CourseHub: A MUGS Design

Alejandro Diaz, Katharina Gschwind, Dylan Lewis
{addiaz15, gschwind, drlewis}@mit.edu
Rec. Instructor: Steve Bauer, 2pm, Rec12

May 2019

Contents

1 Introduction

MIT is a leading technological university, but it lacks a coherent system for courses that supports student submissions and grading in a comprehensive manner. MIT has decided to update 6.033's (Computer Engineering) grading and submissions system necessary for the course of 400 students with many different recitations, tutorials, and staff. Provided the system is flexible enough, MIT would like to extend the system on a campus wide scale to other courses.

On a system-wide scale the current 6.033 system's problems include:

1. Accommodating the specific hierarchy and layout of 6.033 (WRAP instructors and TAs) while being flexible enough to work for other course structures (e.g. labs, recitations).
2. Differentiating between students and staff with different permissions and capabilities that are related to their identities and memberships in groups.
3. Allowing file submissions of group and student work and handling concurrency issues.

We propose a novel system, CourseHub, that corrects these problems and allows students and staff to more effectively perform their tasks and interface with one another in a clear manner. Other problems CourseHub addresses are: selection of design groups, sharing work between students and staff, mistaken submissions, handling deadlines, handling grading policies and maintaining full history of student submissions while maintaining current submission selections.

Our main design goals are correctness, modularity, and abstraction. Correctness is emphasized because first and foremost our system is designed to fix the problems of the current system and minimize new issues. In some cases we tradeoff simplicity in favor of correctness in order to correctly address edge cases. We emphasize modularity and abstraction so our system can accommodate other courses and note that courses could be implemented in parallel on a server if given more resources. Modularity is clear in the organization and division of CourseHub which is composed of the following modules: a file system, a grade database, four classes that organize course data. These modules are independent, and interact internally on the server.
Correctness is emphasized through a well-defined server side transfer protocol, a locking discipline for file uploads and reads, and granularity in member permissions to name a few. Abstraction is most apparent in our division of pertinent course information into separate classes. We begin by delving deeper into the overall system, the modules of the system, and the specific implementations. We end by delineating use cases and evaluating the system based on our design goals and other important metrics. The following figure gives a more general view of our system.

2 Overview

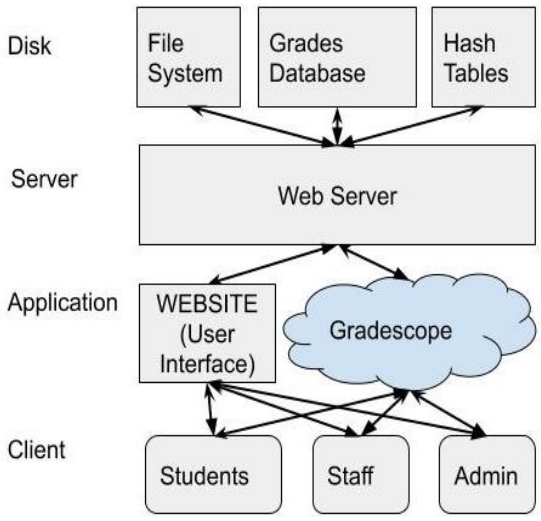

Figure 1: Overall System Diagram, delineating system modules and their channels of communication and data flow

2.1 High Level Description/Intention

CourseHub is composed of one server running one multi-threaded program on its 10 cores. Our system stores the data from our 400 students and the coursework they prepare on disk in the server. As our server runs its processes, it loads the appropriate data it needs from disk to memory and to the cache for performance. These latter operations are automatically handled by the OS Kernel. See Figure 1 for general system flow/organization.
CourseHub's user interface with students and staff is a website. Through this website, CourseHub utilizes student and staff specific functions, but students and staff are never allowed access to the server, including the course-related data stored there. This mitigates security concerns by tightly restricting access to files and directories, minimizing the system's surface area of potential attacks. We identify students and staff accessing the course webserver using MIT-distributed user names called Kerberoses/Kerbs. These provide secure permission allocation, as students and staff must login using these IDs.
The course admin has direct access to course data for logistic and administrative purposes. Course admin, however, are not clients to the webserver. Instead, admin must log in at the OS user level to the server, and interact with CourseHub from the terminal.

