# Beaver Course Management System

Fiona Zhang , Nyle Sykes, Ashwath Thirumalai

Massachusetts Institute of Technology

May 6, 2019

{fionaz, nsykes, ashwath}@mit.edu

Olivia Brode-Roger - Friday 2 PM

# 1 Introduction

The MIT undergraduate course 6.033: Computer System Engineering operates on a fragmented grading infrastructure that splits submitting assignments and viewing grades across multiple different, unconnected systems. Furthermore, this system does not support automatically forming teams, sharing works in progress, and submitting group assignments and necessitates manual verification for team assignments, deadlines, and penalties.

To address these needs, we propose the Beaver Course Management System (BeaverCMS), a modernized infrastructure that facilitates student team work, provides a unified place to view grades, and supports additional video upload functionality while retaining similarly rigorous permissions from the current implementation. This system integrates various MIT-provided modules for file systems, locking, syncing, and identity verification, Gradescope, and a central server containing multiple relational databases. The system aims to automate much of the work currently done manually and provide a seamless, central location to submit and grade assignments, provide and view feedback, and collaborate on team projects. The sections that follow specify our implementations of the modules and communication between them as well as how they achieve our key design goals of implementation simplicity, user simplicity, authorization security, and reliability.

## 1.1 Challenges

Designing this system involves the challenges of dealing with a variety of roles with their own set of permissions and functions. As this system is designed to support the management of a course, it must achieve reliability and correctness for the students, staff, and graders. This system must support a wide variety of use cases such as submitting large files and handling network outages. Moreover, it must automate as many tasks as possible, as even small manual tasks when magnified over hundreds of students can become a significant time burden on course staff.

# 2 System Overview

BeaverCMS extends and integrates five existing systems: the MIT Identity Service (MIDS), MIT File Service (MFS), MIT Sync Service (MSS), MIT Lock Service (MLS), and Gradescope. These modules and their relationships are illustrated in Figure 1 below.

The center of this system is a single mirrored-disk server that stores various relational databases. This central server connects users, who can be students, student graders, or members of the course staff, to the other modules in the system. The system supports submitting assignments and peer reviews to the MFS, submitting large video files through the MSS, and submitting hands-on assignments to Gradescope (4.1). It also supports staff grading and feedback for assignments, automatic grade penalties, and student voting on the final video assignment (4.2). *Brown-outs* are defined as a reduction in bandwidth to a level uniformly distributed between 1 Kb/s and 100 Kb/s for a period uniformly distributed between 10 seconds and 30 minutes which occur approximately once every 48 hours. In the case of a *brown-out*, the system cancels the upload and notifies the user.

The major goals of our system design are user simplicity, implementation simplicity, authorization security, and reliability. User simplicity allows the students and staff to focus on the material of the course rather than worry about a complex grading or submission system and primarily exists to save the time of the course staff. Implementation simplicity allows for the system to be quickly and easily built and maintained. Authorization security is crucial in maintaining users' privacy regarding grades and assignment submissions. Lastly, the design focuses on reliability

