at all times. This would ensure that anyone could use the server computation resources at any time, but would be extremely slow and wasteful of computing resources. On the other hand, we could let anyone use as much computational resources as one wished, but this would potentially allow some malicious actor to adversely affect everyone else. To find some middle ground, SubMIT uses a computational resource allocation scheme typical to modern operating systems: process switching. Each user on the system is allowed to run one process on the server at a time (a file upload, a file download, etc.). In addition, the current computing resources are split evenly among all active users with no preference being given to any user. This also means, however, that there may be many more processes than processors at any given point in time, which is where the switching comes in. Processes will be switched at some interval on the order of milliseconds. This interval strikes a balance between

minimizing overhead timer costs (jeopardized if intervals are too short) and ensuring fair allocation of computing time (jeopardized if intervals are too long).

Storage Resources

Storage resources need to be allocated a bit differently than computational resources, as storage loads are generally (1) less variable and (2) more enduring than computational loads. As a principle of fairness, no user can be allocated more storage than an equivalent peer; however, it is possible that different classes of users are allocated different amounts of storage (e.g. administrators being allotted more storage than students). Exact computations for proposed storage allotments, like computation switching intervals, will be calculated and reported in the final report.

Conclusion

SubMIT utilizes a file system hierarchy with limited access via security at every level, to provide students and staff with a simple system that handles all work and supports staff feedback for grading. By limiting access to files and utilizing our checkout system for student assignment work our system enables students to confirm confidence in the safety and privacy of their work. Furthermore, our SubMIT server hides many of the logistical complexities from students such as dividing them up into recitations and tutorial sections. With the structure and modularity of SubMIT server we deliver a *simple* and *secure* framework for students, while still providing staff the tools necessary to address needs for operation of the class, and overall *utility* for both users. Future iterations of SubMIT will uphold current design principles and develop more *efficient* and *robust* storage and sync services to improve the quality of users utility.