3 File System

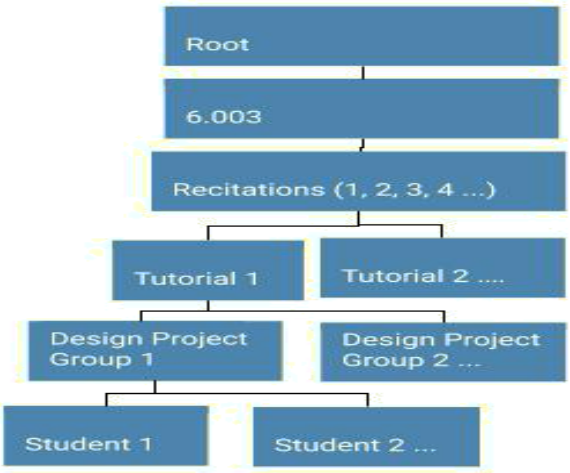

Figure 2: File System Hierarchy for 6.033 using MFS and stored on disk

We implement a file system hierarchy that utilizes the pre-existing MFS (MIT File System) to accommodate: differentiating between recitations, tutorials, and students. This file system allows for the creation of different groups, and groups within groups (For 6.033 it would look similar to Figure 2). For the purposes of 6.033, an example student path might look like /6.033/Rec/Tut/DPGroup/Stud This path makes sense in the case of 6.033 because the course is partitioned into such subgroups and makes for straightforward organization of the course. This infrastructure can easily accommodate the needs of other courses to create more or less groups and groups within groups as aforementioned, a design decision that implements modularity and abstraction. For example, in order to store another course a similar hierarchy could be made in the same root directory with another course directory. Every individual student's directory stores the student's submissions to assignments. A student group's directory stores the group's submissions and shared files.

3.1 Student and Group Directory Capacities

For 6.033 we recommend that student directories have a limit of 20MB and group directories have a limit of 700MB for 6.033, but these limits can be modified for other courses. These numbers are derived from the fact that there are 400 students in 6.033 each with their own directory(400 Ind. Groups) and a group directory (130 project groups). 20MB is plenty for individuals submitting text files and 700MB for DP groups allows 100MB video uploads while leaving space for other files. Altogether this means allocating 99GB to 6.033 which is well within the 240GB storage capacity of disk.
(400 students * 20MB) + (130 Project-Groups * 700MB) = 99GB

4 Software Classes

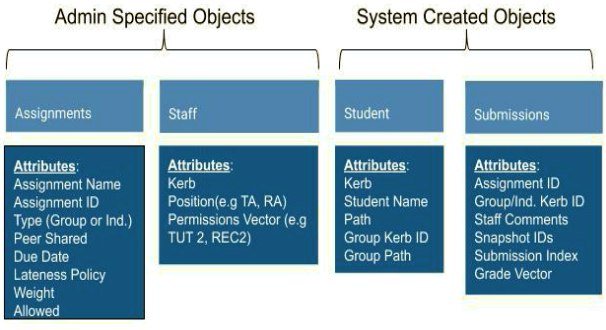

Figure 3: Software Classes and Attributes used to store pertinent system-wide information

For clients of the course, the system operates as a black box. They do not interact directly with any of the data on the server. Instead they interact with a

website, where the server offers them functions that they may use.
To implement this black box system, CourseHub utilizes the following data abstractions to organize its functionality. First, we introduce abstractions to represent the types of clients of our system. Clients are either staff or students of the course. This identity determines what functions are available to them. For example, a staff member can grade a student's work, whereas a student cannot.
We add two more abstractions to this set of "classes", the assignment class and the submissions class, to organize this data and its metadata in our server.
These classes grant CourseHub high modularity, as individual classes' may be implemented and modified without impacting the remaining classes.
Furthermore, this design contributes high flexibility to the system, as it gives a lot of freedom to course admin in establishing staff to student relationships to handle many different courses' hierarchy.

All student and staff objects are stored locally on the server's disk in student and staff hash table databases respectively. Each database hashes every member's kerb to the remainder of their information (constant time lookup). Assignment objects are similarly stored in their own hash table on disk using ID as the hash. To reference any attributes of any class see Figure 3.

4.1 Staff Class

A Staff object represents an individual staff member, and is created by the admin from the Staff Class on their addition to the course. A staff object has the following attributes:

1. Kerberos ID
2. Staff Name
3. Permissions Vector

The Kerberos ID allows secure staff member identification. The Staff Name provides clarity. The permissions vector enumerates all directories staff has access to. For example, a staff member may be recitation instructor 11 who should have access to the work of all students in recitation 11. They would then have recitation 11 directory in their permissions vector thereby granting them access to all student and group directories beneath directory recitation 11. Staff cannot access the data stored in directories that are not nested within the directories in the permissions vector.

Staff members are added and removed from the system by the admin of the course either singly or in bulk by passing CourseHub a csv file with the staff members' names and kerberos'.

4.2 Student Class

A Student object represents an individual student, and is created upon their addition to the course. A student object has the following attributes:

1. Kerberos ID (unique identifier)
2. Student Name
3. Path (path to student directory)
4. Group Kerb ID
5. Group Path

The Kerberos ID allows us to identify the student uniquely.
The student name provides greater clarity to staff.
The path allows quick navigation to the student directory where their individual submissions are stored.
The Group Kerb ID pertains to the student's group. This attribute is initially empty, when the student has not yet been assigned to a group. The updating of this attribute is specified in Section 8.1 Design Group Creation. The group path gives a link to the student's group directory for quick navigation to group work given the student's object.