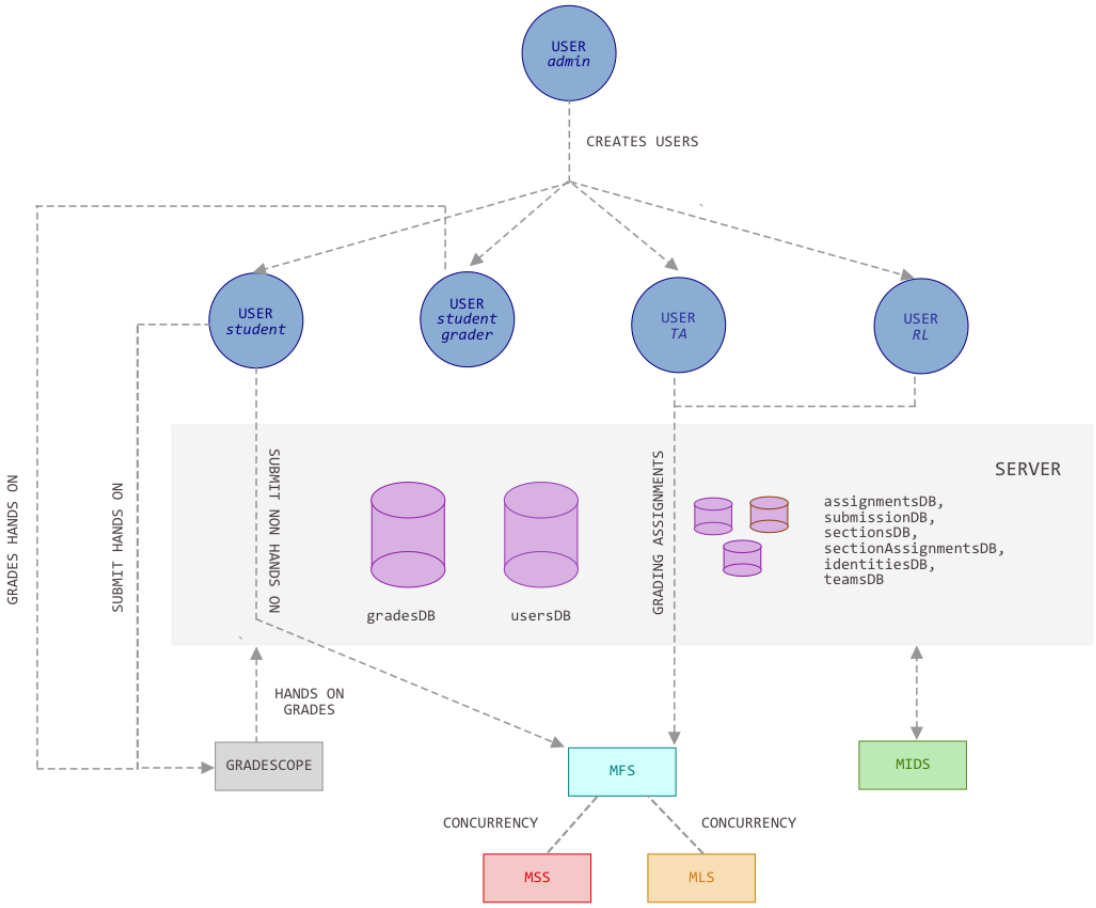


Figure 1: System overview

over performance since confidence in the system from both the student and the staff is more important than the time required to upload an assignment or provide feedback.

# 3 Design

## 3.1 Central Server and Data Structures

The central server is a 10-core 240 GB mirrored-disk system with 16 GB in RAM and 11 GB of cache. This server stores all of the information about the class, including students, grades, assignments, teams, and recitations.

**Table 1: Databases**

| Database Name | Contents |
|---|---|
| *gradesDB* | Student Name (Kerberos name), Assignment ID, Number of Points Received for Assignment, Late Penalty Multiplier (1 if on-time), Timestamp of Last Query, Published (True/False) |
| *assignmentsDB* | Assignment Name, Assignment ID, Total Possible Points, Assignment Due Date, Number of Hours Late Per Letter Grade Deducted |
| *usersDB* | User Kerberos Name, User Status (Active/Inactive), User Role (Student/Student Grader/TA/Administrator) |
| *teamsDB* | Student Kerberos Name, Team Kerberos ID |
| *sectionsDB* | Section ID, Recitation Team ID, Section TA Kerberos Name, Section Instructor Kerberos Name, Section Location, Section Time, Section Type (Recitation/Tutorial) |
| *sectionAssignmentsDB* | Student Kerberos Name, Assigned Section ID |
| *identitiesDB* | Kerberos Name, Home Directory in the MIT File System (MFS) |
| *submissionDB* | Kerberos Name, File Name, Assigment ID, Timestamp, Number of Hours Late (0 if on-time) |
| *recitationTeamsDB* | Kerberos Name, Recitation Team ID |
| *videoVoteDB* | Student Kerberos Name, First Vote Team (team Kerberos name of first-choice video), Second Vote Team, Third Vote Team, Fourth Vote Team, Fifth Vote Team |
| *inProgressSubmissionsDB* | Student Kerberos Name, Assignment ID, Assignment Submission Start Time |

#### 3.1.1 Database Structure

The server stores eleven relational databases linking students, staff, assignments, and grades. The grades database stores grades and grade penalties for all assignments and students. The assignments database stores a list of assignments and information about them. The users database maps Kerberos IDs to course roles for use in permissions. The teams database stores which students are in which Design Project teams. The sections database contains information about the organization of each recitation section, while the section assignments database maps students to their respective sections. The identities database maps MIDS IDs to their corresponding home directories in the MFS. The submissions database stores information about every submission for every student or group. The recitation teams database maps student Kerberos names to their corresponding recitations. The video vote database stores each student's votes for the final Design Project video submission. Lastly, the in progress submissions database records assignment submissions that are currently in progress and whose outcomes need to be communicated to the sender. Table 1 above lists these databases and the variables that they store.

#### 3.1.2 Implementation Simplicity

The relational database structure achieves implementation simplicity due to the ease of modifying and quickly accessing information through SQL querying. These databases are highly flexible, well established, and simple to maintain. Although they struggle with more complex data-types, this is likely not to be a concern with the 6.033 course. Our design only requires the system to store integers and strings in the data fields, and any more complex data types can be stored in the file system. As such, it makes sense to use a relational database structure to optimize our design for storing simple data-types.

## 3.2 Identities and Access Control

BeaverCMS employs a robust user management system that spans both individuals and groups to secure access control to the file system. The system combines MIDS with the teams database, sections database, section assignments database, and the recitation teams database.

### 3.2.1 Identity Control

System users include students, student graders, recitation leaders, teaching assistants, and course administrators. MIDS has existing identities that correspond to every Kerberos ID for individual users. Course staff has permissions to create new MIDS identities via the *create_id* command that correspond to groups of students. These include Design Project teams, recitations, sections, and the entire class. Since MIDS does not support querying for specific roles or checking if a student belongs to a certain Design Project team or recitation, our system stores these relationships in the databases in the central server.

### 3.2.2 Course Initialization

At the beginning of the course, recitation leaders, teaching assistants, and course administrators are delegated permissions to create new Kerberos IDs using the *delegate_kerb_creation* command in MIDS. With the start of the course, the system will create an identity that corresponds to the entire class. After recitations and sections have been assigned from the results of a student survey and compiled into a .csv file, this can be uploaded to the server by the Course Administrator. The server will parse through the file, create the necessary recitation and section MIDS identities, and create the necessary relationships between students and these groups in the databases.

Assignments can be created only by the course admin. The course admin needs to specify the Assignment Name, Total Possible Points, Due Date, and Late Penalty. The server generates the Assignment ID, and adds this as a row in *assignmentsDB*. If late submissions are not allowed at all, then the Late Penalty can be set to zero.

