Distributed Grading

Team Members

Jim Danz, Stefan Muller, Kenny Yu, Tony Ho, Willie Yao, Leora Pearson, Rob Bowden

March 29: Preliminary specification. This should take the form of a short (2-3 page) written document

- What open source you can/will use
- What services are involved
- How the services find each other
- What the service interface is

Problem Statement

In academic classes, grading students' work can easily be the most time-consuming aspect of teaching a course. At the same time, the time required to do a thorough job grading does not seem to be commensurate to the value provided per student. Six hours spent grading assignments for 60 students leads only to a rushed job of grading -- 6 minutes per student. It seems likely (or at least conceivable) that the 60 students as a whole would benefit more from those 6 hours of instructor-time being spent on course development -- lectures, assignments, supplementary material.

Furthermore, the instructor is likely to feel that he or she is not personally getting any value of trudging through student assignments. The goal of our project is to empower students to grade assignments in a way that is both instructive to the students and safe against rigging. We intend to build a distributed system to from the infrastructure that facilitates this goal.

System Overview

The first iteration (V0) of our system will be modeled after the process that Professor Waldo is currently conducting manually for cs262.

Specifically, we will build a system that:

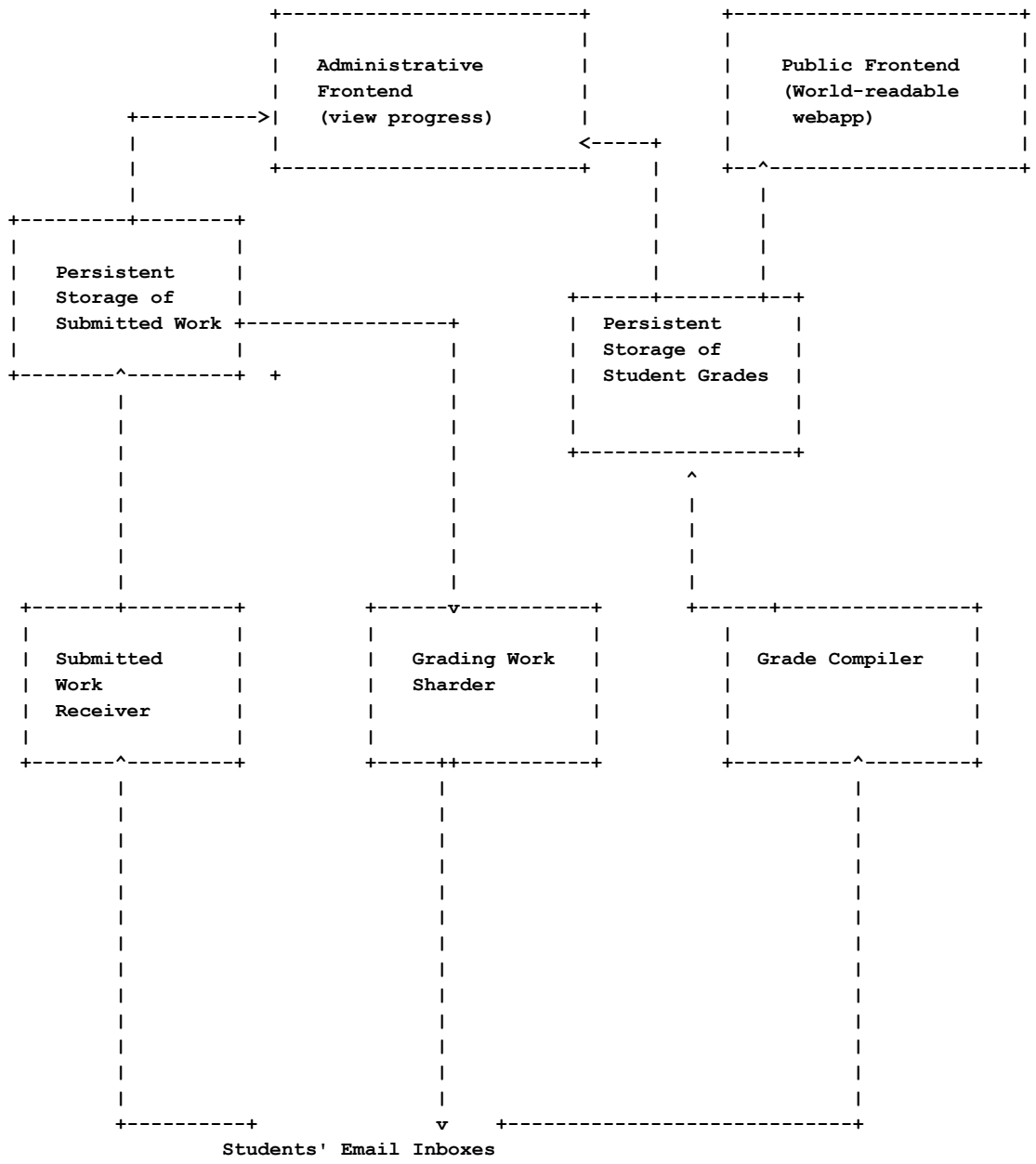
- Receives papers as email attachments
- On a paper deadline, randomly chooses which student should grade which paper, and sends out grading assignments
- Upon receipt of grading assignments, makes graded work available on a world-viewable website