4.3 Assignment Class

We introduce the assignment course as a means of representing individual assignments specified by the admin and/or staff that helps us handle the logistics of running an MIT course. Each assignment object has the following attributes:

1. Assignment name
2. Assignment ID
3. Type - Whether it is a group or individual assignment
4. Peer Shared Boolean
5. Due Date
6. Lateness Policy
7. Course Weight
8. Allowed - Allowable submission file types (e.g. .pdf, .mov)

Assignment ID is a unique assignment key. Contrary to the ID, the assignment name can be changed.
Type determines whether the submission is stored in the individual or group directory.
Peer Shared Boolean determines whether the assignment is peer shared.
Due Date has the form MM/DD/YYYY.
The Lateness Policy is a pointer to a function which CourseHub can call. This function take as input the due date of the assignment, the date of a student's submitted assignment, the grade the student has received for the assignment, and returns the grade for the assignment when accounting for lateness. This attribute allows for a course to account for different lateness policies for different assignments.
The Course Weight attribute represents the individual assignment weight for the overall course's final grade which enables student grade appraisal.
The Allowed attribute specifies the permitted file types for an assignment.

One assignment object maps to exactly one assignment. Assignment descriptions or any files that should be available for students to complete the assignment are treated as data and information contained within the UI of the webserver, and are held as resource files in the `UI_resources` folder in server storage.

4.4 Submission Class

The Submission class represents and organizes the student's/group's submission to an assignment. As with the rest of our system, staff and students do not directly modify the submission objects; instead they rely on the functionality we provide them via the web interface, which the server carries out. A submission object is a .txt file that contains the metadata below, and is located in the directory of the key kerb it belongs to.

Groups/Individuals can "Submit" a submission as soon as the corresponding assignment object is created. A submission object for all relevant kerbs is created upon an assignment creation, so that staff can grade tasks to which students did not submit any files. For example, staff can grade a student's recitation participation without a student submission. Each Submission Object has the following attributes:

1. Assignment ID
2. Key Kerb
3. Submission History: List of Snapshot IDS w/ Timestamp
4. Current Submission
5. Grade Struct
6. Comments for Submission

Assignment ID maps the submission to the corresponding assignment. This enables automated file type checking during submission, staff downloading of all submissions to an assignment, late policy checking, etc. Key Kerb is the kerb representing either a student or group kerb.
Submission History tracks all previous versions of a submission. Every entry contains a unique snapshot ID and timestamp taken upon each submission, as well as the specific individual kerb that committed the submission upload. This kerb is not the same as the group's kerb. The unique snapshot of a submission can be used to revert an old file to its state at the time of submission.
Current Submission is a pointer to the submission history entry which should be graded.
Grade Struct tracks the grades this submission has received from all staff who have graded it and maps the staff kerb to a tuple of student kerb and the staff grade for the individual student. No grades or comments for any submission object are visible to students or other staff members until a staff member publishes the whole assignment's grades.
The comments attribute is a dictionary mapping staff-kerberos to the .txt file (or possibly an audio file annotation) with that staff-kerberos' submitted comments. A .txt file for a staff's comments is submitted via the website and stored in the Key Kerb directory (txt, audio or other file types can be submitted, but they will not contribute to the directory size limit we enforce for students). Subsequent comments for the same staff member overwrite the previous comments. Thus commenters can work independently on the same submission object.

5 Bootstrapping

The admin of the system must bootstrap the CourseHub server. First the admin sets up the server's file system via terminal. This allows them flexibility in organizing class data. Then the admin has to launch the CourseHub program. We assume that the admin starts with a server that no one else can SSH into to minimize security flaws. Admin then pass in a CSV file of all staff members' kerbs, names, and pertaining directories (e.g. ./recitation 12) to create all

staff objects for the class. The "permissions vector" is updated according to the directories, which allows admin to set up recitation groups with potentially multiple recitation team members.

Once staff is added, the admin can add students to the course. First they pass in a CSV file of all students' kerbs and their names, which creates all student objects. CourseHub then automatically and evenly distribute students' individual directories at the lowest level of directories, and update the students' path attributes accordingly. Admin can also specify directories for students in the third column if needed. The initial terminal bootstrapping of roughly 30 directories for tutorials and recitations seems reasonable; more importantly this step allows admin to make highly specialized course structures to fit their needs, which emphasizes correctness in respect to flexibility. Using CSV files to initialize students and staff is simple and the details of the implementation are abstracted away by the system for ease of use. For a time estimate see Section 16.10 Additional Metrics.

6 Grade Database

Grades are contained in a database to enforce modularity and thereby distribute CourseHub across modules that each serve a focused and well-defined purpose. The grade database maintains a cohesive structure of all grades in the course, and can be used to quickly render information about all grades for assignments from all students in the course to the granularity of a single student submission. Staff member permissions determine what grade database information is accessible, addressing any concerns of maintaining student's privacy.
Maintaining the hash table structure of the grade database separated from the class objects and file system keeps our system modular, enhances performance from utilizing the constant lookup time of a hash table, and minimizes the complexity of the implementation for each module in CourseHub. This facilitates CourseHub comprehension and use, as opposed solely maintaining grades within Gradescope and the file system. We tradeoff constantly maintaining current grades that are stored in the Gradescope and file system with this centralized database that updates at times that are most necessary such immediately before notification of posted grades and at least twice a day for any grade modifications. This tradeoff allows us to provide the ease of use for staff and students to extract grades without any issues and to enhance modularity of our system by maintaining a module on the server with a well-defined purpose of storing all grades.