### 3.2.3 Dynamic Assignments

If, for whatever reason, a recitation assignment needs to be switched, recitation leaders, teaching assistants, and course administrators have Kerberos ID access to modifying the central server. Any of the course staff can submit a request to the server to change a student from one recitation or section to another, and the system will handle modifying the database to reflect these changes.

Similarly, when a student drops a class, staff can submit a request to the server to remove this student from all database records. Since their Design Project team is now lacking a member, those students must find a new third member. However, this is simply achieved through modifying the teams database, which is checked on all accesses to Design Project team home directories.

### 3.2.4 Secure Querying

Every access to the file system and every access to the central server must first pass through MIDS. MIDS will verify the user's Kerberos ID. Next, the central server will ensure that the user is a member of a specific group if necessary. A simple query to the database can verify that a recitation leader has permissions to view a file of one of their students or that the Course Administrator has permissions to view the grades of the entire class.

### 3.2.5 Authorization Security

A requisite feature for any upgraded system for this course is a comparable level of authorization security. One strength of the current system is that no user, student or staff, can access information that they do not have permissions to view. This system maintains comparable levels of authorization security as the current course infrastructure since it leverages the MIDS for identity control and all accesses to the central server databases are done only after authenticating the Kerberos token and verifying it corresponds to a course staff member.

## 3.3 File Structure

This system creates a custom file system hierarchy within the MFS that distinguishes between different submissions of the same assignment and stores staff feedback on assignments.

### 3.3.1 File System Hierarchy

Each Kerberos ID in our system, both individuals and groups, have corresponding home directories within the file system. An individual's home directory provides a location for submissions of an individual assignment such as a system critique, and Design Project team home directories provide a location for students in the same team to submit group assignments. Upon creation of a new identity in MIDS and the creation of a corresponding home directory, a protocol creates a "COLLAB" directory that only that Kerberos identity can access, and a "FINAL" directory in the home directory and grants the recitation team staff identity access. Figure 2 below illustrates this file structure.

The "FINAL" directory stores the most recent (or user-chosen) file to be graded, while all other submitted files exist in the main home directory. By default, newly submitted files are stored in the "FINAL" directory, and the previous file submission for this assignment is moved to the main directory. However, the submitting user can designate which submission is their final submission. This action swaps that submission from their main directory with the corresponding submission in the "FINAL" directory.

The "COLLAB" directory is accessible only by the user whose home directory it belongs to. Clients can connect to it through MSS. A user (student or team) can upload or download files to their "COLLAB" directory through MSS. If there are multiple individuals with access to this Kerberos user ID (in the case of a DP team, for instance), this folder allows them to collaborate on files and assignments. In the case of a merge conflict, where two individuals, using the same Kerberos identity, attempt to modify the same file in the "COLLAB" directory at the same time, since we require individuals to use MSS, the tags received will be different and the second upload will fail.

For example, a student can access their directory (and the "COLLAB" directory) as well as the directories of their recitation, section, Design Project team, and the entire class by choosing on the client side which Kerberos ID they would like to use. That student's recitation leader, teaching assistant, WRAP instructors, and the course administrators have access to this student's "FINAL" directory.

### 3.3.2 Nomenclature

Submissions of reading questions, system critiques, Design Project Preliminary Reports, Design Project Reports, and peer reviews to BeaverCMS are named in the formats *reading_question_X_Y*, *critique_X_Y*, *DPPR_Y*, *DPR_Y*, and *peer_review_Y*, respectively, where $X$ identifies which assignment in the series it is and $Y$ identifies the submission number starting at 1. A student's third submission of the second system critique would be named as *critique_2_3.pdf*.

When a member of the staff submits comments for a given submission, this file is stored in the "FINAL" directory with the same name as the submission appended with *_comments_Z* where *Z* keeps track of multiple graders potentially submitting multiple sets of comments. For example, if both the WRAP Instructors and Recitation Teaching Assistant provide comments on the student's third submission of the second system critique, both of these comment files will exist in the "FINAL" directory and will be named, in no particular order, *critique_2_3_comments_1* and *critique_2_3_comments_2*.

### 3.3.3 Implementation Simplicity

The system prioritizes implementation simplicity here to ensure the file system is easy to develop, maintain, and operate. This is enforced through a simple nomenclature for submitted assignment files that distinguishes between different assignments, the number of that assignment, and the number of that submission. This allows the system to be flexible from year to year with different grading protocols for different assignments, as this nomenclature can be easily changed. The hierarchy for the file system is also organized around Kerberos IDs at the highest level. This facilitates adding and removing students from the class, which is likely to be much more of a common occurrence than adding or removing assignments.

While this system does not scale as well due to the lack of organization between the different types of files, we do not anticipate this to be a problem due to the limited number of assignments in this course.

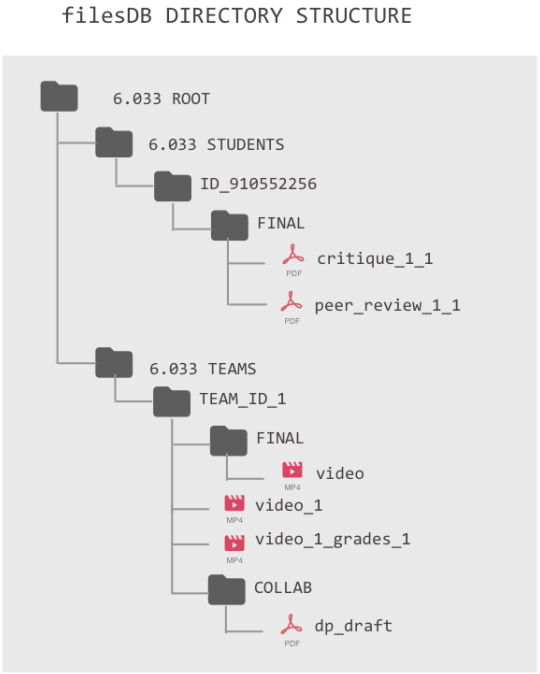


Figure 2: MFS contains a directory for each student and each team.