Services

This entails the following services

- Persistent Storage of Submitted Work
- Persistent Storage of Student Grades
- Submitted Work Receiver: A service to receive work submitted by students
- Grading Work Sharder: A service to comb over submitted work and shard into into grading assignments
- Grade Compiler: A service to receive the grades that students submit and compile scores together
- Two frontends: one for an administrator (course head) to view rich information, and additionally a public frontend for students to see anonymized versions of their scores

Services



Plans to Use Open Source Software

- Zookeeper - we may use this as a distributed lock system
- Java RMI - we will use this as our main RPC mechanism
- MySQL (?) - for persistent storage
    - we may use other non-RDBMS database systems that are key-value stores (e.g. Cassandra, MongoDB) to allow us to easily replicate the persistent storage systems for increased fault tolerance and higher availability (consistency will depend on how these systems deal with the CAP

tradeoff)

- Hadoop MapReduce, and/or Storm (https://github.com/nathanmarz/storm/wiki/Rationale) - for continuous data processing
    - useful because we will receive grades in a continuous fashion
    - allows for easy parallel realtime computation

How Services will Find Each Other

- The services will use RPC (using Java's RMI) to contact each other.
- We will have the services find each other through a configuration file in the format:
    - name of service -> local DNS name for that service

Service Interface

- Persistent Storage of Submitted Work
    - bool put(studentID, workID, workObject)
    - workObject get(studentID, workID)
    - Set<(workID, workObject)> getAllWork(studentID)
    - Set<studentID, workObject> getSubmissions(workID)
- Persistent Storage of Student Grades
    - bool put(studentID, workID, Grade)
    - Grade get(studentID, workID)
    - List<Grade> getAllGrades(studentID)
    - List<Grade> getGradesForSubmission(workID)
- Submitted Work Receiver: A service to receive work submitted by students
    - bool submit(studentID, workID, workObject)
- Grading Work Sharder: A service to comb over submitted work and shard into into grading assignments
    - (shardID, Map<studentID, studentID>) generateShard(workID)
      // generate a studentID -> studentID (grader ---> gradee)
    - Set<studentID> graders(studentID, workID) // get the set of graders for a particular (student,assignment)
    - Map<studentID, studentID> shard(shardID)
      // shardID's will allow us to maintain different versions of shards
- Grade Compiler: A service to receive the grades that students submit and compile scores together
    - bool grade(studentID, workID, Grade)
    - studentID grader(Grade) // retrieve the grader who gave this value
    - (gradeID, Map<studentID, Grade>) compiledGrades(workID)
    - Map<studentID, Grade> grades(gradeID) // allow us to maintain a history of grades so far
- Two frontends: one for an administrator (course head) to view rich information, and additionally a public frontend for students to see anonymized versions of their scores

Plans After v0

As we've stated, our v0 entails simply building a distributed system around the current partially-manual workflow used for grading in cs262.

Upon completion of this, time permitting, we will expand the functionality of our system in the following ways:

- Allow the instructor to specify that an assignment is "split up" in a certain way, so that grading can be sharded at a level of granularity smaller than that of an entire student's submission. The main interpretation here would be problem sets (such as math or theoretical computer science problem sets, as well as some problem-based programming problem sets), with the goal being that grading can be "by problem" rather than one student grading another students' entire submission.
    - To do this, we will likely develop a simple format for an instructor to specify how a problem set is broken up.
    - We may need to build in some error handling to facilitate this (i.e., to reject students' work if they have broken the formatting that divides their problem set)
- Allow instructors to weight grades in unequal ways, and introduce rules around who should grade what
    - After a few assignments, instructors will have access to an interesting trove of data, including:
        - For each student (and for each assignment), the distribution over grades that he has **given**
        - For each student (and for each assignment), the distribution over grades that he has **received**
    - Instructors may want to make use of this data to make decisions about who should grade what, and what weight should be given to grades made by different students.
    - Since this is a distributed systems class rather than a machine learning class, we will not make these functions the emphasis of our work. However, we will provide a clean mechanism by which an instructor can define their own *grade sharding and weighting service**,*** that implements a function from: grading history -> (who should grade what * what weight should be given to each grade)