6.1 Staff Grading

Many assignments are submitted and graded on Gradescope. For a submission that is graded on MUGS, there is a submission object present either in a group directory (such as a group assignment) or an individual student's directory (such as an individual assignment). Whenever a staff member grades the submission via the website, its grade attribute gets updated with that grade. For a group assignment all students can receive differing grades depending on the assignment grading scheme. For example, the grade attribute for a group assignment contains 3-4 grades, one corresponding to each student. See Section 4.4 Submission Class for staff comment information.

Certain assignments receive multiple grades from multiple members of staff. In this situation, the grade we put in the student's grade vector in the grade database is the result of the weighted average of the grades given by the graders. The averages for each grader are determined by the assignment writeup.

For late submissions, we add the grade to the database with the lateness penalty applied. This step happens for Gradescope and grades stored on MUGS before the grades are updated in the Grade Database. GradeHub accounts for illnesses and extenuating circumstances by calling *studentGradeException*() (described below). Because grades reside in either Gradescope or MUGS, the current grades in the grades database are updated whenever the server calls *addMUGSGrades*() and *addGradescopeGrades*() (described below).

Our grading database comes equipped with a functional interface that the server utilizes for processes involving the Grade Database, Gradescope, and MUGS.

6.2 FOR USE BY SERVER ONLY

studentGradeException(studentKerb, assignment) - Staff enters parameters via the website. If the staff has permission for the student kerb then the specific kerb becomes exempt from the lateness policy for a specific assignment when the grade database is being updated. Staff then use their existing ability to edit grades on MUGS via the website or Gradescope as needed.

The server uses these next 2 functions automatically whenever it pulls from Gradescope or the file system right before notification of grading, otherwise, the server updates grades 2x a day to accommodate late submissions or modified grades. These daily pulls occur when expect to have low server traffic, 2AM ET and 2PM ET to prevent overloading the network.

addGradescopeGrades(CSV) - updates the assignment grades from Gradescope in the grades database. Any recent changes to grades pulled from Gradescope in the database are reflected on the CSV returned from the call to *pullGradescopeGrades*(). The server can compare the CSV constructed from *grabGradeReport*() (description below) for the entire course's (which reflects all grades currently stored in the grades database) relevant assignments, against the CSV from *pullGradescopeGrades*(). If any grades are different for any of these assignments, we update the grades in the grades database with the grade on the CSV file from *pullGradescopeGrades*(). (see Grades Database Figure)

addMUGSGrades() - updates grades in grade attributes contained in student submissions objects for all assignments. Similar to the process described above, the server compares current grades for an assignment for the whole course (from the CSV returned from *grabGradeReport*()) against the grades returned by in the file system. If there is a disagreement, the grade database grade is replaced by the one in the file system. (see Figure 4)

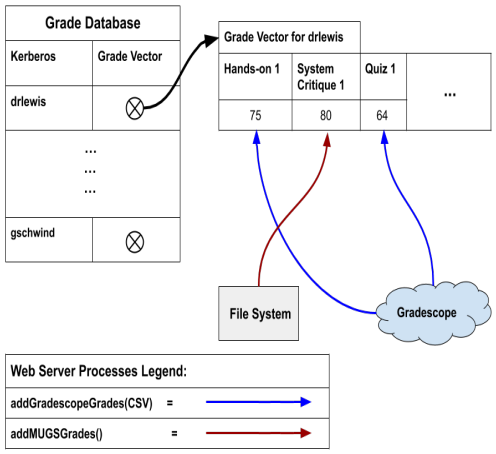

Figure 4: Grade Database with server processes

6.3 FOR USE BY CLIENTS VIA SERVER

grabGradeReport(assignmentIDs, directoryID) - Returns a CSV file containing the staff or student requested grades but the client only sees the grades that their permissions permit them to see. The data to be displayed on the CSV is extracted from the current grades present in the grades database. Students can only see the grades for their entire grade vector up to the most recently admin published grade or specific grades contained within their grade vector. Additionally a column is added on this CSV that has current student final grade averages as calculated from the course weighting attribute for each assignment. This allows staff members to appraise individual students, groups of students, or the entire course (depending on their permissions). Students may assess their own performance as well.

6.4 Grade Reporting

We provide a function for downloading the assignment specific submission objects from a specified directory:

downloadAssignmentSubmissions (directoryID, assignmentID) - Checks user permissions when user sends this request to the server via the web interface. Downloads all the snapshots corresponding to the specific assignment that are within the directory, onto the client's machine. The server sends this data to the client's machine to be downloaded.