# 4 Networking

The following section details specific interactions between modules outlined above.

## 4.1 Assignment Submissions

BeaverCMS's submission protocols are designed to optimize for reliability. This is achieved through failing fast in the case of errors, redundancy in the central server, and sending notifications to the user.

BeaverCMS processes any assignment submission by first attempting to verify the Kerberos token by querying MIDS and receiving the relevant Kerberos name in case of success. BeaverCMS then ensures that the Kerberos name is in *usersDB*. If either of these authentication steps fail, then the system fails and returns an error to the client stating that the provided authentication token is invalid. Failing silently for user submissions should be avoided to ensure that the student knows that they must re-attempt the submission.

### 4.1.1 Notifications

For all of the submissions described below, as soon as the server receives the submission request, it adds it to *inProgressSubmissionsDB*, and as soon as the server completes the submission request, it removes it from the database. There will be a process on the server that constantly scans this database, and in the case that a submission is older than 10 minutes (and hasn't been completed), it will be removed, as soon as a failure notification is emailed to the user. If the server has poor internet connection at the time where it realizes a failed submission, which is very possible as poor network conditions can lead to failed submissions, the database entry is not removed until the server is able to send the notification email.

In general, given a Kerberos ID, the server can email a notification the @mit.edu email associated with that Kerberos ID. The notification will include the assignment name, submission time, and whether or not it was successful.

### 4.1.2 MFS Submissions

Certain assignments, such as system critiques, are required to be submitted from the client through the server and stored in MFS in the file hierarchy described above. A submission request consists of a file, an assignment name, a timestamp, and a Kerberos token. The server stores a submission queue. It exposes a submission function, which takes in a submission request, adds the current timestamp in the timestamp field, and appends it to the end of the submission queue. The server has 100 internal threads that watch the submission queue, and whenever the queue is non-empty, picks the submission request at the front of the queue and processes it. We talk about this aspect of parallel uploading in more detail in Section 4.1.6.

Processing a submission request involves verifying the Kerberos token as described above, getting the Kerberos ID of the submitter, using *assignmentsDB* to verify the assignment name and due date, updating *submissionsDB* to reflect the newly received submission, connecting to MFS and using the *write_file* function to upload the file in the "FINAL" submission folder for this assignment of the Kerberos ID submitting the assignment, and moving any file that was previously in the "FINAL" directory to the student's general submission directory.

### 4.1.3 Video Submissions

Our system supports the functionality for Design Project teams to submit 5 to 8 minute videos and view other teams' video submissions. Since these files are significantly larger (approx. 100

MB) than other files, we implement submission differently in order to maximize reliability.

Videos are stored in the directory that is visible to all 6.033 users (i.e the home directory of the whole class identity) and named with their team Kerberos name. Teams must use MSS to connect to MFS in order to upload their video. Specifically, they use the *sync_upload* function to upload their video to the class folder. When a student or team would like to view another team's video, they can run the *sync_download* function through MSS. We choose to use MSS instead of the MFS submission process for video submissions so that reads and writes are synchronized and to allow upload and download processes to be terminated, using the *kill* function, if they take too long.

To allow for speedy cancellations, video files are broken into 1 MB chunks sequentially. A label is appended to the end of the file name, where the filename of the $i^{th}$ chunk is appended with $_i$. When the client sends these chunks using the *sync_upload* function, it waits for completion of all of them, and then sends a notification to the server indicating the video has been submitted (the server exposes a function to do this).

Upon downloading videos, the client downloads all the chunks corresponding to that video team name, and collates the files in the order of the suffixes.

#### 4.1.4 Peer Review

Our system also supports peer reviews. Upon creation of the peer review assignment, BeaverCMS creates a peer review folder in the home directory of the whole class identity. BeaverCMS copies all the DP teams' DPPRs into this folder, setting the permissions such that each DPPR can be viewed only by its authors. The server exposes a function that allows for a DP team (using their team identity) to add another Kerberos ID (corresponding to the team reviewing them) as a peer reviewer. This function verifies the team's Kerberos token and then changes the permissions on the copy of that team's DPPR in the peer review folder to include the reviewing team. We then provide a server function where a team can download their peer reviewee's DPPR. The function simply scans the peer review folder in MFS and downloads and returns the only file it has permission to view (except their own DPPR).

We expose server functions for submitting and viewing the peer reviews themselves as well. The submit function takes in the peer reviewee's Kerberos name and the peer review file (and verifies the Kerberos token of the peer reviewer) and writes this file to the peer review folder in MFS. The function grants the peer reviewee (and no one else) permission to view the peer review file. The viewing function simply scans the peer review folder and returns the only submitted peer review file which this Kerberos token has access to.

#### 4.1.5 Gradescope Integration

Submitting assignments through Gradescope is handled by Gradescope – BeaverCMS simply needs to pull the grades from Gradescope, which we discuss later.