6.4.1 Notification On Grading

We notify students on a course-wide basis when grades for an assignment become available, and individually when a change has otherwise been made to a grade. For both instances the student is only notified and can can only view the grade if the admin has published the grades for the assignment. Gradehub provides these two types of notifications.

Course-wide notification of a grade release: The admin determines when an assignment is ready to have its grades published. Gradehub then automates an email to all students to their kerberos@mit.edu notifying them of the update, and makes the grades for the assignment available to everyone.

Regrades or grades of late submissions: students are notified via email using their kerb either at 2am or 2pm when the grades database is automatically updated. Students are given access to the grade (given the admin has decided to publish the grades).

7 Permissions

CourseHub keeps track of member permissions independently from MFS via the student and staff classes. CourseHub relies on MIDS (MIT ID Service) to provide a secure way of identifying users and allocating permissions. Students have access to the files and submissions in their directories only via the website, they do not have direct access. Staff can access the student and group directories that are within their permissions (e.g. students in their recitation), but only through the website. Admin have direct access to the entire file system, grade database, and software class objects.

8 Student Group Work and Assignments

Section 3 File System describes how group work is stored on the server. Section 4 Software classes describes how access to a group directory is granted. A group directory is not directly accessible. Instead, our server interfaces with the file system and provides the appropriate functions for members to interact with the directory. For example, the server allows students to download existing files, upload files, make or update a submission (locking issues are covered later). This design decision trades system simplicity for abstraction and correctness. We introduce overhead in implementing the system, but can thereby abstract away details of data storage from clients, simplifying the user experience. We thereby also ensure correctness, as we limit access to course data, prohibiting clients from corrupting each others' data or viewing data beyond their permissions as a member.
We utilize the pre-existing MSS (MIT Sync Service) to synchronize asynchronous updates by different group members within a directory. MSS already allows for the client download of files, editing of the files on a personal computer and uploading to update the server's file. We maintain these functionalities.

A conflict arises if a client tries to update the server's file after the file has been modified with a timestamp that occurred after the client first downloaded the file. In this case the *client cannot upload their changes and must re-download the file, re-edit, and submit* the new file to update the server file.

8.1 Design Project Group Creation

Coursehub supports student project group formation. A student group can be created by a staff member. The staff member must have permission to access the information of all students they want to group together. For 6.033, this allows the system to enforce that the students working on a 6.033 Design Project together are within the same recitation. Students email their staff member (in this case recitation TAs) their grouping preferences and then staff enters the groups to CourseHub. Once approved, a group directory is created in the lowest file system directory level not including the student directories (tutorial group in our case). Then the links between the tutorial directory and the students are broken and student directories are placed in the design group directory, altogether taking milliseconds. The student's attribute "path" must be updated and a new group kerb is generated for the directory that is attributed to all member students. This update keeps the file system strictly organized and facilitates admin use.

8.2 Student to Student Sharing

We support peer-review based assignments which require students to view each others' submissions. Such an assignment must have their Peer Share Boolean attribute set to True. Then students enter the kerb and assignment name in order to access and download the specific submission from another student. So that students cannot simply access all assignments staff must specify which directories students have access to. For example staff specifies that student kerb drlewis has access to directory Recitation 9 for assignment 10. In this case drlewis has access to the current submissions for the specific assignment 10 that are within this directory.

8.3 Rank Ordered Voting

We include support for students to rank or essentially grade the work of other groups. The instructor marks this as an assignment that is peer-shared via the peer shared attribute in the assignment class, allowing students to search for the submissions of other students and download them, and submit comments, and grades/ranks, for the assignment. We give the students the ability to rank one time per peer shared assignment. The student's rankings are submitted via the website and the votes appear for all groups in the course to the staff (the tally totals for the groups are displayed as well). The staff awards credit to students by giving them a participation grade for the assignment. This implementation of rank ordered

voting makes a design tradeoff in simplicity in favor of storage management and flexibility. This one peer sharing functionality can enable students to view all other submissions for an assignment, one specific other student's submission, or a few students' assignments. This flexibility allows admin and staff to use this for whatever combination of peer review a course may call for. This approach also allows for students to access each others' information without having to duplicate data stored on the server, which makes implementing correctness easier.

9 Individual Work and Assignments

Using the submission and assignment classes we already defined and explained thoroughly, all individual tasks of submissions and editing are taken care of. Submitted and uploaded files are treated the same as those submitted by a group, except they are stored in the individual student's directory. If a student wants to simply edit a file they follow a similar protocol to that of a Group and utilize MSS to edit files and update the current server file.