#### 4.1.6 Parallel Uploads

Our submission queue can be as large as necessary – so if all 500 students submit at the exact same time, the queue will have size 500. Putting a new submission request on the queue is a relatively quick process. Putting tasks on a queue and picking from them allows for a more orderly and resourceful approach to parallelizing uploads to the extent possible.

We choose to have 100 different submission processes running simultaneously since we have 10 cores and we'd like to avoid resource contention as much as possible while ensuring that submissions are still relatively fast during bottleneck times (i.e 2 minutes before an assignment

is due). If we assume 90% of students will submit in the 2 minutes before the deadline, we have an average case of 3 submissions every second, so we should be able to ideally support a worst case of 10 submissions every second. As we discuss in Section 5, the average case file upload is 10 seconds, so 100 threads would allow us to support 10 submission every second. As more students are added to the class, we will want to devote more threads to the uploading in order to still be able to support these worst case scenarios.

Allowing many concurrent submission processes to run at the same time accelerates the upload process and ultimately improves the reliability of the system. If a file slows down one submission process or if there is high traffic due to many students attempting to upload at once, this system has built-in redundancy to ensure that a student's valid submission will be recorded and stored. As we talk about in our evaluation section, using this many processes gives us a sufficiently fast expected upload time.

#### 4.1.7 Killing Uploads

In the case of a brown-out, the client automatically kills file uploads in process. The client can also voluntarily kill file uploads if desired.

For uploads that are in the submission queue, their requests are simply deleted from the queue. Once a server process is processing the upload, however, the kill transaction becomes more nuanced. In the case of non-video file uploads, if it's already left the submission queue, we ignore the kill and proceed with the upload anyways (so it's too late to kill a transaction at that point). With assignment file uploads, a student can always submit a new file on top of this one, mark it final, and it's essentially equivalent to killing the original file upload anyways.

In the case of video uploads, we currently upload them to MFS through MSS. If the video upload has already begun, then we can use the MSS kill function to kill the current video upload prematurely. Videos are uploaded via chunks, so the kill signal is sent in between chunks. This way if a user cancels their upload, the file transfer is killed before the entire file is uploaded. Then, the client can also delete all chunks that were already uploaded to MFS.

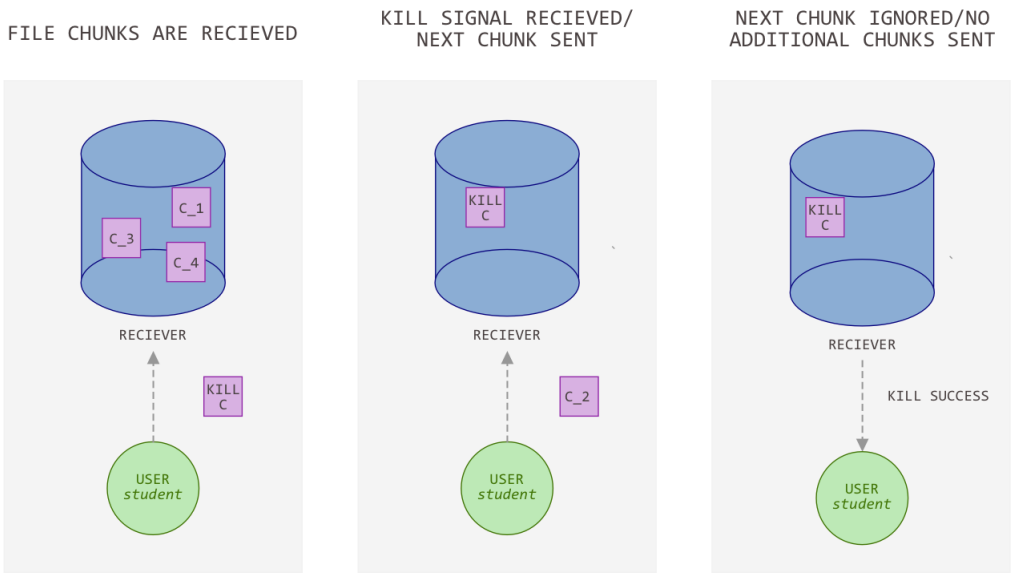


Figure 3: Killing Uploads

### 4.1.8 Reliability

Our submission protocols prioritize system reliability, so that we can provide as strong a guarantee as possible that when a student thinks a file is submitted, it actually is. To this extent, we provide a notification system where upon every successful file upload, the student receives an email notifying them of that. Whenever a submission process finishes processing a submission request, it sends an automatic server-generated email to the Kerberos ID of the submitter notifying them that their submission was successful. Users should not consider their files successfully uploaded until they receive that confirmation email.

To further bolster reliability of the system, BeaverCMS also provides functionality for a client to verify whether an upload was successful. If a client sends a verification request with a Kerberos ID and a file name, BeaverCMS can authenticate the Kerberos ID and check in the corresponding directory if a file with the given file name exists. BeaverCMS then return a true/false result to the user. Figure 4 illustrates the work flow of submitting a file and periodically checking to ensure receipt.

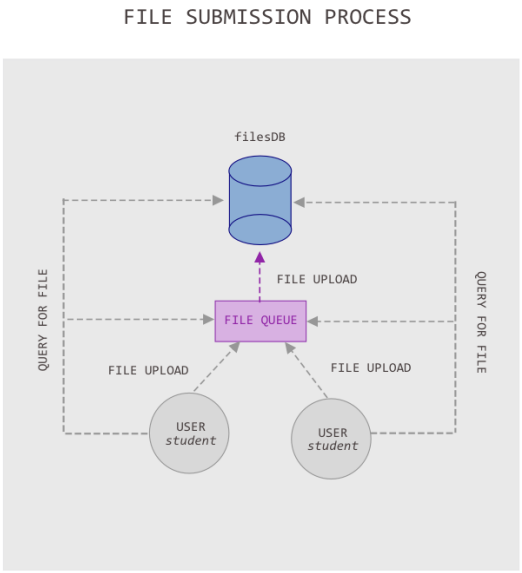


Figure 4: When a file is submitted, it is placed in a queue. The client then periodically queries the server to determine the status of their upload.