10 Client Server Transfer Protocol

The file size determines the Trip Time (TT) that we expect for all packets of a file to transfer. (For trip time analysis, see Section 16.7 in Evaluation)

If the time of an upload or download takes longer than TT, we send a kill() instruction in order to stop the file transfer, we refer to this as timeout. We delete partial transfers on the server. For user-defined kills, we maintain two queues. One queue for handling generic client requests such as uploads, downloads, inputting grades and another queue for maintaining a kill switch. The kill switch is triggered when we have the user press cancel on the on the website. The browser then sends a kill message to the kill switch queue. This queue always has priority over the generic requests queue. These queues operate on the application layer, not interfering with normal TCP, but rather building on top of it. In the case where a user is uploading a document, but they close their laptop before the upload has completed, we ensure that this process gets killed by the timeout we have defined above.

If a file transfer is killed by timeout, the user is sent an error and instructions to resubmit/redownload. In order to ensure the ordered delivery of large numbers of packets we use TCP, but refrain from updating our own system until the entire files have been sent.

By handling the multiple client-server transfer instances above with well-defined behavior integrated into the transfer protocol, we ensure the correctness of our system when it kills processes and handles requests from clients over the network.

11 Locking and Concurrency

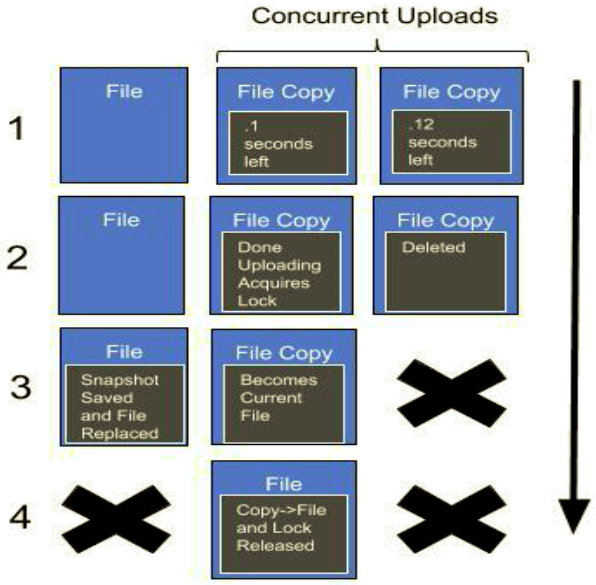

Figure 5: Concurrent File Upload Locking Procedure

We expect issues with order and erroneous uploads if/when multiple people or the same person with different tabs attempts to upload files to the same directory at the same time. To mitigate this problem GradeHub implements a locking discipline. In order to minimize the time that files/directories are locked we allow all uploads to occur simultaneously (1 Figure 5). Files are initially be uploaded as copies for the current file. The first file to finish uploading receives a read/write lock for the directory with the client attached to the lock (2 Figure 5).

We use MLS (MIT Lock Service) for this action. The copy becomes the current file, the snapshot for the old file is recorded in the corresponding submission object, and the lock is released (3,4 Figure 5). This lock prevents any reads or writes during this period, but it ensures correctness. Because the lock is only held for the duration of slight internal modification of files, the lock is held for less than a millisecond

(a negligible amount of time). In the case of concurrent uploads, the other clients who did not receive the lock have their upload canceled and deleted and must re-download the newest copy of the file, redo the changes, and then upload the file again. This process may seem like a nuisance, but this process is already in place in MSS and the process ensures correctness in spite of concurrent uploads.

12 Submission File Type/Size Checking

Upon submission of a file, CourseHub only allows the specified file types for the assignment object that are under the file size limit of 20MB for individual assignments and 100 MB for Group assignments (otherwise an error is thrown and the user is told to re-submit their files in a suggested format under the specific file size limit). This limit can be changed depending on differing class needs. Staff define the allowable file types for each specific assignment when creating the assignment object.

13 Security

Students do not have direct access to the file system and only can interface with the server via the website. Thus students cannot edit grades or tamper with directories. A possible attack is a multiple submission/download attack that overloads the system. To address this possibility we only allow students to upload/download 15 times an hour which we believe is very reasonable and given the worst case scenario of 100MB uploads/downloads our system would take 1/4 second to respond to 15 equivalent uploads given 6GB/sec bandwidth. If this cap is exceeded the corresponding staff are alerted via email. Also if students exceed their directory capacities, staff are also notified and the oldest submissions by a student are deleted while conserving their current active submissions. While it is unfortunate that data is lost in this scenario, we believe that given responsible use of directories students should not come close to filling their storage limits.

14 General System Assumptions

Our assumptions include:

1. The staff is not corrupt or malicious.
2. The Design Project members do not try to sabotage each other.
3. Unique Kerberos IDs provide a secure method of delegating permissions to others.
4. Directory and file lookups take constant time.
5. Access to the disk is given only to the Web Server and the Admin (not students or staff)
6. Our system is able to handle overflow and overflow attacks.