## 4.2 Grading Assignments

### 4.2.1 Staff Grading

In order for a staff member to grade assignments on the MFS, they must pull the assignments from the file system, grade them and write comments in a file, and query the server to submit the grade and file with comments.

When a recitation team staff member wants to download in bulk all of the submissions from their recitation team for a given assignment, they can send a request to BeaverCMS with the assignment name. BeaverCMS can query *recitationTeamsDB* to receive a list of student Kerberos names and query MFS using *download_file* to extract and return the desired files from the "FINAL" directories of these students. If this staff member only wants to download a specific student's submission for one assignment, they can query BeaverCMS with the student's Kerberos

name and assignment name, and BeaverCMS will again query MFS to extract and return the desired file. Both of these methods of extracting submitted assignments from the file system abstract away almost all of the work from the staff member, achieving greater simplicity than the current system.

Upon grading the received submission, the staff member will send BeaverCMS the grade and/or comments along with the student's Kerberos name and assignment name. BeaverCMS names the comments file accordingly and adds it to the student's or team's "FINAL" directory and adds the grade to the *gradesDB* database.

For bulk grading submissions, BeaverCMS allows the staff member to submit a CSV containing each student's Kerberos name, their grade, and a text field containing their comments. BeaverCMS has an internal function that can parse this CSV and process each grading request individually, as described above.

In order for the publication function to work and for all students to see their grades at the same time, the grades are initially designated as unpublished and the client will not display those grades to the student. In order to publish the grades, the staff member runs the publish function along with the name of the assignment. This will update the status of the grades such that the client allows the students to view their grades.

If there are multiple graders grading the same submission by the same student for the same assignment, the system treats this as multiple grading events which corresponds to multiple entries in the *gradesDB* database.

For certain assignments, recitation team staff can manually enter the grades for each student or team. In this case, the recitation team staff can send such a request to BeaverCMS with either a single student or team Kerberos and a grade, or a dictionary mapping Kerberos names to grades. BeaverCMS will authenticate the staff member and ensure the student(s) are in the recitation team, and then add these values to the *gradesDB* database. In these cases, BeaverCMS will also send an email to the student whose grade was manually changed, notifying them of the change.

### 4.2.2 Video Voting

The server exposes a function to vote for videos, whereby a student can submit an ordered list of the Kerberos names of the five videos that they would like to vote for. The function verifies the Kerberos token (and that it corresponds to a student) and adds the Kerberos names to the *videoVoteDB* relational database. Upon the voting period ending, a quick SQL query allows BeaverCMS to determine the winning videos.

### 4.2.3 Grade Reports

We expose a server function that may be used by the course admin and all recitation team staff and returns a CSV containing an up-to-date grade report of all students. If the Kerberos token is from course admin's Kerberos ID, then it returns a grade report of all students, and if it's a recitation team staff member, then it returns a grade report of all students in that recitation team. We also allow the user to specify a specific student's Kerberos ID, and they will only receive the grade report for that student. Finally, the user can also specify a list of assignment names that they would like to get a grade report for (the default is all assignments). This function works by scraping the *gradesDB* relational database and formatting it in a CSV format. This function can be used by the course admin at the end of the semester for the grades meeting, and it can also be used by recitation team staff who want to check on students' performance in the middle of the course.

#### 4.2.4 Grade Penalties

In the current implementation, deadlines and grading penalties are handled manually. BeaverCMS automates the late penalty system, while allowing for staff to manually change the penalty in extenuating circumstances. Upon a submission to the MFS, BeaverCMS queries *assignmentsDB* and compares the listed due date with the current date. If this assignment is late, it marks the number of hours late in *submissionDB* and uses the late penalty rate in *assignmentsDB* to compute a late penalty multiplier and puts this in *gradesDB*.

Recitation staff members will have access to a server function to change the late penalty multiplier in *gradesDB*, to be used in special circumstances ($S^3$ note excusing lateness, submission error, TA excuse, etc.).

#### 4.2.5 Gradescope Integration

Pulling grades from Gradescope is the only case where BeaverCMS must interact with a third-party system. As such, this protocol is made as simple as possible to avoid over-reliance on third-party support.

To pull grades from Gradescope, calls to the *pull_gradescope_grades()* function are made periodically up until the regrade period is over. The result is stored as a CSV data structure and then parsed into inputs for *gradesDB*.

Grades are pulled every minute in the current system. A frequency of once per minute is chosen to keep the BeaverCMS grades almost perfectly in sync with Gradescope. Since the cost of running *pull_gradescope_grades()* is under a minute (justified in Section 5), it suffices to have a single process responsible for pulling grades every minute. Syncing more frequently might require multiple processes, which would be an unnecessary use of resources, and syncing less frequently might lead to BeaverCMS and Gradescope being noticeably out of sync. Since the resource use is limited anyways, we choose to optimize for consistency between BeaverCMS and Gradescope.