15 Use Cases

1. **Initialization of 400 student accounts:**CourseHub is already setup with the appropriate group hierarchy for recitations and tutorials; therefore, what remains is to populate the class with its students, and to assign the students to a specific recitation and tutorial. See Section 5 Bootstrapping for the process. This full process should take on the order of seconds because it involves simply creating 400 directories and student objects.
2. **400 simultaneous submissions:** CourseHub processes submissions as they come and the worst case is described in Section 16.5. In the case where the bandwidth of CourseHub is matched/exceeded the submissions are added to a queue and GradeHub records the timestamp pertaining to the submission along with the kerb ID who submitted the file and processes the submissions accordingly. A submission is not timestamped until it is received on the server side, which makes submissions immune to student tampering of submission timestamps.
3. **Notification of 400 students simultaneously that a grade was posted:** See Section 6.4.1 Notification on Grading
4. **Regrade of quiz question for one student in Gradescope:** After we have pulled initial grades from Gradescope, but someone has new grades entered, we receive a notice from Gradescope that a grade has been changed and within an hour we query Gradescope for the grade and add it into our system. The grade is added into our system by using the assignment name as a identifier to then assign the assignments attributes such as weight and deadlines and then

places the grade in the grade database accordingly. On a regrade, we want to alert the student of the change, so we send an email when a change to the grade database is made to the appropriate student email.

5. **A student drops the course late in the term:** We simply remove the entry for the student in our Grading Database and then delete the student's object containing their information, and permissions to certain groups. This effectively removes them from their group and when an instructor attempts to look at their work or information an error is thrown because they no longer exist.

6. **Staff Member is added late in the term:** A staff member's functionality in our system is defined entirely by their staff object, meaning that to get a late-added staff member up to full speed on the system we only have to instantiate their staff object properly, which the admin can easily add.

7. **End-of-term spreadsheet for grades:** Our Grade Database is readily able to return this end-of-term spreadsheet to the course lecturer with permissions to all grades for all students in the course at the end of the course with the *grabGradeReport*() (see Section 6.3 FOR USE BY CLIENTS VIA SERVER) function our server uses to provide these grade reports to the students and staff contingent on their permissions.

8. **Initialization of a Course in our system:** See Section 5 on Bootstrapping.

16 Evaluation

16.1 What is the communication overhead of your system?

Between the client and the server we expect to use the full bandwidth of 6GB/sec in the cases of simultaneous uploads by many groups. Otherwise on average we expect to be using 1GB/sec at max to handle random submissions and student and staff actions (we expect non uploading/downloading actions to take orders of magnitude less bandwidth than uploading). Between Gradescope and the server we expect notifications and data transmission via csv. As described in section 16.3 csv files should take 10 seconds to upload. Altogether, however Gradescope and server communication is negligible because it happens during low traffic times (Section 6.4.1) and the communication overhead comes primarily from the server client interaction described above.

16.2 On average, how long does it take a student to upload an assignment to the server?

As covered in Section 16.5 in the worst case with all DP groups simultaneously uploading videos, uploading can take up to 2 seconds. In the case of 900 students this number becomes 5 seconds.

16.3 On average, how long does it take for Gradescope grades to be transferred to the server?

To get Gradescope grades to the disk of the server, the server calls *pullGradescopeGrades*(). It takes 10 seconds (as defined by project specs) to export the CSV from Gradescope to the server. The time it takes to add new grades from the Gradescope CSV into our grade database is negligible compared to the 10 sec export time.

16.4 How much data are you storing on the server?

As explained in Section 3.1 we expect the 6.033 to require 99GB at maximum. If 500 students were added to 6.033 then we expect to need 228GB which is still within the 240GB on disk. 6.033 with 1000 students or another large course requires more storage on the server. However, 6.033 is an exception from many other courses in that it requires large files such as videos to be uploaded. Therefore, several courses lacking large file type submissions could be accommodated relatively easily.
(900 students * 20MB) + (300 Project-Groups * 700MB) = 228GB

16.5 What parts of your system limit scale, and what are those limits?

A pressing bottleneck for our system is the network speed, especially if/when hundreds of students are trying to upload at the same time. In the worst case student upload speeds are 600MB/sec, 130 DP groups are submitting videos of 100MB and we are restricted to 10 cores. This speed is in line with the 6GB/sec bandwidth of our sever. Thus 60 people/sec could be serviced which means a potential slow down of our

system that is roughly 2 seconds and not a significant bottleneck. If 6.033 became a course of 900 w/ a maximum of 300 DP groups, we expect a slow down of roughly 5 seconds. This bottleneck does not seem unreasonable given that everyone will not submit at the same time and even if they did the system would recover quickly. If all EECS courses were to use this system, problems would most likely arise; however given that most EECS courses do not require video submissions and if deadlines were staggered, our system would be able to accommodate most of the EECS department. Disk storage capacity is another bottleneck, but has been addressed above. If bottlenecks arise the system will have to include some combination of additional servers, cores, and storage.

Another bottleneck to web server performance could be internal lookup time. We store the bulk of our data to disk, so lookup time is a function of the duration disk lookup takes + the duration of additional computation. However with a 12GHz clock we do not expect this to be a bottleneck.

16.6 How long does it take your system to deliver all student grades to the Course Lecturer?

We query all the entries of our grades database, and copy them in a formatted way into a csv file to deliver to the course lecturer. We assume that it takes a millisecond to pull all the grades for an individual student and copy them into a csv file. This is a generous upper bound. If we then have 600 students in a course (larger than our 400 student estimate for 6.033) then this process takes 600 milliseconds. Meaning that we can return all the requested data in well under 1 second, let alone 10 seconds.

16.7 How long does it take for a file transfer to be killed, if requested?

Since we maintain two queues in our transfer protocol specification, if a user-defined kill is triggered, we expect the file transfer to get killed in a few milliseconds, as the kill switch queue takes priority over the queue that handles all requests from clients to our server. In the worst case, the time it takes for the file transfer to be killed is the Trip Time (TT), which is proportional to the size of the file. The slowest speeds of the network are 600MB/sec. We define TT to be the number of MB*4ms. For example for 100MB of data we expect the TT to be under 400ms or 2/5 of a second. This number is a factor of 3 slower than the slowest expected time and should provide enough cushion for the slowest network speeds to upload.

16.8 How long does it take to create all student accounts at the beginning of the semester?

We assume that we have 600 students to create accounts for, and that for each student it cannot take CourseHub more than 1 millisecond to create an account. This is a generous assumption because creating a student account only entails making an entry in our hash table of student accounts and creating a textfile to store the entry's information. Therefore, the upper bound is 600 milliseconds to create all of our student accounts.

16.9 How usable will users find your system?

Uploading/downloading actions have been covered above (taking at most 5 seconds under the worst circumstances) and all other actions that clients take using the website should be resolved within ms due to the transfer of data on the order of KB using a 6GB/sec transmission. This is reasonable given that delays on the order of seconds are okay from a human standpoint. Also the delay does not impinge upon submission timestamps.

16.10 Additional Metrics

How long does it take to bootstrap the system?

The admin should on the order of minutes to create the file system for the class via the terminal. Assuming the CSV mappings specified in Section 5 Bootstrapping have been created, the creation of the corresponding directories in the file system and the hash tables we use to maintain course data should take on the order of milliseconds. This mixture of CourseHub automation and admin manual input leads to a total setup time within minutes which seems reasonable for the system, and for the valuable time of admin. The use of CSV files to enter staff and students into the system minimizes tedious admin work. We tradeoff simplicity for correctness and user friendliness.

17 Future Work

Future iterations of CourseHub could extend the system's functionality. Currently Coursehub's as-

signment and submissions classes accommodate rich media (.WAV files, .jpg files, etc.) as submission types and as staff comments; however, CourseHub could be altered to allow the submission of code files and their subsequent evaluation via defined test cases. One would merely have to add the appropriate attributes to the class abstractions and their respective functions.

Future work may also focus on redefining the admin role; it may be convenient to access administrator permissions via a remote login to the webserver as staff and students do. We eschewed this online access to administrator capabilities to avoid unnecessary complications in security, but future work could be done in this area.

18 Conclusion

CourseHub addresses the problems of the current 6.033 system and has the flexibility to generalize to other course structures. Furthermore, in its current form CourseHub uses less than half the total storage available on its server, priming the system for use across MIT in significantly larger and different courses.

CourseHub has been designed to be straightforward for both implementers and system users. Our emphasis on modularity resulted in a logical and useful partitioning of system components(file system, software classes, and grade database) lending itself to ease of implementation. Our emphasis on abstraction created a system that obscures responsibility from users, leaving them with a simple user experience. We designed our system to have well-defined behaviors for its many different uses guaranteeing that our system would behave correctly in each circumstance (seen in locking and transfer protocols). We have identified how future work may extend and augment CourseHub and we emphasize that future work can be readily built upon the structures we have already defined. One remaining implementation problem lies in the creation of the client UI (website) and its basic integration with the server.

19 Author Contributions

19.1 Alejandro Diaz

Point person for the Introduction; High Level Description; Specified File System; Locking implementation; Design Choice Arguments; Figures for Gradehub diagram, file system, locking, and classes; Evaluation; and Peer Review Functionality.

19.2 Katharina Gschwind

Point person for Design of Software Classes; High Level Description; Design of Course Metadata storage; Design of Student/Staff/Admin Roles and Functionality; Permissions; Notification; Design Choice Arguments; Evaluation; Security; Conclusion; Writing and Researching Future Work; and Peer Review Functionality

19.3 Dylan Lewis

Point person for Grade Database; Grade Database Diagram; Grade Reporting; Integration with Gradescope; Rank Ordered Voting; Client Server Transfer Protocol; Design Choice Arguments; and Evaluation;

20 Acknowledgements and References

Katrina LaCurts - Course Lecturer
Steve Bauer - Recitation Instructor
Amy Carleton - Tutorial Instructor
Kifle Woldu - Teaching Assistant

We would like to thank the course staff of 6.033 for contributing to our learning of Computer Systems Engineering concepts and for providing us the assistance we needed to bring this design to fruition.

Word Count: 6900