#### 4.2.6 User Simplicity

Throughout the staff grading workflow, we prioritize user simplicity. The major reason for this is that there are often few staff responsible for many students and many grades – a single recitation instructor might have more than 36 students to grade for every assignment. As a result, even slight inefficiencies in usability and the user experience multiply out to be major wastes of time on the part of the course staff. For this reason, we implement an automated late assignment policy, automated syncing between Gradescope and BeaverCMS, and automated production of Grade Reports. Moreover, the staff grading workflow is designed so that staff members can download assignments in bulk for their sections, and submit grades/comments either one at a time or in a bulk format. We hope that these design choices, while they tradeoff implementation simplicity, make the user experience more seamless and efficient.

Where possible, we do preserve the flexibility of the system, giving staff members the ability to override the automated processes (in late policy and *gradesDB*), preserving the strengths of the current system.

# 5 Evaluation

## 5.1 Quantitative Evaluation

### 5.1.1 Communication Overhead

Each query has a 12KB overhead due to the Kerberos verification [1]. For file uploads the components to the upload are:

- Assignment name (20 characters, 20 bytes)
- Course name (10 characters, 10 bytes)
- Home directory (Max path is 260 characters, 260 bytes)
- date (8 bytes)

The upload components add up to <1KB and the Kerberos authentication is 12KB. Overall, there is a ∼13KB constant overhead and a 2x overhead since we need to send the file to server and then server needs to send to MFS.

### 5.1.2 Assignment Uploads

From the system specifications, it is known that:

- MIT backbone network runs at 100 GB/s
- The connection outside MIT is 10 GB/s
- Each wired Ethernet link runs at a maximum of 1 GB/s
- Each wireless link can support up to 600 MB/s
- Each edge device can be supported at 500 MB/s

For the purpose of this measurement, it is assumed that 10 MB/s is available to the user as they may have varying network conditions. The bottleneck in the system is the computer upload speed unless there is a brownout in the system.

**Average Case**

In the average case the following is assumed:

- There is 500 MB/s for the network
- The average video size is 100 MB
- MIT runs at 10 GB/s.

From this we can determine that the average time for a file upload is:

$$100MB * \frac{1}{10MB/s} = 10s \tag{1}$$

Therefore, the average file upload time is 10 seconds.

**Additional Case: 400 Simultaneous Submissions**

To ensure that our system works in even the worst case scenario, it is able to handle 400 simultaneous submissions in the average file upload case.
We assume the same conditions in the previous case:

- There is 500 MB/s for the network
- The average video size is 100 MB
- MIT runs at 10 GB/s.

Even if 1,000 students upload at the same time, the network is able to handle the traffic since:

$$10MB/s * 1000 < 10GB/s \tag{2}$$

**Worst Case**

The worst case upload time is the maximum upload time that the server allows before a upload fails. We set this time to be 10 minutes in our system. This supports the following worst case file upload scenario:

- 0.84 MB/s DSL connection [2]
- 500 MB Video File

We choose 10 minutes at the upper bound on upload times because:

$$500MB * \frac{1}{0.84MB/s} = 600s = 10minutes \tag{3}$$

If a file takes longer than 10 minutes to upload, then the upload fails.

#### 5.1.3 Gradescope Grades

The system pulls grades every minute.

In the average case, it takes a minute from a grade on Gradescope to be transferred to the server. In the worst case, a grade is uploaded to Gradescope right before a brown-out. In that case, the upper bound on a brown-out time is 30 minutes. This means that in the worst case, it takes 31 minutes for a grade to be transferred to the server.

#### 5.1.4 Server Data Storage

To determine how much data is put on the storage, we assume the following:

- 9 databases, per our design
- 240 GB SSD, per the project specification
- 400 students, per the design specification
- 20 assignments per semester
- 18 sections and recitation teams
- Each row take up at most 1KB

With the previous assumptions, we find that the size of each relational database is as follows:

By summing up the size of each database, we find that each semester we have approximately 17,390 KB of relational database data. However, the roughly 130 DP teams each produce a 100 MB video, along with about 40 MB in PDF files for the assignments in the class, which takes up 18 GB. It's easy to see the file size is the dominant factor over the relational database.

By this estimate, we find that the current system can handle a 10X increase in data before overloading the SSD and Main Memory.

| Database | Size of Database |
|---|---|
| gradesDB | #assignments*#students*1 KB = 8000 KB |
| assignmentsDB | #assignments * 1 KB = 20 KB |
| usersDB | #students * 1 KB = 400 KB |
| teamsDb | #teams = #students/2 = 134 KB |
| sectionsDB | #sections * 1 KB = 18 KB |
| sectionAssignmentsDB | #students*1 KB = 400 KB |
| identitiesDB | #students*1 KB = 400 KB |
| submissionsDB | #assignments*#students = 8000 KB |
| recitationTeamsDB | #sections = 18 KB |

Table 2: Estimate of Each Database Size

#### 5.1.5 Scaling

The two major bottlenecks to our system scaling are server storage space and handling the average case where a large fraction of students upload videos at the same time. Both of them scale linearly with the number of students. Given our calculations above, we can support roughly 10x the number of students before our server runs out of memory. In terms of the average case video upload time, we can support 10x the number of students if we allow our video uploads to take an average of 2-3 minutes each, which isn't entirely unreasonable, especially considering the email notification system we have in place.

We could easily handle doubling the number of students in 6.033, however, we wouldn't be able to use this system for every class in EECS without an increase to our server memory.

#### 5.1.6 Grade Downloads

There are #assignments*#students = 20*400 = 8000 grades in the system. Taking the same assumptions from 5.1.4, all the grades take up at most 8000 KB.

It can be assumed that the course lecturer has a standard internet connection. As mentioned in 5.1.2, that speed is 10Mb/s. When the course lecturer runs the download grades function, it takes 7.8 seconds to download all of the grades. That comes from the following:

$$8000KB * \frac{1MB}{1024KB} * \frac{1s}{10MB} = 0.78s \tag{4}$$

#### 5.1.7 File Transfer Kills

As mentioned in section 4, when a file is uploaded, it is split into 1Mb chunks as it is sent over the network. Assuming an average network speed of 10Mb/s, a file transfer can be cancelled within 0.1s. Even with a slow network connection of 1Mb/s, a file transfer can be cancelled within 1s.

For a user to be notified that the transfer was killed, the user needs to receive a cancellation confirmation from the server. The server sends out the confirmations via email. The standard time for an email delivery is around 5 seconds. The expected notification time is 5 seconds for a transfer kill.

#### 5.1.8 Student Account Creation

In order to estimate the time it takes to create all student accounts, the following assumption are taken into account:

- 400 students, per the design specification

- 1 second to create a Kerberos ID
- 1 second to create a directory in MFS

The bulk of the computation time is in creating the Kerberos IDs through MIDS (for DP teams, recitation teams... etc) and creating the home directories in MFS. Initializing the relational databases should be very fast. With the assumptions above, this takes about 13 minutes.

## 5.2 Qualitative Measurements

### 5.2.1 Usability

Usability for both students and staff members was a key factor in designing this system, as this is the area in which current infrastructure most lacks.

#### Student

BeaverCMS offers some functionality that improves student usability from the current course infrastructure. The system has built-in automated functionality for submitting as a group and accessing group files. In this system, all members of a Design Project group can see submitted files and feedback. The file upload notification system allows students to receive email confirmations about the success of a file upload. This is beneficial in the case of network outages prevent a student's assignment from being submitted properly. Additionally, grades are pulled every minute from Gradescope to minimize lag time for a student's grades to be updated in the system.

#### Staff

Course staff will also find the system significantly more usable than the current system as it was designed to minimize the number of manual tasks necessary. At the beginning of the course, the system automatically populates the database with students, recitations, and sections. Permissions are granted to course staff to make any changes to a specific student's recitation or section, and the server handles all changes in student or group permissions in the database automatically. As a function of the database structure, system also provides staff the ability to easily pull grades from all students in a grade report at the end of the semester to submit to the registrar. Additionally, where grade penalties are currently manually assigned by course staff, BeaverCMS automatically enforced grade penalties and gives staff permissions to manually adjust them in certain exceptional circumstances.

The grades populate every minute, which allows for user to have a more accurate snapshot of their grades. The interval of a minute was chosen such that the user would have an optimal user experience.

File cancellations are all very speedy. It takes less than 0.1 seconds to cancel a file. The notification speed of 5 minutes (the time it takes to send an email) could be more optimal, but in this case reliability was chosen as a higher priority for the user. The email notification is still the most reliable method to deliver a notification.

The file transfer time out interval of 10 minutes is also an optimal length for the user. The time was deliberately chosen to be as short as necessary to allow the user to be notified as soon as possible in the event of a failure. This quick feedback is especially important when it is close to the project deadline. The time out window is no shorter than 10 minutes because then the system becomes less usable as large files and/or slow connections could trigger upload failures.

#### 5.2.2 Flexibility

The scaling concerns of our system have been discussed thoroughly, so we think the biggest bottlenecks to flexibility are the ability to create isolated instances of courses and support different assignment or grading flows. Our system is not specific to any particular assignment or assignment structure, and we support general creation of assignments, submissions, and late penalties. However, some classes may desire completely different submission or grading workflows, which our system does not provide at the moment. Some classes may want to integrate third party software beyond Gradescope, which we do not currently support. We also would need to modify our relational database structure to account for multiple isolated classes. Our file hierarchy and home directory model, however, can support multiple classes.

Because of the general nature of the file hierarchy and upload processes, we don't anticipate any issues with different file formats.

## 6 Conclusion

Our proposed system accomplishes the goals set out by the 6.033 staff: support of forming teams, sharing works in progress, submitting group assignments, providing a uniform place to view grades, and allowing for additional video upload functionality. Built with simplicity in mind, our file system uses an intuitive naming structure, and our grading process automates much of the work on behalf of the course staff. Our system achieves a comparable level of security as the current implementation through integration with MIDS and proper access controls on our databases. To ensure reliability in our system, our central server supports parallel submission processes and is fault tolerant in the case of a brown-out.

## 7 Author Contribution

All of us worked on creating the high level design. **Nyle Sykes** primarily worked on the file hierarchy design. **Fiona Zhang** worked on the grade file system and the diagrams. Additionally, she led efforts to generate estimates for the evaluation section. **Ashwath Thirumalai** primarily worked on the networking design. All of us had a role in writing each section of the report and revising based on feedback.

## 8 Acknowledgments

A special thanks to Olivia Brode-Rogers and Jessie Stickgold-Sarah for their feedback. We would also like to thank Katrina LaCurts and the course staff for an instructive and engaging semester.

## References

[1] ShaneC33. *MaxTokenSize and Kerberos Token Bloat.* Just Blog'n, blogs.technet.microsoft.com/shanecothran/2010/07/16/maxtokensize-and-kerberos-token-bloat/.

[2] *The Average DSL Connection Speed.* Techwalla, www.techwalla.com/articles/the-average-dsl-connection-speed.